

YILDIZ TEKNİK ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

'Relp' Par. Prog. Dili Tas. ve Uyg.

DOKTORA TEZİ

Süha Futacı

1995

YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

‘RELP’ PARALEL PROGRAMLAMA DİLİ
TASARIMI VE UYGULAMASI

Bilgisayar Yük.Müh. Süha FUTACI

F.B.E. Bilgisayar Bilimleri Mühendisliği Anabilim Dalında
hazırlanan
DOKTORA TEZİ

Tez Danışmanı : Prof. M.Yahya KARSLIGİL (Y.T.Ü.)

İSTANBUL, Mart 1995

YILDIZ TEKNİK ÜNİVERSİTESİ
KÜTÜPHANE DOKÜMANTASYON
DAİRE BAŞKANLIĞI

Kot	: R 368 76
Alındığı Yer	: F.B.Enstitüsü
Tarih	: 19.9.1996
Fatura	: -
Fiyatı	: 500 Bin
Ayniyat No	: 1/7
Kayıt No	: 52476
UDC	:
Ek	:

Y.T.O.

KÜTÜPHANE DOK. DAI. BAŞKANLIĞI

İÇİNDEKİLER	36
TEŞEKKÜR	ix
TÜRKÇE ÖZET	x
İNGİLİZCE ÖZET	xi
1. GİRİŞ VE GENEL BİLGİLER :	1
1.1 Dağıtılmış Bilgisayar Sistemleri :	1
1.2 Dağıtılmış Bilgisayar Sistemlerine Yönelik Programlama Dilleri	4
1.3 Eş Zamanlı Mesaj İletimine Dayanan Paralel Programlama Dilleri	8
1.3.1 "OCCAM" Programlama Dili :	8
1.4 Asenkron Mesaj İletimine Dayanan Paralel Programlama Dilleri	12
1.5 İletişim ve Eş Zamanlı Çalışmada Randevu Mekanizmasını Kullanan Paralel Programlama Dilleri	13
1.5.1 "Ada" programlama dili	13
1.5.2 "Concurrent C" programlama dili	14
1.6 Nesne Temelli Paralel Programlama Dilleri	15
1.6.1 "EMERALD" Programlama dili	16
1.7 Paralel Fonksiyonel Programlama Dilleri	16
1.7.1 "ParAlfl" programlama dili	17
1.8 Mantıksal Paralel Programlama Dilleri	19
1.8.1 "Concurrent PROLOG" programlama dili	19
1.8.2 "PARLOG" programlama dili	20
1.9 Dağıtılmış Veri Yapılarına Dayanan Paralel Programlama Dilleri	22
1.10 Veri Akış Paralel Programlama Dilleri	24
1.10.1 "VAL" programlama dili	25

1.11 Genel deęerlendirme	26
2. "RELP" PROGRAMLAMA DİLİNİN GENEL TANITIMI	28
2.1 "RELP" Dilinin hedeflenen uygulama alanı (Domain) .	28
2.1.1 Karar Destek Sistemleri	28
2.2 "RELP" Dilinin tasarlanmasının amacı	30
2.3 "RELP" Dilinin uygulanmasının amacı	32
2.4 "RELP" Dilinin genel özellikleri	32
2.5 "RELP" Dilinin bloklarının genel tanıtımı	40
2.5.1 "GENERATE" (Generate Attribute) bloęu genel yapısı	41
2.5.2 "GENERATEE" (Generate Element) bloęu genel yapısı	41
2.5.3 "GENERATET" (Generate Tuple) bloęu genel yapısı	41
2.5.4 "TEST" bloęu genel yapısı	42
2.5.5 "APPLY" bloęu genel yapısı	42
2.5.6 "APPLY CUMULATIVE" bloęu genel yapısı . . .	42
2.5.7 "COMPUTE" bloęu genel yapısı	42
2.5.8 "COMFIX" (Compute Fix) bloęu genel yapısı .	43
2.5.9 "INPUT" bloęu genel yapısı	43
2.5.10 "REPORT" bloęu genel yapısı	43
2.5.11 "TRANSFER" bloęu genel yapısı	43
2.5.12 "SORT" bloęu genel yapısı	43
2.5.13 "COUNTBY" (Count By) bloęu genel yapısı . .	44
2.5.14 "SINK" bloęu genel yapısı	44
2.5.15 "FILTER" bloęu genel yapısı	44
2.5.16 "CARTESIANP" (Cartesian Product) bloęu genel yapısı	44

2.5.17 "APPEND" bloğu genel yapısı	45
2.5.18 "DELETE" bloğu genel yapısı	45
2.5.19 TXCOUNT (Transaction Count) bloğu genel yapısı	45
2.5.20 "COND" (Condition) bloğu genel yapısı	45
2.5.21 "NOTMEMBER" (Not Member Test) bloğu genel yapısı	46
2.5.22 "MERGE" (Merge Transactions) bloğu genel yapısı	46
3. RELP DİLİ GENEL SÖZ DİZİMİ YAPISI	46
3.1 Genel	47
3.1.1 Programın ana yapısı	47
3.1.2 İsimlendirme kuralları	47
3.2 Blokların söz dizimi yapıları	49
3.2.1 "INPUT" bloğu söz dizimi yapısı	49
3.2.2 "GENERATEE" (Generate Element) bloğu söz dizimi yapısı	50
3.2.3 "GENERATET" (Generate Tupple) bloğu söz dizimi yapısı	51
3.2.4 "GENERATE" (Generate Attribute) bloğu söz dizimi yapısı	51
3.2.5 "APPLY" bloğu söz dizimi yapısı	52
3.2.6 "REPORT" bloğu söz dizimi yapısı	53
3.2.7 "COMPUTE" bloğu söz dizimi yapısı	54
3.2.8 "COMFIX" (Compute Fix) bloğu söz dizimi yapısı	56
3.2.9 "TEST" bloğu söz dizimi yapısı	56
3.2.10 "CARTESIANP" (Cartesian Product) bloğu söz dizimi yapısı	58

3.2.11 "SORT" bloğu söz dizimi yapısı	59
3.2.12 "TRANSFER" bloğu söz dizimi yapısı	59
3.2.13 "COUNTBY" (Count By) bloğu söz dizimi yapısı	60
3.2.14 "FILTER" bloğu söz dizimi yapısı	61
3.2.15 "SINK" bloğu söz dizimi yapısı	61
3.2.16 "MERGE" bloğu söz dizimi yapısı	61
3.2.17 "TXCNT" (Transaction Count) bloğu söz dizimi yapısı	62
3.2.18 "COND" (Condition) bloğu söz dizimi yapısı	62
3.2.19 "NOTMEMBER" (Not Member Test) bloğu söz dizimi yapısı	64
3.2.20 "APPLY CUMULATIVE" bloğu söz dizimi yapısı	65
3.2.21 "APPEND" bloğu söz dizimi yapısı	66
3.2.22 "DELETE" bloğu söz dizimi yapısı	66
4. "RELP" PROGRAMLAMA DİLİ ANLAMSAL YAPISI	67
4.1 Genel	67
4.1.1 Kuyruk ve Hareket birimleri (transaction)	67
4.1.2 İsimlendirme ortamları (context) ve kapsamları (scope)	68
4.1.3 İsim bildirimleri	68
4.1.4 Değişken ve saha tipleri	69
4.1.5 Değişkenler, aritmetik ve mantıksal ifadeler	69
4.2 Blokların anlamsal yapılarının ayrı ayrı açıklanması	69
4.2.1 "INPUT" bloğu anlamsal yapısı	70

4.2.2	"GENERATEE (Generate Element) bloğu anlamsal yapısı	71
4.2.3	"GENERATET (Generate Tupple) bloğu anlamsal yapısı	71
4.2.4	"GENERATE" (Generate Attribute) bloğu anlamsal yapısı	72
4.2.5	"APPLY" bloğu anlamsal yapısı	72
4.2.6	"REPORT" bloğu anlamsal yapısı	74
4.2.7	"COMPUTE" bloğu anlamsal yapısı	74
4.2.8	"COMFIX" (Compute Fix) bloğu anlamsal yapısı	77
4.2.9	"TEST" bloğu anlamsal yapısı	78
4.2.10	"CARTESIANP" bloğu anlamsal yapısı	79
4.2.11	"SORT" bloğu anlamsal yapısı	80
4.2.12	"TRANSFER" bloğu anlamsal yapısı	81
4.2.13	"COUNTBY" (Count By) bloğu anlamsal yapısı	81
4.2.14	"FILTER" bloğu anlamsal yapısı	82
4.2.15	"SINK" bloğu anlamsal yapısı	82
4.2.16	"TXCNT" (Transaction Count) bloğu anlamsal yapısı	83
4.2.17	"COND" (Condition) bloğu anlamsal yapısı	83
4.2.18	"NOTMEMBER" (Not Member Test) bloğu anlamsal yapısı	85
4.2.19	"APPLY CUMULATIVE" bloğu anlamsal yapısı	86
4.2.20	"APPEND" bloğu anlamsal yapısı	87
4.2.21	"DELETE" bloğu anlamsal yapısı	87
4.2.22	"MERGE" bloğu anlamsal yapısı	88

4.3 RELP programlama dilinin "ilişkisel tamam" (Relationally Complete) olduğunun gösterilmesi	89
5. RELP DİLİNİN SORGULAMA ALT KÜMESİNİN BAZI SORGULAMA DİLLERİ İLE KARMAŞIKLIK YÖNÜNDEN KARŞILAŞTIRILMASI	95
5.1 Operatör karmaşıklığı	96
5.2 Sorgulama karmaşıklığı	97
5.3 Programlama dili karmaşıklığı	97
5.4 Mutlak karmaşıklık	97
5.5 İlk karmaşıklık (Initial Complexity)	98
5.6 RELP dilinin karmaşıklığı	98
5.7 Örnek bir veri tabanı ve sorgulamalar kümesi temel alınarak RELP dilinin diğer sorgulama dilleri ile karmaşıklık yönünden karşılaştırılması	101
5.7.1 Sorgulama karmaşıklıklarının karşılaştırılması	101
5.7.2 RELP programlama dili karmaşıklığı	108
5.7.3 Karşılaştırma sonuçları	112
6. DİLİN UYGULAMASI (IMPLEMENTATION)	112
6.1 Genel	112
6.1.1 Dilin işleyiş yapısının kısa açıklaması	113
6.1.2 Dil uygulamasının fazları, kararlar ve sebepleri	115
6.1.2.1 İlk faz, dil işleyiş yapısının tasarımı	115
6.1.2.1.1 Performansı etkileyen hususlar	115
6.1.2.1.1.1 İletişim düşünceleri	115
6.1.2.1.1.2 Süreç başlatma-program yükleme	116

6.1.2.1.1.3 Sanal işlemci kullanımı ve yük dağıtımı	116
6.1.2.1.2 Kod üretimi ve derleme	116
6.1.2.1.3 Program çalıştırma-çalışma zamanı ortamı	117
6.1.2.2 İkinci faz, kişisel bilgisayar uygulaması	117
6.1.2.3 Üçüncü faz, UNIX tek işlemcili platform üzerinde uygulama	118
6.1.2.4 Dördüncü faz, derleyici geliştirme	118
6.1.2.5 Beşinci faz, çoklu işlemci ortamında denemeler	119
6.2 Dilin işleyiş yapısının detaylı açıklaması	119
6.2.1 Veri yapıları	120
6.2.1.1 Sanal işlemci (süreç) tablosu	120
6.2.1.2 Süreç aktivasyon tablosu	122
6.2.1.3 Mesaj bloğu veri yapıları	122
6.2.1.4 Kuyruk tablosu	122
6.2.2 İlişki tablosu yapısı	123
6.2.3 Global tablolar	125
6.2.4 Sanal işlemcilerin genel yapısı	128
6.2.4.1 Sanal işlemci bloklarında yapılan özel uygulamalar	129
6.2.5 Derleyicinin kaynak programı icra dizisine çeviriş yöntemi	130
6.2.5.1 RELP Derleyicisinin LEX ve YACC kullanılarak oluşturulması	132
6.2.5.2 Derleyicide kullanılan önemli veri yapıları ve değişkenler	134
6.2.5.3 Derleyici algoritması	136

7. DENEY ORTAMI VE DENEY SONUÇLARI	145
7.1 Genel	145
7.2 Deneyin amacı	145
7.3 Deney şartları	147
7.3.1 Deneyin gerçekleştirildiği donanım platformu	147
7.3.2 Prototip sistem yazılım geliştirme ve deney ortamı	148
7.4 Ölçümlerin daha hassas olması için alınan önlem	148
7.5 Ölçülecek büyüklük	148
7.6 Ölçüm birimi	148
7.7 Ölçüm için seçilen uygulamalar	148
7.8 Deneylerin ve sonuçlarının sergilenmesi	149
7.9 Diğer deneyler	151
7.10 Deneylerdeki saptamalar	156
7.11 Deney sonuçlarının değerlendirilmesi	159
8. TARTIŞMA, SONUÇ VE ÖNERİLER	160
KAYNAKLAR	168
EK-1 RELP PROGRAMLAMA DİLİ SÖZ DİZİMİ YAPISININ BNF(BACKUS NAUR FORM), EBNF (EXTENDED BACKUS NAUR FORM) VE SÖZ DİZİMİ DİYAGRAMLARI KULLANILARAK AÇIKLANMASI	175
EK-2 DENEY SONUÇLARI	207
EK-3 YACC VE LEX ARAÇLARI İÇİN YAPILAN TANIMLAMALAR.	238
EK-4 DİL ORTAMI VE SANAL İŞLEMCİLERİN KAYNAK KODLARI.	296
EK-5 LÜGATÇE	447
ÖZGEÇMİŞ	

ÖZET

Bu çalışmada, "REL P" adı verilen bir paralel programlama dili geliştirilmiştir. Bu Programlama Dilinin uygulama alanı Hızlı Karar Destek Sistemlerinde İlgili Veri İşlemidir.

Paralel işlemeye yönelik bilgisayar donanımı artık erişilebilir. Fakat klasik bilgisayar mimarileri için tasarlanmış olan yazılım bu donanımdan yeterli düzeyde faydalanmaz.

TEŞEKKÜR

Doktora çalışmamda bilgisi ve örnek kişiliğiyle ışık tutan Sevgili Hocam Sayın Prof. M. Yahya KARSLIGİL'e tezimi yönlendirmede gösterdiği büyük ilgiden ve katkılarından dolayı sonsuz teşekkür ederim.

Tez çalışmam esnasında kullandığım dört işlemcili bilgisayar donanımını geçici süre ile kullanıma veren Motorola Bilgisayar Grubuna, Bu donanımın Türkiye'de bulunma süresi dolup, geriye gönderilme zorunluluğu doğduğunda, prototip REL P dilinin paralel çalışmasını izlemek üzere zaman ayırıp, kıymetli görüşlerini esirgemeyen İstanbul Teknik Üniversitesi Kontrol ve Bilgisayar Mühendisliği Bölümü Öğretim Üyeleri, Sayın Prof.Dr. Emre HARMANCI, Sayın Prof. Dr. Nadir YÜCEL ve Sayın Doç. Dr. Bülent ÖRENCİK'e ve aynı nedenle, Hocam Sayın Prof. M. Yahya KARSLIGİL'e, ayrıca sonsuz teşekkür ederim.

Ayrıca, iş hayatımın yanında, akademik çalışmamın devamına izin veren T.C.Genel Kurmay Başkanlığı ve Deniz Kuvvetleri Komutanlığı ile Setra A.Ş. Yönetim Kuruluna teşekkür ederim.

Senelerce süren yoğun tempodaki Doktora çalışmamda, özveriyle bana destek veren, Sevgili Eşim Serpil FUTACI ve Oğullarım Toygun FUTACI ile Seyhun FUTACI'ya ayrıca sonsuz teşekkür ederim.

Bu çalışmamın sonucunda, daha gerçekçiliğe sahip ve paralel donanımdan etkin bir şekilde yararlanıldığı görülmüştür. Uygulamaya bağlı olarak, seri ve paralel çalışmaları karşılaştırıldığında dört işlemcili bir sistemde, 3.5'e ulaşan hızlanma kaydedilmiştir. Bu çalışmada elde edilebilirliği yapılan bir yazılım için bir örnek, örnek bir uygulamadır. REL P dilinde yazılan uygulamaların karşılaştırılması, 3.5'e ulaşan hızlanma kaydedilmiştir. Bu çalışmada elde edilebilirliği yapılan bir yazılım için bir örnek, örnek bir uygulamadır. REL P dilinde yazılan uygulamaların karşılaştırılması, 3.5'e ulaşan hızlanma kaydedilmiştir.

ÖZET

Bu çalışmada, "RELP" adı verilen bir paralel programlama dili geliştirilmiştir. Bu Programlama Dilinin uygulama alanı Hızlı Karar Destek Sistemlerinde İlişkisel Veri İşlemedir.

Paralel işlemeye yönelik bilgisayar donanımı artık erişimimizdedir. Fakat klasik bilgisayar mimarileri için tasarlanmış olan yazılım bu donanımdan yeterli düzeyde yararlanamamaktadır.

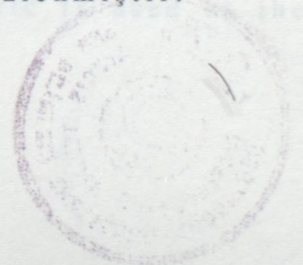
Bu kullanılmayan potansiyeli göz önüne alan bazı Veri Tabanı Yönetim Sistemi üreticileri, yeni ürünler ortaya koymaktadır. Fakat bu ürünlerin pek çoğu eski Veri Tabanı İşleme ve Sorgulama Dillerini kullanmaya devam etmektedir.

Seçilen uygulama alanı için, paralel donanımdan iyi yararlanan, programcının zihninde paralel işlemeyi uygun biçimde canlandıran ve değişik paralel programlama dillerinin en iyi özelliklerini üzerinde toplayan, tümüyle yeni bir programlama dilinin geliştirilmesi gerektiğine inanılmaktadır.

RELP Programlama Dili, Fonksiyonel Programlama Dilleri, Mantıksal Programlama Dilleri, Nesne Temelli Programlama Dilleri, Veri Akış Programlama Dilleri ve İlişkisel Programlama Dillerinin, genel kabul gören, iyi özelliklerinin bir çoğunu üzerinde toplayacak şekilde tasarlanmıştır. RELP dili kontrol yapısı veri akış yapısı, veri modeli ise ilişkisel veri modelidir. Programın yapı taşları olan dil blokları, akan verilere, ilişki ve veri dönüştürme fonksiyonlarını uygular. Bu nedenle RELP "Uygulama Tabanlı" (applicative) bir dildir. RELP dilinin blokları, verilerden oluşan mesajları alan ve işleyen nesneler olarak düşünüldüğünde nesne temelli programlama mantığı kullanılabilir.

Programlama dilinin gerçekleştirilebileceğini göstermek ve paralel donanımdan yararlanma derecesini ölçmek için, dilin tasarımından sonra bir prototip geliştirme çalışması yapılmıştır. Bu amaçla prototip, dört işlemcili paylaşılan bellek MIMD mimarisindeki bir donanım platformu üzerinde çalıştırılmıştır. RELP dilinin sorgulama alt kümesi, bazı sorgulama dili modelleri ile karmaşıklık yönünden karşılaştırılmıştır.

Bu çalışmanın sonucunda, dilin gerçekleştirilebileceği ve paralel donanımdan etkin bir biçimde yararlandığı görülmüştür. Uygulamaya bağlı olarak, seri ve paralel çalıştırma zamanları karşılaştırıldığında dört işlemcili bir sistemde, 3.8'e ulaşan hızlanma katsayılarının elde edilebileceği saptanmıştır. Bir yazılım ölçütü seçilerek, örnek bir uygulamada, RELP dilinde yazılan sorgulamaların karmaşıklıklarının, SQL'de ve İlişkisel Cebirde yazılan sorgulamaların karmaşıklıklarına çok yakın değerlerde olduğu gözlenmiştir.



ABSTRACT

In this work, a Parallel Programming Language, namely "RELP", has been developed. The domain of this Programming Language is "Relational Data Processing in Fast Decision Support Systems."

While the parallel processing computer hardware is available, existing Software Base designed for conventional computer architectures is not capable of fully utilizing these hardware platforms.

In realization of this unused potential, some Database Vendors are releasing new products to make use of this hardware resource. But most of them are still using the previous programming languages for Database Processing.

It is believed that, for this application domain, a, totally new programming language should be developed which takes the commonly accepted best properties of different parallel programming languages, so that the parallel hardware capacity is properly utilized and also the computational model in the minds of programmers is changed accordingly.

The language RELP borrows the good properties of the Functional Languages, Logical Languages, Object Based Languages, Data Flow Languages and Relational Languages. RELP control structure is Data Flow control structure and Data Model is Relational Data Model. The RELP blocks, that are the program building blocks of the language, apply relations and data transformation functions to the data streams. So, RELP is an applicative language. By thinking the RELP blocks as the objects receiving and manipulating messages made up of data, object based programming logic can be applied.

A prototype implementation of the language was done after the design, just for showing that the language can be realized and for measuring its parallel processing hardware utilization level. The prototype was run on a four processor shared memory parallel MIMD architecture for this purpose. The complexity of the query subset of the RELP language was compared against some query language models in a sample application.

As a result of this work, it has been shown that the language can be realized and it utilizes parallel hardware efficiently. When the elapsed times of serial and parallel runs of some specific programs were compared, depending on the application, it was found that the speedup, up to 3.8 is achievable. Taken a sample application, the complexities of the queries written in RELP language were found to be close to the complexities of the queries written in SQL and Relational Algebra when a selected software metric is used as the measurement tool.

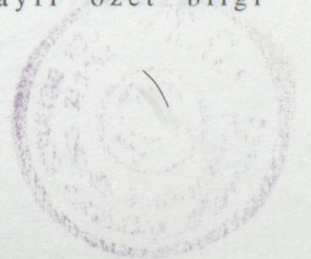
1. GİRİŞ VE GENEL BİLGİLER :

1.1 Dağıtılmış Bilgisayar Sistemleri :

Dağıtılmış Bilgisayar Sistemleri (Distributed Computer Systems) geçen on yıl içinde son derece önem kazanmıştır. Otuz yıllık "von Neumann" Bilgisayar Mimarisi ile daha fazla performans elde etmek gittikçe daha güçleşmiş ve bir problemin birden fazla işlemcinin işbirliği içinde çalışması ile çözülmesi esasına dayanan paralel işlemciler mimarisi gündeme gelmiştir. Aslında paralel işleme fikri yeni olmayıp daha önce Charles Babbage tarafından ortaya atılmıştır. Bu fikrin günümüzde gerçeğe dönüşebilmesinin nedeni, donanım olarak kolay gerçekleştirilebilir hale gelmesidir(Cybenko, 1992).

Paralel işlemenin pratik olarak kullanıma girmesi için , uygun işletim sistemi ve programlama dillerine sahip olunması gereklidir. Dağıtılmış bilgisayar sistemleri felsefesine uygun, çok değişik donanım mimarisi ve programlama sistemleri geliştirilmiştir. Bazı dağıtılmış bilgisayar sistemlerinde programlama, klasik programlama dillerinde, bazılarında ise, dağıtılmış, paralel sistemlere uygun tasarımlanmış programlama dillerinde gerçekleştirilmektedir (Comerford,1992). Ancak günümüze kadar, paralel işleme yazılımı kalitesi donanım kalitesinin gerisinde kalmıştır ve geride kalma eğilimindedir(Cybenko,1992).

Bu bölümde, önce, dağıtılmış bilgisayar sistemlerine yönelik programlama dillerinde, paralel işleme ve paralel süreçler arası iletişimi temin için kullanılan standart yöntemlere değinilecek, daha sonra, verilen bu genel bilgilerin yardımıyla, RELP programlama dilinin geliştirilmesi sürecinde değerlendirmeye alınan, ön plandaki bazı paralel programlama dilleri ile ilgili ayrı ayrı özet bilgi verilecektir.



Dağıtılmış bilgisayar sistemlerini, " Kendi belleğine sahip olan, kendi programını icra eden ve aralarındaki iletişim sistemi üzerinden mesaj alışverişi ile haberleşip yardımlaşarak sonuç üreten işlemciler sistemi" şeklinde tanımlayabiliriz. İletişim sistemi olarak "Hypercube", "Transputer Network", "Local Area Network-(LAN)" ve "Wide Area Network-(WAN)" gibi iletişim sistemlerinden bahsedebiliriz. Dar Alan Bilgisayar ağları (Local Area Network) tipik olarak aynı bina ve bölge içinde 10-16 Megabit/saniye iletişim hızlarında bağlantı sağlarlar. Fiber Optik iletişim ortamına dayanan FDDI(Fiber Distributed Data Interface) ile bu hız 100 Megabit/saniye düzeyine çıkmıştır. Fiber-Kanal (Fibre-Channel) standardında iletişim bağlantıları ile 266 Megabit/Saniye hızına ulaşılmıştır. 1 Gigabit/Saniye hızına ulaşmak üzere çalışmalar yapılmaktadır (Sounders,1993). Geniş Alan Bilgisayar Ağları (Wide Area Network) ise şehirler ve ülkeler arasında tipik olarak 14.4-64 Kilobit/saniye hızlarında iletişim sağlamaktadır. ATM (Asynchronous Transfer Mode) iletişim teknolojisi ile geniş alan bağlantılarında 155 Megabit/Saniye hızlarına ulaşılabilenekte, 2.5 Gigabit/Saniye hızlarına ulaşmak için çalışmalar yapılmaktadır (Riezenman,1994). LAN ve WAN bağımsız işlemciler arasında iletişim kurarak yukarıdaki tanıma uysa da, bu tip iletişim sistemleri işlemciler arasında yeterince hızlı ve güvenilir bir iletişim sağlayamazlar, bu tip bir iletişim sistemini kullanan dağıtılmış bilgisayar sistemleri "Gevşek Bağlantılı-Loosely Coupled" olarak sınıflandırılırlar. Diğer yandan "Hypercube" , "Transputer Network" gibi iletişim sistemleri yüksek süratli ve güvenilir bir yol yapısı üzerinden veri transferi gerçekleştirebilirler. Bu tip iletişim sistemlerinde hız, 200 Mbyte/saniye hızına ulaşmıştır (IEEE Spectrum,1994). Bu da 100 byte büyüklüğünde bir mesajın 0.5 Mikrosaniyede iletildiği anlamına gelmektedir. Geleceğe dönük olarak, VME yolu tabanlı "Autobahn" projesi 400 Mbyte/saniye yol hızını hedeflemektedir. Bu tip iletişim sistemlerini kullanan dağıtılmış bilgisayar sistemleri "Sıkı Bağlantılı-Closely Coupled" olarak sınıflandırılır.

Uygulamanın özelliğine bağlı olarak programın değişik işlemciler üzerinde çalışan bölümleri aralarında sık veya seyrek iletişimde bulunma gereksinimindedir. Eğer iletişim frekansı yüksek ve iletişimde bulunacak paralel icra edilen program bölümleri küçük ise bu tür paralel çalışmaya "**Küçük Parçalı-Fine Grain**" paralel çalışma adı verilmektedir. Eğer paralel olarak icra edilen program bölümleri büyük ve iletişim frekansı düşük ise bu tür paralel çalışma "**Büyük Parçalı- Course Grain**" paralel çalışma adını almaktadır. "**Küçük Parçalı-Fine Grain**" paralel çalışmanın söz konusu olduğu uygulamalarda "**Sıkı Bağlantılı-Closely Coupled**" mimarilerin uygulanması gereklidir. "**Büyük Parçalı-Course Grain**" paralel çalışmanın gerekli olduğu uygulamalarda "**Gevşek Bağlantılı-Loosely Coupled**" mimarilerin kullanılması ekonomik yönden uygun olmaktadır.

Yukarıda anlatılan dağıtılmış mimariler, diğer bir paralel çalışma mimarisi olan ortak bellekli çok işlemciler mimarisinde benzetişimle uygulanabilir. Bu tip bir çalışmada işlemciler arasında gönderilen mesajlar iletişim alt sistemi yerine, paylaşılmakta olan bellek üzerinden iletilirler. Prensip olarak bellek paylaşımlı çok işlemci mimarileri işlemci sayısı yönünden limitlidirler.

Programların bölümlerini paralel olarak icra ederek işlem zamanını kısaltmak, aynı programın kopyalarını birden fazla işlemci üzerinde paralel olarak çalıştırarak, tüm sistemin komple hata olasılığını azaltan, hataya toleranslı sistemler oluşturmak, dağıtılmış bilgisayar sistemlerinin önde gelen amaçlarıdır. Gerçek zaman işletim sistemlerinde her işlevin ayrı işlemci üzerinde çalıştırılması ile harici olaylara cevap verme zamanını kısaltmak diğer bir uygulama alanıdır. Yapısı yönünden dağıtılmış bilgisayar sistemlerine uygun olan "**Mesaj İşleme**" ve "**Elektronik Posta**" uygulamaları da kullanım alanlarına ilave edilebilir.



Dağıtılmış bilgisayar sistemlerinde, programın bölümleri, değişik işlemciler üzerinde paralel olarak çalışırlar. Bu bölümlerden her biri, icra zincirinin belirli noktalarında diğer işlemciler üzerinde çalışan bölümler ile eş zamanlama yapmak, ara sonuçların karşılıklı alışverişini yapmak ve ortak kullanılan kritik veri yapıları üzerindeki icra sırasını belirlemek üzere koordinasyonu sağlamak zorundadır. Her Dağıtılmış Bilgisayar Programlama Sistemi, bu işlemlerin yapılmasını sağlayacak şekilde, işlemcilerin tahsisi ve süreçlerin eş zamanlaması için, mekanizmalar sağlamak zorundadır.

1.2 Dağıtılmış Bilgisayar Sistemlerine Yönelik Programlama Dilleri

Özel olarak, Dağıtılmış Bilgisayar sistemlerinde programlama için tasarlanmış, birçok programlama dili mevcuttur. Özel programlama dillerinin amacı, programcı için, paralel çalışmayı dil düzeyinde destekleyen, gerekli soyutlamalara ve üst düzeyliliğe sahip bir ortam yaratmak, kolay anlaşılır, ve taşınabilir programlar yazılmasını kolaylaştırmaktır. Dağıtılmış Bilgisayar programlama dillerini incelediğimiz zaman bir çoğunun CSP (Communicating Sequential Processes) Modelini uyguladığını görürüz. Bu modelde değişik işlemciler üzerinde çalışan süreçler birbiriyle mesaj alışverişi yolu ile iletişim sağlar. Bir çok dil için bu yeterlidir. Bazı araştırmacılar ise, von Neuman mimarisinin problemlerine sahip algoritmik dillerin izlerini taşıyan, bu yaklaşımdan saparak, paralel yapıya uygun Mantıksal (Logic), Nesne Yönelimli (Object Oriented) ve Fonksiyonel programlama dillerine ağırlık vermektedirler. Atama yan etkisinin (Side effect) bulunmadığı Fonksiyonel Programlama Dillerinde, değişik fonksiyonlar biribirini etkilemeden paralel olarak çalıştırılabilirler. Mantıksal Programlama Dillerinde değişik amaçlar (Goals) biribirine paralel olarak denenebilirler. Nesne Yönelimli (Object oriented) Programlama Dillerinde, nesneler aktif süreçlere halinde değişik işlemcilere atanabilirler.

Dağıtılmış Bilgisayar Programlama Dillerinde, paralel çalışmanın ifade edilişi çeşitli şekillerde olabilir. Paralel çalışma, kendi statüsü ve veri alanı bulunan bağımsız süreçlerin, işletim sisteminin sağladığı süreçler arası iletişim olanaklarıyla haberleşerek paralel olarak icra etmesi şeklinde ifade edilebilir. Nesne temelli dillerde paralel çalışma, nesneler ile ifade edilmektedir. Bir nesne hem veriyi hem de veriyi işlemede icra edeceği metodu içerir. Nesne, diğer nesnelerle mesaj alışverişi yaparak haberleşir. Bir nesnenin bünyesinde olası her mesaj için bir metod yer alır. Bu metodlar nesnenin mensup olduğu sınıfın (Class) belirlediği standart metodlardır.

Diğer bir ifade şekli ise OCCAM programlama dili "PAR" yapısında olduğu gibi, paralel çalışması istenen deyimlerin bir paralel çalıştırma bloğu içine yerleştirilmesidir. Atama Yan Etkisinin bulunmadığı (Side effect free) programlama dillerinde ifadelerin işlenme sırası önemini kaybetmekte, böylece her ifade ayrı ayrı işlenebilmektedir. Bu sayede derleyici kolaylıkla paralel işleyecek program bölümlerini ortaya çıkartabilmektedir. Sonuç olarak bu dillerde paralel çalışmanın özel yapılar kullanılarak ifade edilmesine gerek kalmamaktadır. Mantıksal (Logic) Programlama dillerinde aynı mantıksal değişkenleri paylaşmayan önermeler paralel olarak denenebilirler. Bu dillerde de paralel çalışmayı özel yapılar kullanarak belirlemek gerekli olmayabilir.

Dağıtılmış Bilgisayar programlama dilleri belirli bir program bölümünü belirli bir işlemci ile ilişkilendirmeye yönelik yapılar içerebilirler. Bu bağlantılama, hesaplamanın amacına ve bilgisayar mimarisine bağlı olarak değişir. İletişimden doğan kayıplar ile programın bölümlerinin paralel çalışmasından kaynaklanan kazanç uygun şekilde dengelenmelidir. Eğer hatalara karşı tolerans amaçlanıyorsa her sürecin bir kopyası alternatif bir işlemcide çalışacak şekilde yüklenmelidir. Programlama dili, programcıya, program bölümlerinin işlemcilere atanması konusunda kontroller

verebilir veya derleyici derleme fazında statik olarak bu belirlemeyi kendisi yapabilir. Alternatif olarak, çalışma fazında süreçler dinamik olarak yaratılıp, işlemcilerle sistem tarafından atanabilirler. gelece kadar bekler. Eğer bir "select" deyimi içindeki "guard" Paralel süreçler arasındaki iletişim, mesaj iletişimi veya veri paylaşımı şeklinde olabilir. Mesaj iletişiminde çeşitli teknikler uygulanmaktadır. Bu tekniklerden ilki, Noktadan noktaya senkron veya asenkron mesaj transferidir. Senkron transferde mesajı gönderen süreç, alan süreç mesajı alana kadar, bloke olur ve bekler. Asenkron iletişimde gönderen süreç mesajın alınmasını beklemez, mesajlar alacak süreç için sistem tarafından kuyrukta bekletilir. Bu teknik tek yönlü mesaj akışı sağlar.

İkinci teknik olarak belirlediğimiz "**Randevu**" tekniğinde, mesajı kabul edecek süreç içinde bildirimi yapılan, formal parametrelere de sahip olabilen, "bildirilen giriş noktası", (declared entry point) mesajı gönderen süreç tarafından gerçek parametreler kullanılarak çağırılır ve alıcı süreç tarafından "kabul" işlemi yapılarak giriş bildiriminde yer alan deyimler icra edilir. Alıcı süreç çıkış parametrelerine değer atayarak gönderen sürece bu değerleri geri gönderebilir. Her iki süreç, giriş noktası çağırılıp, giriş noktası için bildirimi yapılan deyimler icra edilir edilmez serbest kalarak bağımsız olarak icraya devam ederler. Görüldüğü gibi iletişim eş zamanlı olarak yapılmakta ve çift yönlü olarak gerçekleşmektedir.

"Uzak Yordam Çağırma- Remote Procedure Call" üçüncü teknik olarak klasik bir programlama dilindeki yordam çağırma işlemine benzemektedir. Ancak çağırılan yordam diğer bir süreç içinde yer almaktadır. Çağırılan süreç, çağırılan sürecin yordamı tamamlamasına kadar bloke olur. Bazı yapılarda, CSP de olduğu gibi, Select deyimi içinde, bir seri, "guard" deyimi yer alır. Bu "guard" deyimleri, bir mantıksal ifade ve bir harici sürece iletişim çağırısı kısmından oluşurlar. Bir "guard" deyimi eğer mantıksal ifadesi TRUE sonucunu verir ve iletişim çağırısı yapılan süreçten bir mesaj

gelirse başarıya ulaşmış olur. Eğer bir guard başarıya ulaşır ise bu guard ile ilgili deyimler icra edilir. Bir guard ancak mesaj beklediği süreç sona ererse başarısız olur. Aksi takdirde mesaj gelene kadar bekler. Eğer bir "select" deyimini içindeki "guard" deyimlerinden birden fazlası aynı anda başarılı olursa bir tanesi rastgele seçilir. Eğer bir select deyimini içindeki tüm "guard" deyimleri başarısız olursa tüm "select" deyimini başarısız olur. Mantıksal programlama dillerinde kullanılan "Guarded Horn" deyimini aynı kategoride yer almaktadır. Bu durumda bir liste içinde yer alan "guard" deyimleri (Guard goals) denenir. Başarıya ulaşan "guard" deyimlerinden bir tanesi seçilir ve bu noktadan itibaren bu "guard" deyimini ile ilgili önermeler denenir.

İletişim paylaşılan değişkenler ve dağıtılmış veri yapıları vasıtasıyla da yapılabilir. Bu iletişim şekli paylaşılan bellek söz konusu olmadığı durumlar için de geçerlidir. Paylaşılan değişkenler ile iletişimde, bu tip bir değişkene değer atama hakkına sahip olan süreç, gerekli olduğu an atamayı yapar, atama hakkı olmayan süreç ise böyle bir atama yapılana kadar askıya alınarak bekler. Bu mekanizma, iki süreç arasında bir iletişim kanalı oluşturur.

Dağıtılmış veri yapıları kullanıldığında bütün süreçler mantıksal olarak paylaşılan genel bir veri yapısı kullanırlar. "Linda" programlama dilinde kullanılan "Tuple Space" buna bir örnektir. Bu veri yapısına paralel olarak yapılan erişimler sistem tarafından seri hale getirilerek sırayla uygulanır. Mesaj iletişiminden farklı olarak, iletişim tüm işlemcilerin katılımıyla sürdürülür. Bu veri yapısına mesaj gönderen bir işlemcinin bu mesajı hangi sürecin kullanacağı konusunda bir belirleme yapması söz konusu değildir. Benzeri şekilde bu veri yapısından mesaj alan bir sürecin, mesajın kaynağı hakkında bilgi sahibi olmasına gerek yoktur.

Dağıtılmış Bilgisayar Sistemlerine yönelik bir programlama dili, sistem hatalarını da hesaba katmak zorundadır. Hedef alınan

uygulama alanı hataya karşı tolerans sağlamayı amaçlamasa dahi, sistem, bir hata durumunda verileri iyi tanımlı bir konumda bırakacak şekilde oluşturulmalıdır. Bu mekanizma kalıcı değişiklikleri oluşturan Hareketlerin bölünememesi ve kurtarma imkanlarını sağlamalıdır.

Bu bölümde, yukarıda verilen genel bilgilerin ışığı altında, kullanımı belirgin bir şekilde yaygınlaşmış olan dağıtılmış programlama dilleri ile ilgili ayrı ayrı özet bilgi sunulacaktır. Bu dillerin özelliklerine kısaca değinilmesinin nedeni, paralel programlama dillerinin genel özelliklerinin gözden geçirilmesi ve RELP programlama dili tasarımında kullanılan özellikler ile kullanılmayan özelliklerin okuyucu tarafından net bir şekilde ayırt edilebilmesinin sağlanmasıdır.

1.3 Eş Zamanlı Mesaj İletimine Dayanan Paralel Programlama Dilleri

Hoare tarafından 1978 yılında geliştirilen (Hoare,1978) CSP (Communicating Sequential Processes) programlama dili biribiriyle sadece eş zamanlı (senkron) mesaj iletişimi ile haberleşen sıralı (sequential) kontrol yapısında süreçlere dayanmaktadır. CSP kendini takip eden ve aynı özelliğe dayanan bir çok dile rehber olmuştur. Bunlardan en önemlisi "Inmos Transputer" paralel işlemci mimarisinin temel programlama dili olarak kullanılan OCCAM programlama dilidir (Transputer Development System,1990).

1.3.1 "OCCAM" Programlama Dili :

Occam programlama dili programcının programını biribiriyle mesaj iletişiminde bulunan paralel süreçler halinde yazmasını sağlar. Occam biribiriyle mesaj iletişimi yoluyla bağlantılı bir çok sürecin, tek bir süreç gibi görülebilmesini sağlayarak sistem hiyerarşisini

temin eder (Occam Programming Manual,1984). Herhangi bir detay düzeyi ele alındığında programcı yapısal ve yönetimi basit süreçler bazında çalışır.

Occam programlama dilinde sıralı yapılacak işlemler "SEQ" bloğu içinde yer alırlar . Şekil-1.3.1-1'de, içinde atama deyimleri yer alan bir OCCAM , SEQ bloğu görülmektedir.

```
SEQ
  x:=3
  y:=x+7
  z:=x+1
  x=(y+z)/2
```

Şekil-1.3.1-1 Occam SEQ bloğuna örnek.

Paralel yapılacak işlemler ise PAR bloğu içinde yer almaktadır. Şekil-1.3.1-2 de gösterilen tipik PAR bloğunda yer alan iki atama deyiminden hangisinin önce yapılacağı belirli değildir. Biri diğerinden önce icra edebileceği gibi., her iki deyim ayrı işlemciler tarafından aynı anda icra edilebilir (Geraint,1987).

```
PAR
  x:=v-2
  z:=x+3
```

Şekil-1.3.1-2 Occam PAR bloğu.

Occam programlama dilinde paralel süreçlerin arasındaki iletişim, kanallar üzerinden yapılan giriş ve çıkışlar ile gerçekleştirilir. Giriş ve çıkış kanal işlemleri söz dizimi yapısı şekil 1.3.1-3 te yer almaktadır. '!' çıkış, '?' ise giriş işaretidir. 'channel' kanal bildiriminde bildirilen bir kanal ismidir. Çıkış işleminde '!' işaretinin sağında yer alan ifadenin sonucu bulunur ve işaretin solunda ismi verilen kanala gönderilir. Giriş işleminde ise belirtilen kanaldan okunan değer '?' işaretinin sağındaki değişkene depolanır.


```
channel ! expression
```

```
channel ? variable
```

Şekil-1.3.1-3 OCCAM kanal giriş ve çıkış işlemleri.

Occam programlama dilinde, değişkenler ve kanalların bildirimi zorunludur. Benzeri şekilde Süreçler de bildirilirler. Şekil-1.3.1-4'te bir OCCAM program kesiti verilmiştir. Bu programda "üretici" ve "alıcı" adlı iki süreç bildirilmiştir. Her iki süreci PAR yapısı içinde aktive eden süreç ise, "kopru" adıyla bildirdiği kanalı bu süreçlere parametre olarak geçirmektedir. Sonuç olarak, değişik lokal kanal isimlerine yönelik giriş çıkış işlemleri yapan paralel süreçler, (üretici ve alıcı) efektif olarak bu kanal üzerinden (kopru) veri iletişimi yapmaktadırlar.

Occam ALT yapısında bir şart ve/veya bir giriş (kanaldan okuma) yapısından oluşan alternatifler mevcuttur. Alternatiflerden birisinin başarıya ulaşması şartın TRUE değerini alması ve giriş yapısının okuma yapabileceği bir verinin kanaldan ulaşması halinde olur. Alternatiflerden hangisi başarıya ulaşırsa o alternatif ile ilgili deyimler icra edilir. Birden fazla alternatif aynı anda başarılı olursa bir tanesi rastgele seçilir. Şekil-1.3.1-5'te bir ALT yapısı örneği bulunmaktadır.

Bu örnekte, eğer "siyah.secildi" adlı değişken TRUE değerine sahip ve "siyah" adlı kanalda bir giriş değeri mevcut ise çıkış kanalına "siyah" kanalından okunan veri gönderilir. Benzeri şekilde eğer "beyaz.secildi" adlı değişken TRUE değerine sahip ve "beyaz" adlı kanalda bir giriş değeri mevcut ise çıkış kanalına "beyaz" kanalından gelen veri gönderilir.


```

PROC uretici(CHAN cikis.kan)=
  WHILE TRUE
    VAR a:
    SEQ
      ..... a hesaplanır.....
      cikis.kan ! a
    : .....
PROC alici(CHAN giris.kan)=
  WHILE TRUE
    VAR b:
    SEQ
      giris.kan ? b
      ..... b'yi kullanarak hesap yapılır....
    : .....
CHAN kopru:  --kanal bildirimi--
PAR
  uretici(kopru) --kopru adlı kanal cikis.kan adlı formal
                parametre ile eşleştii--
  alici(kopru)  ---kopru adlı kanal giris.kan adlı formal
                parametre ile eşleştii--
  .....

```

Şekil-1.3.1-4 Occam programlama dilinde süreçler arasında kanallar üzerinden veri iletimi.

TIME adlı özel bir kanaldan, o anki zamanı okumak ve ALT yapısının sonsuz kilitlenmesinin önlenmesi mümkündür. OCCAM çoklu bağlantılı Transputer ağlarında kullanılmak üzere tasarlanmıştır. Programlama dilinde mevcut kanal yapısı, iki transputer arasındaki bağlantı ile bire bir ilişkilendiğinden bilgisayar mimarisi ile programlama dili yapısı uyum göstermekte, bunun sonucu olarak da program etkinliği artmaktadır. OCCAM, sinyal işleme, imaj işleme, süreç kontrol, benzetişim ve nümerik analiz konularında yaygın olarak kullanılmıştır.

```

ALT
  siyah.secildi&siyah ? x
    cikis.kanal ! x
  beyaz.secildi&beyaz ? x
    cikis.kanal ! x

```

Şekil-1.3.1-5 OCCAM ALT yapısı.

OCCAM kanal soyutlaması bu çalışmada tasarımı yapılan RELP dilinin bünyesinde standart giriş ve standart çıkış kanalları şeklinde yer almıştır. Farklı olarak RELP dilinde kanal bildirimi özelliği yoktur. OCCAM ile RELP arasındaki temel benzerlik, paralel çalışan program bölümleri arasında kanallar üzerinden değişken soyutlaması altında veri transferi yapılmasıdır.

1.4 Asenkron Mesaj İletimine Dayanan Paralel Programlama Dilleri

Asenkron mesaj iletişiminde gönderen süreç alan sürecin mesajı almasını beklemeden icrasına devam eder bu dillere güzel bir örnek NIL (Network Implementation Language) programlama dilidir. (Strom, 1986)(Bal et al,1989). NIL süreçler arasındaki iletişimde değişkenlerin paylaşılmasını ve pointer işlemlerini yasaklar.

NIL programlama dilinde paralel işleme girecek birimler süreçlerdir (Process Model). Süreçler dinamik olarak yaratılır ve iletişim kanallarını kullanarak aralarında mesaj iletişiminde bulunurlar. NIL'de süreçler aynı zamanda modüler yapıyı sağlayan birimlerdir.

NIL'de iletişim kanalları dinamik olarak yaratılır. İletişim kanalları "PORT" adını alan kuyruk yapılarıdır. Herhangi bir zamanda bir kuyruğa bir süreç sahiptir. Bu hak süreçten sürece mesaj olarak transfer edilebilir. NIL'in bu özelliği bu çalışmada tasarlanan RELP programlama dilinin kuyruk soyutlamasına örnek teşkil etmiştir. Bir süreç duruma göre giriş kuyruklarından bazılarını çıkış kuyruklarından bazılarına yönlendirebilir. Bir giriş kuyruğu birden fazla çıkış kuyruğu ile ilişkilendirilebilir. OCCAM "ALT" yapısına benzeyen, "Guard" tipinde giriş için bir yapı NIL'de de mevcuttur.

NIL dilinin tek işlemcili bilgisayar konfigürasyonunda çalışan versiyonu geliştirilmiş ve bir iletişim uygulamasının geliştirilmesinde kullanılmıştır.

1.5 İletişim ve Eş Zamanlı Çalışmada Randevu Mekanizmasını Kullanan Paralel Programlama Dilleri

1.5.1 "Ada" programlama dili

Ada programlama dili Amerika Birleşik Devletleri Savunma Bakanlığı tarafından desteklenen bir proje çerçevesinde bir grup bilim adamı tarafından tasarlanmıştır bir programlama dilidir (Reference Manual for the ADA Programming Language, 1980).

Ada programlama dilinin geliştirilmesinin ardında yatan neden, tüm Savunma Bakanlığı programlarının, gerçek zaman uygulamalarından klasik bilgi işlem programlarına kadar aynı programlama dilinde geliştirilmesi isteğidir.

Ada programlama dilinde paralel yapı taşlarını kendi içinde sıralı işlem yapan ve "Task" adı verilen süreçler oluşturur. Bir "Task" diğer "Task"lar ile iletişim şeklini tanımlayan bir "belirleme" bölümü ve icra edilebilir deyimlerden oluşan "gövde" bölümünden oluşur.

```
TASK A is
..... Belirleme bölümü
end A;

TASK body A is
..... Gövde bölümü
end T;
```

Şekil-1.5.1-1 ADA task yapısı.

Tasklar arası iletişim ve eş zamanlama genellikle randevu mekanizması ile gerçekleşir. Aynı zamanda paylaşılan değişkenler üzerinden iletişim de mümkündür. Randevu mekanizması daha önce tanımlandığı biçimde çalışır. Girdi Çağırma işlemlerinin şarta bağlı olarak kabul edilmesi söz konusu değildir.

1.5.2 "Concurrent C" programlama dili

Concurrent C programlama dili AT&T Bell Laboratuvarlarında C programlama dili dağıtılmış bilgisayar sistemlerinde kullanılacak şekilde genişletilerek tasarlanmıştır.

Concurrent C Programlama dilinde yer alan süreçler ADA Task yapısındadır. Concurrent C terminolojisinde "Transaction" Ada'daki girdi bildirimine karşılık gelmektedir. Süreçler dinamik olarak yaratılabilir ve bu işlem esnasında ortaya çıkan süreç tanıtıcıları (Process Id) değişkenlere depolanabilir veya parametre olarak geçirilebilirler.

Süreçler randevu mekanizması ile aralarında iletişim kurarlar. Concurrent C "Transaction"ları ADA entry bölümlerinin aksine çağırıcıyı yapan sürece değer geri döndürebilirler. Buna ilaveten Concurrent C zamanuyumsuz mesaj iletişimini kabul eden "Transaction"lara sahiptir.

Concurrent C accept deyimi ADA'dan farklı olarak, "Transaction" çağırılarındaki yer alan gerçek parametrelere bağlı şartlı kabul işlemi yapabilir. Yine belli bir öncelik derecesine göre bu çağırıların kabulü mümkündür. "Transaction" çağırıları fonksiyon çağırıları gibi ifadelerin içinde yer alabilirler. Bir "transaction" çağırısının belirli bir zaman içinde sonuçlanmaması halinde alternatif bir ifadenin işleme koyulması şeklinde faydalı bir deyim de Concurrent C içinde yer almaktadır. Bu sayede eş zamanlama yapamayan süreçlerin tüm sistemi kilitlemesi önlenir. Concurrent C, ADA programlama dilindeki "select" deyimine çok benzer bir "select" deyimine sahiptir.

REL P programlama dilinin tasarımında, ADA ve Concurrent C gibi dillerin paralel yapı özelliklerinden yararlanılmamıştır.

1.6 Nesne Temelli Paralel Programlama Dilleri

Nesne temelli programlama gittikçe daha popüler hale gelmektedir. Nesne temelli paralel programlama dillerinden çoğunda her bir nesne bir süreç ile temsil edilir.

Nesne temelli diller programcıya, biribiriyle etkileşen bir çok otonom nesneler bazında program tasarlama ve yazma olanağı sağlarlar. Dağıtılmış nesne temelli programlama sistemlerinde süreçler ve nesneler arasındaki ilişki açısından iki model vardır. Bunlar pasif ve Aktif nesne modelleridir. Pasif nesne modelinde bir süreç tek bir nesne ile ilişkilendirilmez ve bir çok nesne içinde nesnelerin operasyonlarını icra eder. Aktif nesne modelinde ise her nesne bir süreç ile ilişkilendirilir.

Nesne temelli dillerde bir nesnenin aynı anda birden fazla isteğe cevap vermesi söz konusudur. Her talep için ayrı bir nesne konumu (state) tutulması ve bu konumların biribirine karışmaması bu sistemlerde önemli bir işlevdir.

Nesnelerin biribiriyle etkileşimi ve biribirilerinin operasyonlarını çağırımları da düzenli olarak yürütülmesi (Object Interaction Management) gerekli bir alandır. Burada çağırılan nesnenin dağıtılmış sistem içinde yerinin tesbit edilmesi birinci adımdır.

Aktif nesne modelinde nesneleri temsil eden süreçler arasında Mesaj İletişimi kullanılır. Pasif nesne modelinde bir dizi nesne operasyonu çağırma işlemi tek bir süreç tarafından yapıldığından, sırasıyla her nesne operasyonu yordam çağırma operasyonuna çok benzer bir şekilde çağırılarak bu süreç içinde icra edilir (Chin,1991). nesne temelli popüler programlama dillerinden Concurrent Smalltalk, Clouds, Eden, Amobea, Argus ve Emerald dilleri örnek olarak verilebilir.

Bu çalışmada tasarımı yapılan RELP programlama dilinde, dil blokları sabit yapıdaki nesneler olarak düşünülmüş, nesneler arasında sadece verilerden oluşan mesajların alış verişinin yapılması öngörülmüştür. Sabit yapıdaki RELP nesnelerinin iletişimde aktif nesne modeli kullanılmıştır.

1.6.1 "EMERALD" Programlama dili

Emerald Washington Üniversitesinde geliştirilmiş bir programlama dilidir.(Black, 1987) Emerald tüm girdileri nesne olarak kabul eder. Değişkenler, dosyalar hep nesne olarak kabul edilirler. Emerald sınıf (class) ve miras (inheritance) mekanizmasına sahip değildir. Paralel çalışma aktif nesnelerin süreçler halinde ayrı çalışması şeklindedir (Aktif obje model). Bu süreçler işlemciden işlemciye yer değiştirebilirler.

Bir nesnenin, ismi, taşıdığı verisi, ve bir dizi operasyonları vardır. Nesneler birbirinin operasyonlarını aktive ederek iletişimde bulunurlar, herhangi bir anda bir nesne içinde birden fazla aktif olarak sürmekte olan operasyon bulunabilir. Bu operasyonlar nesnenin verisine bir monitör yapısında paralel olarak erişirler.

Bu çalışmada tasarımı yapılan ve Nesne Temelli yapıya sahip olan RELP programlama dilinde de sınıf ve miras mekanizmasına yer verilmemiştir. Nesneler aralarında mesaj olarak sadece veri alışverişinde bulunurlar. Her RELP nesnesi (bloğu) sadece bir metoda sahip olup, diğer nesnelerden aldığı verilere bu sabit metodu uygular. Bazı RELP nesneleri bünyelerinde, veri kısmı olarak ilişki tablolarını içerirler.

1.7 Paralel Fonksiyonel Programlama Dilleri

Fonksiyonel programlama dillerindeki paralel çalışmaya uygun yapıları sayesinde direk olarak paralel programlama dili olarak

kullanılabilirler(Hudak,1989). Paralele sokulabilecek hesaplamaların değişik işlemcilere tahsisi için özel yapılar bu programlama dillerinin yapısına ilave edilerek etkin paralel çalışma sağlanabilir.

1.7.1 "ParAfl" programlama dili

Hudak tarafından Yale Üniversitesinde geliştirilmiştir. (Hudak,1986) Fonksiyonel programlama dilleri küçük parçalı paralel çalışmaya direk olarak uygun bir yapıya sahiptir. Bu durumda fiziki işlemci sayısından daha fazla süreç ortaya çıkmaktadır. ParAfl her paralel çalışabilecek fonksiyonun hangi fiziki işlemci üzerinde çalışması gerektiğini belirleyecek yapıya sahiptir. "\$ on proc" yapısında "proc" bir ifadedir ve işleme sokulduğunda çıkan sayısal değer geçerli bir fiziki işlemci numarası olmalıdır.

(diff(x) \$ on (\$ self+1))+(sin(z) \$ on k)

ifadesi işlemci İ üzerinde icra ediliyorsa işlemci İ+1 diff(x) fonksiyonunu, işlemci k ise sin(z) fonksiyonunu paralel olarak icra edecektir.

Süreçler arası iletişim için belirgin bir dil yapısı kullanılmasına gerek yoktur ve kullanılmamıştır. Bir hesaplama bir diğer fonksiyonun sonucuna ihtiyaç duyuyor ise bu sonuç ilgili fonksiyon tarafından sağlanana kadar bloke olur. ParAfl anlamsal yapısı "Tembel İşleme" ("Lazy Evaluation") tekniği üzerine oturtulmuştur.

Bu teknikte, bir ifadenin sonucu kesinlikle gerekli olmadıkça ifade işleme koyularak sonucu bulunmaz. Şekil-1.7.1'de yer alan ParAfl programında a değişkeni b değişkeninin değerine bağlı olmasına rağmen sıraları, bu deyimlerin sırası önemli olmadığından ters yazılmıştır. En son sırada bulunan fonksiyon tanımına dikkat edilirse eğer p TRUE ise sadece y formal parametresinin değeri geri döndürülmektedir.


```

a == b*b;
b == 2;
result f(a,b);
f(x,y) == if p then y else x+y

```

Şekil-1.7.1 ParAfl Lazy Evaluation örneği.

Bu durumda $f(a,b)$ fonksiyon çağırısında eğer p TRUE çıkarsa sadece b nin değeri önemli olacak ve $b==2$, bağlanması bu noktada (gerektiği için) icra edilecektir. Bu durumda $a==b*b$ hiç icra edilmeyecektir. Eğer p NIL(FALSE) sonucunu verecek olsaydı a ve b nin sonuçlarının bulunması gerekecekti. Bu durumda a hesaplanmaya çalışıldığında b nin değerine ihtiyaç duyulacak ve $b==2$ bağlanması yapılacak daha sonra a buna dayalı olarak hesaplanacaktı. Bu teknikte, bir ifadenin sonucuna ihtiyaç olmadıkça ifade işleme koyulmaz. Ancak bu dil içinde, programcının icra sırasını ve zorunluluklarını belirleyebildiği istek ifadeleri (eager expressions) yer alabilir.

Bu çalışmada tasarlanan RELP programlama dilinde, program içinde paralel çalışacak bölümlerin, belirli fiziki işlemcilere atanmasına yönelik bir yapı kullanılmamıştır.

Her RELP bloğu, belirli bir veri kümesini girişten alıp, çıkışta belirli bir veri kümesi üreten, bir fonksiyon gibi düşünülebilir. Bu da RELP dilinin fonksiyonel yapısını göstermektedir. RELP blokları kendilerine giriş verileri akmaya başladığında uygulama yapmaya başlayıp, akış sona erdiğinde uygulamayı durdururlar. Yani bir RELP bloğu, bir başka RELP bloğu tarafından ancak veri gönderilerek aktive edilebilir. Bu mekanizma veri akış yapısı ile fonksiyonel yapının bulunduğu ideal bir noktadır (Almasi et al, 1989)(*)

(*) Bknz. (Almasi et al,1989) Pg.190-191

1.8 Mantıksal Paralel Programlama Dilleri

Bir çok mantıksal paralel programlama dili AND/OR paralel işleme tekniğini ve paylaşılan değişkenler kullanır. Örnek olarak Concurrent PROLOG ve PARLOG verilebilir. (Almasi et al,1989), (Shapiro,1989).

1.8.1 "Concurrent PROLOG" programlama dili

Ehud Shapiro tarafından tasarlanmıştır (Shapiro,1989). Paralel çalışma "VE" ("AND") mantıksal ilişkisindeki mantıksal hedeflerin (Goals) ve guarded Horn clause yapısında yer alan mantıksal hedeflerin paralel işlenmesine dayanır. Paralel süreçler paylaşılan değişkenler üzerinden haberleşir. Eş zamanlama için de değişkenler kullanılır. PARLOG programında X ve Y ile Z ve W hedef kombinasyonları birbirine paralel olarak denenecek. Her kombinasyon

Bir mantıksal hedef süreci, atama yapma hakkına sahip olduğu değişkene atama yapar. Bu değişkeni sadece okunabilir bir değişken olarak gören ve değer atanmamış ise bloke olup bekleyen diğer bir mantıksal hedef süreci bu değişkeni okuyarak icrasına devam eder. Bu şekilde hem iki süreç arasındaki iletişim hem de eş zamanlama başarılmış olur. Şekil-1.7.1'de yer alan üç adet hedef arasında X değişkeni bir kanal oluşturur. X değişkenine değer atamaya yetkili olan A hedefi, X ile bir değer eşleştirdiğinde bu değer, X'i sadece okuyabilen B ve C hedefleri tarafından kullanılabilir. B ve C hedefleri, X'e değer atanana kadar askıya alınıp, X üzerinden A ile eş zamanlanırlar.

hedef_A($X^{\wedge}$,Y),hedef_B($X^?$,Z,W),hedef_C(M, $X^?$)

Şekil-1.8.1 Paylaşılan değişkenler üzerinden iletişim ve eş zamanlama.

1.8.2 "PARLOG" programlama dili

Clark ve Gregory tarafından Londra Üniversitesinde geliştirilmiştir. (Clark, 1986). PARLOG, "AND" ve "OR" paralel çalışma şekillerine sahiptir. İki adet "AND" operatörü vardır. "&" operatörü mantıksal hedeflerin sıralı olarak ard arda incelenmesini, "," operatörü ise hedeflerin paralel incelenmesini sağlar. Her iki durumda operatör ile bağlanan mantıksal hedeflerin her ikisinin de başarılı olması veya bir tanesinin başarısız olmasına kadar denenmesi söz konusudur. "." operatörü ise mantıksal hedeflerden bir tanesinin başarılı olmasına veya her ikisinin başarısız olmasına kadar paralel olarak denenmesini sağlar. Bir ";" karakteri, kendisinden sonra gelen hedeflerin ancak kendisinden önce gelen hedeflerin başarılı olmaması durumunda araştırılmasını sağlar. Şekil 1.8.2-1'de yer alan PARLOG programında X ve Y ile, Z ve W hedef kombinasyonları birbirine paralel olarak denenecek, her kombinasyon ise kendi içinde sıralı olarak (Önce X sonra Y ve önce Z sonra W şeklinde) denenecektir. İkinci kural ancak birinci kural başarısız ise denenecektir.

1. A -(X&Y),(Z&W);
2. A -R,S,T.

Şekil-1.8.2-1 PARLOG paralel işleme yapısı.

Parlogda iletişim ve eş zamanlama paylaşılan değişkenler üzerinden yapılır. Her bir ilişki için yapılan "mode" bildiriminde giriş ve çıkış değişkenleri belirlenir(Şekil 1.8.2-2). Bu mode bildirimi ile ilişkili kural ancak giriş parametrelerinden birisinin değer atanmamış konumdan değer atanmış konuma geçmesi ile başarılı oluyor ise ve bu değer diğer süreç tarafından atanmamış ise, kuralın icrası askıya alınır ve beklemeye girer. Çıkış parametresi pozisyonunda yer alan gerçek parametre ise kural başarılı olduğunda, karşılık gelen lokal değişkenin o anki değerini alır.

Şekil-1.8.2-2'de yer alan "mode" bildiriminde "x" giriş, "const-list" ise çıkış parametresidir.

```
mode construct(x?, const-list^).
```

Şekil-1.8.2-2 Parlog İletişim ve Eş Zamanlama yapısına örnek.

Parlog da Concurrent PROLOG gibi, nondeterminizm yapısı olarak Guarded Horn Clause kullanır. Çıkış değişkenleri ancak terim seçildikten sonra sahip oldukları değerleri dışarıya verebilirler. Şekil-1.8.2-3'de "Guarded Horn Clause" genel yapısı yer almaktadır.

```
X :- H1,H2,H3.....,Hn : G1,G2,G3,.....,Gn.
```

Şekil-1.8.2-3 "Guarded Horn Clause" yapısı.

Bu yapıda X hedefinin TRUE sonucunu alabilmesi için H1,...,Hn hedefleri birbirine paralel olarak denenir, başarılı olanlardan bir tanesi, Hx, rastgele seçilerek buna karşılık gelen Gövde hedef Gx'in denenmesine başlanır. Şekil-1.8.2-4'de bu konuda örnek bir program yer almaktadır.

```
P(X):-R(X):S(X).
P(X):-T(X):U(X).
P(X):-V(X):Y(X).
R(1):-A(1).
V(2):-B(2).
T(3):-C(3).
```

Şekil-1.8.2-4 Guarded Horn Clause örneği.

Bu örnekte R,V ve T birbirine paralel olarak denenmekteyken her üçü de X değişkenine birbirinden bağımsız olarak sırasıyla 1,2 ve 3 atayacaktır. Bunlardan biri rastgele seçilecektir. Eğer R(1)

seçildiyse bir sonraki denenecek hedef $S(1)$ dir. Eğer $V(2)$ seçilirse bir sonraki denenecek hedef $Y(2)$ dir. Aynı husus $T(3)$ için de geçerlidir. Bu seçim yapıldıktan sonra $P(X)$ kuralını kullanan kural için X değeri belirlenmiş olur. Bu ana kadar, X değeri gizli tutulur. Programcılar bu yapıyı kullanırken dikkatli olmalıdırlar, örneğin eğer $R(1)$, $U(3)$ ve $Y(2)$ TRUE ise ve $R(1)$ "guard"ı seçilip $S(1)$ başarısız çıkar ise P kuralı başarısız olur, oysa V veya T guardları da başarılı olduğundan bunlardan biri de seçilebilir ve takip eden gövde hedefleri de başarılı olacağından P başarılı çıkardı.

Bu çalışmada tasarımılanan RELP programlama dilinde gerçekler veri tabanındaki tablolarda depolandığı takdirde, mantıksal sorgulamaların ace programlanabilmesi mümkündür. Bu mekanizmada mantıksal değişkenlerin yerini RELP Hareket Değişkenleri almaktadır.

1.9 Dağıtılmış Veri Yapılarına Dayanan Paralel Programlama Dilleri

Dağıtılmış veri yapıları "Linda", "Tuple Space Smalltalk" ve "Orca" programlama dillerinde kullanılmaktadır.

"Linda" Yale Üniversitesinde David Gelertner ve arkadaşları tarafından geliştirilmektedir. Linda "Tuple Space" adlı çok değişik bir iletişim mekanizması kullanmaktadır. Linda programcayı paralel çalışma mantığını devamlı aklında tutması zorunluluğundan kurtarmakta ve paralel programlamayı sıralı programlamayla aynı düzeye getirmektedir.

Dağıtılmış veri yapıları aynı anda birden fazla süreç tarafından işlenebilen veri yapılarıdır. "Linda" programlama dilinde kullanılan dağıtılmış veri yapısı olan "Tuple Space" tüm süreçler tarafından paylaşılan bir global bellektir. Bu veri yapısının elemanları

Pascal'daki kayıt yapısına benzer yapıdaki kayıtlardır. Örneğin (9098, "Cüneyt Sağlam",95) üç sahaya sahip bir kayıttır. "Tupple Space" üzerinde süreçler tarafından yapılabilecek üç operasyon vardır. "Out" "Tupple Space"e bir kayıt ekler. "in" "Tupple Space"den bir kayıt okur ve kaydı iptal eder. "read" aynı okuma işlemini kaydı iptal etmeden yapar. Kayıtlara erişim içerik kullanılarak yapılır.

Örneğin 9098 numaralı öğrencinin ismi ve notunu okumak için:

`read(9098, var isim, var not)` (Demand Driven) iletişimi işlemi yapılmalıdır. Birden fazla uyuşan kayıt var ise bir tanesi rastgele seçilir, tekil anahtar sahaların (Unique Keys) kullanımı programcının sorumluluğundadır. Kayıtlar üzerinde değer değiştirme işlemleri, önce kaydın "in" işlemi ile okunması, gerekli değerin değiştirilmesi, ve değişmiş kaydın "out" işlemi ile tekrar "Tupple Space"e yazılması ile yapılır. Bir sürecin okumak istediği kayıt eğer "Tupple Space" içinde yok ise, süreç bu kayıt "Tupple Space"e bir diğer süreç tarafından yerleştirilene kadar bloke olur. Bu sayede, süreçler arasında eş zamanlama sağlanır. Süreçler arasında iletişim "Tupple Space"e gönderilecek veriyi içeren kaydın gönderen süreç tarafından depolanması ve alıcı süreç tarafından okunması ile gerçekleştirilir.

"Linda" Programlama Dilinde süreçler ortak bir işi, eş işlemler yapacak şekilde paylaşarak paralel olarak gerçekleştirirler. (Cooperative Co-Workers.) Her süreç aynı işi yaptığından süreçleri fiziki işlemciler ile ilişkilendirmek için bir yapıya ihtiyaç duyulmamıştır. Bu paralel çalışmada süreçler yapılacak işleri "Tupple Space"den okur, sonuçları ve ürettikleri yeni yapılacak işleri "Tupple Space"e yerleştirirler.

Linda programlama dilinin kayıt uyuşturma ve kayıt okuma (binding) özelliği bu çalışmada tasarlanan RELP programlama dilinde "İlişki Uygulama" işleminde kullanılmıştır. Ancak RELP'de uyuşan

kayıtların tümü seçilerek işlem görmektedir. F çıkış kenarlarından birisine transfer ederek yapar. (Harowitz, 1987).

Şu ana kadar kontrol yapısı itibarıyla, kontrol güdümlü (Control Driven) ve iletişim yapısı itibarıyla mesaj iletişimine dayanan paralel programlama dillerinden Occam, Ada, Concurrent C örnekleri verilmiştir. Concurrent PROLOG ve PARLOG gibi Mantıksal Paralel Programlama Dilleri, patern güdümlü (Patern Driven) kontrol yapısına ve paylaşılan bellek tipinde iletişim yapısına sahiptir. Fonksiyonel paralel programlama dilleri ise kontrol yapısı itibarıyla talep güdümlü (Demand Driven), iletişim yapısı itibarıyla ise mesaj iletişimine dayanan yapıdadır. Diğer önemli bir paralel programlama dilleri sınıfı ise kontrol yapısı yönüyle Veri Güdümlü (Data Driven) iletişim yapısı yönüyle Mesaj iletişimine dayanan Veri Akış Programlama Dilleridir. (Data Flow Programming Languages) Bu diller özel Veri Akış Bilgisayar Mimarilerine, yani özel donanıma dayalı olarak tasarlanmıştır.

1.10 Veri Akış Paralel Programlama Dilleri

Veri akış programlama dillerinde yazılan programlarda, işleme girecek veriler, programdaki operasyonlar arasında akış halindedir. Bu modelde programı bir ağ(graph) olarak düşünebiliriz. Bu ağda düğümler(node) operasyonları, kenarlar (edges) ise verilerin aktığı yolları temsil ederler. Bu ağa Veri Akış Dilleri terminolojisinde "Veri Akış Ağı-Data Flow Graph" adı verilir. Bir Üst Düzeyli Veri Akış Programlama dili, program deyimleri ile ifade edilen hesaplamayı, Veri Akış bilgisayarında icra edilmek üzere Veri Akış Ağına çevirir. Veri Akış Ağında sadece operasyonları temsil eden düğümler bulunmaz. Şartlı veri transferi ve dağıtımı yapan "seçici-selector" ve "dağıtıcı-distribütör" düğümler de mevcuttur. Seçici düğüm mantıksal TRUE verisi geldiğinde ve T kenarından bir veri geldiğinde bu veriyi çıkışına geçirir. Aynı şekilde mantıksal FALSE verisi aldığı anda ve F kenarından bir veri geldiğinde bunu çıkışına geçirir. Dağıtıcı düğüm ise benzeri işlemi, giriş kenarındaki veriyi,

aldığı mantıksal veriye bağlı olarak T veya F çıkış kenarlarından birisine transfer ederek yapar. (Horowitz,1987).

1.10.1 "VAL" programlama dili

VAL programlama dili MIT'de (Massachusetts Institute of Technology) geliştirilmiştir(Dennis,1979). VAL dili salt uygulama bazlı (Purely Applicative) bir veri akış dilidir. Fonksiyonel programlama dilleri ailesindendir. Deyimler yerine sadece ifadeler ve fonksiyonlar yer alır. Bir ifadenin işleme girmesinden önce değişkenlere değer atamak mümkündür(let-in kombinasyonu) bu değişkenlerin değeri sadece bu ifade için geçerli olup ifadenin işleme girmesinden sonra geçersizdir.

Böylece yan etki (side effect) söz konusu olmamaktadır.Şekil-1.10.1-1'de bu yapıya bir örnek verilmiştir.

```
let x:real; t: real;
    t:= a+4;
    f:=t*3.1417;
in t*f endlet
```

Şekil 1.10.1-1 Val "let-in" yapısı.

VAL programlama dilinde paralel ifadeler üç ana bölümden oluşur:

1. Menzil tanımı (Range Spesification)
2. Ortam tanımı.
3. Sonuçların toplanması.

Şekil-1.10.1-2'de bir VAL paralel ifadesi örneği yer almaktadır.

```
forall K in (1,100)
    L:boolean:= asal(K);
    construct L
endall
```

Şekil-1.10.1-2 Paralel asal sayı testi.

Şekil-1.10.1.-2'deki örnek program 1 ile 100 arasındaki sayıların asal sayı testlerini yapmakta, mantıksal tipe sahip 100 elemanlı L dizinli değişkeni içindeki, asal sayılara karşılık gelen elemanlara TRUE değeri atamaktadır. Asal fonksiyonu her dizinli değişken elemanı için (donanımın sağladığı imkanlar ölçüsünde ve derleyicinin değerlendirmesine bağlı olarak) ayrı ve paralel olarak çalışabilir. VAL sıralı olarak işlenmesi gereken ifadeler için de gerekli yapılara sahiptir (Horowitz,1987).

Bu çalışmada tasarımı yapılan RELP programlama dili, VAL gibi veri akış kontrol yapısına sahip olmasına rağmen, veri akışı olgusunu programcıya daha açık bir biçimde sunmaktadır. Diğer bir farklılık olarak, RELP Makro Veri Akış Modeli uygulamaktadır. (Almasi et al,1989)(*) Bu modelde, aritmetik ve mantıksal operatörler yerine , veri yapıları üzerinde çalışan daha komplike operatörler arasında veri akışı söz konusudur. Yani daha büyük parçalı paralelleştirme yapılmaktadır.

1.11 Genel değerlendirme

İşlemciler arasındaki iletişim hızı ve bilgisayar sistemlerinin kaynakları büyüdükçe, dağıtılmış bilgisayar sistemlerine yönelik programlama dilleri daha çok kullanılacaktır. Bu sayede programcılar altta yatan paralel sistemin kompleks yapısından soyutlanmış bir programlama ortamında yazacakları paralel programlar ile daha hızlı çalışan uygulamalar geliştirebileceklerdir. Günümüzde bilgi işlem sistemlerinde çok işlemcili konfigürasyonlar kullanıma sunulmuştur. Bu işlemci gücünden sadece iletişim sistemi düzeyinde değil programlama dili düzeyinde yararlanmak ancak paralel işlemeye yönelik tasarlanmış programlama dilleri ile mümkün olabilecektir.

(*) Bknz. (Almasi et al,1989) pg 190

(*)Bknz. Makro Dataflow (Almasi et al,1989) pg 357.

Değişik tip paralel programlama dillerinin güçlü oldukları noktalara eğilecek olursak; Fonksiyonel ve Uygulama Tabanlı (Applicative) programlama dillerinin programın değişik bölümleri arasında paylaşılan değişkenler aracılığıyla oluşan etkileşimi ve bunun neticesindeki istenmeyen Yan Etkiyi (Side Effect), yok ettiğini söyleyebiliriz. Yine bu dillerdeki indirgeme(reduction) özelliği, bir operasyonun sadece, başka bir operasyonun gerçekleşebilmesi için kendisinden sonuç istendiğinde devreye girmesini sağlayan Tembel İşlemeye (Lazy Evaluation) izin verir. Bu sayede lüzumsuz işlem yapılmamış olur.

Veri Akış programlama dillerinde, operasyon için gerekli olan veriler, işlemin yapılacağı noktaya ulaşır ulaşmaz, işlem yapılır. Burada herhangi bir merkezi kontrol mekanizmasının güdümü söz konusu değildir. Bu nedenle Veri Akış Dilleri, bir programdaki potansiyel olarak paralelleştirilebilecek tüm bölümleri ortaya çıkarıp kullanabilir. (Almasi et al,1989)(*).

Mantıksal paralel programlama dilleri, belirgin bir süreç modeline, süreç konumuna, süreçler arası iletişim ve eş zamanlama modeline sahiptir. Bu model, mantıksal değişken soyutlaması ile birleştiğinde, bu dillerin paralel işlemedeki gerçek gücü ortaya çıkar.(Shapiro,-1989)(**)

Nesne Temelli programlama dilleri, gerçek dünyanın en kolay modellenenebildiği programlama dilleridir. Bu diller bir programın otonom nesneler kümesi şeklinde oluşturulmasını teşvik eder. Bu nesneler, ayrı işlemciler üzerinde paralel olarak çalıştırılabilirler. Nesneler arasındaki iletişim ise paralel süreçler arasındaki iletişim için uygun bir soyutlamadır. Özetle nesne temelli paralel programlama dilleri, paralel işleme şemasına doğal bir uyum gösterirler.

(*) Bknz (Almasi et al,1989) pg.190.

(**)Bknz. (Shapiro,1989) pg.497

Dağıtılmış veri yapılarına dayanan programlama dilleri, paralel çalışan program birimleri arasındaki tüm iletişimi dağıtılmış veri yapıları üzerinden gerçekleştirirken, programcıyı, paralel çalışma ve aynı anda olan olayları düşünmekten kurtararak, sıralı program mantığını uygulayabilmesini sağlar.

Sıralı Süreçlerin mesaj iletişimine dayanan CSP tipi dillerin avantajları ise sırasıyla, paralel işlem modelinin basitliği, değişik paralel mimarilere uyarlama kolaylığı ve iyi bir donanım kontrolünün sağlanabilmesidir. Ancak birçok araştırmacı bu tip, algoritmik dillerin, yapılarından tamamıyla çıkartılamayan seri çalışma nedeniyle, paralel işlemede tercih edilmemesi gerektiğini savunmaktadır.(Bal et al,268)(*)

2. "RELP" PROGRAMLAMA DİLİNİN GENEL TANITIMI

2.1 "RELP" Dilinin hedeflenen uygulama alanı (Domain)

RELP programlama dili ilişkisel operasyonlar ile bilgi işleme yönelik olarak paralel bilgisayar mimarileri hedeflenerek tasarlanmıştır. Özellikle karar sürecini hızlandıran hızlı sorgulama uygulamalarında, karar destek sistemlerinde veri tabanı sorgulamalarında kullanılması öngörülmektedir.

2.1.1 Karar Destek Sistemleri

Karar Destek Sistemleri, Verimlilik'ten çok Etkinliği hedefler. Verimlilik, bir işi, tanımlı performans kriterlerine göre en iyi şekilde yapmak ile ilgilidir. Etkinlikte ise yapılması gerekenin tam olarak ne olduğunun belirlenmesi ve seçilen kriterin geçerliliğinin doğrulanması önem kazanır. Etkinlik öğrenme ve adapte olmayı gerektirir; hatalı başlangıç ve tekrar tekrar aynı işin yapılması riskini taşır. Verimlilikte ise çalışma alanının daraltılarak zaman

(*) Bknz (Bal et al,1989) pg. 268.

ve maliyetin en aza indirgenmesi çabası vardır. Esas olarak programatik bir süreçtir. Verimlilik, yapılan işin en az maliyet ve zamanda yapılmasını hedefler. Etkinlikte ise yapılan işin doğru olarak yapıldığından emin olunması hedeflenir.

Geleneksel otomatik bilgi işlem, yapılan işlerde zaman ve maliyetin azaltılmasında, doğrusal programlama ve kritik yol planlaması gibi uygulamalarla, daha çok, kaynakların dağıtılmasında verimliliğin arttırılmasına yönelmiştir, etkinlikle ilgilenmez. Bilgisayar genel olarak verimliliği hedef alan bir araçtır. Örnek olarak bir bankanın her ay sonunda bilanço çıkarması işini bilgisayarla yapmak, yapılan işteki verimliliği büyük ölçüde arttıracaktır. Ancak, aynı bankanın yatırımlarını hangi alanlara yöneltmesi gerektiği gibi, yarı yapısal bir problemin çözümünde verilecek kararı desteklemeye, geleneksel bilgi işlemin verimlilik yönelimli araçları yetersiz kalır.

Dolayısıyla Karar Destek Sistemlerinin üzerinde çalıştığı problemlerin kesin çözüm yolları ve kuralları yoktur. Yönetici kararın oluşmasında aktif rol oynar. Karar Destek Sistemi, karar sürecinde sağladığı araçlarla yöneticiye destek olmayı hedefler. Hedef yöneticilerin karar sürecinde yeterliliklerini ve yeteneklerini arttırarak etkin kararlara ulaşılmasına destek olmaktır. Karar Destek Sistemlerinin yöneticilere faydası, karar sürecini otomatize etmeye, hedefleri tanımlamaya, yada çözümler empoze etmeye kalkmayan, kendi kontrolleri altında bir destek aracı sağlamasındadır.

Karar Destek Sistemlerinde bilgi veya veri genellikle acilen alınacak bir karar için gerekli olur. Bu veriler, bilgi işlemde kullanılan tüm detay verileri içermez, çoğu zaman bilgi işlemdeki verileri özetleyen istatistiki verilerdir. Yine bir çok uygulamada verilerin çok duyarlı ve en son ana kadar işlenmiş olması gerekmez. Çoğu zaman, geçmiş dönemlere ait genellemeler yapmaya yetecek kadar duyarlı olmaları amaca hizmet eder. Yapısal olmayan veriler programın kullanımı esnasında tanımlanarak yapılandırılabilir.

Geleneksel bilgi işlem uygulamalarının aksine programın akışı, kullanıcının o anki duruma göre soracağı sorulara bağlı olarak gelişir. Yeni sorular, yeni akışlar ortaya çıkartır. Bu işlem karar süreci içinde zincirleme olarak yinelenir. Karar Destek Sistemlerinde kullanıcılar bilgi işlem personeli değildir ve genellikle bir programlama dilinin ayrıntılarını öğrenmeye zamanı olmayan yöneticilerdir. Bu nedenle program veya programlama aracı olabildiğince basit olmalıdır. (1984)(Almasi et al., 1989). yani eski problemi olmayan (side effect free) fonksiyonel ve nesne temelli

Özetle, karar destek sistemlerinde hızla üretilen soruları, hızla üretilen uygulamalar ile göğüslemek, bu sorulara hızla cevap vermek, başlangıçta istenen yapıda olmayabilecek veri organizasyonu içinde, verileri anında yeni yapıda organize edebilmek ve bütün bunları çok kolay bir şekilde yapabilmek önem arz etmektedir. Veri Akış Modeli ve Nesne Temelli yapı ile sağlanan yeni programlama

2.2 "RELP" Dilinin tasarlanmasının amacı

Bu bağlamda programlama dilinin yeterli soyutlamalara sahip olması Geleneksel olarak, donanım gelişmeleri, yazılım gelişmelerinden önce gerçekleşmiş ve sonuçta yaratılan donanım kapasiteleri yıllar boyunca kullanım dışı olarak kalmıştır(Cybenko,1992). Bu çalışmada, şimdiki ve gelecekteki, paralel bilgisayar mimarisindeki gelişmelerden verimli olarak yararlanmayı sağlayacak Karar Destek Sistemlerine yönelik bir dilin tasarımı ve prototip uygulaması hedeflenmektedir. yeni ne olursa olsun, istenen işlem noktasına, istenen verilerin yitirilmeden organizasyonu içinde ulaştırılabilmesi sağlan-

Halen bilgi işlemde kullanılmakta olan programlama dillerinin, yeni gelişen paralel bilgisayar mimarilerden yararlanacak hale getirilmeleri oldukça güçtür. Her şeyden önce bu dillerin anlamsal yapıları programcının zihninde, paralel işlemeden yararlanma fikrini yaratmamaktadır. Bu dillerin derleyicileri içinde yapılacak karmaşık program akış analizleri neticesinde paralel çalışabilecek program bölümlerinin ayırt edilmesi ve bu bölümlerin paralel işlemciler üzerinde çalıştırılması yaklaşımından çok, yeni paralel programlama dillerinin tasarlanması daha olurlu görülmektedir. Paralel bilgisayar

mimarilerine yönelik pek çok yeni programlama dili geliştirilmekte ve bu dillerin ticari sistemler haline dönüştürülmesi süreci işlemektedir (Almasi et al., 1989) (Bal et al., 1989). Veri Tabanı Yönetim Sistemi üreten firmalar, paralel işlemcilerden yararlanma konusunda atılım yapmaktadırlar. (Informix Technical Brief, 1994).

Paralel işlemede çekici özelliklere sahip olan Veri Akış Modelinin uygulandığı (Ben Lee et al., 1994)(Almasi et al., 1989), yan etki problemi olmayan (side effect free) fonksiyonel ve nesne temelli bir programlama dilinin, paralel işlemciler mimarisi üzerinde etkin olduğu takdirde, paralel programlama dilleri arasında önemli bir yer alabileceği düşünülmüştür.

Programlama kolaylığı diğer bir faktör olarak ele alınmıştır. Veri Akış Modeli ve Nesne Temelli yapı ile sağlanan yeni programlama konseptinin programlama kolaylığı sağlayacağı değerlendirilmiştir. Bu bağlamda programlama dilinin yeterli soyutlamalara sahip olması öngörülmüştür (Agha, 1993).

REL P programlama dili, Karar Destek Sistemlerinde, uygulamaları hızla ve basit bir biçimde üretebilecek ve paralel işlemeden yararlanarak hızla sonuç verecek şekilde tasarlanmıştır. Dilde veri akış kontrol yapısı kullanılmasıyla, verilerin başlangıçtaki statik organizasyonu ne olursa olsun, istenen işlem noktasına, istenen verilerin yine isten organizasyon içinde ulaştırılabilmesi sağlanmıştır. Dilin yapısının Görüntüsel Programlamaya olanak tanıması sayesinde, kullanımı basit kullanıcı arayüzlerinin yaratılabilmesi mümkündür.

Tez konusu seçimi esnasında İngilterede Londra Brunel Üniversitesinde ve British Science and Information library'de, 33000 makaleyi kapsayan BIDS Bilgisayar Veri Tabanında, CD-ROM kataloglarında, A.B.D. US Naval Postgraduate School kütüphanesinde ve Bilgisayar Sisteminde, INTERNET'de, ve konu ile ilgili

periodiklerde detaylı tarama çalışması yapılmıştır. yapılan geniş literatür araştırması sonucunda, seçilen uygulama alanına dönük ve yukarıdaki özelliklerin tümünü sağlama yönünde atak yapan bir programlama dili tasarım/uygulama çalışmasına rastlanmamıştır.

Böyle bir çalışmaya rastlanamamasına rağmen, benzer bir çalışma olsa dahi, yapılacak bağımsız, özgün tasarım ve uygulamanın bu konuda ilave deneysel sonuçlar temin edeceği düşünülmüştür.

2.3 "RELP" Dilinin uygulanmasının amacı

RELP dilinin tasarımı esnasında, dilin amaçlanan, paralel işlemcilerden yararlanma gücünün sınanması için, GPSS programlama dili kullanılarak benzetim programları yazılmış, ve olumlu sonuçlar elde edilmiştir. Ancak böylesine değişik bir dilin, uygulanabilir olduğunu göstermenin tek yolu, dilin prototipinin gerçek paralel işleme ortamında yaratılıp, denenmesinden geçmektedir. Bu nedenle, tasarlanan programlama dilinin, Çok İşlemcili Paylaşılan Bellek MIMD bilgisayar ortamında bir uygulaması yapılmıştır. Prototip derleyici kullanılarak, dört işlemcili paylaşılan bellek yapısındaki bir sistemde yapılan deneylerin sonuçları bölüm 7' de açıklanmıştır.

2.4 "RELP" Dilinin genel özellikleri

Önerilen RELP dili, Fonksiyonel , Veri Akış, Nesne Temelli, ve Mantıksal programlama dillerinin izlerini taşımaktadır. RELP Dili, nesne temelli (Object Based) olup, nesne yönelimli (Object Oriented) değildir. Makro Veri Akış modelini takip eden dil, diğer veri akış (Data Flow) dillerinden bir çok bakımdan farklı olup, klasik paralel donanım mimarilerinde uygulanabilir yapıdadır. Dil Yan Etkinin bulunmadığı Uygulamalı (Applicative) yapıya sahiptir. Mantıksal Programlama dillerinde kullanılan patern uyumlama, gerçek arama, kural doğrulama, tipindeki işlemler bu dilde kolaylıkla programlanabilmektedir.

Böylece, ilişkisel bir veri tabanı üzerinde veri işleme, sorgulama, mantıksal sonuç çıkarma işlemlerini, paralel alt yapıdan maksimum yararlanarak, yan etkiden uzak, güvenli bir şekilde gerçekleştirebilecek bir Programlama dili yaratılmıştır. Ayrıca, RELP, programcıya, problemi modelleme kolaylığı sağlayan nesne temelli bir yapı da sunmaktadır. (*)

RELP Programlama dilinin tasarımı sürecinde, konu ile ilgili birçok makale ve kitap kaynak olarak kullanılmıştır. Bu kaynaklar'da yer alan bilgiler tasarıma genel bir yön vermiş, yeni dilde kullanılacak dil özellikleri ile ilgili fikirlerin oluşmasını sağlamıştır.

Veri Tabanı, Dağıtılmış Veri Tabanları ve İlişkisel Veri İşleme dilleri ile ilgili, (Ceri,1984), (Danforth,1992), (Date,1988), Fonksiyonel programlama dilleri ile yapılan araştırmada kaynak olarak (Almasi et al, 1989), (Backus,1978), (Bal et al, 1989), (Chen,1991), (Fradet et al,1991), (Haltstead, 1991), (Hudak,1986), - (Hudak,1989), (MacIennan,1981) kullanılmıştır.

Bu kaynaklardan; (Bal et al,1989) Paralel Programlama Dilleri, Mantıksal programlama dilleri ile ilgili yapılan incelemede ise, (Abiteboul et al,1991), (Bal et al,1989), (Brogi et al,1991), (Clark et al,1986), (Saphiro,1989) kaynakları üzerinde yoğunlaşmıştır.

İzleni içerici geniş kapsamlı makalelerdir.

Nesne yönelimli ve nesne temelli programlama ile ilgili, (Almasi et al, 1989), (Bal et al, 1989), (Chin,1991), (Danforth, 1988), (Kaiser,1992), (Lieberman,1987), (Yonezawa,1986), (Yonezawa,1993), (Grimshaw,1993) kaynakları incelenmiştir.

Veri Akış Programlama dilleri ile ilgili (Almasi, 1989) (Arvind, 1989), (Bal et al,1989), (Chiola,1993), (Gabb,1984), (Pagan,1981),

Paralel bilgisayar mimarisi ve donanımı üzerine (Almasi et al,1989), (Athas et al, 1989), (Dally,1991), (Gottlieb,1991), (Kumar,1991), (Pritchard,1991), (Transputer Development System,1990),

Veri Akış Programlama dilleri ile ilgili (Almasi, 1989) (Arvind, 1989), (Bal et al,1989), (Chiola,1993), (Gabb,1984), (Pagan,1981),

(*) Bir paralel programlama dilinde Veri Akış ve Fonksiyonel Programlama dillerinin özelliklerinin birlikte bulunmasının avantajları (Almasi et al, 1989)'da sayfa 190-191'de detaylı olarak anlatılmıştır.

(Tucker, 1988) kaynakları kullanılmıştır.

Paralel yazılım konusunda, (Akl,1989) (Almasi et al,1989) (Fox,1991), (Gotlieb,1991), (Holt,1978), (Lakshmivaran,1990), (Montry,1991), (Snir, 1991), (Lewis,1993), kaynaklarından yararlanılmıştır.

Programlama dilleri teorisi, dil tanımlaması ve uygulaması konularında, (Andrews,1988), (Andrews,1991), (Chen, 1991), (Gehani,1993), (Gordon,1979), (Horowitz,1983), (Li,1991), (MacLennan,1981), (Burke,1991), (Pagan,1981), (Agha,1993) kaynaklarından, Veri Tabanı, Dağıtılmış Veri Tabanları ve İlişkisel Veri İşleme dilleri ile ilgili, (Ceri,1984), (Danforth,1992), (Date,1981), (Demers et al,1992), (Informix-Online,1994), (MacLennan, 1983), (Maier,1987), (Ozsu,1991), (Parsaye,1989), (Salzberg,1988) kaynaklarından faydalanılmıştır.

Bu kaynaklardan, (Bal et al,1989) Paralel Programlama Dilleri, (Hudak,1986) Fonksiyonel Programlama Dilleri, (Shapiro,1989) Paralel Mantıksal Programlama dilleri, (Chin et al, 1991) Nesne Temelli Programlama dilleri ile ilgili bir çok dilin genel özelliklerini içeren geniş kapsamlı makalelerdir.

RELDP dili, etkin bir paralel bilgi işlem için, çoklu-bağımsız-işlemciler (Multicomputer) ve paylaşılan-bellekli-çoklu-işlemciler(Shared Memory Multiprocessor) MIMD mimarilerinde uygulanabilecek şekilde tasarlanmıştır.

Dil, Uygulamalı ("Applicative") bir yapıya sahiptir. En önemli "hesaplama konumu değiştiren operasyon", İlişki-Uygulama ("Relation Application") operasyonudur. İlişki uygulama operasyonu, dile "Uygulamalı" (Applicative) özellik kazandırmaktadır. Fakat kontrol akışı fonksiyonların bir diğerini çağırması şeklinden değişik olarak GPSS'de ("General Purpose Simulation System")

olduğu gibi "Transaction" (hareket eden veri) akışına bağlanmıştır. Bu da dile Veri Akış dillerinin özelliğini kazandırmaktadır. İlişkiler ("Relations") bellekte ve diskte fiziki olarak temsil edilirler. Veri akış kontrol şeması sayesinde Veri Tabanı Programcısının karşılaştığı en büyük zorluk olan Veri Tabanı Dolaşımı (Database Navigation) basitleşmiştir.

Bu dilde ilişki tablolarının içinde yer alan veri elemanları veya klavyeden yeni girilen veriler programcının belirlediği şekilde sistemin içinde akarlar. Bu veri elemanlarının bir veya birden fazlasının biraraya gelerek oluşturdukları akış halindeki birimlere "Hareket" adı verilir. Bu Hareketler programcı tarafından arka arkaya dizilerek oluşturulan bloklar dizisi içinde, blokların içine girip çıkarak akarlar. Eğer her Hareket bir mesaj olarak düşünülürse bu akışın bloklar arasında asenkron mesaj alışverişi şeklinde gerçekleştiği söylenebilir. Dil yapısında değişik fonksiyonlara sahip standart bloklar yer almaktadır. Programcı tarafından belirlenen operasyonların geçiş yapan Hareketlere uygulandığı bloklar da mevcuttur. Hareketler asıl veri ile birlikte ara sonuç olarak oluşturulan verileri de program içinde yararlı olarak kullanılacakları noktalara birlikte taşıyabilirler. Böylece, veri, Hareketler halinde organize olarak dil blokları üzerinden sistem içinde bloktan bloğa akar. Her blok, içinden geçen Harekete belirli bir işlem yapar. Hareketin içindeki veriye dayalı mantıksal test işlemi yaparak Hareketin bloktan çıktıktan sonra gideceği yeni bloğu tayin eden, Hareketlerin akış yönünü değiştiren, Hareketin içindeki verilere dayalı olarak hesaplama yaparak sonuç olarak bulduğu yeni verileri Harekete ilave eden, Hareket içindeki verileri ekrana formatlı olarak yazan blok ise bu sonuçların ekranda görülmesini sağlayan, Sisteme Hareket girişi sağlayan çeşitli tip bloklar vardır. Bu tabi olarak düşünülmelidir. Bu kısıya, Hareketleri birer genli, mesaj iletimine dayanmaktadır. Hareket'ler blokların önünde İGİÇ(İlk Giren İlk Çıkar-FIFO) veri yapılarında kuyruğa sokulurlar.

Her dil bloğu ayrı bir icra birimi olması nedeniyle, sistemde işlemcilere önceden statik olarak atanabilirler. Bu fazda statik yük dengeleme yapmak mümkündür. Her bir nesne bloğu aktif bir süreç tarafından temsil edildiğinden, uygulamaya bağlı olarak, altta yatan işletim sistemi olanaklarından yararlanılarak dinamik yük dengeleme (Dynamic Load Balancing) yapılabilir.

Sistem içinde Hareket eden "Hareketler" kendi statüleri bulunduğundan, Hareketli nesneler olarak kabul edilebilirler. Operasyonları yapan bloklar ise dilin içinde yer alan standart nesnelerdir. Sınıf ve miras mekanizması(Class and Inheritance) bulunmayan bu nesne yapısına, "Nesneye Yönelimli- Object Oriented" yerine "Nesne Temelli-Object Based" yapı adını vermemiz daha uygun olacaktır. Programcı, sistemi, "hareket eden nesneler" ve bu nesneleri alıp, kabul edip yeni hareket eden nesneler üreten "Değiştirici Nesneler" olarak canlandırabilir.

Dil, veri akış programlama dillerinin temel özelliği olan icra sırası bağımsızlığını korumak üzere özel olarak tasarımlanmıştır. Yan etki (Side effect) yaratan, veri tabanına yönelik işlemler, özel olarak belirli bloklar içinde izole edilmiştir. (*) Değişken bazında ise, global değişkenler elimine edilmiştir. Bloklar içindeki değişkenlere diğer bloklar tarafından erişim söz konusu olmadığından ve her blok içindeki değişiklikler, bir Hareketin varması ve üzerinde işlem yapılması sürecine kısıtlı olduğundan, yan etki olasılığı ortadan kaldırılmıştır.

Bu programlama dilindeki diğer önemli gelişme ise, programlama konseptindeki kolaylıktır. Programlamada tecrübeli olamayan bir kişiyi bu programlama dilinde program yapmak üzere eğitime tabi tuttuğumuzu düşünelim. Bu kişiye, Hareketleri birer gemi, nesne bloklarını ise birer liman gibi düşünmesini söyleyebiliriz. Bu

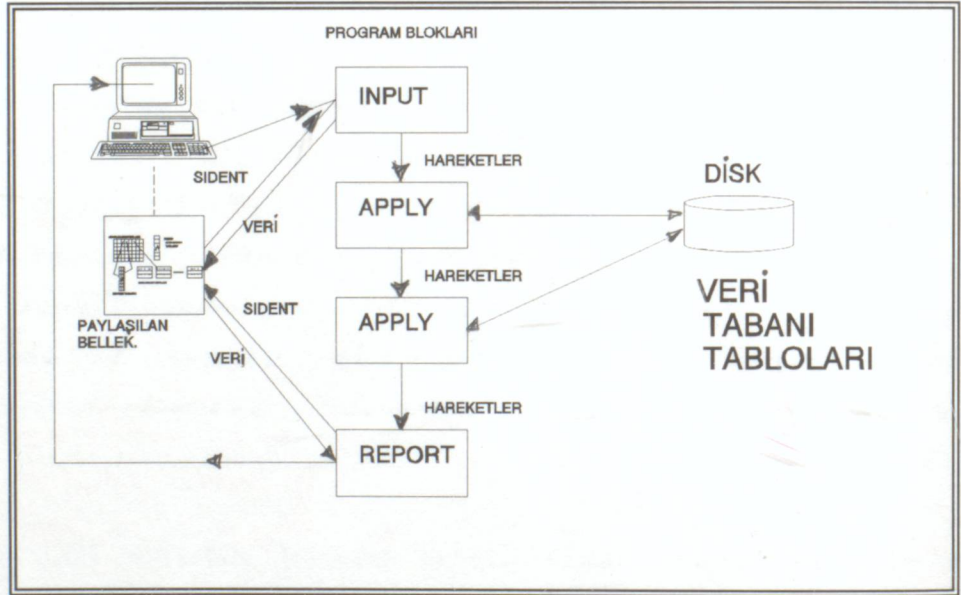
(*)Veri Tabanı Değiştirme, Ekleme ve Silme işlemleri.



gemilerin (Hareket) limanlardan (dil bloklarından) yükleme boşaltma yaptığını ve bu limanlar arasında seyrettiğini düşünebilir. Bu tip bir programlama konseptine uygun olarak tasarımlanacak grafik kullanıcı arayüzleri sayesinde, kullanıcı, grafik ekranda, grafik simgeler olarak görebildiği, blok kütüphanesindeki bloklardan seçim yapıp, bunları ekranda iletişim ilişkisiyle ilişkilendirip program yapabilir.

Programcıların nesne temelli düşünmelerinin en doğal yöntem olduğu bilinmektedir. Bu tip bir dil veri tabanı uygulama programlarının yazılışını kolaylaştırarak son kullanıcı konumunda olan kişilerin genel programlama gayretine daha büyük katkıda bulunmasını sağlayacaktır.

Dilin genel işleyiş yapısı şekil 2.4'de sergilenmiştir.



Şekil 2.4 Dilin Genel İşleyiş yapısı.

"Çok sayıda hareket eden Hareketler ve iletişim maliyeti bu tip bir dili ters yönde etkilermi " sorusuna cevap olarak şunları söyleyebiliriz. Halihazırda 1 Gigabyte/saniye hızında işlemciler arası yol tasarımları hızla sürmektedir. Günümüzde 200 Megabyte/saniye

hızında işlemci kartları arası ve işlemci kartları ile paylaşılan bellek arasında transfer yapan "VME AutoBahn" yol yapısı gerçekleşmiştir. (IEEE Spectrum, 1994) Bu da günümüzde 100 byte'lık bir mesajın 0.5 mikrosaniye mertebesinde transfer edilebildiği, gelecek on yıl içinde ise nanosaniye mertebesinde işlemci kartlar arasında transfer edilebileceğini göstermektedir. Özellikle büyük parçalı paralel çalışmanın söz konusu olduğu uygulamalarda bu sayede büyük kazançlar elde edilebilecektir. Ana bellek ve ikincil bellek kapasitelerindeki büyük artışlar ve ekonomik skaladaki düşüş, işlemci zamanı lehine bu kaynakların daha çok kullanımını sağlayan uygulamaları cesaretlendirmektedir.

4. Başvırmaya uygun bir DİS (donanım) Hareket (Transaction) Akışı
RELAP Dili Paylaşılan Bellek Çok İşlemci MIMD Mimarilerinde ve LAN-WAN gibi dağıtılmış mimarilerde de kullanılabilecek yapıdadır. LAN ve WAN iletişim hızları artmakta ve İstemci-Sunucu veri tabanı mimarileri de Hareket iletişimine dayanmaktadır. Hareket iletişimini farklı ölçekte kullanan böyle bir dilin de büyük kuruluşlar seviyesinde dağıtılmış bilgisayar ağlarında kullanılabileceğini doğal görmemiz gerekir. Bu tip bir ortamda, ağ içindeki, çok işlemcili merkezi bilgisayarlardan daha iyi yararlanılırken, programın belirli blokları, ilişkili oldukları kalıcı verilerin depolandığı bilgisayarda yer alır. Böylece, bir dağıtık veri tabanı şemasında ilişki tablolarının sadece gereken noktalarda depolanması sağlanır ve Hareketler ağ üzerinde bilgisayardan bilgisayara akarak sorgulama sonuçlarının oluşmasını sağlarlar. Bu dil ile dağıtılmış bilgisayar sistemleri üzerinde çalışan uygulamalar çok daha basit bir şekilde yazılarak uygulamaya geçirilebilirler.

5. Dindeki her blok paralel çalışmaya aday bir bileşendir. Programcı
Geleneksel olarak, donanım gelişmeleri, yazılım gelişmelerinden önce gerçekleşmiş ve sonuçta bu kapasiteler yıllar boyunca kullanılmadan kalmıştır. Bu çalışmada, gelecekte oluşacak, donanım gelişmelerinden zamanında yararlanmayı sağlayacak bir dilin tasarımı ve uygulaması hedeflenmektedir. Dilin başlangıçta ticari olarak yaygın kullanımına başlanan çok işlemcili paylaşılan bellek

mimarilerinde kullanılabileceği, daha sonra iletişim hızlarının artmasıyla LAN ve WAN yoluyla gevşek bağlı dağıtık mimarilerde kullanım bulabileceği değerlendirilmektedir.

REL P dilinin genel özellikleri aşağıda özetlenmiştir:

- 1.Kontrol yapısı Veri akışı yapısıdır.
- 2.İlişkisel bir programlama dilidir. Temel Veri yapısı olarak ilişki tabloları kullanılır. Ara sonuç ve asıl sonuç olarak oluşabilecek ilişkiler özel olarak programlanmadıkça kalıcı olarak oluşturulmazlar.
- 3.Nesne Temellidir.
- 4.Hesaplama modeli GPSS benzeri Hareket (Transaction) Akış modelidir. İlişkisel kayıtlar ve elemanlar program içinde belirli noktalarda üretilirler ve sistem içinde hareket ederek bloklara girip çıkarak , bloklar boyunca akarlar, bloklar tarafından başka kayıtlara dönüştürülebilirler ve genel olarak rapor bloğuna vararak, bir raporun gerekli boş kısımlarını veri ile donatarak sistemi terkederler.
- 5.Dışarıdan veri girişi "INPUT obje bloğu" vasıtasıyla yapılır. Kullanıcı tarafından girilen değerlerden oluşan bir Hareket bu blokta yaratılarak bir sonraki bloğa yollanır.
6. "İlişki uygulaması"(*) (APPLY) dilin en önemli operasyonudur. Bir Hareket, "APPLY" bloğuna girdikten sonra kendisine blokta ismi belirtilen ilişki uygulanır sonuçta oluşan yeni Hareket/Hareketler bloğu terkederken yeni bir bloğa giderler.
- 7.Paralel çalışan dil blokları arasındaki iletişim, asenkron mesaj iletimi (Hareketlerin İletimi) yapısındadır.
- 8.Dildeki her blok, paralel çalışmaya aday bir bölümdür. Programcı dildeki blokların mümkün olduğu ölçüde paralel çalışacağını kabul eder, yani paralel işleme dilin yapısı içinde otomatikman yer

(*) Buradaki "İlişki Uygulaması" operasyonu, formal anlamda İlişkiler Teorisindeki, İlişki Uygulama işlemidir. Fonksiyon Uygulaması (Function Application) İlişki Uygulamasının özel halidir, çünkü Fonksiyonlar İlişkilerin alt kümesidir.

almaktadır. Programcının paralel çalışmayı dikte etmek için ayrıca bir belirleme yapmasına gerek ve olanak yoktur.

9.Bloklar, paralel fiziki işlemciler ile dil çalışma zamanı ortamı (Language Run Time Environment) tarafından otomatik olarak ilişkilendirilirler. Bu ortam otomatik yük dengeleme fonksiyonuna sahiptir. Dilde belirli blokları belirli fiziki işlemciler ile ilişkilendirmeye yönelik bir olanak sunulmamaktadır.

10.Dilde iki veri tipine yer verilmiştir.

- a. Sayısal.
- b. Karakter Dizini.

11.Dildeki değişkenler iki gruba ayrılır.

- a.Blok içi değişkenler.
- b.Hareket değişkenleri.

Blok içi değişkenler, bazı blok tiplerinde, kullanıldıkları anda otomatik olarak bildirimi yapılan ve blok içinde kullanılan değişkenlerdir. Bir Hareket, üzerindeki depolama birimlerinde depoladığı verileri sistem içinde bir bloktan diğer bir bloğa taşır. Bu depolama birimleri temel veri soyutlamaları olan "Hareket Değişken İsimleri" (Transaction Variable Names) ile referanslanarak kullanılırlar.

12. Blok içi değişkenlerin geçerli olduğu bölge, (scope) blok ile sınırlıdır. Hareket değişkenlerinin geçerli olduğu bölge ise Hareketin bir bloktan diğer bloğa geçişini sağlayan kuyruk ile sınırlıdır.

2.5 "RELP" Dilinin bloklarının genel tanıtımı

Aşağıda dilin sabit blok tipleri listelenmiş ve kısaca açıklanmıştır.

- 1.GENERATE (GENERATE)
- 2.GENERATEE (GENERATE ELEMENT)
- 3.GENERATET (GENERATE TUPLE)
- 4.TEST (TEST)
- 5.APPLY (APPLY)

- 6.COMPUTE (COMPUTE)
- 7.COMFIX (COMPUTE FIX)
- 8.INPUT (INPUT)
- 9.REPORT (REPORT)
- 10.SORT (SORT)
- 11.COUNTBY (COUNT BY)
- 12.TRANSFER (TRANSFER)
- 13.SINK (SINK)
- 14.FILTER (FILTER)
- 15.CARTESIANP (CARTESIAN PRODUCT)
- 16.APPEND (APPEND A RECORD)
- 17.DELETE (DELETE A RECORD)
- 18.TXCNT (COUNT TRANSACTIONS)
- 19.COND (CONDITION BLOCK)
- 20.NOTMEMBER (NOT MEMBER)
- 21.APPCUM (APPLY CUMULATIVE FUNCTION)
- 22.MERGE (MERGE TRANSACTIONS)

2.5.1 "GENERATE" bloğu genel yapısı

Bu blok argümen olarak verilen ilişkinin, yine argümen olarak verilen tek bir sahasını esas alarak, bir ilişkisel projeksiyon işlemi yapar, bu projeksiyon ile ortaya çıkan kümede yer alan elemanlar birer Hareket halinde bir sonraki bloğa gönderilirler.

2.5.2 "GENERATEE" (Generate Element) bloğu genel yapısı

Bu blok bir veya daha fazla sabit verinin, belirlenen Hareket değişkenlerine atanarak, oluşturulan bir adet Hareket içinde bir sonraki bloğa gönderilmesini sağlar.

2.5.3 "GENERATET" (Generate Tuple) bloğu genel yapısı

Bu blok, parametre olarak verilen bir ilişkinin, yine parametre

olarak belirtilen sahalarını içeren kayıtlarını Hareketler halinde üreterek bir sonraki bloğa gönderir.

2.5.4 "TEST" bloğu genel yapısı

Bu bloğa giren Hareket programcı tarafından belirlenmiş olan mantıksal ifadenin değerlendirilmesi sonucu "True" çıkması halinde belirli bir bloğa "False" çıkması halinde başka bir bloğa gönderilir.

2.5.5 "APPLY" bloğu genel yapısı

Bu blok bir ilişkiyi içine giren Hareketin temsil ettiği, kayda uygular, sonuç olarak oluşturulan ve bir eleman veya kaydı temsil eden Hareket bloktan çıkarak yoluna devam eder. İlişki uygulama operasyonu, bloğa giren Hareketin seçilen sahalarının içeriğinin verilerin, ilişki içinde bu sahalar karşılık gelen sahaların içeriği ile karşılaştırılması, ve eğer tümünde tam bir uyum varsa, bu uyumu sağlayan kayıtların seçilen sahalarını içeren Hareketlerin bloğu terk etmesi işlemidir. Bu blokta giriş yapan her bir Hareket için uygulanan ilişkinin yapısına göre birden fazla Hareket çıkış olarak üretilebilir.

2.5.6 "APPCUM" (Apply Cumulative) bloğu genel yapısı

Bu blok seçilen bir birikimsel fonksiyonu, içinden geçen Hareketlerin belirli bir Hareket değişkeni üzerinde uygular ve gerekirse seçimlik olarak grup bazında tasnif edilmiş veriler bazında birikimsel sonuç elde edebilir.

2.5.7 "COMPUTE" bloğu genel yapısı

Bu bloğa bir Hareket girdiğinde bir veya daha fazla aritmetik ifade, icra edilir ve ara sonuçlar geçici değişkenlere atanabilirler.

Yerel değişkenler, Hareket değişkenleri bu aritmetik ifadelerin işleme giren kısımlarını oluşturabilirler. Sonuçlar çıkış Hareket değişkenlerine atanır. Çıkış Hareketi hesaplanmış sonuçları ve/veya girişte alınıp değiştirilmeden bırakılan değişkenlerin içeriğini taşıyabilir.

2.5.8 "COMFIX" (Compute Fix) bloğu genel yapısı

Compute blok ile aynı özellikleri taşır ancak, kalıcı yapıdaki ilişki tablolarında veri değiştirme işlemleri bu blok ile yapılır.

2.5.9 "INPUT" bloğu genel yapısı

Klavyeden giriş verisi kabul eder, ve bu veri kümesi bir Hareket oluşturacak hale geldiğinde yeni bir Hareket oluşturarak bir sonraki bloğa yollar.

2.5.10 "REPORT" bloğu genel yapısı

Report obje bloğu, bloğa gelen Hareketlerin içeriğini programcı tarafından belirlenen formata göre çıkış aracında gösterir.

2.5.11 "TRANSFER" bloğu genel yapısı

Transfer bloğu, bloğa gelen her Hareketten belirtilen miktarda kopya çoğaltarak belirlenen hedef bloklara birer kopya yollar. Hedef blok adresleri bloklarının etiketleri olarak belirtilir.

2.5.12 "SORT" bloğu genel yapısı

Giren Hareketlerin belirtilen değişkenlerini esas alarak, her Hareketi sıralı bağlı liste veri yapısında doğru sırada depolayarak Hareketleri biriktirir ve son Hareketin alınmasından sonra, tüm biriktirilen Hareketleri yeni sıra ile yayınlar.

2.5.13 "COUNTBY" (Count By) bloğu genel yapısı

Count obje bloğu gelen Hareketleri BY kısmında belirtilen parametreleri (Hareket değişkenleri) esas alarak tasnif ederek sayar, sayım sonuçları giriş Hareketlerine ilave edilerek çıkış Hareketleri üretilir. Her çıkış Hareketi ait olduğu sınıfın sayım sonucunu da taşıyarak bloğu terkeder.

2.5.14 "SINK" bloğu genel yapısı

Bu blok girişten aldığı her Hareket için parametre olarak verilen

2.5.15 "FILTER" bloğu genel yapısı

Filter bloğu, parametre olarak belirtilen Hareket değişkenlerini esas alarak, içinden geçen Hareketleri tekrarları ortadan kaldıracak şekilde süzer. Belirtilen Hareket değişkenlerinde aynı değerleri taşıyan Hareketlerden sadece ilki bu bloktan geçebilir, diğerleri sistemden çıkartılırlar.

2.5.16 "CARTESIANP" (Cartesian Product) bloğu genel yapısı

Girişten gelen Hareketlerin temsil ettiği her kaydı, blok parametresi olarak verilen ilişkinin belirtilen sahalarını içeren her kaydı ile eşleştirip birleştirerek, oluşturduğu kayıtları çıkış kuyruğuna gönderir.

Tüm giriş kayıtlarının bir ilişki oluşturduğu düşünülürse (İlişki A), parametre olarak verilen ilişkinin, yine parametre olarak verilen sahaları bazında bir projeksiyon işlemi yapıлып yeni bir ilişki bulunursa (İlişki B), sonuçta bu bloğun ürettiği tuple'lar, ilk ilişki ile ikinci ilişkinin kartezyen çarpımından elde edilecek ilişkinin kayıtlarıdır (İlişki $A \times B$).

2.5.17 "APPEND" (Append a Record) bloğu genel yapısı

Bu blok girişten aldığı her Hareket için, parametre olarak verilen ilişkiye bir kayıt ekler, bu kaydın belirtilen sahaları, karşılık gelen Hareket değişkenlerinin içerdiği değerleri alırlar. Geri kalan sahalar ise boş bırakılır.

2.5.18 "DELETE" (Delete a Record) bloğu genel yapısı

Bu blok girişten aldığı her Hareket için parametre olarak verilen ilişkiden bir kayıt siler. Bu kaydın seçimi, belirli Hareket değişkenleri ile ilişkinin belirli sahaları arasında oluşturulan eşitlik şartına göre yapılır. Bir veya birden fazla eşitlik şartının birlikte önerilmesi ile oluşturulan eşitlik şartı, ilişki içinde bu şarta uyan kaydın araştırılıp bulunmasına yöneliktir. Bloğa her Hareket girişinde bu şarta uygun ilk ilişki kaydı bulunup silinir.

2.5.19 TXCOUNT (Count transactions) bloğu genel yapısı

Bu blok çok basit bir blok olup, içinden geçen Hareketleri hiç bir kriter gözetmeksizin sayar ve sonucu tek bir Hareket halinde bir sonraki bloğa gönderir. Bu Hareketde yer alan sonuç _Cnt sistem değişkeni altında depolanmıştır.

2.5.20 COND (Condition) bloğu genel yapısı

Bu blok bir referans kuyruğa (REF) bir de standart giriş kuyruğuna (INP) sahiptir. Referans kuyruktan sadece bir Hareket okunur ve bu Hareket referans Hareket olarak kabul edilir. Standart giriş kuyruğundan ise arka arkaya gelen Hareketler okunur.

Standart kuyruktan okunan her Hareketin belirtilen değişkenin içerdiği değer, referans kuyruktaki Hareketin belirtilen değişkeninin içerdiği değer ile bir şart ifadesi içinde karşılaştırılır. Sonuç

'Yanlış' (FALSE) ise standart girişten gelen Hareket durdurulur ve çıkış kuyruğuna verilmez. Sonuç 'Doğru' (TRUE) ise standart girişten gelen Hareket çıkış kuyruğuna gönderilir.

2.5.21 "NOTMEMBER" (Not Member test) bloğu genel yapısı

Bu blok giren Hareketin belirli değişkenlerini parametre olarak verilen ilişkinin sahaları ile karşılıklı ilişkilendirir. Eğer bloğa giren herhangi bir Hareketin bu belirli değişkenlerinin içerikleri, ilişki içinde karşılık gelen sahalar altında herhangi bir kayıt içinde birlikte yer almıyorsa, Hareket çıkış kuyruğuna gönderilir, aksi takdirde Hareket bloke edilerek sistemden çıkartılır.

2.5.22 "MERGE" (Merge Transactions) bloğu genel yapısı

Bu blok birden fazla giriş kuyruğundan içine giren Hareketleri tek bir çıkış kuyruğuna gönderir. Efektif olarak birden fazla Hareket akış yolunu tek bir Hareket akış yoluna birleştirir.

3. RELP DİLİ GENEL SÖZ DİZİMİ YAPISI

RELP dili söz dizimi yapısının yapı taşlarını bloklar oluşturur. Her blok kendi içinde özel bir söz dizimi yapısındadır. Kuyruklar ve kuyruklara paralel olarak oluşturulan 'isim tanımlama ortamları' (context) dilin anlamsal (semantic) yapısı içinde yer alırlar.

Bu bölümde sadece söz dizimine ait yapılardan bahsedilecek, dilin anlamsal yapısına çok gerekmedikçe değinilmeyecek, anlamsal yapı diğer bir bölümde açıklanacaktır.

Dilin "Backus Naur Form" ve "Extended Backus Naur Formda" hazırlanmış söz dizimi formal tanımı ile söz dizimi kurallarının diagramları Ek-1 ' dedir.

3.1 Genel

3.1.1 Programın ana yapısı

Program ard arda dizilmiş bloklardan oluşmuştur. Bloklar biribirinden noktalı virgül karakteri ile ayrılırlar. Program '#' karakteri ile sonlandırılır. Eğer herhangi bir blok diğer bloklar tarafından özellikle adresleniyor ve bu yolla Hareket kabul ediyorsa blok anahtar kelimesinden önce etiket (Label) gelebilir.

3.1.2 İsimlendirme kuralları

1. Tüm İlişki İsimleri (Relation) 'R_' ön eki ile başlar ve 18 karaktere kadar büyük harflerden oluşur. Bu dizinin ilk karakteri hariç diğer karakterleri 0 ile 9 arasında rakkamlar olabilir.

Bu kuralın Regüler İfade (Regular Expression) kullanılarak tarifi aşağıdadır:

$R_ (A-Z)(A-Z0-9)^*$ (3.1.2.1)

Örnekler :

R_BABA

R_ANASTOK1

2. Değişken isimleri (Variable Names) '_' ön eki ile başlar. Bu ön eki takibeden ilk karakter bir büyük harf olmalıdır. İlk karakteri takibeden dizin, 17 karaktere kadar küçük harfler veya 0 ile 9 arasında rakkamlardan oluşur.

Bu kuralın Regüler İfade (Regular Expression) kullanılarak tarifi aşağıdadır:

$_ (A-Z)(a-z0-9)^*$ (3.1.2.2)

Örnekler : $)*((')(9-9)+$

_Ders1

_Parcano

3. Tüm attribute isimleri '_' ön eki ile başlar ve 18 karaktere kadar küçük harflerden oluşur. Bu dizinin ilk karakteri hariç diğer karakterleri 0 ile 9 arasında rakkamlar da olabilir.

Bu kuralın Regüler İfade (Regular Expression) kullanılarak tarifi aşağıdadır:

$_(a-z)(a-z0-9)^*$ (3.1.2.3)

Örnekler : $2)(A-Z)^*$

_kod2

_ogul

4. Karakter dizini sabitleri, ' " ' ön eki ile başlar ve a ile z arasında büyük küçük harfler ile 0 ile 9 arasında rakkamların rastgele sırayla dizilmesinden oluşur.

Bu kuralın Regüler İfade (Regular Expression) kullanılarak tarifi aşağıdadır:

$"(A-Za-z0-9)(A-Za-z0-9)^*$ (3.1.2.4)

Örnekler : ya açıklanarak ve örnekler verilecektir.

"F16

"Ahmet

5. Sayısal sabitler , sadece reel sayısal sabitler kullanılmakta olup, pozitif veya negatif tamsayıyı takibeden nokta karakteri ve bunu takiben pozitif tamsayı veya 0'dan oluşur.

(+-' ')(0-9)*('.'')(0-9)+

(3.1.2.5)

Örnekler : T anahtar kelimesi ile başlar, bunu takiben % karakteri bu etiket alan parantez içinde ilk verinin ekranı okunacağı 123.345 rakamlar sırasıyla satır ve sütun numarası olarak yer alırlar. Bu -2.0 GET anahtar kelimesi izler. GET anahtar kelimesinden sonra +456.01 belirtilen koordinatlarda başlık olarak gösterilecek karakter dizisi yer alır. Bu karakter dizisini takiben okunacak verinin tipi

6. Etiket (Label) : Program içinde belli blokların isimlendirilmesinde kullanılan ve bu blokların başında yer alan etiketler (Label) sadece büyük harflerden oluşan 20 karaktere kadar karakter dizinleri şeklinde isimlendirilirler. anahtar kelimesi harf yukarıdaki yapıda blok içinde yer alırlar. Blok sonu noktalı virgül karakteri ile belli (A-Z)(A-Z)*

Örnekler :

SONBLOK (10,15) GET 'Adı' CHAR 'Adı'

TOPLAMANOKTASI (10,15) GET 'Soyadı' CHAR 'Soyadı'

(10,15) GET 'Yaşı' NUM 'Yaşı'

Blok isimleri de etiketler gibi büyük karakterlerden oluşan dizinler şeklinde olduğundan bu anahtar isimlerin blok etiketi (label) olarak kullanılması yasaktır.

Açıklama :

3.2 Blokların söz dizimi yapıları

Bu bölümde blokların söz dizimi yapıları, her bir blok için ayrı ayrı sırasıyla açıklanacak ve örnekler verilecektir.

3.2.1 "INPUT" bloğu söz dizimi yapısı

Açıklama :

Input blok standart girişten veri okuyan ve bu verileri Hareket olarak sisteme katan bir bloktur.

Örnek :

GENERATEL 'Adı' 'Adı', 'Soyadı' 'ACAR', 'Yaşı' '25'



Söz Dizimi :

GENERATE anahtar kelimesi ile başlar, bunu takiben % karakteri ön ekini alan parantez içinde ilk verinin ekrandan okunacağı koordinatlar sırasıyla satır ve sütun numarası olarak yer alırlar. Bu diziyi GET anahtar kelimesi izler. GET anahtar kelimesinden sonra ekranda belirtilen koordinatlarda başlık olarak gösterilecek karakter dizisi yer alır. Bu karakter dizisini takiben okunacak verinin tipi karakter dizisi ise CHAR, sayısal veri ise NUM olarak belirtilir. En son girdi olarak da girilen verinin depolanacağı değişken ismi yer alır. İlk veri okumayı takibeden veri okumaları virgül ile birbirinden ayrılmış olarak INPUT anahtar kelimesi hariç yukarıdaki yapıda blok içinde yer alırlar. Blok sonu noktalı virgül karakteri ile belirlenir.

Örnek :

```
INPUT %(10,15) GET "Adi CHAR _Adi,
          %(11,15) GET "Soyadi CHAR _Soyadi,
          %(12,15) GET "Yasi NUM _Yasi;
```

3.2.2 "GENERATEE" (Generate element) bloğu söz dizimi yapısı**Açıklama :**

Generate Element Blok, program içinde programcı olarak sabit olarak belirtilmiş bir veya birden fazla data elemanını birer değişken ile özdeşleştirerek tek bir Hareket olarak sisteme katar.

Söz Dizimi :

Blok GENERATEE anahtar kelimesi ile başlar. Bunu takiben birbirinden virgül ile ayrılmış 'Değişken ismi, eşittir karakteri ve sabit veri' kombinasyonları yer alır.

Örnek :

```
GENERATEE _Adi="Ali, _Soyadi="ACAR, _Yasi=22.0 ;
```


3.2.3 "GENERATET" (Generate Tupple) bloğu söz dizimi yapısı

Açıklama :

Generate tupple bloğu bir ilişkinin tüm kayıtlarını sırasıyla ele alır ve her kaydın belirtilen sahalarını bir Hareket içinde sisteme katar. "APPLY" bloğu söz dizimi yapısı

Söz dizimi :

Blok GENERATET anahtar kelimesi ile başlar, bunu takiben ilişki ismi ve biribirinden virgül ile ayrılmış saha listesi yer alır. Blok noktalı virgül karakteri ile sonlandırılır.

Örnek :

GENERATET R_BABA _baba, _ogul ; Bu örnekte R_BABA ilişkisinin tüm kayıtlarında _baba ve _ogul sahaları altında depolanmış veriler, çift çift her kayıt için bir Hareket olmak üzere sisteme katılacaktır.

3.2.4 "GENERATE" (Generate Attribute) bloğu söz dizimi yapısı

Açıklama :

Bu blok bir ilişkinin tek bir sahası (attribute) içinde yer alan verileri bir projeksiyon işlemi gibi ayırıp, her bir veri bir Hareket oluşturacak şekilde sisteme katar. Bu bloğun GENERATE TUPLE bloğundan en önemli farkı tek bir attribute üretmesi yanında, üretilen bir verinin birden fazla tekrar etmemesi ve ilişki tablosundan başka bir veri yapısına dayanılarak daha hızlı bir mekanizma ile üretilmesidir.

Söz dizimi :

Blok GENERATE anahtar kelimesi ile başlar. Bunu takiben ilişki ismi (Relation Tablosu ismi) ve saha ismi yer alır. Blok noktalı virgül karakteri ile sonlandırılır.

Örnek :

```
GENERATE R_BABA _baba ;
GENERATE R_PARCA _parcano ;
GENERATE R_KOD _kod1;
```

3.2.5 "APPLY" bloğu söz dizimi yapısı

Açıklama :

Bu blok giriş kuyruğundan aldığı Hareketin belirtilen değişkenlerini uygulama operasyonunun yapılacağı ilişkinin (Relation) giriş saha listesinde belirtilen sahaları ile ilişkilendirerek, bu değişkenlerin içerdiği veriler ile karşılık gelen sahalar altında ilişki içinde yer alan veriler arasında tam uyum bulunan kayıtları seçer. Seçilen bu kayıtların, çıkış saha listesinde belirtilen saha içeriklerini Hareket olarak sisteme katar. Apply bloğuna giren Hareket içinde bulunup, herhangi bir saha ile eşleştirilmeyen ve çıkış saha listesinin sonunda ':' karakteri ile ayrılan bölümde yer alan değişkenlerin içeriği üretilen Hareketlerin sonuna eklenerek bir sonraki bloğa nakledilirler. Böylece apply bloğuna giren Hareket içinde yer alan ancak eşleştirmede kullanılmayan faydalı veriler bir çeşit "bypass" operasyonu ile çıkış Hareketi içinde de yer alırlar.

Söz dizimi :

Blok APPLY anahtar kelimesi ile başlar. Bunu takiben giriş değişkenleri ile söz konusu ilişkinin sahaları arasındaki ilişkilendirmeler, birbirinden virgül ile ayrılmış 'değişken ismi , "|" karakteri, saha ismi' kombinasyonlar halinde yer alır.

Bu listeden sonra "/" karakteri ve söz konusu ilişkinin ismi yer alır. İlişki isminden sonra "\" karakteri ayırma için kullanılır. Birbirinden virgül ile ayrılmış çıkış saha isimleri bu karakteri takip eder. Eğer giriş Hareketinde yer alan değişken veya saha isimleri altında depolanmış veriler aynen çıkışa yansıtılacaksa, bu değişken veya saha isimleri iki nokta karakterini takiben birbirinden virgül ile ayrılmış şekilde bir liste halinde sonda yer alırlar.

Bu listedeki saha isimleri ile çıkış saha listesinde yer alan saha isimlerinin çakışmaması programcının sorumluluğundadır. (Bu nevi bir çakışma aynı zamanda anlamsal bir hatanın da göstergesidir.)

Örnek :

APPLY _Numarasi|_no,_Kismi|_kisim /R_SINIF\ _adi,_soyadi:_Boy

Bu örnekte giren Hareketde _Numarasi, _Kismi ve _Boy değişkenlerinin bulunduğu bunlardan sadece _Numarasi ve _Kismi değişkenlerinin uygulama operasyonunda kullanıldığı kabullenilmiştir. Uygulama sonunda uyum sağlayan kayıtların _adi ve _soyadi sahalarının içerikleri ve giriş Hareketindeki faydalı bilgi olarak çıkışa yansıtılmak istenen _Boy değişkeni içeriği çıkış Hareketinde yer alacaktır.

3.2.6 "REPORT" bloğu söz dizimi yapısı

Açıklama :

Report blok giriş Hareketi içinde yer alan verileri belirtilen formatta ve koordinatta ekranda gösterir.

Söz dizimi :

Blok REPORT anahtar kelimesi ile başlar. Bunu takiben '' karakteri ve köşeli parantez içinde birbirinden virgül ile ayrılmış satır ve sütun olarak ekran koordinatları yer alır. SAY anahtar kelimesi , başlık için kullanılacak karakter dizisi, NUM veya CHAR olmak üzere tip göstergesi ve değişken/ saha ismi bunu izler. Takibeden veriler için REPORT anahtar kelimesi hariç bu yapı virgül ile bir önceki yapıdan ayrılmış olarak kullanılır.

Örnek :

REPORT ?(10,15) SAY "ADI CHAR _adi,
?(11,15) SAY "NO NUM _Numarasi;

3.2.7 "COMPUTE" bloğu söz dizimi yapısı

Açıklama :

Compute blok giriş Hareketindeki sayısal değişkenlerin değerlerini esas alarak, bir veya birden fazla aritmetik atama deyimiyle hesaplamalar yapılmasını sağlayan ve bulunan sonuç değerler ile istendiği takdirde giriş Hareketinde yer alan diğer değerleri çıkış Hareketinde birleştirerek bir sonraki bloğa yollayan bir bloktur.

Söz dizimi :

Blok COMPUTE anahtar kelimesi ile başlar. İlişkilendirme, arama, atama ve çıkış grupları bu anahtar kelimeyi takip eder. Arama grubu seçimlik bir grup olup, ihtiyaca göre kullanılır.

İlişkilendirme grubunda giriş Hareketinde yer alan değişken ve saha isimleri , compute blok içinde karşılık gelecek değişken isimleri ile eşleştirilirler. Bu yeni değişkenler sadece compute blok içinde geçerli değişkenler olup, bu eşleştirme 'Değer Geçirme (Call by Value)' yöntemindeki formal parametre gerçek parametre ilişkilendirilmesi gibidir. Söz dizimi yapısı 'Değişken/Saha ismi,|" karakteri, Lokal değişken ismi' şeklinde olup bu kombinasyonlar biribirinden virgül karakteri ile ayrılırlar.

Arama grubu, Compute Blok içinde bir ilişkinin, belirli bir eşlik kriterine uyum sağlayan kaydının belirli bir sahasına referans yapılacak ise kullanılır.

Eşlik kriteri giriş Hareketindeki bir veya birden fazla değer, söz konusu ilişkinin bir veya birden fazla anahtar saha içeriğinin karşılıklı eşliğini şart koşar. Her Hareketin girişinde bu Hareketde yer alan değerler esas alınarak söz konusu ilişkideki kayıt seçilir ve arama grubunda eşitlik kriterinde yer alan veya sadece ismi belirtilen sahalara "compute" blok içindeki değişken isimlerine dahil olarak hesaplamada kullanılabilir duruma girerler. Ancak bu

değişkenler (saha isimleri) aritmetik atama deyiminin sol tarafında bulunamazlar.

Bu grup LOCATE anahtar kelimesi ile başlar ve bunu ilişki ismi takip eder. İlişki isminden sonra birbirinden virgül ile ayrılmış 'saha ismi, "=" karakteri, Compute Blok Değişkeni veya gerçel sayı' kombinasyonları yer alır. Eğer belirtilen ilişki sahası eşlik şartı içinde yer almıyorsa bu kombinasyonda sadece saha ismi yer alır. Eşlik kriteri içinde yer alsın almasın bu kombinasyonlarda yer alan ilişki sahaları "Compute" Blok içinde aritmetik ifadelerin sağ tarafında hesaplama içinde kullanılabilirler.

Atama grubu :

Bu grupta aritmetik atama deyimleri birbirinden virgül karakteri ile ayrılmış olarak yer alırlar. Aritmetik atama deyimi sol tarafında Compute Blok ilişkilendirme grubunda otomatik bildirimi yapılan değişkenlerden biri bulunabilir. Sağ tarafta ise -, +, *, / gibi aritmetik operatörler operatör, Compute Blok değişkenleri, arama bloğunda deklare edilmiş ilişki saha isimleri ve sayısal sabitler ise operand olarak yer alır.

* ve / operasyonları + ve - operasyonlarına göre önceliklidir.

Çıkış grubu :

Çıkış grubunda, compute blok içinde tanımlı değişkenler ve saha isimleri birbirinden virgül ile ayrılmış olarak bulunurlar. Bu değişkenler ve/veya sahalar çıkış Hareket içeriğini oluştururlar.

Örnek :

```
COMPUTE _En|_Eni ,_boy|_Boyu, _Parcano|_Parcano
LOCATE R_PARCA _parno=_Parcano, _yukseklık
ASSIGN _Hacim=_Eni*_Boyu*_yukseklık
OUTPUT _Hacim,_Parcano;
```


3.2.8 "COMFIX" (Compute Fix) bloğu söz dizimi yapısı

Açıklama :

Bu blok Compute Blok ile aynı işlevi görür ve aynı söz dizimi yapısındadır. Farklı olarak arama grubunda yer alan ilişki sahaları aritmetik ifadelerde sol tarafta yer alabilirler.

Buna ilaveten, farklılık olarak, arama grubunda ismine yer verilen sahalar karakter tipinde değişkenlerin değerlerinin atanmasını sağlayan 'attribute' ismi, eşittir karakteri, karakter tipindeki değişken' yapısında atama deyimleri yer alabilir. Bu sayede giriş Hareketinde yer alan karakter dizisi tipindeki verilerin, arama grubunda yer alan ilişkinin, eşlik kriterleri altında seçilen kayıtlarının yine arama grubu içinde ismine yer verilen sahalarına işlenebilmesi sağlanmıştır. Bu blok içinde hesaplamalar yapıldığı gibi tek saha günleme işlemleri de böylece yapılabilir. 

3.2.9 "TEST" bloğu söz dizimi yapısı

Açıklama :

Test blok compute bloğa çok benzeyen bir bloktur. Test blok compute bloktan farklı olarak tek bir mantıksal ifade ile mantıksal TRUE veya FALSE sonucunun elde edilmesini ve bu sonuca göre giriş yapan Hareketin TRUE halinde belli bir bloğa, false halinde diğer bir bloğa gönderilmesini sağlar. Giriş Hareket içerikleri çıkışta aynen korunur.

Söz dizimi :

Blok TEST anahtar kelimesi ile başlar. Bunu Compute Bloktaki aynı yapıya sahip olan ilişkilendirme ve arama grubu takip eder. Arama grubu ihtiyaca göre seçimlidir, hiç kullanılmayabilir.

Atama grubunda tek bir mantıksal atama deyimini yer alabilir. Bu atama deyiminin genel yapısı aşağıda özetlenmiştir. Mantıksal

değişken ismi rastgele seçilecek bir değişken ismi olup otomatik bildirimi yapılmış olur.

sayısal değişken = mantıksal ifade

mantıksal ifade ::= aritmetik ifade

mantıksal ifade ::= karakter dizin sabiti = karakter tipinde

mantıksal ifade ::= karakter dizin sabiti != karakter tipinde

mantıksal ifade ::= mantıksal ifade mantıksal operatör

mantıksal ifade ::= mantıksal hayır operatörü mantıksal ifade

Karşılaştırma operatörleri aşağıda listelenmiştir.

Büyüktür >

Küçüktür <

Büyük veya eşit >=

Küçük veya eşit <=

Eşit değil !=

Eşittir ==

Mantıksal operatörler aşağıda listelenmiştir.

Mantıksal VE (AND) &&

Mantıksal VEYA (OR) ||

Mantıksal HAYIR (NOT) operatörü !

Karşılaştırma operatörleri hepsi aynı öncelik derecesinde ve soldan sağa öncelik kazanacak şekilde tüm aritmetik operatörlerden daha düşük önceliklidir. Mantıksal operatörler karşılaştırma operatörlerinden daha düşük öncelik derecesine sahiptir.

Mantıksal hayır operatörü mantıksal operatörlerden daha düşük öncelik derecesine sahiptir. Önceliklerin aynı olduğu durumda soldaki operatör sağdakinden önceliklidir. Test bloğunda Compüte Bloktan farklı olarak, çıkış grubu yer almamaktadır. Çıkış grubu

yerine birbirinden virgöl ile ayrılmış iki etiket hemen atama grubunu takiben yer alır. Birinci etiket mantıksal atama sonucunda mantıksal YANLIŞ (FALSE) elde edilmesidurumunda çıkış Hareketinin gönderileceği bloğun markalandığı etikettir. İkinci etiket ise mantıksal atama sonunda mantıksal DOĞRU (TRUE) elde edilmesi durumunda çıkış Hareketinin gönderileceği bloğun etikettir.

Örnek :

```
TEST _Yas|_Yas, _Boy|_Boyu
ASSIGN _Sonuc = _Boyu > 1.75 && _Yas < 25.0
UYGUNDEGIL,UYGUN
```

3.2.10 "CARTESIANP" bloğu söz dizimi yapısı

Örnek :

Açıklama :

Bu blok her giriş Hareketi için, Hareketin temsil ettiği kayıt ile, blok parametresi olarak verilen ilişkinin içindeki tüm kayıtların ilişkinin belirtilen sahalarını içeren hallerini, ilişkisel kartezyen çarpım mantığı içinde eşleştirerek sonuçta oluşan kayıtları çıkış kuyruğuna gönderir.

Söz dizimi :

Blok CARTESIANP anahtar kelimesi ile başlar. Bu anahtar kelimeyi takiben, kartezyen çarpıma girecek ilişki ismi ve bunu da takiben bir ilişkilendirme listesi yer alır. İlişkilendirme listesinde çıkışta ilişkinin hangi sahalarının yer alacağı ve bu sahaların değişken isimleri ile eşleştirilmesi yer alır. İlişkilendirme listesi birbirinden virgöl ile ayrılmış, ilişkilendirme girdilerinden oluşur. Her bir ilişkilendirme girdisinde ilk olarak, ilişkinin çıkışta yer alacak saha ismi yer alır. Bunu takiben "|" karakteri ve bunu da takiben bu sahanın eşleştirildiği çıkış Hareket (transaction) değişkeni ismi yer alır. Bu tür ilişkilendirmenin sebebi giriş Hareket değişkeninde yer alan Hareket değişken isimleri ile çıkış Hareket değişkeni içinde yer alacak ilişki saha isimlerinde bir çakışmayı önlemek üzere programcıya olanak vermektir. Zira giriş Hareket değişkeninde

yer alan değişkenler çıkışta aynen yer alırlar. Örneğin giriş Hareket değişkeni olarak "_adi" mevcutsa, ve çarpım yapılan ilişkinin içindeki "_adi" sahası çıkışa verilecek çarpım sonucu oluşturulan kayıtda yer alacaksa, bir çıkış Hareketi içinde iki adet "_adi" değişkeni olacak ve bu hataya sebep olacaktır. Böyle bir durumda programcının ilişkilendirme girdisi olarak:

_adi | _Adi

belirlmesi gerekecektir.

Blok ";" karakteri ile sonlandırılır.

Örnek :

CARTESIANP R_OGRENCI _adi | _Adi, _soyadi | _Soyad ;

3.2.11 "SORT" bloğu söz dizimi yapısı

Açıklama :

Bu blok giren Hareketleri belirtilen değişkenlerini esas alarak sıralı bir liste haline getirir ve Hareket dizi sonu işaretini (-1) aldığı anda bu sırayla Hareketleri çıkış kuyruğuna gönderir.

Söz dizimi :

Blok SORT anahtar kelimesi ile başlar. Bunu takiben sıralamada esas alınacak olan ve Hareket değişken isimleri soldan sağa azalan önem sırasında birbirinden virgül ile ayrılmış olarak yer alır.

Örnek :

SORT _Adi, _Soyadi, _Yasi ;

3.2.12 "TRANSFER" bloğu söz dizimi yapısı

Açıklama :

Bu blok giriş Hareketini blok tanımında belirtilen etiket ile

markalanmış bloğa yönlendirerek bu bloğa giriş yapmasını sağlar. Eğer birden fazla etiket tanımlanmışsa Hareketin kopyaları bu etiketler ile markalanmış blokların tümüne gönderilir.

Söz dizimi :

Blok TRANSFER anahtar kelimesi ile başlar. Bunu takiben etiket veya etiketler (biribirinden virgül ile ayrılmış olarak) yer alır.

Örnek :

TRANSFER BLOK1 ;

(Bu örnekte giriş Hareketi bir sonraki blok yerine BLOK1 ile markalanmış bloğa gönderilecektir.)

TRANSFER BLOK1, BLOK2, BLOK3 ;

(Bu örnekte giriş Hareketinden üç kopya yaratılacak ve sırasıyla, BLOK1, BLOK2 ve BLOK3 ile markalanmış bloklara birer kopya gönderilecektir.)

3.2.13 "COUNTBY" bloğu söz dizimi yapısı

Açıklama :

Count blok giriş Hareketlerini belirtilen değişkenlerini esas alarak sayar. Bu Hareket değişkenleri içinde yer alan veriler esas alınarak geçen Hareketler tasnif edilir ve her bir cinsten kaç adet geçtiği hesaplanır. Daha sonra bu Hareketler çıkış kuyruğundan sırayla bir sonraki bloğa gönderilirken, her birinin sonuna tasnifte dahil olduğu sınıfın bulunan sayısı eklenir.

Söz dizimi :

Blok COUNTBY anahtar kelimesi ile başlar ve bunu takiben biribirinden virgül ile ayrılmış, Hareket içindeki verileri temsil eden değişken/saha isimleri yer alır. Bu şekilde bir veya daha fazla değişken yada saha ismi COUNT anahtar kelimesinden sonra yer alabilir.

Örnekler :

COUNTBY _Kilosu, _Yasi;

COUNTBY _Parcano;

3.2.14 "FILTER" bloğu söz dizimi yapısı**Açıklama :**

Bu blok belirtilen Hareket değişkenlerini esas alarak geçen Hareketleri bu değişkenlerinde aynı değerleri taşıyan Hareketlerden birden fazlası çıkışa geçmeyecek şekilde filitreler. Belirtilen değişkenlerinde aynı verileri içeren Hareketlerden ilki bu bloktan serbestçe geçerken takip eden Hareketler bloke edilip geçirilmez.

Söz dizimi :

Blok FILTER anahtar kelimesi ile başlar. Bu anahtar kelimeyi biribirinden virgül ile ayrılmış Hareket değişkenleri/sahaları takip eder. Bu liste tek bir değişken/sahadan ibaret de olabilir.

Örnek :

FILTER _Parcano;

FILTER _Numarasi, _Kismi;

3.2.15 "SINK" bloğu söz dizimi yapısı**Açıklama :**

Bu blok dildeki en basit blok olup görevi , giriş kuyruğundan okuduğu Hareketleri sistem dışına çıkartmak yani sonlandırmaktır.

Söz dizimi :

Blok sadece SINK anahtar kelimesinden ibarettir.

3.2.16 "MERGE" bloğu söz dizimi yapısı**Açıklama :**

Bu blok birden fazla kaynaktan kendisine adreslenen Hareketleri

geldikleri giriş kuyruklarından okuyup, hepsini tek bir çıkış kuyruğunda bir sonraki bloğa gönderir. Blok birden fazla etiket ile isimlendirilebilir. Bu etiketler aynı zamanda TRANSFER ve TEST bloklarının, MERGE bloğuna gönderi yapmak üzere kullandığı etiketlerdir. Bir TRANSFER/TEST bloğunun çıkışında bulunan etiket eğer MERGE bloğunda etiket listesinde de yer alıyorsa bu iki blok arasında bir kuyruk oluşmuş demektir.

Söz dizimi :

Blok MERGE anahtar kelimesi ile başlar. Bunu biribirinden virgül ile ayrılmış etiketler takip eder.

Örnekler :

MERGE ORIJIN1, ORIJIN2, ORIJIN3;

3.2.17 "TXCNT" bloğu söz dizimi yapısı

Açıklama :

Bu blok diğer bir basit blok olup, içinden geçen Hareketleri hiç bir kriter gözetmeksizin sayar ve sonucu tek bir Hareket halinde bir sonraki bloğa gönderir. Bu Hareketde yer alan sonuç _Cnt sistem değişkeni altında depolanmıştır.

Söz dizimi :

Blok sadece TXCNT anahtar kelimesinden ibarettir.

3.2.18 "COND" bloğu söz dizimi yapısı

Açıklama :

Bu blok bir referans girişe (REF) bir de standart girişe (INP) sahiptir. Bu girişler soldan sağa sırayla iki adet etiket ile temsil edilirler ve diğer bloklar tarafından referanslanırlar. Bu blok içinde ayrıca sayısal veya karakter dizisi tiplerinin karşılaştırma operatörlerini kullanan ve referans girişteki Hareketinin herhangi bir

değişkeni ile standart girişten giren Hareketin herhangi bir değişkeninin karşılaştırılması ile oluşturulan basit şart ifadesi yer almaktadır. Referans girişten sadece bir Hareket okunur ve bu Hareket referans Hareket olarak alınır. Standart girişten ise arka arkaya giriş kuyruğundan giren Hareketler okunur. Standart girişten okunan her Hareketin belirtilen değişkenin içerdiği değer, referans girişteki Hareketin belirtilen değişkeninin içerdiği değer ile yine belirtilen karşılaştırma operatörü ile karşılaştırılır. Sonuç 'Yanlış' (FALSE) ise standart girişten gelen Hareket bloke edilir ve çıkış kuyruğuna verilmez. Sonuç 'Doğru' (TRUE) ise standart girişten gelen Hareket çıkış kuyruğuna gönderilir.

Söz dizimi :

Blok COND anahtar kelimesi ile başlar. Bunu takiben referans girişini temsil eden etiket yer alır. Daha sonra standart girişi temsil eden etiket ve bunu takiben de şart ifadesi yer alır. Şart ifadesi yapısında ise önce REF anahtar kelimesi ve bunu takiben referans girişteki Hareketin karşılaştırmada kullanılacak Hareket değişkeni/sahası ve onu da takiben karşılaştırma operatörü yer alır. Karşılaştırma operatörleri aşağıda listelenmiştir.

Büyüktür >

Küçüktür <

Eşit değil !=

Büyük veya eşit >=

Küçük veya eşit <=

Eşittir ==

Karşılaştırma operatöründen sonra INP anahtar kelimesi ve bunu takiben standart girişten gelen Hareketin karşılaştırmada kullanılacak Hareket değişkeni/sahası yer alır.

Örnek :

```
COND BAZ , STNDGIRIS REF _Yasi == INP _Yas;
```


Bu örnekte BAZ referans girişi temsil eden programcı tarafından seçilmiş bir etiket, benzeri şekilde STNDGIRIS de standart girişi temsil eden bir etikettir. _Yasi değişkeni referans port'daki Hareketin karşılaştırmada kullanılacak değişkeni. _Yas ise standart girişteki Hareketin karşılaştırmada kullanılacak değişkenidir. REF ve INP ise bu konuda okunabilirliği arttıran Söz Dizimi katkılarıdır.

3.2.19 "NOTMEMBER" bloğu söz dizimi yapısı

Açıklama :

Bu blok giren Hareketin belirli değişken/sahalarını belirli bir ilişkinin sahaları ile karşılıklı ilişkilendirir. Eğer bloğa giren herhangi bir Hareketin bu belirli değişken/sahalarının içerikleri ilişki içinde karşılık gelen sahalar altında herhangi bir kayıt içinde birlikte yer almıyorsa, Hareket çıkış kuyruğuna gönderilir, aksi takdirde Hareket bloke edilerek sistemden çıkartılır. Eğer ilişki belirtilen ilişki sahaları esas alınarak ilişkisel projeksiyon operasyonu ile elde edilmiş bir yeni ilişki olarak düşünülürse, ve bu ilişki de bir kayıtlar kümesi olarak ele alınırsa, bloğun fonksiyonu bu küme üzerinde 'elemanı değildir', ('not member') operasyonu olarak düşünülebilir. 'Elemanı değildir' testinden geçen Hareketler çıkış kuyruğuna gönderilir, geçemeyenler bloke edilerek sistemden çıkartılır.

Söz dizimi :

Blok NOTMEMBER anahtar kelimesi ile başlar. Bunu takiben aynı 'Apply' bloğunda olduğu gibi bir ilişkilendirme listesi yer alır. Bu liste birbirinden virgül ile ayrılmış 'Hareket değişkeni/sahası ismi, "|" karakteri, ilişki sahası ismi' kombinasyonlarından oluşur. Bu liste bitiminde '/' karakteri ve bunu takiben ilişki ismi yer alır.

Örnek :

NOTMEMBER _Adi|_adi, _Soyadi|_soyadi / R_PERSONEL ;

3.2.20 "APPLY CUMULATIVE" bloğu söz dizimi yapısı

Açıklama :

Açıklama : Bu blok belirli birikimsel fonksiyonların, blok içinden geçen Hareketlerin belirli değişkenlerine birikimsel olarak uygulanması ve elde edilen sonucun her bir Harekete _Result adlı sistem değişkeni altında ilave edilmesi amacıyla tasarlanmıştır. Bu blokta istenirse biribirinden virgül ile ayrılmış bir Hareket değişken/saha listesi tanımlanarak giren Hareketlerin bu Hareket değişkenlerinde taşıdıkları değerler esas alınarak sınıflara ayrılması ve birikimsel fonksiyonun bu sınıflar içinde kalan Hareketlere ayrı ayrı uygulanması sağlanabilir. Bu durumda her Hareket mensup olduğu sınıf için elde edilen birikimsel fonksiyon sonucunu, _Result adlı sistem değişkeninde taşıyarak bir sonraki bloğa gidecektir.

Söz dizimi :

Blok APPLY anahtar kelimesi ile başlar. Bunu aşağıda listelenen birikimsel operatörlerden biri izler. Basit formda, tasnif söz konusu olmadığı durumda blok ON anahtar kelimesi ve bunu takiben birikimsel fonksiyonun uygulanacağı Hareket değişkeni ismi ile sona erer. Eğer tasnif söz konusu ise yukarıdaki yapının sonunda BY anahtar kelimesi ve biribirinden virgül ile ayrılmış Hareket değişkenlerinden oluşan tasnife esas oluşturan liste yer alır.

Açıklama :

Operatörler :

SUM (Birikimsel toplama)

PROD (Birikimsel çarpma)

MAX (Sayısal dizinin maksimum değerdeki elemanı)

MIN (Sayısal dizinin minimum değerdeki elemanı)

AVG (Sayısal dizinin aritmetik ortalaması)

Örnek :

APPLY AVG ON _Puan BY _Sınıf,_Kısım;

3.2.21 "APPEND" bloğu söz dizimi yapısı

Açıklama :

Append blok verilen bir ilişki üzerine yeni kayıtlar ekleyen bir bloktur. Giriş yapan Hareketlerin belirli değişkenleri ile ilişkisinin sahaları karşılıklı ilişkilendirilir. Her Hareket girişinde ilişkiye yeni bir kayıt ilave edilir. Bu kaydın ilişkilendirmede ismi belirtilen sahalara, Hareketin karşılık gelen değişkenlerinin değerleri yerleştirilir. İlişkilendirmede ismi geçmeyen sahalara NULL veri atanır.

Söz dizimi :

Blok APPEND anahtar kelimesi ile başlar. Bunu LOCATE anahtar kelimesi, ilişki ismi ve birbirinden virgül ile ayrılmış, ' ilişki saha ismi , '=' karakteri , Hareket değişken ismi' kombinasyonlarından oluşan ilişkilendirme listesi izler. Buradaki '=' karakteri atama operasyonu olarak düşünülebilir.

Örnek :

Program blokları arasında "Hareket" adı verilen veri bloğumları

APPEND LOCATE R_PARCA _parcano=_Parno, _parcaismi=_Parisim;

3.2.22 "DELETE" bloğu söz dizimi yapısı

Açıklama :

Bu blok giren Hareket tarafından tetiklenen kayıt iptali işlevini üstlenmiştir. Update bloğunda olduğu gibi bir ilişki ismi ve bu ilişkisinin sahaları ile Hareket değişkenleri arasında bir ilişkilendirme listesi tanımlanır. Ancak burada ilişkilendirmede yer alan '=' karakteri update bloğunda olduğu gibi atama manasında değil, eşitlik şartı manasındadır. Bir Hareket bloğa girdiğinde, ilişkilendirme listesinde yer alan Hareket değişkenlerinin değerleri, karşılık gelen ilişki sahalalarının değerleri ile kayıt bazında karşılaştırılarak, tanımlı ilişkinin içinde ilerlenir. Sahalarının değerleri, aranan değerlere, hep birlikte, tam uyum gösteren ilk kayıt seçilir ve bu

kayıt iptal edilmiş olarak etiketlenir.

Söz dizimi : İsimler, aritmetik işlemler, veri tabanına yönelik Blok DELETE anahtar kelimesi ile başlar. Bunu LOCATE anahtar kelimesi, ilişki ismi ve birbirinden virgül ile ayrılmış, ' ilişki saha ismi , '=' karakteri , Hareket değişken ismi' kombinasyonlarından oluşan ilişkilendirme listesi izler. Buradaki '=' karakteri eşitlik şartı olarak düşünülmelidir.

4. "RELP" PROGRAMLAMA DİLİ ANLAMSAL YAPISI

Bu bölümde RELP Dili anlamsal ve pragmatik yapısı doğal dil kullanılarak anlatılacaktır.

4.1 Genel

4.1.1 Kuyruk ve Hareket birimleri (transaction)

Program blokları arasında "Hareket" adı verilen veri oluşumları akar. Her bir "Hareket" bir veya birden fazla veriyi temsil eden Simge (token) içerebilir.

Simgeler "Simge Deposu" (Token Store) adını verdiğimiz veri yapısında depolanan gerçek verilere tam sayı indeks oluştururlar. Sistem iki adet Simge Deposu kullanır. Bunlardan birisinde Veri Tabanına depolanmış veriler kalıcı olarak bulunmaktadır. Geçici Simge deposunda ise kalıcı olarak depolanmayacak geçici olarak kullanılan veriler depolanır. Bir Hareket içindeki geçici ve kalıcı nitelikteki veriler böylece Simgeler ile temsil edilirler.

Hareket birimleri, bir programın icrası esnasında, program blokları arasında oluşturulan kuyruk yapıları üzerinden, bloktan bloğa akarlar. Her blok, girişten aldığı her Hareket üzerinde daha önceden programlanmış olduğu işlemleri yaparak gerekiyorsa çıkış kuyruğuna

veya kuyruklarına yeni Hareketler gönderir.

Mantıksal işlemler, aritmetik işlemler, veri tabanına yönelik işlemler, Hareket akışını veya sırasını değiştiren işlemler bloklar içinde yer alırlar.

4.1.2 İsimlendirme ortamları (context) ve kapsamları (scope)

Hareket birimleri içinde yer alan veriler'in isimlendirilmesi değişken ve saha isimleri ile yapılır. Diğer bir deyimle Hareket birimleri içindeki verilere değişken ve saha isimleri ile referans yapılır.

Programda her kuyruk, bir isimlendirme ortamı (Context) oluşturur. Hareket Değişkenleri bu kapsam içinde tanımlıdır. Bu değişkenler kuyruğun çıkışında yer alan blok içinde kullanılacaksa değer geçirme yöntemiyle, bu blok içinde tanımlı değişkenlere değerlerini geçirirler.

"Compute" Bloğu gibi bazı bloklar içinde kuyruklardan ayrı, blok içinde geçerli isimlendirme ortamı (Context) vardır. Bu ortam, blok içinde kullanılan değişkenlerin tanımlandığı ortamdır. Yukarıda belirtildiği gibi, Hareket Değişkenlerinin değerleri Hareketin bloğa vardığı anda karşılık gelen blok değişkenlerine aktarılır.

4.1.3 İsim bildirimleri

İsim bildirimleri otomatik olarak yapılmaktadır. Değişken ve saha (Attribute) isimleri ilk kullanıldıkları noktada bildirilmiş olurlar ve kullanıldıkları ortama göre otomatik tip alırlar.

"Input" Bloğunda ve "Generate" bloklarında, yani sisteme giriş noktalarında değişken ve saha isimleri belirli tipler ile ilişkilendirilebilmektedir. Çıkış bloğunda yine veri tipi belirtilerek, değeri

çıkış birimine yazılacak değişkenin/sahanın uygun formatta yazılması sağlanır.

4.1.4 Değişken ve saha tipleri

Dilin ilk tasarımında sadece iki veri tipine yer verilmiştir. Bunlar:

1. Karakter dizini. (CHAR)
2. Sayısal (NUM)

tipleridir. Sayısal tip, prototip dilin uygulamasında, altta yatan depolama ve işleme esasları yönünden Kayar Noktalı C programlama dili veri tipine eş olarak tanımlanmıştır.

Karakter dizini tipi ise, söz dizimi kurallarında belirtilen karakterlerden oluşan karakter dizilerini değer olarak kabul eden tiptir.

4.1.5 Değişkenler, aritmetik ve mantıksal ifadeler

"Compute" ve "Compute Fix" bloğunda yer alan aritmetik atama deyimleri ile "Test" Bloğunda yer alan mantıksal atama deyimi bir çok programlama dilinde olduğu gibi, aritmetik ve mantıksal sonuçların blok içinde tanımlı değişkenler ve sabitlerin kullanımıyla hesaplanması amacına hizmet eder. Bu konudaki operatörler, operatör öncelikleri, ve kurallar, bloklar ile ilgili detaylı açıklamalarda yer almaktadır.

4.2 Blokların anlamsal yapılarının ayrı ayrı açıklanması

Bu bölümde, blokların her biri ile ilgili anlamsal yapı detaylı olarak açıklanacaktır. Blokların her birinin yaptığı iş çok basit olduğundan, formal anlamsal tanım yapılmasından ziyade tabii dil kullanılarak açıklama yapılmasının yeterli hassasiyeti sağlayacağı değerlendirilmiştir. Her bir blok aşağıdaki formatta açıklanacaktır.

1. Giriş kuyruğu düzeni ve giriş kuyruğundan okunacak Hareket birimlerinin özellikleri.

2. Bloğun işlevi.

3. Bloğun kısıtları.

4. Bloğun çıkış kuyruk düzeni ve çıkış kuyruğuna/kuyruklarına gönderilecek Hareket birimlerinin özellikleri.

4.2.1 "INPUT" bloğu anlamsal yapısı

Giriş Kuyruğu : Yoktur. Kullanıcı tarafından klavyeden girilen veriler kümesi standart girişten okunur.

Bloğun işlevi : Her bir satırda:

1. Parantez içinde tanımlanan satır ve sütun koordinatlarına , GET anahtar kelimesinden sonra ' ' karakteri ile belirlenen başlık karakter dizini yazar.

2. Bu noktada tanımlanmış tipi (CHAR veya NUM) esas alarak girişten gelen veriyi okur.

3. Bu veriyi geçici Simge deposunda arar bulamazsa yeni veri olarak geçici Simge deposuna yerleştirir. Karşılık gelen geçici Simgeyi çıkışa gönderilecek Hareket birimine satırın en sonunda belirtilen değişken ismi altında depolar.

Tüm satırlar bittiğinde kullanıcıya "DEVAM E/H (Evet veya Hayır)" sorusunu yöneltir. Ve cevap "evet" ise işlemler tekrar eder, cevap "hayır" ise çıkışa bitiş Simgesini içeren bir Hareket yazarak durur.

Bloğun kısıtı : Blok maksimum Hareket birimi genişliği olan maksimum 100 adet değişkene değer okuyabilir.

Çıkış kuyruğu : Çıkışta tek bir kuyruk olup, her giriş tamamlandığında oluşturulan Hareket birimi çıkış kuyruğuna yazılır.

Arka arkaya yapılan girişler sonucu kullanıcı "DEVAM E/H" sorusuna "H" (hayır) cevabını verdiğinde çıkış kuyruğuna sonlandırma Simgesi olan -1 yazılır.

4.2.2 "GENERATEE (Generate element) bloğu anlamsal yapısı

Giriş Kuyruğu : Giriş kuyruğu yoktur. Giriş verileri direk olarak biribirinden virgül ile ayrılmış değişkenlere atanan sabit verilerdir. Değişkenler, tiplerini otomatik olarak kendilerine atanan veri tipine bağlı olarak alırlar. (Sayısal-NUM veya Karakter dizini-CHAR) Bu verilere karşılık gelen Simgeler çıkış Hareket birimine ilgili oldukları değişken isimleri altında depolanırlar.

Bloğun işlevi : GENERATEE anahtar kelimesini takibeden atama kombinasyonları içinde yer alan verilere ait Simgeler karşılık geldikleri değişken isimleri altında çıkış Hareketine yerleştirilir.

Bloğun kısıtı : Blok toplam 10 adet sayısal veya karakter dizini sabit girişine izin verir. Karakter dizini maksimum 256 karakter uzunluğunda olabilir.

Çıkış kuyruğu : Çıkışta tek bir kuyruk olup, oluşturulan bir adet Hareket birimi çıkış kuyruğuna yazılır. Bunu takiben hemen akış sonlandırma standart Hareketi olan (-1) çıkış kuyruğuna yazılır.

4.2.3 "GENERATET (Generate tuple) bloğu anlamsal yapısı

Giriş Kuyruğu : Giriş kuyruğu yoktur. Anahtar kelime GENERATET'yi takiben yer alan ilişki esas alınarak bu ilişkiye ait iste halinde belirtilen sahaları içeren kayıtlar sırayla ilişkiden okunur.

Bloğun işlevi : GENERATEE anahtar kelimesini takibeden tüm kayıtlarının, hemen ilişki isminin ardından biribirinden virgül ile ayrılmış olarak belirtilen sahalarını içeren halleri çıkış Hareketleri

halinde çıkış kuyruğuna yazılır. İlişkiden alınarak Hareket birimine atanan Simgeler Hareket birimi içinde ait oldukları ilişki sahasının ismi ile isimlendirilirler.

Bloğun kısıtı : Blok toplam 10 adet sahadan oluşan kayıt girişine izin verir.

Çıkış kuyruğu : Çıkışta tek bir kuyruk olup, oluşturulan kayıtlar Hareket birimleri halinde çıkış kuyruğuna yazılır. Bunları takiben akış sonlandırma standart Hareketi olan (-1) çıkış kuyruğuna yazılır.

4.2.4 "GENERATE" (Generate attribute) bloğu anlamsal yapısı

Giriş Kuyruğu : Giriş kuyruğu yoktur. Anahtar kelime GENERATE'i takiben yer alan ilişki esas alınarak bu ilişkiye ait belirtilen sadece bir sahayı içeren kayıtlar sırayla ilişkiden okunur.

Bloğun işlevi : GENERATE anahtar kelimesini takibeden ilişki isminin ardından belirtilen sadece bir sahayı içeren kayıtlar çıkış Hareketleri halinde çıkış kuyruğuna yazılır. İlişkiden alınarak Hareket birimine atanan Simgeler Hareket birimi içinde söz konusu saha ismi ile isimlendirilirler.

Bloğun kısıtı : Blok sadece bir sahanın içeriğini üretir.

Çıkış kuyruğu : Çıkışta tek bir kuyruk olup, oluşturulan kayıtlar Hareket birimleri halinde çıkış kuyruğuna yazılır. Bunları takiben akış sonlandırma standart Hareketi olan (-1) çıkış kuyruğuna yazılır.

4.2.5 "APPLY" bloğu anlamsal yapısı

Giriş kuyruğu : Bir adet giriş kuyruğu vardır. Bu kuyruktan

sırayla kayıtlar okunur. Bu kayıtlar içinde yer alan verileri temsil eden Simgelerin yerleştiği Hareket gözleri, kuyruk isim ortamında (Context) değişken isimleri veya saha isimleri ile isimlendirilmiş durumdadır. Programcı tarafından istenen değişken ve saha isimleri uygulamanın yapıldığı ilişkinin istenen saha isimleri ile eşleştirilirler. Bu eşleştirmeler Hareket Değişken ismini takiben "pipe" karakteri ve ilişki saha isminden oluşur. Bu tür eşleştirmeler biribirinden virgül ile ayrılırlar ve bir liste oluştururlar. Bu listeye eşleştirme listesi denir, bir veya birden fazla eşleştirme içerebilir.

Bloğun işlevi : Blok eşleştirme listesinden sonra "/" sembolü ardından belirtilen ilişkiyi, giriş kaydına eşleştirme listesindeki saha isimlerini ve bu saha isimleri ile eşleştirilmiş Hareket değişkenlerinin değerlerini esas alarak uygular (Relational Application). Bu uygulama esnasında, efektif olarak, eşleştirme listesinde belirtilen ilişki saha isimleri esas alınıp bir ilişkisel projeksiyon işlemi yapıp, ortaya çıkan ilişkinin sahaları karşılık gelen Hareket değişkenlerinin değerleri ile her kayıt için karşılaştırılır. Tüm sahaları tam bir uyum içinde belirtilen değerleri içeren bir veya daha fazla kayıt seçilerek, bu kayıtların "çıkış saha isimleri listesinde" yer alan saha içerikleri çıkış kuyruğuna yazılır. Giriş Hareket biriminde yer alan tüm değişkenlerin eşleştirmede kullanılması zorunlu değildir. Bu değişkenler, istendiği takdirde, çıkış sahaları listesinden sonra iki nokta karakterini takiben biribirinden virgül ile ayrılmış olarak yer alabilirler. Bunun anlamı ise her üretilen çıkış Hareketinin sonuna bu değişkenlerin giriş Hareketinde içerdiği değerlerin aynı değişken isimleri altında ilave edileceğidir.

Blok kısıtı : Blok 10 saha üzerinden uygulama yapmaya kısıtlıdır.

Çıkış kuyruğu : Bloğun bir çıkış kuyruğu vardır. İlişki ismini takiben "\" sembolünün ardından biribirinden virgül ile ayrılmış çıkış saha isimleri yer alır. İlişkisel uygulama operasyonu esnasında

sırayla kayıtlar okunur. Bu kayıtlar içinde yer alan verileri temsil eden Simgelerin yerleştiği Hareket gözleri, kuyruk isim ortamında (Context) değişken isimleri veya saha isimleri ile isimlendirilmiş durumdadır. Programcı tarafından istenen değişken ve saha isimleri uygulamanın yapıldığı ilişkinin istenen saha isimleri ile eşleştirilirler. Bu eşleştirmeler Hareket Değişken ismini takiben "pipe" karakteri ve ilişki saha isminden oluşur. Bu tür eşleştirmeler biribirinden virgül ile ayrılırlar ve bir liste oluştururlar. Bu listeye eşleştirme listesi denir, bir veya birden fazla eşleştirme içerebilir.

Bloğun işlevi : Blok eşleştirme listesinden sonra "/" sembolü ardından belirtilen ilişkiyi, giriş kaydına eşleştirme listesindeki saha isimlerini ve bu saha isimleri ile eşleştirilmiş Hareket değişkenlerinin değerlerini esas alarak uygular (Relational Application) . Bu uygulama esnasında, efektif olarak, eşleştirme listesinde belirtilen ilişki saha isimleri esas alınıp bir ilişkisel projeksiyon işlemi yapıp, ortaya çıkan ilişkinin sahaları karşılık gelen Hareket değişkenlerinin değerleri ile her kayıt için karşılaştırılır. Tüm sahaları tam bir uyum içinde belirtilen değerleri içeren bir veya daha fazla kayıt seçilerek, bu kayıtların "çıkış saha isimleri listesinde" yer alan saha içerikleri çıkış kuyruğuna yazılır. Giriş Hareket biriminde yer alan tüm değişkenlerin eşleştirmede kullanılması zorunlu değildir. Bu değişkenler, istendiği takdirde, çıkış sahaları listesinden sonra iki nokta karakterini takiben biribirinden virgül ile ayrılmış olarak yer alabilirler. Bunun anlamı ise her üretilen çıkış Hareketinin sonuna bu değişkenlerin giriş Hareketinde içerdiği değerlerin aynı değişken isimleri altında ilave edileceğidir.

Blok kısıtı : Blok 10 saha üzerinden uygulama yapmaya kısıtlıdır.

Çıkış kuyruğu : Bloğun bir çıkış kuyruğu vardır. İlişki ismini takiben "\" sembolünün ardından biribirinden virgül ile ayrılmış çıkış saha isimleri yer alır. İlişkisel uygulama operasyonu esnasında

giriş Hareketindeki değerlere, değerleri tam uyuşan ilişki kayıtlarının, bu sahalarının içerikleri, çıkış kuyruğuna yazılır. Eğer çıkış saha isimlerinin ardından, iki nokta sembolünü takiben bir giriş Hareket değişkenleri listesi yer alıyorsa, bu listede yer alan değişkenlerin giriş Hareket birimindeki değerleri çıkış Hareket biriminin sonuna eklenir. Bu değerler çıkış Hareket biriminde de aynı isimleri alırlar.

4.2.6 "REPORT" bloğu anlamsal yapısı

Giriş kuyruğu : Bloğun bir giriş kuyruğu vardır. Giriş kuyruğundan okunan Hareket birimlerinde tanımlı olan değişken isimleri blok içinde yazma komutlarında aynen kullanılır.

Bloğun işlevi : Blok yazma satırlarından oluşmuştur. Her bir satırda parantez içinde satır ve sütun olarak yazma işleminin yapılacağı ekran koordinatları, bu noktada gösterilecek başlık karakter dizini, gösterilecek verinin tipi (NUM-sayısal veya CHAR-karakter dizini) ve giriş Hareket birimi değişken ismi yer alır.

Her Hareket birimi giriş kuyruğundan okunduğunda, yazma satırlarında belirtilen değişkenlerin içerikleri bu kurallara uygun olarak ekrana yazılır.

Blok kısıtları : 100 yazma satırı ile limitlidir.

Çıkış kuyruğu : Yoktur.

4.2.7 "COMPUTE" bloğu anlamsal yapısı

Giriş kuyruğu : Bloğun bir giriş kuyruğu vardır. Giriş kuyruğundan okunan Hareket birimi içindeki değişkenler, bir ilişkilendirme listesi ile "Compute Blok" içinde geçerli olan değişkenler ile eşleştirilerek ilişkilendirilirler. Her Hareket birimi okunuşunda,

Hareket birimi içindeki her değişken içinde depolanmış veri karşılık gelen "compute blok" değişkenine kopyalanır.

Bloğun işlevi : Compute blok giriş Hareketindeki sayısal değişkenlerin değerlerini esas alarak, bir veya birden fazla aritmetik atama deyimiyle hesaplamalar yapılmasını sağlar. Bulunan sonuç değerler ve istendiği takdirde giriş Hareketinde yer alan değerlerden bazıları çıkış Hareketine birleştirilerek çıkış Hareketi bir sonraki bloğa yollanır.

Blok gövdesi aşağıdaki gruplardan oluşur:

1. İLİŞKİLENDİRME GRUBU
2. ARAMA GRUBU
3. ATAMA GRUBU
4. ÇIKIŞ GRUBU.

Bu gruplardan sadece ARAMA GRUBU seçimlidir.

1. İlişkilendirme grubu : Hareket birimi içindeki değişkenler bir ilişkilendirme listesi ile "Compute" Blok içinde geçerli olan değişkenler ile eşleştirilerek ilişkilendirilirler. Her Hareket birimi okunuşunda, Hareket birimi içindeki her değişken içinde depolanmış veri karşılık gelen "compute" blok değişkenine bu grup içinde kopyalanır.

2. Arama grubu : Arama grubu, bir ilişkinin belirli bir eşlik kriterine uygun kaydının üzerinde işlem yapılacak tuple olarak seçilmesini sağlar. Bu kayıd arama grubundaki listede yer alan sahalarının isimleri hesaplamalar içinde değişken olarak kullanılabilir. Bu sayede "compute" blok içinde bir ilişkinin belirli bir kaydının içindeki saha değerlerine erişilerek hesaplamada kullanılması sağlanabilir. Eşlik kriteri giriş Hareketindeki bir veya birden fazla değer, söz konusu ilişkinin bir veya birden fazla

anahtar saha içeriğinin karşılıklı eşliğini önerir. Her Hareketin girişinde, bu Hareketde yer alan değerler esas alınarak söz konusu ilişkideki kayıt seçilir ve arama grubunda eşitlik kriterinde yer alan veya sadece ismi belirtilen sahalar "compute" blok içindeki değişken isimlerine ilaveten hesaplamada kullanılabilir duruma girerler. Ancak bu değişkenler (saha isimleri) aritmetik atama deyiminin sol tarafında bulunamazlar.

Bu grup LOCATE anahtar kelimesi ile başlar ve bunu ilişki ismi takip eder. İlişki isminden sonra biribirinden virgül ile ayrılmış 'saha ismi, eşittir karakteri, Compute Blok Değişkeni veya gerçel sayı' kombinasyonları yer alır. Eğer belirtilen ilişki sahası eşlik şartı içinde yer almıyorsa bu kombinasyonda sadece saha ismi yer alır. Eşlik kriteri içinde yer alsın almasın, bu kombinasyonlarda yer alan ilişki sahaları Compute Blok içinde aritmetik ifadelerin sağ tarafında kullanılabilirler.

3. Atama grubu : Bu grupta aritmetik atama deyimleri biribirinden virgül karakteri ile ayrılmış olarak yer alırlar.

Aritmetik atama deyimlerinde aşağıdaki kurallar geçerlidir.

.İşleme giren kısım :

1. Giriş ilişkilendirme Listesinde yer alan Compute Blok değişkenleri.
2. Sayısal Sabitler. (Kayar noktalı aritmetik tipinde sayılar)
3. Arama grubunda bildirilmiş olan ilişki saha isimleri.
4. Daha önce atama deyimi sol tarafında yer alarak otomatik bildirimi yapılmış değişken isimleri.

İlişki saha isimleri ve sabitler aritmetik atama deyimi sol tarafında bulunamazlar. Atama deyimi sol tarafında ilişkilendirme listesinde yer alan "Compute" blok değişkenleri bulunabileceği gibi o ana kadar bildirilmemiş değişken isimleri de yer alabilir. Bu tür

değişkenler otomatik olarak kullanıldıkları anda bildirilmiş olurlar.

Aritmetik ifade içinde kullanılan ilişki saha isimleri, arama grubu tarafından aranarak bulunmuş ilişki kaydı içinde, bu saha ismi altında depolanmış veriyi temsil ederler.

Operator olarak aşağıdaki operatörler geçerlidir.

(Öncelik sırasına göre)

1. Parantez içindeki ifade.
2. Monadic -
3. Çarpma "*" ve Bölme "/".
4. Toplama "+" ve Çıkarma "-"

4. Çıkış grubu : Çıkış grubunda, compute blok içinde tanımlı değişkenler ve saha isimleri biribirinden virgül ile ayrılmış olarak bulunurlar. Bu değişkenler ve/veya sahalara çıkış Hareket içeriğini oluştururlar.

Blok kısıtları : Arama grubunda maksimum 10 girdi bulunabilir.

Çıkış kuyruğu : Çıkış kuyruğu bir adet olup, hesaplama sonunda çıkış grubunda belirtilen değişken/saha içerikleri çıkış Hareket birimine yerleştirilerek çıkış kuyruğuna gönderilir.

4.2.8 "COMFIX" (Compute fix) bloğu anlamsal yapısı

Bu blok Compute Blok ile aynı işlevi görür ve aynı söz dizimi yapısındadır. Farklı olarak arama grubunda yer alan ilişki sahalara aritmetik ifadelerde sol tarafta yer alabilirler.

Buna ilaveten, arama grubunda ismine yer verilen sahalara karakter tipinde değişkenlerin değerlerinin atanmasını sağlayan;

'attribute ismi, eşittir karakteri, karakter tipindeki değişken'

yapısında atama deyimleri yer alabilir. Bu sayede, giriş Hareketinde yer alan karakter dizisi tipindeki verilerin, arama grubunda yer alan ilişkinin, eşlik kriterleri altında seçilen kayıtlarının ismine yer verilen sahalarına işlenebilmesi sağlanmıştır. Bu blok içinde hesaplamalar yapıldığı gibi tek saha günleme işlemleri de böylece yapılabilir.

4.2.9 "TEST" bloğu anlamsal yapısı

Giriş kuyruğu: Compute bloğunda olduğu gibidir.

Blok işlevi : Test bloğu anlamsal yapısı Compute Blok ile aynıdır. Farklı olarak aritmetik atama deyimleri yerine sadece bir adet mantıksal atama deyimini yer almaktadır. Mantıksal atama deyiminin yapısı Söz Dizimi kurallarında detaylı olarak açıklanmıştır. Test bloğunda ayrıca bir Çıkış Grubu yoktur. Girişteki Hareket değişken isimleri ve yapısı çıkışta aynen korunur. Yani, giriş Hareket birimi aynen çıkışa kopyalanır. Mantıksal atama deyimini sonucu DOĞRU (TRUE) ise çıkış Hareket birimi DOĞRU kuyruğuna, YANLIŞ (FALSE) ise YANLIŞ kuyruğuna gönderilir.

Blok kısıtları : Arama grubunda maksimum 10 girdi bulunabilir.

Çıkış kuyruğu : Test bloğunun iki adet çıkış kuyruğu vardır. Bu kuyruklar programcının belirlediği şekilde farklı bloklara bağlantı sağlarlar. dolayısıyla, kuyruklar Atama Grubunun hemen ardından, birbirinden virgül ile ayrılmış blok etiketleri ile özdeşleştirilebilirler.

Eğer mantıksal atama deyimini sonunda sonuç YANLIŞ (FALSE) çıkarsa çıkış Hareket birimi ilk blok etiketi ile markalanmış bloğa giden kuyruğa, eğer DOĞRU (TRUE) çıkarsa, ikinci blok etiketi ile markalanmış bloğa giden kuyruğa gönderilir.

4.2.10 "CARTESIANP" bloğu anlamsal yapısı

Giriş kuyruğu : Bir giriş kuyruğu vardır. Giriş Hareket Birimi (Input Transaction) içinde yer alan değişkenler direk Çıkış Hareket Birimi içinde yer alırlar.

Blok İşlevi : Giriş kuyruğundan gelen kayıt dizisi bir kayıt kümesi, dolayısıyla bir ilişki olarak ele alındığında, bu ilişki ile blok parametresi olarak belirtilen ilişkinin, ilişkisel kartezyen çarpım operasyonuna tabi tutulması sonucu oluşacak kayıtlar bu bloğun çıkışına gönderilirler. Bir farklılık olarak, parametre ilişkinin sadece parametre olarak belirtilen sahaları bu çıkış kaydında yer alırlar. Bu saha isimleri bir ilişkilendirme listesi içinde, çıkış Hareket değişken isimleri ile ilişkilendirilirler. Böylece giriş Hareket değişken isimlerinin yeni eklenen kayıt girdileri içinde yer alan saha isimleri ile çakışması önlenabilir. Bir ilişkinin kendisi ile kartezyen çarpım işlemine tabi tutulması da mümkündür. Aşağıda bir ilişkinin kendisi ile kartezyen çarpımının yapılması ile ilgili bir örnek yer almaktadır.

Örnek : İki sahadan oluşmuş ("_Adi" ve "_Soyadi") biri R_ISIM ilişkisinin kendi kendisi ile kartezyen çarpımı RELP dilinde aşağıdaki şekilde yapılabilir:

```
GENERATET R_ISIM _adi, _soyadi;
CARTESIANP R_ISIM _adi|_Ad, _soyadi|_Soyad ;
REPORT ..... _adi,
           ..... _soyadi ,
           ..... _Ad,
           ..... _Soyad;
```

Eğer ilişki aşağıdaki içerikte ise ;

```
R_ISIM
_adı                               _soyadi
```

AHMET	SOLMAZ
KASIM	TARHAN

Çıkış aşağıdaki gibi olacaktır.

AHMET SOLMAZ AHMET SOLMAZ
AHMET SOLMAZ KASIM TARHAN
KASIM TARHAN AHMET SOLMAZ
KASIM TARHAN KASIM TARHAN

Blok kısıtı : Blok en çok 10 adet çıkış ilişkilendirme girdisi tanımına müsaade eder. Giriş Hareket birimi içinde n adet değişken varsa çıkış Hareket biriminde maksimum $n+10$ adet girdi bulunur.

Çıkış grubu : Tek çıkış kuyruğu vardır. Bir sonraki bloğa bağlantı sağlar.

4.2.11 "SORT" bloğu anlamsal yapısı

Giriş kuyruğu : Bir adet giriş kuyruğu vardır. Giriş Hareket Birimi içinde yer alan değişkenler, direk blok parametresi olarak kullanılır.

Blok İşlevi : Blok, "SORT" anahtar kelimesinden sonra soldan sağa azalan önem sırasında sıralanan Hareket değişkenlerinin taşıdığı değerlere bağlı olarak, giren Hareket birimlerini sıralama işlemine tabi tutar.

Giriş kuyruğundaki bütün Hareketler okunduktan sonra bütün Hareket birimlerini çıkış kuyruğuna sıralı olarak gönderir. Giriş Hareket birimi isim yapısı Çıkış Hareket Biriminde de aynen korunur.

Blok kısıtı : Blok maksimum 10 Hareket Birimi Değişkenini esas alarak sıralama yapabilir.

Çıkış grubu : Tek çıkış kuyruğu vardır. Bir sonraki bloğa bağlantı sağlar.

4.2.12 "TRANSFER" bloğu anlamsal yapısı

Giriş kuyruğu : Bir giriş kuyruğu vardır. Hareket birimi değişkenleri ile ilgili bir işlem yapılmaz, girişteki Hareket aynen çıkış kuyruğuna/kuyruklarına gönderilir. Değişken isim yapısı da aynen korunur.

Blok İşlevi : Blok, giren Hareket birimini, bir bloğa veya daha fazla bloğa, kopyalarını çıkartmak yoluyla gönderir. TRANSFER anahtar kelimesini takip eden blok etiketleri, Hareketin hangi bloklar gönderileceğini belirler. Giriş Hareketi birden fazla kopya haline ayrılarak bu etiketler ile markalanmış bloklar ile bağlantı sağlayan kuyruklara gönderilir.

Blok kısıtı : Bir transfer bloğu en çok 10 bloğa transfer işlemi yapabilir.

Çıkış kuyruğu : Bir veya birden fazla çıkış kuyruğu olabilir. Her çıkış kuyruğu, "Transfer" bloğunu başka bir bloğa bağlar. Bu bloklara "hedef blok" adı verilebilir. Hedef bloklar program içinde blok etiketleri ile markalanmışlardır. Transfer bloğu içinde bu etiketlere referans yapılarak gönderi yapılır. Dolayısıyla, transfer bloğundaki her blok etiketi bir kuyruğa karşılık gelir.

4.2.13 "COUNT" bloğu anlamsal yapısı

Giriş kuyruğu : Bir giriş kuyruğu vardır. Giriş Hareket birimindeki değişkenler direk olarak blok parametresi olarak kullanılırlar.

Blok işlevi : "Count" blok, giriş Hareketlerini COUNT anahtar kelimesini takiben belirtilen Hareket değişkenlerini esas alarak sayar. Bu Hareket değişkenleri içinde yer alan veriler esas alınarak geçen Hareketler tasnif edilir ve her bir cinsten kaç adet geçtiği hesaplanır. Daha sonra bu Hareketler çıkış kuyruğundan sırayla bir

sonraki bloğa gönderilirken, her birinin sonuna tasnifte dahil olduğu sınıfın sayaç değeri eklenir.

Bu sayacın standart değişken ismi "_Cnt" olarak verilmiştir. Çıkışa belirtilen Hareket değişkenleri ve "_Cnt" isimli değişkeni taşıyan Hareket birimleri gönderilir.

Blok kısıtları : En çok 10 Hareket değişkeni esas alınarak sayım yapılabilir.

Çıkış kuyruğu : Çıkış kuyruğu bir tanedir. Bu Hareket için sayaç değerini bir artırarak toplam giriş Hareket sayısını hesaplar.

4.2.14 "FILTER" bloğu anlamsal yapısı

Giriş kuyruğu : Giriş kuyruğu bir adettir. Giriş Hareket birimindeki değişkenler direk olarak blok parametresi olarak kullanılırlar.

Blok işlevi : Bu blok, belirtilen Hareket değişkenlerini esas alır, ve bu değişkenlerinde aynı değerleri taşıyan Hareketlerden birden fazlası çıkışa geçmeyecek şekilde geçen Hareketleri süzer. Belirtilen değişkenlerinde aynı verileri içeren Hareketlerden ilki bu bloktan serbestçe geçerken, takip edenler sistemden çıkarılırlar. Giriş Hareketinin isim yapısı çıkış Hareketinde aynen korunur.

Blok kısıtı : Blok maksimum 10 Hareket değişkenini esas alarak süzme yapabilir.

Çıkış kuyruğu : Çıkış kuyruğu bir tanedir. Her birimi yapısı ve içeriği değiştirilmeden çıkış Hareket birimine yazar.

4.2.15 "SINK" bloğu anlamsal yapısı

Blok işlevi : Bu blok bir referans giriş (RBF) bir de standart

Giriş Bloğu : Giriş kuyruğu bir tanedir. Hareket değişkenlerine yönelik bir işlem yapılmaz.

Blok içinde ayrıca sayısal veya karakter dizisi türlerini kapsar.

Blok işlevi : Blok giriş kuyruğundan okuduğu Hareketleri sistem dışına çıkartır.

Çıkış kuyruğu : Bloğun çıkış kuyruğu yoktur.

4.2.16 "TXCNT" bloğu anlamsal yapısı

Giriş Kuyruğu : Bir giriş kuyruğuna sahiptir. Hareket değişkenleri üzerinde bir işlem yapılmaz.

Blok işlevi : Blok giriş kuyruğundan okuduğu her Hareket için sayaç değerini bir arttırarak toplam giren Hareket sayısını hesaplar. En son Hareket okunduktan sonra blok, içinde sayaç değerini taşıyan tek bir Hareketi çıkış kuyruğuna yazar. Bu sayaç değeri _Cnt adlı Hareket değişkeni ismi altında çıkış Hareketinde depolanır.

Blok kısıtı : Yoktur.

Çıkış kuyruğu : Bir çıkış kuyruğu vardır. Bu kuyruğa içinde giriş Hareketleri sayısı olan tek bir çıkış Hareketi yazılır.

4.2.17 "COND" bloğu anlamsal yapısı

Giriş kuyruğu : Bloğun iki adet giriş kuyruğu vardır. Bunlardan ilki "referans kuyruk" ikincisi ise "standart giriş kuyruğu" olarak adlandırılır. Giriş Hareket biriminde yer alan Hareket değişkenleri aynen blok içinde kullanılırlar. Giriş Hareket birimi yapısı ve içeriği, değiştirilmeden çıkış Hareket birimine yansır.

Blok işlevi : Bu blok bir referans girişe (REF) bir de standart girişe (INP) sahiptir. Bu girişler soldan sağa sırayla iki adet etiket ile temsil edilirler ve diğer bloklar tarafından adreslenebilirler. Bu blok içinde ayrıca sayısal veya karakter dizisi tiplerinin karşılaşt-

tırma operatörlerini kullanan ve referans girişteki Hareketin herhangi bir değişkeni ile standart girişten giren Hareketin herhangi bir değişkeninin karşılaştırılması ile oluşturulan basit şart ifadesi yer almaktadır. Referans girişten sadece bir Hareket okunur ve bu Hareket referans Hareket olarak alınır. Standart girişten ise arka arkaya giriş kuyruğundan giren Hareketler okunur. Standart girişten okunan her Hareketin belirtilen değişkenin içerdiği değer, referans girişteki Hareketin belirtilen değişkeninin içerdiği değer ile yine belirtilen karşılaştırma operatörü ile karşılaştırılır. Sonuç 'Yanlış' (FALSE) ise standart girişten gelen Hareket sistemden çıkarılır ve çıkış kuyruğuna verilmez. Sonuç 'Doğru' (TRUE) ise standart girişten gelen Hareket çıkış kuyruğuna gönderilir.

Blok başında COND anahtar kelimesinin ardından biribirinden virgül ile ayrılmış iki etiket yer alır. Bu etiketlerden ilki, diğer bloklar tarafından referans giriş kuyruğuna Hareket göndermek için kullanılır. İkinci etiket ise diğer bloklar tarafından standart giriş kuyruğuna Hareket göndermek üzere kullanılır.

Karşılaştırma ifadesi, REF ön eki ile belirtilmiş, referans giriş Hareketinin değişkeni ile INP ön eki ile belirtilmiş standart giriş Hareketi değişkeninin aşağıdaki karşılaştırma operatörleri ile karşılaştırılmasını sağlar:

Büyüktür >

Küçüktür <

Eşit değildir !=

Büyük veya eşit >=

Küçük veya eşit <=

Eşittir ==

Karşılaştırma sonucu DOĞRU (TRUE) değerini verirse, standart girişten gelen Hareket aynen çıkış kuyruğuna kopyalanır. Eğer sonuç YANLIŞ (FALSE) çıkarsa giriş Hareketi sistemden çıkarılır ve çıkışa gönderilmez.

Blok kısıtı : Blok referans kuyruktan sadece bir giriş Hareketi okumaya programlanmıştır. Bu kuyruğa gönderilen ikinci Hareket akış sonu (-1) Hareketi olmalıdır.

Çıkış kuyruğu : Çıkış kuyruğu tektir. Çıkış Hareketleri, Giriş Hareketlerinin isim yapısını ve içeriğini aynen yansıtır.

4.2.18 "NOTMEMBER" bloğu anlamsal yapısı

Giriş kuyruğu : Bir giriş kuyruğu vardır. Giriş Hareket değişkenleri, ilişkilendirme listesinde, eşleştirme çiftlerinin ilk elemanı olarak kullanılırlar. Bu çiftlerin ikinci elemanları ise, işlemin uygulandığı ilişkinin saha isimleri olabilir.

Blok işlevi : Bu blok giren Hareketin belirli değişkenlerini belirli bir ilişkinin sahaları ile karşılıklı ilişkilendirir. Eğer bloğa giren herhangi bir Hareketin bu belirli değişkenlerinin içerikleri, ilişki içindeki karşılık gelen sahalar altında herhangi bir kayıt içinde birlikte yer almıyorsa, Hareket çıkış kuyruğuna gönderilir, aksi takdirde Hareket, sistemden çıkartılır. Eğer, ilişki sahaları, ilişki sahaları esas alınarak ilişkisel projeksiyon operasyonu ile elde edilmiş bir yeni ilişki olarak düşünülürse, ve bu ilişki de bir kayıtlar kümesi olarak ele alınırsa, bloğun fonksiyonu bu küme üzerinde 'elemanı değildir' ,('not member') operasyonu olarak düşünülebilir. 'Elemanı değildir' testinden geçen giriş Hareketleri çıkış kuyruğuna gönderilir, geçemeyenler ise sistemden çıkartılır.

Blok kısıtları : İlişkilendirme listesinde maksimum 10 eşleştirme bulunabilir.

Çıkış kuyruğu : Yukarıda belirtilen şart sağlandığında, çıkış kuyruğuna, giriş Hareketi aynen kopya edilir. Dolayısıyla, isim yapısı ve içeriği aynen çıkışa yansıtılmış olur.

4.2.19 "APPLY CUMULATIVE" bloğu anlamsal yapısı

Giriş kuyruğu : Bir giriş kuyruğu vardır. Okunan Hareketin değişkenleri blok içinde kullanılırlar.

Blok işlevi : Bu blok belirli fonksiyonların, blok içinden geçen Hareketlerin belirli bir değişkenine uygulanması ve elde edilen sonucun her bir Harekete _Result adlı sistem değişkeni altında ilave edilmesi amacıyla tasarımlanmıştır. Birikimsel fonksiyonun uygulanacağı Hareket değişkeni ON anahtar kelimesinin ardından yer alır. Bu blokta istenirse BY anahtar kelimesi kullanılıp bunu takiben, biribirinden virgül ile ayrılmış bir Hareket değişken listesi tanımlanarak giren Hareketlerin bu Hareket değişkenlerinde taşıdıkları değerler esas alınarak sınıflara ayrılması ve birikimsel fonksiyonun bu sınıflar içinde kalan Hareketlere ayrı ayrı uygulanması sağlanabilir. Bu durumda her Hareket, ait olduğu sınıf için elde edilen birikimsel fonksiyon sonucunu, _Result adlı sistem değişkeninde taşıyarak bir sonraki bloğa gidecektir.

Birikimsel fonksiyonlar aşağıda listelenmiştir.

SUM (toplama)
 PROD (çarpma)
 MAX (Sayısal dizinin maksimum değerdeki elemanı)
 MIN (Sayısal dizinin minimum değerdeki elemanı)
 AVG (Sayısal dizinin aritmetik ortalaması)

Blok kısıtı : Birikimsel fonksiyon sadece bir Hareket değişkenine uygulanabilir. Tasnif seçeneği kullanıldığında maksimum 10 Hareket değişkeni tasnif için esas alınabilir.

Çıkış kuyruğu : Çıkış kuyruğu tek olup, Giriş Hareketleri çıkışa aynen yansıtılırlar. Dolayısıyla aynı isim yapısı ve içeriktedirler. Ancak çıkış Hareketinin sonuna, birikimsel fonksiyonun sonucunu içeren "_Result" adlı değişken ilave edilmiştir. Takip eden blok

veya bloklar bu değişkeni kullanarak bu blok içinde çıkan sonucu kullanabilirler.

4.2.20 "APPEND" bloğu anlamsal yapısı

Giriş kuyruğu : Bir giriş kuyruğu vardır. Giriş Hareket değişkenleri blok içinde saha isimleri ile ilişkilendirmede direk olarak kullanılırlar.

Bloğun işlevi : "Append" blok, verilen bir ilişki üzerine yeni kayıtlar ekleyen bir bloktur. Giriş yapan Hareketlerin belirli değişkenleri ile ilişkinin sahaları karşılıklı ilişkilendirilir. Her Hareket girişinde ilişkiye yeni bir kayıt ilave edilir. Bu kaydın, ilişkilendirmede ismi belirtilen sahalarına, Hareketin karşılık gelen değişkenlerinin değerleri yerleştirilir. İlişkilendirmede ismi geçmeyen sahalara NULL veri atanır.

Blok kısıtları : Yeni eklenen kayda maksimum 10 başlangıç değeri atanabilir(*).

Çıkış kuyruğu : Giriş kuyruğundan gelen Hareketler aynen çıkış kuyruğuna geçirilir.

4.2.21 "DELETE" bloğu anlamsal yapısı

Giriş kuyruğu : Bir giriş kuyruğu vardır. Giriş Hareket değişkenleri blok içinde saha isimleri ile ilişkilendirmede direk olarak kullanılırlar.

Blok işlevi : Bu blok giren Hareket tarafından tetiklenen, kayıt silme işlevini üstlenmiştir. "Append" bloğunda olduğu gibi bir ilişki ismi ve bu ilişkinin sahaları ile Hareket değişkenleri arasında bir

(*) Bu bir dil kısıtı olmayıp ilk dil uygulamasındaki kısıt olarak kabul edilmelidir.

ilişkilendirme listesi tanımlanır. Ancak burada ilişkilendirmede yer alan '=' karakteri "append" bloğunda olduğu gibi atama anlamında değil, eşitlik şartı anlamındadır. Bir Hareket bloğa girdiğinde, ilişkilendirme listesinde yer alan Hareket değişkenlerinin değerleri, karşılık gelen ilişki sahalarının değerleri ile kayıt bazında karşılaştırılarak, tanımlı ilişkinin içinde ilerlenir. Sahalarının değerleri, aranan değerlere, birlikte, tam uyum gösteren ilk kayıt seçilir ve bu kayıt iptal edilmiş olarak markalanır.

Blok kısıtı : Maksimum 10 Hareket değişkeni esas alınarak iptal işlemi yapılır.

Çıkış kuyruğu : Giriş kuyruğundan gelen Hareketler aynen çıkış kuyruğuna geçirilir.

4.2.22 "MERGE" bloğu anlamsal yapısı

Giriş kuyruğu : Merge blok birden fazla giriş kuyruğuna sahiptir. Bu kuyrukların her biri Merge bloğuna çıkış yapan bloklardan biri ile bağlantı sağlar.

Blok işlevi : Bloğun işlevi değişik kuyruklardan okunan Hareket birimlerinin hepsini tek bir çıkış kanalına kanalize etmektir. Blok birden fazla etiket ile isimlendirilebilir. Bu etiketler aynı zamanda "transfer" ve "test" bloklarının, "merge" bloğuna gönderi yapmak üzere kullandığı etiketlerdir. Bir "transfer" veya "test" bloğunun çıkışında bulunan etiket eğer "merge" bloğunda etiket listesinde de yer alıyorsa bu iki blok arasında bir kuyruk oluşmuş demektir.

Bu etiketlerden ilki referans etiket olup, bu kuyruktan okunan Hareket biriminin isim yapısı, çıkış kuyruğuna gönderilen Hareketlerin isim yapısı olarak uygulanır. Burada programcıya düşen önemli görev birden fazla kaynaktan gelip tek bir çıkış kuyruğuna kanalize edilen Hareketlerin hepsinin Hareket birim uzunlukları, değişken

tipleri ve anlamlarının birinci kuyruktan gelen Hareketin yapısına uygun olmasının sağlanmasıdır.

Blok kısıtı : Blok maksimum 10 ayrı kaynak bloktan gelen Hareket akışlarını tek bir kuyruğa kanallize edebilir.

Çıkış kuyruğu : Tek çıkış kuyruğu olup, ilk giriş kuyruğunun Hareket ve isim yapısını aynen taşır.

4.3 RELP programlama dilinin "ilişkisel tamam" (relationally complete) olduğunun gösterilmesi

Önergeler Kalkülüs'ünün (Predicate Calculus), sorgulama dillerinde kullanılmasının fikri ilk olarak Kuhns tarafından önerilmiş (Kuhns,1967). Codd tarafından ise, ilişkiler üzerinde Önergeler Kalkülüs'ü kullanan İlişkisel Kalkülüs (Relational Calculus) önerilmiştir. (Codd,1972).

İlişkisel Cebir ise, yine ilişkiler üzerinde tanımlı işlemlerden oluşan, ve üst düzeyli veri dillerine temel teşkil eden bir cebir olarak, yine Codd tarafından önerilmiştir. (Codd,1972).

Codd her iki formal tanımı yaptığı makalesinde, "İlişkisel Tamamlık" (Relational Completeness) tanımını da yapmıştır.

Bu tanım şu şekilde tercüme edilebilir: Bir programlama dili, İlişkisel Kalkülüs'de, (Relational Calculus) İlişkisel Kalkülüs ifadeleri ile tanımlanabilen her ilişkiyi, içinde yer alan uygun deyimlerle ortaya çıkartabilecek özelliğe sahip ise, "İlişkisel Tamam" (Relationally Complete) olarak nitelenir. (E.F.Codd,1972).

Aynı makalede, herhangi bir İlişkisel Kalkülüs ifadesinin, anlamsal olarak eşdeğer bir İlişkisel Cebir ifadesine dönüştürülebileceği, yani İlişkisel Cebirin de "İlişkisel Tamam" olduğu gösterilmiştir.

Bu durumda, herhangi bir dilin "İlişkisel Tamam" olduğunu göstermek için, dilin, ilişkisel cebirdeki beş temel cebrik operasyondan her birine karşılık gelen operasyonlara sahip olduğunu göstermek yeterli olmaktadır (Date,1981).

Bu beş operasyon şu şekilde sıralanabilir :

Union (Birleşim)

Difference (Fark)

Product (Çarpım)

Selection (Seçim)

Projection (Projeksiyon)

REL P programlama dilinde bu operasyonların herbirine karşılık gelen operasyonlar vardır. Ancak REL P dilinin veri akış yapısı, gerekli akışların yaratılması amacıyla, bu operasyonlara ilaveten bazı diğer REL P dili bloklarının kullanılmasını zorunlu kılmaktadır. Bu bölümde gösterilmesi amaçlanan husus, REL P dili kullanılarak İlişkisel Cebirde yapılabilen işlemler ile eş fonksiyonda işlemlerin yapılabileceğidir.

Union (Birleşim) :

A ve B gibi iki ilişkinin birleşimi, ya A yada B'ye ait olan tüm kayıtların oluşturduğu küme olarak tanımlanır.

REL P programlama dilinde karşılık gelen operasyonu MERGE bloğu yerine getirmektedir.

MERGE bloğunun 1 numaralı giriş kuyruğundan gelen kayıtlar kümesine (İlişkiye) A, 2 numaralı giriş kuyruğundan gelen kayıtlar kümesine B dersek, çıkış kuyruğundan gönderilmek üzere oluşturulan kayıtlar kümesi A ile B ilişkilerinin birleşimidir.

Difference (Fark) :

A ve B gibi iki ilişki ele alındığında, A-B, ilişkisi, A'ya ait olup B'de yer almayan tüm kayıtlar kümesidir.

REL P programlama dilinde karşılık gelen operasyonu NOTMEMBER bloğu yerine getirmektedir.

NOTMEMBER bloğunun giriş kuyruğundan gelen kayıtlar A ilişkisinin tüm kayıtları ise ve NOTMEMBER bloğunda parametre olarak yer alan ilişki ismi B ise, saha ilişkilendirmeleri de tüm B saha isimleri ile karşılık gelen tüm A saha isimleri arasında yapıyorsa, bloğun çıkışında, B içinde yer almayan A kayıtları, yani A-B ilişkisinin kayıtları yer alır.

Extended Cartesian Product (Çarpım) :

A ve B gibi iki ilişkinin, Kartezyen Çarpımı, a kaydı $(a_1, a_2, \dots, a_m)$, A ilişkisinden, b kaydı $(b_1, b_2, \dots, b_n)$ B ilişkisinden olmak üzere, seçilen a ve b kayıtları ile oluşturulan tüm (a, b) permütasyonlarında a ile b'nin uç uca eklenmesi (concatenation) sonucu oluşan t kayıtlarının kümesi olarak tanımlanır. Oluşturulan t kaydı $(a_1, a_2, \dots, a_m, b_{m+1}, b_{m+2}, \dots, b_{m+n})$ olarak gösterilir.

REL P programlama dilinde karşılık gelen operasyonu CARTESIANP bloğu yerine getirmektedir.

CARTESIANP bloğunun giriş kuyruğundan gelen kayıtlar A ilişkisinin tüm kayıtları ise ve CARTESIANP bloğunda parametre olarak yer alan ilişki ismi "B" ise, B ilişkisinin çıkışa gönderilecek saha isimleri çıkış saha isimleri olarak parametre listesinde yer alıyorsa, CARTESIANP bloğundan çıkış kuyruğuna gönderilen kayıtlar A ile B ilişkisinin Kartezyen Çarpımı olan ilişkinin tüm kayıtlarıdır.

Selection (Seçim) :

Cebrik Seçim operasyonu, verilen ilişkinin, verilen mantıksal önermenin sonucunun Doğru (True) çıkmasını sağlayan kayıtlarını seçip, sadece bu kayıtlardan oluşan kümeyi meydana getiren bir operasyondur. Buradaki önerme, sadece gündemdeki bir kayıt esas alındığında, bu kaydın sahaları kullanılarak yaratılan basit karşılaştırma ifadeleri, ve "Doğru" (True) yada "Yanlış" (False) sonucu veren bu ifadelerin mantıksal operatörler ile birleştirilmesi yoluyla oluşturulur. RELP programlama dilinde karşılık gelen operasyonu TEST bloğu yerine getirmektedir.

Interception (Kesim) :

TEST bloğunun giriş kuyruğundan gelen kayıtlar A ilişkisinin tüm kayıtları ise, seçimde kullanılacak önerme, TEST bloğunun mantıksal ifadesi olarak verilmiş ise, TEST bloğunun "Doğru" (True) çıkış kuyruğundan, önermenin "Doğru" sonucunu verdiği kayıtlar çıkar.

Projection (Projeksiyon) :

Projeksiyon operasyonu, verilen ilişkinin, belirtilen sahalarını içeren dikey alt kümesini oluşturup, bu kayıtlar kümesi içindeki tekrar eden kayıtları tek kayıtlara indirger.

RELP programlama dilinde karşılık gelen operasyon GENERATET ve FILTER blok kombinasyonu ile yapılmaktadır.

Verilen bir A ilişkisi ve projeksiyona tabi tutulacak saha isimleri GENERATET bloğu parametresi olarak verildiğinde, bu bloğun çıkışında sadece belirtilen sahaları içeren kısmi kayıtlar üretilir. Bu kayıtlar içinde tekrar edenler olabileceğinden GENERATET bloğu çıkışında elde edilen kayıtlar FILTER bloğuna giriş olarak beslenir ve FILTER bloğu parametresi olarak aynı saha isimleri kullanılırsa, FILTER bloğu çıkışında projeksiyon işlemi sonucu elde edilen kayıt kümesine denk kayıt kümesi elde edilir.

Bu noktaya kadar beş temel İlişkisel Cebir operasyonunun karşılıklarının RELP programlama dilinde nasıl oluşturulduğunu göstermiş bulunuyoruz. Bu, RELP dilinin ilişkisel tamam olduğunu gösterse bile, geride kalan ve temel ilişkisel cebir operasyonları ile ifade edilebilen Kesişim (Intersection), Bölme (Division) ve Doğal Katışım (Natural Join-Equijoin-Eşitlik şartı ile katışım) gibi İlişkisel Cebir operasyonlarının RELP dili karşılıklarını da vermek uygun olacaktır, zira bu operasyonlar "İlişkisel Uygulama" (Relation Application) operasyonu ile RELP dili içinde çok pratik ve güçlü bir şekilde yapılmaktadır.

Intersection (Kesişim) :

A ve B gibi iki ilişkinin kesişimi olan ilişki A ve B'de ortak olan kayıtlardan oluşur.

RELP programlama dilinde kesişim işlemi APPLY bloğu kullanılarak yapılır.

APPLY bloğunun giriş kuyruğundan gelen kayıtlar A ilişkisinin tüm kayıtları ise, APPLY bloğuna parametre olarak verilen ilişki ismi "B" ise, giriş ilişkilendirme listesinde tüm A ilişkisi sahaları, tüm karşılık gelen B ilişkisi sahaları ile ilişkilendirildiyse ve çıkış saha listesinde tüm B saha isimleri yer alıyorsa, çıkış kuyruğuna gönderilen kayıtlar A ilişkisi ile B ilişkisinin kesişimi olan ilişkinin kayıtlarıdır.

Division (Bölme) :

Bu operasyon bölünen A gibi ilişkinin $m+n$ dereceli kayıtlarını, B gibi bir ilişkinin n dereceli kayıtlarına bölüp, m dereceli kayıtlar oluşturan bir operasyondur. A ilişkisinin $(m+i)$ numaralı sahası ile B ilişkisinin (i) numaralı sahası içerikleri aynı veri kümesinden gelmelidir. A ilişkisinin ilk m sahası tek bir komposit saha X, ve

son n sahası Y kompozit sahası olarak düşünülürse, A 'nın (x,y) gibi veri çiftleri içeren kayıtlardan oluştuğu düşünülebilir. Aynı şekilde B ilişkisi tek değerlerden (y) oluşan bir küme gibi düşünülebilir. Bu durumda A ilişkisinin B ilişkisine bölünmesi, B ilişkisinde yer alan tüm y değerleri için A ilişkisinde yer alan (x,y) çiftlerinin x değerlerinin elde edilmesi anlamını taşır.

REL $\bar{P}$ programlama dilinde bölme işlemi APPLY bloğu kullanılarak yapılır.

APPLY bloğunun giriş kuyruğundan giren kayıtlar B ilişkisinin tüm kayıtları ise, APPLY bloğuna ilişki ismi parametresi olarak A ilişkisinin ismi verilmiş ise, A giriş ilişkilendirme listesinde yukarıda tanımda belirtilen n adet B ilişkisi sahası, karşılık gelen n adet A ilişkisi sahası ile ilişkilendirilmekteyse, çıkış sahaları listesinde yukarıdaki tanıma göre m adet A sahası yer almaktaysa, bu şekilde düzenlenmiş bir APPLY bloğunun çıkışında A ilişkisinin B ilişkisine, n adet saha üzerinden bölümünden elde edilecek ilişkinin kayıtları yer alır.

Natural Join - Equijoin (Eşitlik şartı ile katışım) :

Bu katışım işlemi A ile B gibi iki ilişkinin kartezyen çarpımı sonucu elde edilen ilişkide, katışımında esas alınan ve eşleştirilen A ve B ilişkisi sahalarının içerikleri birbirine eşit olan kayıtların seçilmesi ve ortak saha isimlerinin sadece birini kapsayan projeksiyon işleminin yapılmasıdır.

REL $\bar{P}$ programlama dilinde katışım APPLY bloğu kullanılarak yapılır.

APPLY bloğunun giriş kuyruğundan gelen kayıtlar A ilişkisinin tüm kayıtları ise, APPLY bloğuna verilen ilişki ismi parametresi " B " ise, giriş ilişkilendirme listesinde, A ve B ilişkilerinin üzerlerinde

eşitlik şartı önerilen i adet saha karşılıklı ilişkilendirildiyse, çıkış saha isimlerinde B'nin tüm saha isimleri yer alıyorsa, APPLY bloğu "direkt geçiş yapan isimler listesinde" girişte ilişkilendirme listesinde yer almayan tüm A ilişkisi sahaları yer alıyorsa, sonuçta bu blok çıkışında yer alan kayıtlar A ile B ilişkisinin i adet saha üzerinden eşitlik şartına göre katışımı sonucu elde edilen ilişkinin kayıtlarıdır. Görüldüğü gibi APPLY bloğunu değişik şekillerde kullanılarak üç değişik İlişkisel Cebir operasyonunu gerçekleştirebilmekteyiz. Bu ise operasyonun ifade gücünü ve esnekliğini sergilemektedir.

5. RLP DİLİNİN SORGULAMA ALT KÜMESİNİN BAZI SORGULAMA DİLLERİ İLE KARMAŞIKLIK YÖNÜNDEN KARŞILAŞTIRILMASI:

İyi bir Veri Tabanı Sorgulama dilinin hem kullanıcıları tarafından basit ve etkin olarak algılanması hem de bilgisayarda işleme yönünden etkin olması gerekir. Soyutlamalar yolu ile bilgisayar mimarisi ile olan ilişkisi belirsizleştirilen ve kullanıcıya rahat programlama olanağı sağlayan çok üst seviyeli dillerde bu kolaylıkların bedeli dilin bilgisayarda çalışma performansındaki düşüş ile ödenir.

Dilin kullanıcı kolaylıkları yönünden "Kullanıcı-Etkin" olması özelliğini ölçmek için somut kriterler bulunmamaktadır. Bu tür dil karşılaştırmaları çoğunlukla informal olup bu konuda formal araçların yokluğu söz konusudur.

Orman tarafından kullanılan yöntemde, seçilen bir referans dile göre göreceli karmaşıklık ölçülmektedir (Orman,1991). Bu yöntemde minimum ve bağımsız komponentlerden oluşturulmuş fonksiyonel bir programlama dili (Functional Data Language-FDL) doğrusal karmaşıklığa sahip olarak kabul edilmiş ve referans ölçütü olarak kullanılmıştır.

SQL (Structured Query Language) , NDL (Network Data Language) ve RA (Relational Algebra) gibi ayrı sınıflardaki sorgulama dilleri FDL'e göre karmaşıklık yönünden değişik ölçütler kullanılarak ölçülmüştür. Diller hakkında bu yöntemle bulunan sonuçlar ile kullanıcılar tarafından algılanma biçimi arasında yakın ilişki bulunmuştur. Yazar, bulunan ölçütlerin kullanıcının karmaşıklık algılaması için iyi birer tahmin aracı olduğunu öne sürmektedir (Orman, 1991).

Bu yöntemde aşağıdaki karmaşıklık ölçütleri tanımlanmıştır.

1. Operator Karmaşıklığı.
2. Sorgulama Karmaşıklığı.
3. Programlama Dili Karmaşıklığı.
4. Mutlak Karmaşıklık.
5. İlk Karmaşıklık.

5.1 Operatör karmaşıklığı

Ele alınan programlama dilinin her bir operatörü için ayrı ayrı belirlenir. Ölçülecek dilin her bir operatörü için karşılık gelen FDL ifadesi bulunur. Bu FDL ifadesindeki elemanter operasyon sayısı, ele alınan dil operatörünün karmaşıklığı olarak belirlenir.

Örnek : İlişkisel Cebirde (RA) kısıtlama (Restriction) operatörünün karmaşıklığı şu şekilde bulunmaktadır.

$R(A = c)$ R ilişki, A Domain, c sabit olmak üzere.

Operatörünün FDL karşılığı,

$$\begin{array}{l} rER \\ cEA(r) \end{array} \quad \left| \quad r \right.$$

olup burada iki FDL operatörü yer aldığından RA kısıtlama operatörü karmaşıklığı 2 olarak belirlenir.

5.2 Sorgulama karmaşıklığı *complexity*

Her sorgulama için sorgulamada yer alan tüm operatörlerinin karmaşıklıklarının toplamı olarak belirlenmiştir.

5.3 Programlama dili karmaşıklığı

Sorgulama cümlesi büyüklüğünden (query size) sorgulama karmaşıklığına bir fonksiyon olarak tanımlanmıştır. Bu ölçüt, karmaşıklığın, sorgulama büyüklüğü arttıkça yükselmesindeki değişim hızını esas alır. Buradaki sorgulama büyüklüğü sorgulama cümlesinin FDL karşılığı içinde yer alan operatör sayısı, sorgulama karmaşıklığı ise sorgulama cümlesi içinde yer alan dil operatörlerinin karmaşıklıkları toplamıdır.

SQL, NDL ve RA için yapılan ölçümler ve eğri uydurma yöntemiyle elde edilen sonuçlar (Orman,91)'de yer almaktadır. Bu sonuçlara göre SQL kuadratik , RA doğrusal, NDL ise logaritmik karmaşıklık fonksiyonlarına sahip olarak bulunmuştur.

5.4 Mutlak karmaşıklık

Gerçekte kullanıcıların tümünün algıladığı ve üzerinde anlaştığı en ideal karmaşıklık ölçütü olup, ancak istatistiki sonuçlar ile izah edilebilir.

Kullanılan yöntemde (Orman,1991) ölçüt olarak kabul edilen FDL yapı taşları sayısı ile kullanıcıların sorgulama formülasyon zamanı ele alınıp ölçüldüğünde doğrusal ilişki bulunmuştur. Bu da yazar tarafından ileri sürülen ölçütün Kullanıcı Karmaşıklık Algılaması için iyi bir tahmin aracı olduğunu göstermektedir.

5.5 İlk karmaşıklık (initial complexity)

Dilin tüm operatörlerinin karmaşıklıklarının toplamı olarak tanımlanmaktadır.

5.6 RELP dilinin karmaşıklığı

RELP Dili de fonksiyonel bir dil olduğundan FDL ile aynı aileden olması nedeniyle, operatörlerinin karşılıkları kolaylıkla FDL operatörleri olarak ifade edilebilmektedir.

Veri Tabanı Dillerinin sadece Sorgulama Özelliklerinin karşılaştırılmasını amaçladığımızdan RELP dilinin sorgulamaya dönük bloklarından oluşan bir alt kümesi esas alınarak analiz yapılmıştır.

Aşağıda Tablo 5.6'da bu blokların FDL'deki karşılıkları FDL ifadeleri şeklinde yer almaktadır. Her bloğun yanında karmaşıklık da yer almaktadır.

Tablo 5.6 RELP Bloklarının FDL karşılıkları ve karmaşıklıkları.

R BLOĞU	FDL KARŞILIĞI	KARMAŞIKLIK
INPUT	1	0
GENERATEE	1	0
GENERATE	$r \in R \mid A(r)$	2
GENERATET	$r \in R \mid r$	1

R BLOĞU	FDL KARŞILIĞI	KARMAŞIKLIK
APPLY	$ \begin{array}{c c} r \in (R) & A_1(r) \\ c_1 \in B_1(r) & A_2(r) \\ c_2 \in B_2(r) & . \\ . & . \\ . & . \\ c_n \in B_n(r) & A_m(r) \end{array} $ Burada $B_1 \dots B_n$ giriş sahaları, $A_1 \dots A_m$ çıkış sahalarını gösterir.	minimum 3 (1 + giriş saha sayısı + çıkış saha sayısı) (1 + n + m)
TEST	$ \begin{array}{c c} r \in R & \\ \text{test}(r) & r \end{array} $ ("R" giriş kayıt kümesi, "test" mantıksal değer(0 veya 1) geri döndüren bir mantıksal fonksiyondur)	2
CONDBLOK	$ \begin{array}{c c} r \in R & \\ \text{cond}(r) & r \end{array} $ (cond fonksiyonu giriş kayıt kümesi R'a ait kayıtlar üzerinde bir şart araştırarak doğru veya yanlış sonuç üreten mantıksal bir fonksiyondur.)	2
FILTER	$ \begin{array}{c c} r \in R & r \end{array} $ (R ilişkisi, orjinal ilişki R'nün, içinde hiç bir kaydın birden fazla kez yer almadığı kopyasıdır)	1
TRANSFER	$ \begin{array}{c c} & r, r, r, \dots \end{array} $	0

R BLOĞU	FDL KARŞILIĞI	KARMAŞIKLIK
MERGE	$r \in R$ $s \in S$ $t \in T$ $(R, S, T, \dots, \text{giriş kayıt kümeleridir})$	minimum 2 (Giriş kayıt kümesi sayısına eşit karmaşıklığıdır)
TXSAY	$ \text{cardinality}(R)$ (R giriş kayıt kümesi olup, cardinality fonksiyonu verilen kümenin eleman sayısını bulup tek bir sonuç veren fonksiyondur.)	1
COUNTBY	$r \in \text{countby}(R) \mid r$ (R giriş kayıt kümesi olup, $\text{countby}(R)$ fonksiyonunun sonucu geriye döndürülen küme, her $l \in R$ kaydının R içindeki tekrar sayısı m olmak üzere, $r=1, m$ dir (l kaydının sonuna m sayısı eklenerek bulunan kayıt)	1
NOTMEMBER	$r \in R$ $r \notin S$ (R giriş kayıt kümesi, S ise parametre ilişkidir.)	2
REPORT	$ r$	0

R BLOĞU	FDL KARŞILIĞI	KARMAŞIK-LIK
COMPUTE	$r \in R$ $s = \text{compute}(r)$ s (R giriş Hareket kümesi, compute ise kayıt parametre kabul edip sonuç olarak hesaplanmış değerlerden oluşan tek bir kayıt üreten bir fonksiyondur.)	2
CARTESIANP	$r \in R$ (r, s) $s \in S$ (S kayıt kümesi giriş Hareket kümesi olup, R ilişkisi ise operasyonun uygulandığı ilişkidir.)	2

ILK KARMAŞIKLIK

21

5.7 Örnek bir veri tabanı ve sorgulamalar kümesi temel alınarak RELP dilinin diğer sorgulama dilleri ile karmaşıklık yönünden karşılaştırılması

5.7.1 Sorgulama karmaşıklıklarının karşılaştırılması

Orman, değişik sorgulama dillerinin, sorgulama karmaşıklığını karşılaştırmak üzere aşağıdaki ilişki tablolarından oluşan bir örnek veri tabanı üzerinde belirli sorgulamalar tanımlayarak bunların her dildeki sorgulama programlarını yazmış ve her sorgulamanın karmaşıklığını ölçerek listelemiştir. (Orman,1991).

Veri Tabanı "Üretici" (SUPPLIER), "Parça" (PART), "Proje" (PROJECT) ve "Gönderi" (SUPPLY) adlı ilişkilerden oluşmuş, üretim uygulamasına dönük bir veri tabanıdır.

İlişki yapıları aşağıdaki gibidir:

SNO, LOCATION (SUPPLIER) : Her üreticinin numarası (SNO) ve yeri (LOCATION) , üretici (SUPPLIER) adlı ilişki içinde saha (attribute) olarak tanımlanmıştır.

PNO, TYPE, PRICE (PART) : Her parçanın numarası (PNO) , tipi (TYPE) ve fiyatı (PRICE) , "Parça" (PART) adlı ilişkinin içinde saha (attribute) olarak tanımlanmıştır.

JNO, LOCATION (PROJECT) : Her projenin numarası (JNO) ve yeri (LOCATION) , "Proje" (PROJECT) adlı ilişkinin içinde saha (attribute) olarak tanımlanmıştır.

LNO, SNO, PNO, JNO, Q (SUPPLY) : Her gönderinin numarası (LNO) , üretici numarası (SNO) , parça numarası (PNO) , proje numarası (JNO) ve miktarı (Q) , "Gönderi" (SUPPLY) adlı ilişki içinde saha (attribute) olarak tanımlanmıştır.

Aynı sorgulamaların RELP programlama dilindeki karşılıkları ve karmaşıklıkları şekil 5.7.1.2 den şekil 5.7.1.9'a kadar olan şekillerde gösterilmiştir.

Bulunan karmaşıklıkların diğer dillerin aynı sorgulamalar için karmaşıklıkları (Orman,1991) ile karşılaştırması her sorgulama sonunda ayrıca belirtilmiştir. Buna ilaveten 12 sorgulamanın karmaşıklıkları da Tablo 5.7'de yer almaktadır.(*)

(*) RELP dışındaki dillerin sorgulama karmaşıklıkları (Orman,91)'de yer aldığı şekilde verilmiştir.

Parça 15'i sağlayan Üreticiler.
(Suppliers of part 15.)

GENERATED Pno-15.

Tablo 5.7 Seçilen veri tabanına dayalı olarak yapılan 12 sorgulamanın kamaşıklıkları.

SORGULAMA	FDL	RA	SQL	NDL	RELP
Şikagodaki üreticiler	2	4	3	15	3
Üretici 210 tarafından sağlanan parçalar.	3	4	3	29	3
Üretici 210 dışındaki üreticilerden sağlanan parçalar.	3	4	3	31	5
210 veya 206 nolu üretici tarafından sağlanan parçalar.	6	10	5	52	5
210 ve 206 nolu üreticiler tarafından sağlanan parçalar.	6	10	10	54	7
Şikagodan sağlanan parçalar.	6	8	5	31	7
Şikagodan Şikagoya gönderilen parçalar.	10	14	9	31	12
Sağlanan parça tipleri.	5	6	3	16	6
Sağlanmayan parça tipleri.	8	10	17	36	13
Şikago dışındaki şehirlerden sağlanan parçalar.	6	8	5	37	6
Şikagodan , şikago dışındaki şehirlere sağlanan parçalar.	10	14	9	47	12
Üretici 212 tarafından proje 14'e sağlanan A tipindeki parçalar.	8	12	13	40	10

Parça 15'i sağlayan üreticiler.
(Suppliers of part 15.)

```
GENERATEEE _Pno=15;
APPLY _Pno|_pno /R_SUPPLY\ _snum ;
REPORT ..... _snum ;
```

(Toplam Karmaşıklık 3)

DİL KARMAŞIKLIK

FDL :3
SQL :3
RA :4
NDL :24
RELP :3

Şekil 5.7.1.1 Sorgulama örneği-1

Parça 15'in sağlandığı yer.
(The location that supply part 15)

```
GENERATEEE _Pno=15;
APPLY _Pno|_pno /R_SUPPLY\ _snum;
APPLY _snum|_snum /R_SUPPLIER\ _location;
REPORT ..... _location;
```

(Toplam karmaşıklık 6)

DİL KARMAŞIKLIK

FDL :5
SQL :5
RA :8
NDL :24
RELP :6

Şekil 5.7.1.2 Sorgulama örneği-2

Parça 15 ve parça 16'yı birlikte sağlayan üreticiler.
(The suppliers who supplied part 15 and part 16.)

```
GENERATEE _Pno1=15, _Pno2=16 ;
APPLY _Pno1|_pno /R_SUPPLY\ _snum: _Pno2;
APPLY _snum|_snum, _Pno2|_pno /R_SUPPLY\ _snum;
REPORT ..... _snum ;
```

(Toplam karmaşıklık 7)

DİL KARMAŞIKLIK

FDL :6
SQL :10
RA :10
NDL :53
RELP :7

Şekil 5.7.1.3 Sorgulama örneği-3

Parça 15'i sağlamayan üreticiler.
(The suppliers who did not supply part 15)

```
GENERATEE _Pno=15.0;
CARTESIANP R_SUPPLIER \ _snum|_Snum ;
NOTMEMBER _Snum|_snum, _Pno|_pnum / R_SUPPLY;
REPORT ..... _Snum;
```

(Toplam karmaşıklık 4)

DİL KARMAŞIKLIK

FDL :7
SQL :10
RA :8
NDL :34
RELP :5

Şekil 5.7.1.4 Sorgulama örneği-4

Tüm parçaları sağlayan üreticiler.
(The suppliers who supplied all parts.)

```
GENERATE R_PART _pno ;
TXSAY ;
TRANSFER AA,BB;
BB APPLY _pno|_pno /R_SUPPLY\ _snum,_pno;
FILTER _snum,_pno;
COUNTBY _snum;
CONDBLOK AA,BB AA _Cnt== BB _Cnt;
FILTER _snum;
REPORT ..... _snum;
(Toplam karmaşıklık 10)
```

DİL KARMAŞIKLIK

FDL :9
SQL :15
RA :16
NDL :46
RELP :10

Şekil 5.7.1.5 Sorgulama örneği-5

Parça 15'i sağlayıp parça 16'yı sağlamayan üreticiler.
(The Suppliers who supplied part 15 but not part 16)

```
GENERATEE _Pno1=15, _Pno2=16 ;
APPLY _Pno1|_pno/R_SUPPLY\ _snum: _Pno2;
NOTMEMBER _snum|_snum, _Pno2|_pno/R_SUPPLY;
REPORT ..... _snum;
```

(Toplam karmaşıklık 6)

DİL KARMAŞIKLIK

FDL :10
SQL :18
RA :10
NDL :52
RELP :7

Şekil 5.7.1.6 Sorgulama örneği-6

Bütün projelere herhangi bir parçanın aynısından sağlayan üreticiler

(The suppliers who supplied some part to all projects
(The same part to all projects)).

```
GENERATE R_PART _pno;
CARTESIANP R_SUPPLIER \ _snum | _Snum;
APPLY _pno | _pno, _Snum | _snum
      / R_SUPPLY \ _pno, _snum, jnum;
```

```
FILTER _pno, _snum, _jnum;
COUNTBY _pno, _sno;
```

```
TRANSFER BB;
```

```
GENERATE R_PROJECT _jnum;
TXSAY;
```

```
TRANSFER CC;
```

```
CONDBLOK CC, BB CC _Cnt == BB _Cnt;
FILTER _snum;
REPORT ..... _snum;
```

Toplam karmaşıklık 16

DİL KARMAŞIKLIK

```
FDL :13
SQL :21
RA :18
NDL :72
RELP :16
```

Şekil 5.7.1.7 Sorgulama örneği-7

Bütün proje mevkilerine tüm parça tiplerini sağlayan üreticiler. (The suppliers who supplied all part types to all project locations.)

```
GENERATE R_SUPPLIER _snum;
CARTESIANP R_PART \ _pno| _Pno;
CARTESIANP R_PROJECT \ _loc| _Loc;
COUNTBY _snum;
TRANSFER AA,BB;
```

```
BB APPLY _Loc|_loc / R_LOCATION\ _jno: _snum, _Pno;
FILTER _jno, _snum, _Pno;
APPLY _jno|_jno, _snum|_snum, _Pno|_pnum
/R_SUPPLY\ _snum, _pnum, _jno;
COUNTBY _snum;
TRANSFER CC;
```

```
CONDBLOK AA,CC AA _Cnt== CC _Cnt ;
FILTER _snum;
REPORT ..... _snum;
```

Toplam karmaşıklık 21

DİL	KARMAŞIKLIK
FDL	:16
SQL	:27
RA	:28
NDL	:68
RELP	:21

Şekil 5.7.1.8 Sorgulama örneği-8

5.7.2 RELP programlama dili karmaşıklığı

Ele alınan örnek uygulama ve sorgulamaların karmaşıklıkları kullanılarak Sorgulama Büyüklüğünden , Sorgulama Karmaşıklığına bir fonksiyon olarak tanımlanan RELP programlama dili karmaşıklığı incelenmiştir. "X" bağımsız değişkeni, sorgulama büyüklüğünü temsil eden FDL sorgulama programı karmaşıklık değerlerini alacak şekilde, "Y" bağımlı değişkeni ise RELP programlama dilinde yazılmış sorgulamaların karmaşıklık değerlerini alacak şekilde Tablo 5.7.2'de gösterilen (X,Y) veri çiftleri üzerinde "En küçük kareler" metodu kullanılarak doğru uydurma (Least Squares Regression) çalışması yapılmıştır. Bulunan değerler ve doğru denklemi Şekil 5.7.2'dedir.

Tablo 5.7.2 Sorgulama Büyüklüğü-RELP sorgulama karmaşıklığı
X,Y veri çiftleri.

X (FDL KARMAŞIKLIĞI- SORGULAMA BÜYÜKLÜĞÜ)	Y (RELP KARMAŞIKLIĞI)
2	3
3	3
3	5
6	5
6	7
6	7
10	12
5	6
8	13
6	6
10	12
8	10
3	3
5	6
6	7
7	5
9	10
10	7
0	0
13	16
16	21

Bulunan doğrusal korelasyon katsayısı, doğrunun verilere uyumunu gösterecek şekilde, ideal değer olan 1 değerine çok yakın çıkmıştır. Bu durumda seçilen örnek uygulama esas alındığında RELP dilinin

doğrusal programlama dili karmaşıklığına sahip olduğunu söyleyebiliriz. 5.7.2-2'de yer almaktadır.

$$\sum x=142, \sum y=164, \sum x^2=1244, \sum y^2=176$$

$$\sum x.y=1454, x_{ort}=6.761905, y_{ort}=7.809524$$

$$\sum (x-x_{ort}) (y-y_{ort})=345.0476$$

$$\sum (x-x_{ort})^2=283.8095$$

$$\sum (y-y_{ort})^2=479.2381$$

$$n=21$$

En küçük kareler doğrusu :

$$a = \frac{\sum y \sum x^2 - \sum x \sum xy}{n \sum x^2 - (\sum x)^2}$$

$$b = \frac{n \sum xy - \sum x \sum y}{n \sum x^2 - (\sum x)^2}$$

$$a = -0.4265$$

$$b = 1.2107$$

$$y = a + b.x$$

$$y = -0.4265 + 1.2107x$$

Doğrusal Korelasyon Katsayısı :

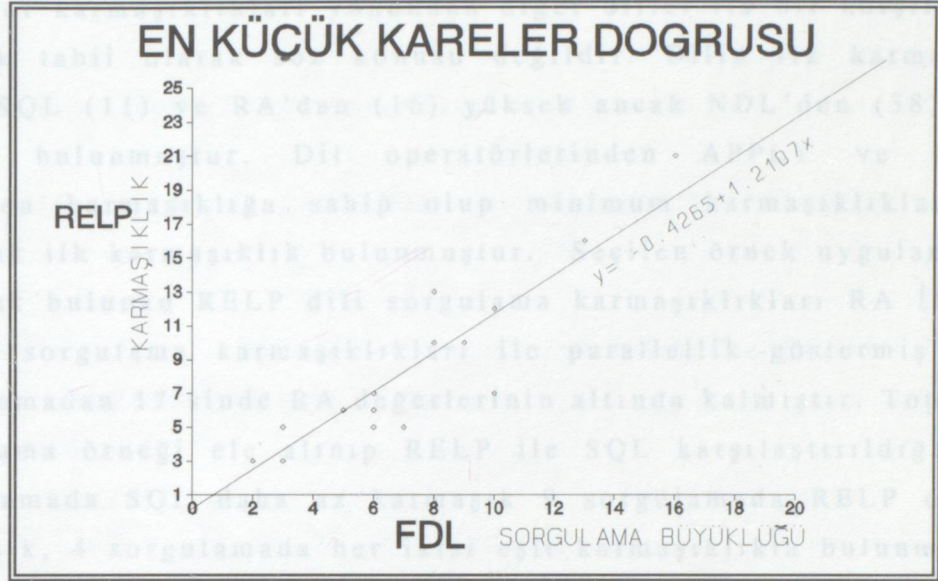
$$r = \frac{\sum (x-x_{ort}) (y-y_{ort})}{\sqrt{\sum (x-x_{ort})^2} \cdot \sqrt{\sum (y-y_{ort})^2}}$$

$$r = \frac{345.0476}{368.7985}$$

$$r = 0.925546$$

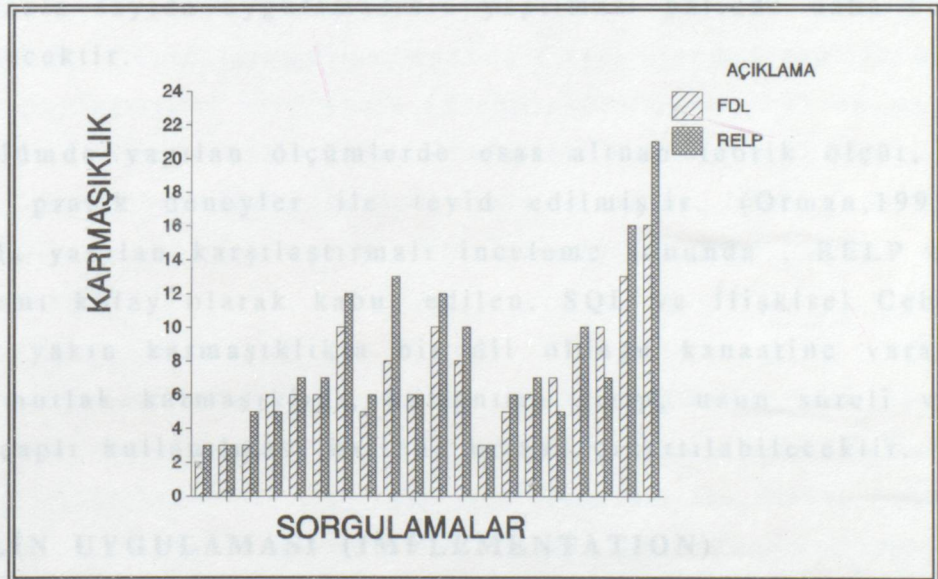
Şekil 5.7.2-1 En küçük kareler doğrusu ve Doğrusal Korelasyon Katsayısının bulunuşu.

En küçük kareler doğrusunun ve verilerin grafik olarak gösterimi Şekil 5.7.2-2'de yer almaktadır.



Şekil 5.7.2-2 En küçük kareler doğrusu ve verilerin dağılımı.

FDL ve RELP karmaşıklıklarının sütun grafik olarak karşılaştırılması Şekil 5.7.2-2'de yer almaktadır.



Şekil 5.7.2-2 Örnek sorgulamalarda FDL ve RELP karmaşıklıkları.

5.7.3 Karşılaştırma sonuçları

Operator karmaşıklıkları yönünden diğer diller ile bir karşılaştırma yapmak tabii olarak söz konusu değildir. Dilin ilk karmaşıklığı (21), SQL (11) ve RA'dan (16) yüksek ancak NDL'den (38) düşük olarak bulunmuştur. Dil operatörlerinden APPLY ve MERGE değişken karmaşıklığa sahip olup minimum karmaşıklıkları esas alınarak ilk karmaşıklık bulunmuştur. Seçilen örnek uygulama esas alınarak bulunan RELP dili sorgulama karmaşıklıkları RA (İlişkisel Cebir) sorgulama karmaşıklıkları ile paralellik göstermiş ve 20 sorgulamadan 17'sinde RA değerlerinin altında kalmıştır. Toplam 20 sorgulama örneği ele alınıp RELP ile SQL karşılaştırıldığında, 7 Sorgulamada SQL daha az karmaşık 9 sorgulamada RELP daha az karmaşık, 4 sorgulamada her ikisi eşit karmaşıklıkta bulunmuştur.

Programlama Dili Karmaşıklığı cinsinden yapılan karşılaştırmada, seçilen örnek uygulama esas alındığında, RELP dilinin referans dil FDL ile lineer karmaşıklık ilişkisinde olduğu sonucuna varılmıştır. Bu husus sadece örnek uygulamaya kısıtlı olup, RELP dilinin çok daha fazla sayıda uygulamasının yapılması halinde daha belirginleşebilecektir.

Bu bölümde yapılan ölçümlerde esas alınan teorik ölçüt, belirli ölçüde pratik deneyler ile teyid edilmiştir. (Orman,1991). Bu aşamada yapılan karşılaştırmalı inceleme sonunda , RELP dilinin, kullanımı kolay olarak kabul edilen, SQL ve İlişkisel Cebir gibi dillere yakın karmaşıklıkta bir dil olduğu kanaatine varabiliriz. Dilin mutlak karmaşıklığı, kullanıma girip, uzun süreli ve daha geniş çaplı kullanılması halinde ortaya çıkartılabilecektir.

6. DİLİN UYGULAMASI (IMPLEMENTATION)

6.1 Genel

Bu bölümde önce dilin işleyiş mekanizması ile ilgili kısaca bilgi

verilecek, daha sonra dilin uygulaması konusunda, tasarım ve uygulama esnasında alınan önemli kararlar ve sebepleri açıklanacaktır.

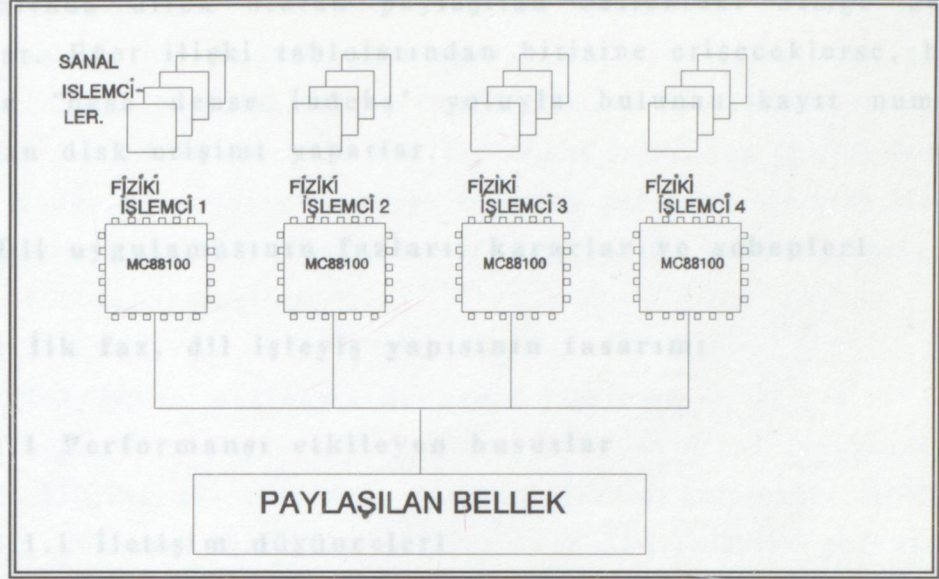
6.1.1 Dilin işleyiş yapısının kısa açıklaması

REL P dili program geliştirme ortamı şu kısımlardan meydana gelmiştir.

1. UNIX System V Release 4 Version 4.0 İşletim Sistemi.
2. REL P Derleyicisi (rr).
3. vi editörü.
4. C derleyicisi (cc).
5. Sanal blok işlemcileri. (Virtual Processors)
6. Veri tabanı kolaylık programları.

Sanal blok işlemcileri, her blok tipi için yazılan programın bir süreç halinde çalışması ile oluşturulurlar. Bu işlemciler, genel olarak, ait oldukları bloğun tipine göre, giriş kuyruğundan Hareket okuyup, gerekiyorsa ilişki tablosuna erişip, çıkış kuyruğuna Hareket gönderirler. Sanal işlemciler mevcut fiziki işlemcilere ilk başlatma anında paylaştırılır ve Uyuma (Sleep) konumunda her an uyandırılmaya ve çalışmaya hazır görev beklerler. Her bir blok tipi için, her fiziki işlemci üzerinde birden fazla o bloğa özgü sanal işlemci bulunabilir. Sanal İşlemci-Fiziki İşlemci ilişkisi şekil 6.1.1'de gösterilmiştir.

Simge deposu, sanal işlemcilere görev atamada veri iletiminde kullanılan veri yapıları ile kuyruklar, paylaşılan bellekte yer alırlar. İlişki tabloları ile ilgili veri yapıları ise diskte bulunurlar. Herhangi bir Sanal işlemci bir ilişki tablosuna erişim yapacak şekilde görev üstlenmiş ise, kullanılan ilişki tablosunun hash indeks yapısı dosya açma aşamasında sanal işlemci tarafından belleğe okunur.



Şekil 6.1.1 Sanal İşlemci-Fiziki İşlemci ilişkisi.

vi editörü ile yazılan RELP kaynak programları, rr derleyicisi ile önce c dili ara kaynak programına, sonra da cc derleyicisi ile hedef bilgisayar makina koduna çevrilir. Bu program, RELP kaynak programındaki talimatlara uygun olarak, sanal işlemcileri seçmek, bu işlemcilere görev dağıtmak ve bu işlemcilere giriş ve çıkış kuyrukları atayarak aralarında iletişimi sağlamak görevini üstlenmiştir.

Bu program çalıştığında, yük paylaştırma kriterleri kullanarak boşta olan sanal işlemcilerden seçim yapar. Bu işlemciler ile ilgili paylaşılan bellekteki veri yapılarına atanacak görev ile ilgili parametreleri işler. Bundan sonra ise bu işlemcilere aktivasyon mesajı gönderir. Mesaj alan sanal işlemciler, uyuma konumundan çalışma konumuna geçerler ve paylaşılan bellekte kendileri ile ilgili veri yapılarına erişip işlem talimatlarını ve parametrelerini alırlar. Bu talimatlara göre giriş kuyruklarından okuma yapıp, gerekiyorsa ilişki tablolarına erişip, çıkış kuyruklarına çıkış vermeye başlarlar. İşlemlerini bitiren sanal işlemciler tekrar uyuma konumuna geçerler.

Sanal işlemciler Simgeleri gerçek veriye çevirme ihtiyacında olduklarında direk olarak paylaşılan bellekteki Simge deposuna ulaşırlar. Eğer ilişki tablolarından birisine erişeceklerse, bellekte bulunan "hash dense indeks" yoluyla bulunan kayıt numarasına doğrudan disk erişimi yaparlar.

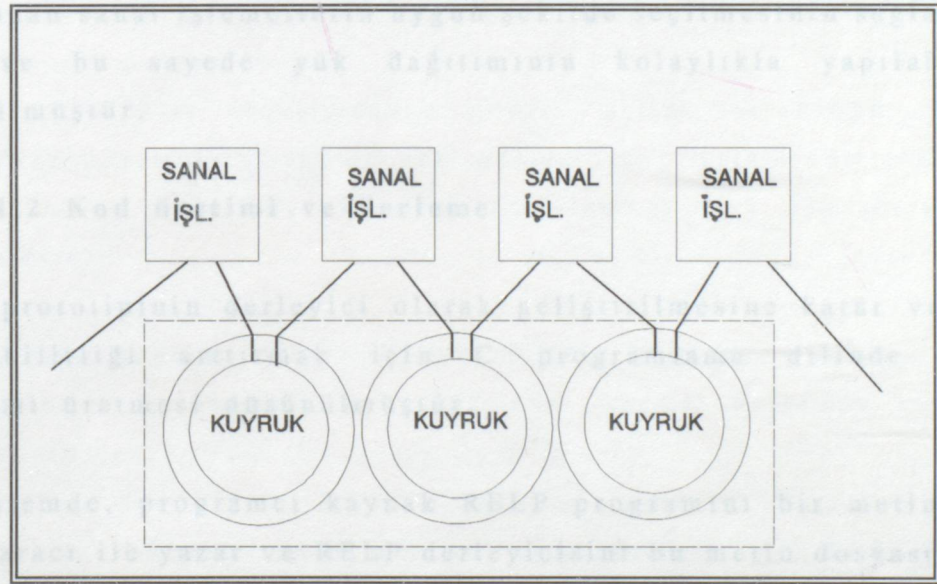
6.1.2 Dil uygulamasının fazları, kararlar ve sebepleri

6.1.2.1 İlk faz, dil işleyiş yapısının tasarımı

6.1.2.1.1 Performansı etkileyen hususlar

6.1.2.1.1.1 İletişim düşünceleri

GPSS benzetim dili ile yapılan benzetimler sonucunda dilin bloklar arasındaki iletişim hızına performans yönünden hassas olduğu ortaya çıkartılmış ve bloklar arasında yapılacak tüm iletişimin paylaşılan bellek (shared memory) mekanizmalarına dayandırılması kararlaştırılmıştır.



Şekil 6.1.2.1.1.1 Kuyruk yapısı.

Yine performansı arttırmak üzere standart UNIX kuyrukları yerine paylaşılan bellekte özel kuyruk uygulaması yapılmıştır. iletişimi hızlandırmak için kuyruk veri yapısı Dairesel İĞİÇ (İlk giren ilk çıkar) (Circular FIFO- First in First Out) olarak tasarlanmış, her kuyrukta tek yönlü iletişim söz konusu olduğundan tek okuyucu, tek yazıcı modelinden yararlanılarak bu yapı üzerinde simafor ile erişim kullanılmamıştır. Bloklar arasında oluşturulan kuyruk yapıları şekil 6.1.2.1.1.1'de gösterilmiştir.

6.1.2.1.1.2 Süreç başlatma-program yükleme

Kişisel bilgisayar üzerinde yapılan ilk uygulamalar sonucunda, program yükleme zamanı maliyetinden kurtulmak gerektiği anlaşılmış, blokların aktif nesne modeline uygun olarak aktif süreçler halinde her an icraya hazır tutulmasına karar verilmiştir.

6.1.2.1.1.3 Sanal işlemci kullanımı ve yük dağıtımı

Sanal işlemcilerin, fiziki işlemciler üzerine yüklenmesi ve bu haritanın ana program tarafından bilinmesi ile ana program tarafından sanal işlemcilerin uygun şekilde seçilmesinin sağlanabileceği ve bu sayede yük dağıtımının kolaylıkla yapılabileceği düşünülmüştür.

6.1.2.1.2 Kod üretimi ve derleme

Dilin prototipinin derleyici olarak geliştirilmesine karar verilmiş. Taşınabilirliği arttırmak için C programlama dilinde kaynak programı üretmesi düşünülmüştür.

Bu sistemde, programcı kaynak RELP programını bir metin geliştirme aracı ile yazar ve RELP derleyicisini bu metin dosyası ismini parametre olarak vererek çalıştırır. Bundan sonra ara kod olarak C programlama dili kaynak programı üretilip, bu program da

otomatikman cc derleyicisi ile derlenerek ana program çalışmaya hazır hale gelir.

6.1.2.1.3 Program çalıştırma-çalışma zamanı ortamı

Sanal işlemciler ve ana programın, paylaşılan bellek üzerinden haberleşmesiyle görev dağıtımının yapılması en hızlı metod olarak düşünülmüştür.

Sanal işlemcilerin iş yapmazken işlemci zamanından kullanmaması ve aynı zamanda her çalışışta tekrar yüklenmelerini engellemek için, ana programdan mesaj bekler (uyku-sleep) konumunda sleep-queue'da beklemesine karar verilmiştir. Seçilen sanal işlemciler süreçler arası mesaj olanağı (Inter Process Communication Message Facility) kullanılarak, ana program tarafından gönderilen mesajlar ile uyandırılırlar.

6.1.2.2 İkinci faz, kişisel bilgisayar uygulaması

Dilin uygulaması tamamiyle aşağıdan yukarı (bottom-up) programlama teknikleri kullanılarak yapılmıştır.

Programlama dili tasarımından sonra, dilin her bloğu kişisel bilgisayar üzerinde Turbo C derleyicisi ile programlanmıştır. Giriş kuyrukları ve çıkış kuyruklarının uygulaması disk dosyaları olarak yapılmıştır.

Bu ortamda yapılan zaman ölçümlerinde, program yükleme zamanı ve bloklar arası iletişimin performansı önemli derecede etkilendiği görülmüştür.

Örneğin kuyrukların hard disk'te uygulanması ile RAM DISK'te uygulanması arasında önemli performans farkı görülmüştür.

Kişisel bilgisayar uygulamasında her blok ile ilgili yazılan C programlarının detaylı test ve hatadan arındırma işlemleri yapılmıştır.

6.1.2.3 Üçüncü faz, UNIX tek işlemcili platform üzerinde uygulama

Bu fazda, kişisel bilgisayarda yazılan C programları UNIX sistemine aktarılmış, derlenip, uyumsuzluklar giderilmiştir.

Paylaşılan bellek iletişim mekanizması oluşturulmuştur. Bu sabit UNIX tabanlı platformda UNIX System V Release 4 (Simetrik Sanal işlemciler tek işlemci üzerinde arka planda (background) süreç olarak çalıştırılarak, tek işlemci üzerinde, süreçler arası iletişim mekanizması denenmiştir.

Bu ortamda süreç yaratma zamanları, iletişim zamanları incelenmiştir.

Bu fazda henüz derleyici geliştirilmemiş, derleyici çıkışında oluşturulacak ana programın benzetimini yapan bir program yazılarak testler yapılmıştır.

6.1.2.4 Dördüncü faz, derleyici geliştirme

Bu fazda, daha önceden söz dizimi ve anlamsal yapısı tasarlanmış dilin yacc (Yet Another Compiler Compiler) ve lex araçları kullanılarak geliştirilmesine başlanmıştır.

Her blok için adım adım derleyici C programlama dilinde yazılmış, örnek programlar derlenmiş, çıkan hatalar düzeltilmiştir.

Bu fazda yacc tarafından bulunan , tutarsızlıklar dilin söz dizimi yapısında düzeltilmiştir.



Derleyicinin geliştirilmesi ile ilgili detaylar, "yacc" ve "lex" tanımları EK-3'de verilmiştir.

6.2.2 Veri yapıları

Derleyicinin çalışması çok sayıda RELP deneysel programları yazılarak test edilmiştir. Bu konuda "PARÇA VERİ TABANI" Örneği (Date,1981) ele alınarak yazılan 26 program derlenip çalıştırılarak test edilmiştir. (Süreç) tablosu.

6.3 Sürüş aktivasyon tablosu

6.1.2.5 Beşinci faz, çoklu işlemci ortamında denemeler

4. Kaynak tablosu.

Bu fazda 4 adet MC88100 RISC işlemciye ve 16MB belleğe sahip UNIX tabanlı platformda UNIX System V Release 4 (Simetrik Çoklu İşleme - Symmetric Multi Processing SMP) İşletim Sistemi üzerinde denemelere başlanmıştır.

SANAL İŞLEMÇİ TABLOSU

Sanal işlemcilerin programları değiştirilerek, her cins sanal işlemciden en az birinin belirli fiziki işlemcilere sabit olarak atanmaları sağlanmıştır.

Daha önceden yazılan programlar bu ortamda çalıştırılarak değişik veri yükleri altında geçen zaman (elapsed time) cinsinden performans ölçümleri yapılmış ve kaydedilmiştir. Yine belirli programlar 1, 2 , 3 ve 4 işlemcide çalıştırılarak sonuçlar alınmıştır.

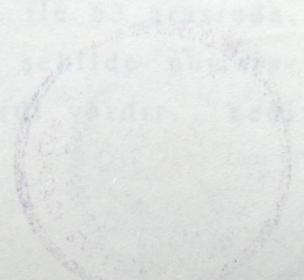
Bu deneylerde elde edilen sonuçlar ve analizleri bölüm 7'de yer almaktadır.

Sekil 6.2.1 Tablolar ve yapıları.

6.2 Dilin işleyiş yapısının detaylı açıklaması

Dilin işleyişi beş ana bölümde açıklanacaktır.

1. Veri Yapıları.
2. İlişki tablosu yapısı ve erişim metodları.
3. Global tablolar.
4. Sanal işlemcilerin genel yapısı.



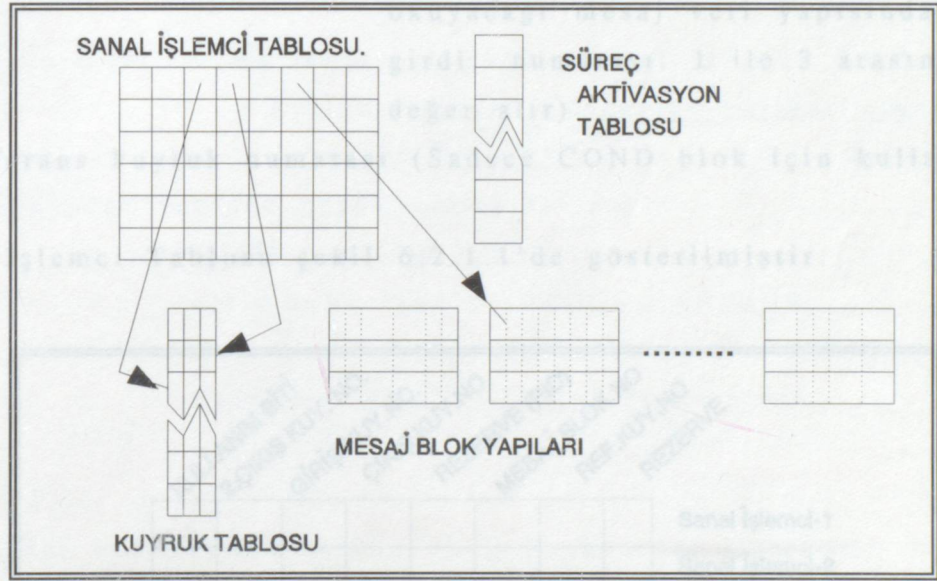
5. Derleyicinin kaynak programı icra dizisine çeviriş yöntemi.

6.2.1 Veri yapıları

Dilin işleyişinde rol oynayan veri yapıları şu şekilde sıralanabilir.

- Sanal İşlemci (Süreç) tablosu.
- Süreç aktivasyon tablosu.
- Her cins blok için ayrı ayrı mesaj bloğu veri yapıları.
- Kuyruk tablosu.
- Global tablolar (Simgede deposu)

Tablolar ve yapıları Şekil 6.2.1' de gösterilmiştir.



Şekil 6.2.1 Tablolar ve yapıları.

6.2.1.1 Sanal işlemci (süreç) tablosu

Her blok cinsi için bu uygulamada üç sanal işlemci çalıştırılmaktadır. Bu durumda blok tipi sayısının üç misli kadar sanal işlemciye (sürece) ihtiyaç olmaktadır. Bu sanal işlemciler 1 ile 63 arasında, aynı cinsten olanlar arka arkaya numara alacak şekilde numaralanmıştır. Her bir işlemci için bu tabloda bir girdi vardır. Yedi

6.2.1.2 Süreç aktivasyon tablosu

Bir sütunluk olan bu tabloda her sanal işlemci için bir bit bulunmaktadır. Sanal işlemciler mesaj ile aktive olsalar dahi bu tablodaki bitleri 1 ise işleme başlarlar 0 ise 1 oluncaya kadar beklerler. Bu bitler ilave eş zamanlama olanağı sağlar.

6.2.1.3 Mesaj bloğu veri yapıları

Her tip blok için bir mesaj veri yapısı bulunur. Üç girdiye sahiptir. Her girdi, bu tipten üç adet sanal işlemcinin birisine dinamik olarak çalışma zamanında atanmaktadır. Her girdide blok tipine göre değişik parametreler bulunur. Bu parametrelerin genel yapısı aşağıdadır..

- a. Kullanım bit'i (Bu mesaj bloğu bir sanal işlemci tarafından kullanılıyorsa 1 değerini kullanılmıyorsa 0 değerini taşır.)
- b. Bloğa göre değişen parametreler (Örnek :
İlişki ismi, (APPLY)
giriş saha ismi , (APPLY, APPEND....)
Yarı kod sayısı. (COMPUTE)
Yarı kod dizisi (COMPUTE,COMFIX,
v.b.)
- c. Hareket uzunluğu. (Bloğun okuyup yazacağı Hareketin sabit uzunluğu.)

Her blok cinsi için bu girdilerin yapısı program içinde "anayapı.h" başlık dosyası içinde detaylı olarak dökümanite edilmiştir ve EK-4'te yer almaktadır.

6.2.1.4 Kuyruk tablosu

Kuyruk tablosunda 200 adet kuyruk için girdi bulunmaktadır. Her

girdide bir kullanım bit'i tutulmaktadır. Eğer kuyruk kullanılıyorsa kullanım bit'i 1 kullanılmıyorsa 0 değerini alır.

6.2.2 İlişki tablosu yapısı

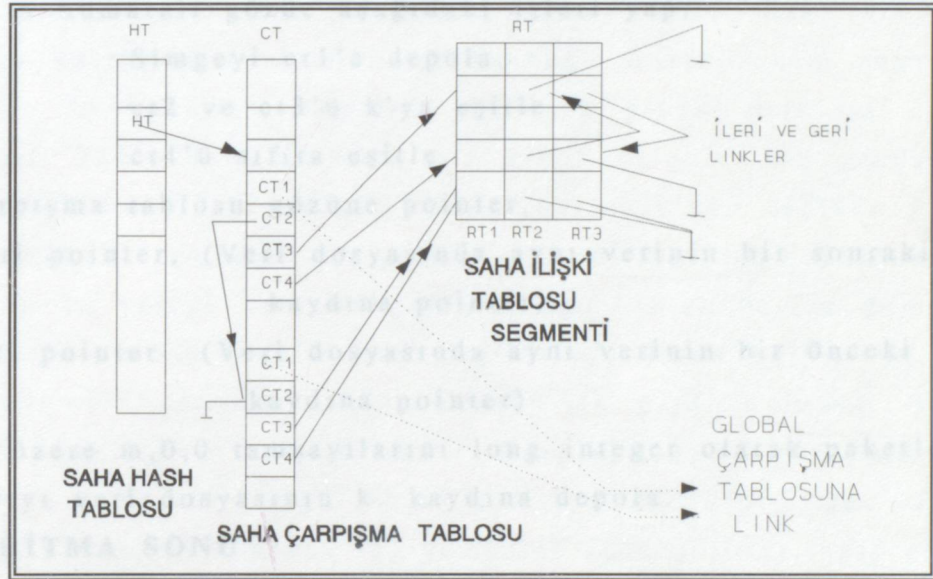
Bir ilişki tablosu her saha için ayrı dosyalar halinde diskte temsil edilmektedir. Her saha üç adet dosya halinde tutulmaktadır. Bunlardan ilki hash tablosunu (ht) , ikincisi çarpışma tablosunu (collision table), üçüncüsü (rt) ise saha içeriğinde yer alan verileri depolamaktadır. Saha dosya isimleri İlişki ismi ile saha isminin birleştirilmesi ve sonlarına tiplerine göre hash tablosu için ".h", çarpışma tablosu için ".c" ve veri dosyası için ".r" ekleri eklenmesi ile oluşturulurlar.

Veri dosyasında her bir veri girdisinde, çarpışma tablosuna tam sayı işaretçi (pointer) ve dosya içindeki aynı verinin depolandığı kayıtlara "ileri" ve "geri" tamsayı işaretçiler tutulur. Bu üç işaretçi tek bir "long integer"a paketlenerek tek bir "long integer" olarak depolanır. İşlemler esnasına bu paket açılır, kullanılır, gerekiyorsa değiştirilir, tekrar "long integer"a paketlenerek yerine yerleştirilir. Böylece 3 yerine 1 long integer kullanılarak girdi başına 2 byte kazanılır.

Herhangi bir Simge ele alındığında ve bu Simgeye hash fonksiyonu uygulandığında bulunan indeks ile hash tablosuna girilir. Hash tablosu girdisi boş ise bu veri daha önce hiç girilmemiş demektir. Simge direk çarpışma tablosuna bir sonraki giriş olarak işlenir, bu çarpışma tablosu indeksi bulunan hash tablosu gözünde saklanır. (Çarpışma olmasa da ilk veri çarpışma tablosuna yerleştirilmektedir. Bunun sebebi hash tablosunun tek tam sayı göz olarak optimize edilmeye çalışılmasıdır. Aksi halde long integer veri tipinde Simge depolanması için her hash tablosu gözünün long tipinde olması gerekecek ve yer israfı olacaktır.)

Çarpışma tablosunun her gözünde, "long integer" tipindeki Simgeyi depolayacak bir göz, veri dosyasında bu verinin temsil edildiği ilk kayıda tam sayı pointer, veri dosyasında bu verinin temsil edildiği son kayıda tam sayı pointer, ve eğer çarpışma varsa, buket içinde yer alan bir sonraki çarpışma gözüne bir tam sayı pointer yer alır. Bağlı listeleri sonlandırmada NULL pointer olarak 0 tamsayısı kullanılır.

Bu veri yapısı şekil 6.2.2 de gösterilmiştir.



Şekil 6.2.2 İlişki sahası veri yapısı.

Aşağıda tek sahadan oluşan bir ilişki tablosuna yeni bir kayıt giriş algoritması verilmiştir.

ALGORITMA BAŞLANGICI

veri dosyasına k. kaydı ekle.

Simgeyi ele alıp hash fonksiyonunu uygula.

Bulunan hash indeks ile hash tablosuna eriş.

Eğer ilgili hash tablosu gözü boş ise ;

çarpışma tablosuna m. girdiyi ekle.

Verini Simgelyi ct1'e depola. Bu gibi bloklarda verinin kendisi
 ct2 ve ct3'ü k'ya eşitle. Simgelyi (Simge) kullanılıp paylaşılan
 bellek ct4'ü sıfıra eşitle. Depozuna erişilerek, veri, bir değişkenden
 Eğer ilgili hash tablosu gözü dolu ve değeri s ise ; Depozunun ve
 Simgeli çarpışma tablosuna m. girdiyi ekle. Bu iki tablodan oluşur.
 Bunlar çarpışma tablosunda s. göze git. İkinci ise "Global Çarpışma
 Tablosu" p=0 pointer değerine rastlanmadıkça yap hash fonksiyonu ile
 Global Hash p = ct4 bu göze için gereken bir indise çevrilir. Bu
 indise bukette en son göze erişildiğinde. Aynı hash girdisine birden
 fazla bulunan gözde ct4'ü m'e eşitle. Çarpışma tablosunda bir-çok
 bağlı m numaralı gözde aşağıdaki işleri yap. Bir sonraki
 girdinin çarpışma Simgelyi ct1'e depola. Aynı nokta tam sayı olarak
 gösterir. Eğer ct2 ve ct3'ü k'ya eşitle. Bu girdi boş ise Global
 Çarpışma Tablosu ct4'ü sıfıra eşitle. Bu girdi okunur. Bu girdiye veri
 m = çarpışma tablosu gözüne pointer, bu çarpışma tablosu girdisine
 0 = ileri pointer, (Veri dosyasında aynı verinin bir sonraki Girdi
 numarası bu veriye kaydına pointer) göze olarak geriye döndürülür.
 0 = geri pointer (Veri dosyasında aynı verinin bir önceki
 kaydına pointer) ile girdi bulunur. Buradan
 olmak üzere m,0,0 tamsayılarını long integer olarak paketlenir.
 bu sayıyı veri dosyasının k. kaydına depola.

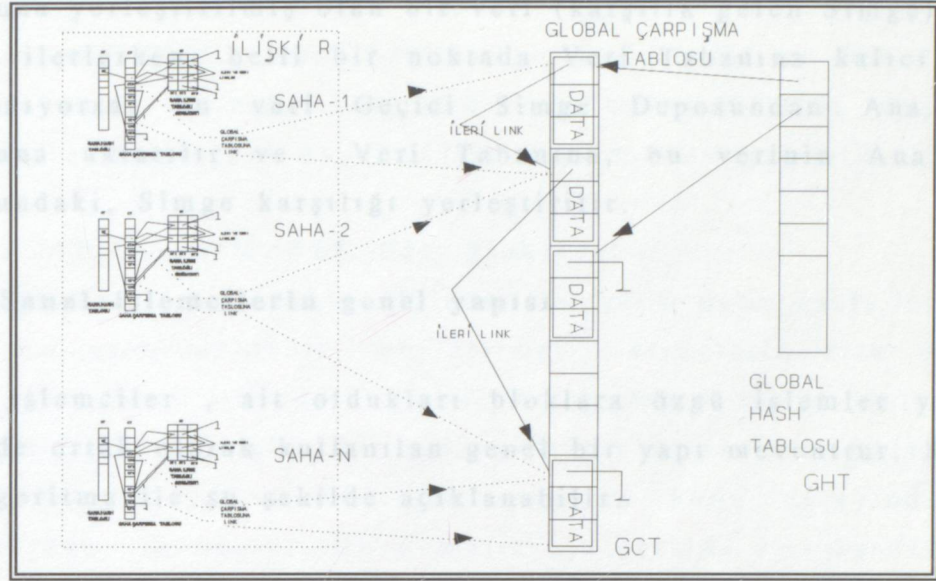
ALGORİTMA SONU

6.2.3 Global tablolar

REL P dili çalışma ortamında, akışa uğratılan verilerin kendisi
 değil, verileri temsil eden Simgelerdir ("Token"). Veri Tabanına
 giren her veriye sadece ve sadece bir Simge karşılık gelir. INPUT
 bloğunda girilen veri Simge Deposu adı verilen global tablolarda
 aranır. Eğer bu veri daha önce girilmiş ve karşılığında bir Simge
 mevcut ise, bu Simge veri yerine sisteme sokulur. Eğer veri daha
 önce girilmemişse bu veriye atanacak Simge atanarak sisteme bu
 yeni Simge sokulur. Bir çok REL P bloğu verinin kendisi ile ilgili
 işlem yapmaz, ancak COMPUTE, SORT, REPORT gibi bloklar,

verinin kendisine ihtiyaç duyarlar. Bu gibi bloklarda verinin kendisi gerektiğinde sadece bir tamsayı işaretçi(Simge) kullanılıp paylaşılan bellekteki Global Simge Deposuna erişilerek, veri, bir değişkenden okunma hızında okunur. Bunu sağlayan Global Simge Deposunun ve Simgelerin yapısıdır. Global Simge Deposu iki tablodan oluşur. Bunlardan ilki "Global Hash Tablosu" ikincisi ise "Global Çarpışma Tablosu" dur. Sisteme giren her veri, bir hash fonksiyonu ile Global Hash Tablosuna giriş için gereken bir indise çevrilir. Bu indis ile Global Hash Tablosuna erişilir. Aynı hash girdisine birden fazla veri karşılık geldiği takdirde çarpışma tablosunda bir tekli bağlı liste oluşturulur. Bu bağlı listede "ileri bağ" bir sonraki girdinin çarpışma tablosunda bulunduğu noktayı tam sayı olarak gösterir. Eğer veriye karşılık bulunan girdi boş ise , Global Çarpışma Tablosunun sonuna bir girdi eklenir. Bu girdiye veri depolanır, Hash tablosu girdisine, yeni çarpışma tablosu girdisine işaret eden bir pointer yerleştirilir. Yeni Çarpışma Tablosu Girdi numarası bu veriye karşılık gelen Simge olarak geriye döndürülür. Eğer hash tablosu girdisi boş değil ise, burada bulunacak pointer takip edilerek çarpışma tablosundaki ilk girdi bulunur. Buradaki veri ile eldeki veri karşılaştırılır. Eğer eşitse, bu çarpışma tablosu girdisinin numarası Simge olarak geriye döndürülür. Eğer eşit değil ise aynı bağda bir sonraki kayıda işaret eden pointer takip edilerek, aynı çarpışma buketi içinde bir sonraki girdi bulunup aynı işlemler yapılır. Bu işleme, ileri pointer NULL olarak bulunana kadar devam edilir. Eğer buketin sonuna gelinmesine rağmen veri bulunamadıysa, yeni bir çarpışma tablosu girdisi tablo sonuna eklenir, bu girdi çarpışma bağına bağlanır ve bu girdinin "ileri" bağına NULL pointer atanarak, bu girdi numarası Simge olarak geriye döndürülür.

Görüldüğü üzere klasik bir hash tablosu veri yapısı içinde veriler depolanmakta ve bu verilerin çarpışma tablosundaki indisleri Simge olarak kullanılmaktadır. Böylece istendiğinde bu verilere Simge değerini kullanarak direk olarak erişmek mümkün olmaktadır. Global tablolar ile ilişki tablolarının ilişkisi şekil 6.2.3'te gösterilmiştir.



Şekil 6.2.3 Global Tablolar ve İlişki Tabloları.

Sisteme giren veriler ancak Veri Tabanında kalıcı olarak depolanırlarsa Global Hash Tablosuna ve Global Çarpışma Tablosuna işlenirler. Sisteme girip veya sistem içinde oluşturulup Veri Tabanı Tablolarına kalıcı olarak depolanmayan veriler ise geçici bir Simge deposunda depolanırlar. Bundan amaç, geçici nitelikteki verilerin ve ara sonuçların Ana Simge Deposunu devamlı olarak işgal etmesini önlemektir. Bu amaçla Global Hash Tablosu ve Global Çarpışma Tablolarının birer benzeri müsvette (scratch) tablolar olarak oluşturulmuştur. Bu tablolar sistem devreye girdiğinde boştur ve sistem çalıştıkça yavaş yavaş dolarlar. Veri Tabanı Yöneticisinin isteği ile veya periyodik olarak bu tablolar boşaltılabilir. Zira bu Simge Deposundaki veriler geçici niteliktedirler. Tabloların yapısı ve kullanılma şekli asıl tabloların yapısı ve kullanılma şekliyle aynıdır. Bir veri INPUT bloğu ile sistem girdiğinde henüz kalıcı olarak depolanıp depolanmayacağı belli değildir. Bu veri Geçici Simge Deposuna yerleştirilir. Bu tip verilere karşılık gelen tamsayı Simgeler -1'den daha küçük negatif tamsayılardır. Ana Simge Deposuna ait olan Simgeler ise pozitif tamsayılardır. Bu şekilde blokların içinde verinin kendisine ihtiyaç duyulduğunda, hangi

Simge deposundan alınacağına karar verilebilmektedir. Geçici Simge Deposuna yerleştirilmiş olan bir veri (karşılık gelen Simge) sistem içinde ilerlerken, belli bir noktada Veri Tabanına kalıcı olarak depolanıyorsa, bu veri Geçici Simge Deposundan Ana Simge Deposuna aktarılır ve Veri Tabanına, bu verinin Ana Simge Deposundaki, Simge karşılığı yerleştirilir.

6.2.4 Sanal işlemcilerin genel yapısı

Sanal işlemciler, ait oldukları bloklara özgü işlemler yapsada, tümünde ortak olarak kullanılan genel bir yapı mevcuttur. Bu yapı bir algoritma ile şu şekilde açıklanabilir.

(Giriş ve çıkış kuyruğu olan standart blok tipi kabullenilmiştir.)

ALGORITMA BAŞLANGICI

- Paylaşılan bellek segmentleri ile bağlantı kurar.
- Süreçler arası mesaj iletişimi olanağı ile bağlantı kurar.
- Kendisine mesaj gelmesini bekler.
- Mesaj geldiğinde paylaşılan bellekte kendisine ayrılan veri yapısından yapacağı işlemler ile ilgili parametreleri ve talimatları okur. Varsa giriş ve çıkış kuyruk numaralarını öğrenir.
- Giriş kuyruğundan akış sonu Hareket birimi (-1) gelinceye kadar Hareket okur. Her Hareket ile ilgili blok tipine özgü işlemler yapar, çıkış Hareketi/Hareketleri yaratır ve çıkış kuyruğuna yazar.
- İşi bittiğinde çıkış kuyruğuna, akış sonu Hareket birimi (-1) yazar.
- Paylaşılan bellekteki veri yapılarına erişerek, kullanım bitlerini sıfırlayarak bu girdileri tekrar kullanıma açar.

ALGORİTMA SONU

6.2.4.1 Sanal işlemci bloklarında yapılan özel uygulamalar

Sanal işlemci bloklarından bazıları yukarıda anlatılan Genel Yapıya ilaveten özel ve daha karmaşık uygulamaya ihtiyaç duyar. Bu bloklar ile ilgili özel uygulamalar aşağıda özetlenmiştir.

"COMPUTE ve COMPUTE FIX Blokları"

Bilindiği üzere bu bloklar içinde bir veya daha fazla aritmetik ifade yer alabilmektedir. Bu aritmetik ifadelerin icra edilmesi görevi ise sanal işlemciye aittir. Bu nedenle bu blokların sanal işlemcileri (Süreç) içine bir yarı kod icra edici "sanal bilgisayar" inşa edilmiştir. Yarı kod seti, makina kodu detayında basit komutlardan oluşmuştur. Bu komutlar şu şekilde sıralanabilir.

- operand1 (Monadic eksi)
- + operand1 operand2 (Toplama)
- operand1 operand2 (Diadic çıkarma)
- * operand1 operand2 (Çarpma)
- / operand1 operand2 (Bölme)

Burada operand olarak belirtilen işleme giren kısımlar, sanal işlemci içine yerleştirilen sanal bilgisayarın blok içi değişkenlere karşılık gelen, işlem göz (register) numaralarıdır.

Derleyici blok içindeki aritmetik ifadeleri, sanal bilgisayarın komut dizilerine, yani bu yarı kod sistemi ile kodlanmış kod dizilerine) çevirir. Derleyici bir işlem yığıt (Stack) veri yapısı kullanarak, aritmetik ifade içinde her karşılaştığı işlemin operand'larını belirleyerek yığıt veri yapısına sokar ve karşılaştığı operatörlerin işlem önceliklerine göre yığıttan çıkartıp düzenleyerek, ard arda icra ettiğinde aritmetik ifadede belirtilen işlemleri yapan komut dizisini üretir.

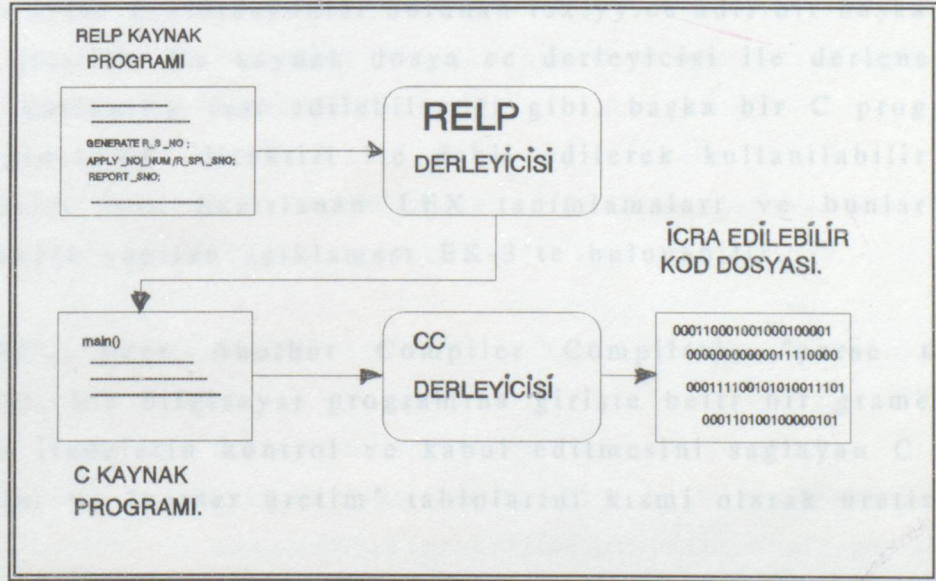
Bu komut dizileri, Sanal İşlemciye çalışma zamanında, ana program tarafından iletilir. Bu komut dizilerini alan sanal işlemci, giriş

kuyruğundan her yeni Hareket birimi okunduğunda, oluşan veri konfigürasyonu üzerinde bu komut dizisini çalıştırarak hesap yapar, Blok içinde otomatik bildirilmiş değişkenler, giriş Hareket değişkenlerinin blok içi değişkenlerine yansımış halhalleri, Kalıcı ilişki tablosu girdileri hesaplamada aritmetik ifadeler içinde temsil edildikleri isimler yoluyla hesaplamaya girebilirler. Yeni oluşan veriler aşağıdaki veri gözlerine depolanabilirler.

1. Blok içi değişkenlere karşılık gelen, blok içi geçici gözlerle.
2. Eğer "Compute Fix" Bloğu söz konusu ise ilişki tablolarının seçilmiş kayıtlarının seçilmiş sahalarına. Bu blokların çıkış grubu içinde ismi geçen blok değişkenlerinin değerleri, çıkış Hareket birimi vasıtasıyla çıkış kuyruğuna yollanır.

6.2.5 Derleyicinin kaynak programı icra dizisine çeviriş yöntemi

Derleyici, RELP kaynak programını okuyarak, her bir blok için bloğun işlevine uygun C kaynak program deyimlerini üretir. Bu C kaynak programı cc derleyicisi ile derlendiğinde icra edilebilir program elde edilir. Dilin derleme akışı şekil 6.2.5'de gösterilmiştir.



Şekil 6.2.5 Derleme akışı.

Derleyici tek geçište (Single Pass) biçimbirimsel çözümlemeyi (Lexical Analysis), Sözdizimsel çözümlemeyi (Syntax Analysis), Anlamsal çözümlemeyi (Semantic Analysis) ve kod üretimini (Code Generation) yapar.

REL P derleyicisinin yazılmasında aşağıdaki yol izlenmiştir:

1. LEX biçimbirimsel çözümleyici üreticisi, çağırıldığında girişden karakter karakter bir metin parçası okuyarak, bu metin parçası daha önceden belirlenerek kendi bünyesine işlenmiş etiketlerden (lexical token) birine uyduğunda bu etiket numarasını geri veren yylex() adlı bir C fonksiyonu üretir. LEX üreticisine yönelik giriş tanımlamaları, etiketler, bu etiketlerden herhangi biri tanındığında yapılacak ilk işlemlerin C programlama dilinde ifadesinden oluşan aksiyonlar ve gereken diğer C programlama dili direktiflerinden oluşur. Bu tanımlar bir kaynak dosyaya yazıldıktan sonra, UNIX sisteminde,

\$lex <kaynak dosya adı>

komutu verilir. Bu işlemten sonra içinde yylex() fonksiyonu ve gerekli diğer deklarasyonlar bulunan lex.yy.cc adlı bir başka kaynak dosya üretilir. Bu kaynak dosya cc derleyicisi ile derlenerek tek başına kullanılıp test edilebileceği gibi, başka bir C programının içine **#include** direktifi ile dahil edilerek kullanılabilir. REL P derleyicisi için hazırlanan LEX tanımlamaları ve bunların yarı kodlamayla yapılan açıklaması EK-3'te bulunabilir.

2. YACC (Yet Another Compiler Compiler), "parse tablosu" üreticisi, bir bilgisayar programına girişte belli bir gramere göre yazılan ifadelerin kontrol ve kabul edilmesini sağlayan C fonksiyonlarını ve "parser üretim" tablolarını kısmi olarak üretir.

YACC üreticisine verilecek giriş dosyasında bu gramer kuralları

YACC tarafından beklenen BNF formuna benzer bir formda verilmelidir. Bu kurallara uyan etiket kombinasyonları tanındığında yapılacak işlemler (aksiyonlar) C programlama dilinde yazılan program parçaları halinde bu dosyada gerekli noktalarda belirtilmelidir. Bu tanımlar hazırlandığında UNIX de;

```
$yacc <kaynak dosya adı>
```

Komutu verilerek üretimin yapılması sağlanmalıdır. YACC, yyparse() adlı bir C fonsiyonunu ve "Left Lookahead Shift Reduce" bir parser için gerekli konum tablolarını içeren y.tab.cc adlı bir C kaynak programını üretir. "yyparse()" adlı fonsiyon, ya kullanıcı tarafından yazılan ya da LEX yardımı ile üretilen "yylex()" adlı fonsiyonu çağırarak, girişten, hatırlanan bir sonraki etiketi alır ve bu etiket akışının kurallara uyumunda konum tabloları aracılığıyla kontrol eder. bu akışta kurallara bir uyumsuzluk söz konusu olduğunda "yyerror()" adlı bir fonsiyon ve kullanıcının sağlayacağı diğer fonsiyonlar çağırılarak işlem yapılır. Yacc tarafından üretilen program bir derleyicinin "parser" kademesini oluşturmada kullanılabilir. Bunun yapılabilmesi için derleyici yazarı tarafından YACC'ın sağladığı programın ilk şekli üzerine çok miktarda C program parçası ve C fonsiyonu ilave etmesi, bunların bir kuralın etiket akışıyla incelenmesi esnasında tam zamanında devreye girip çıkmalarının sağlanması gerekmektedir.

REL P derleyicisi için hazırlanan YACC tanımı ve bu kurallara ilave olarak yazılan C programlarının açıklamaları EK-3'tedir.

6.2.5.1 REL P Derleyicisinin LEX ve YACC kullanılarak oluşturulması

Derleyicinin oluşturulmasında önce REL P biçimbirimsel etiketleri tesbit edilmiş ve hangi karakter dizilerinin etiket olarak alınacağı hangilerinin YACC tarafından üretilen programın incelemesine

birakılacağına karar verilmiştir. LEX giriş tanımları buna göre hazırlanarak, LEX vasıtasıyla deneysel bir "etiket hatırlayıcı-biçimbirimsel çözümleyici" program üretilmiş ve test edilmiştir.

Bir sonraki kademede RELP dilinin tasarlanmış olan grameri YACC giriş tanımları halinde hazırlanmıştır. YACC tarafından bu tanımlardan üretilen program ile LEX tarafından üretilen program birleştirilerek, bir RELP "Söz Dizimi Kuralı Hatırlayıcı veya Sözdizimsel Çözümleyici" hazırlanmıştır.

Bu kademeye kadar söz dizimi kurallarındaki uyumsuzluklar nedeniyle YACC tarafından üretilen "Shift/Reduce" ve "Reduce/Reduce" hataları, (yacc -v <kaynak dosya adı>) komutuyla y.output dosyası üretilip tek tek incelenmiş ve "yyparse()"ın takip edeceği yollar tesbit edilmiştir. Bu hataların giderilmesi için bazı gramer kuralları, RELP orjinal tanımları değiştirilmeden ilave kurallarla geliştirilmiş ve "yyparse()"ın çözümleme esnasında düşebileceği çatışmalar (parsing conflicts) önlenmiştir. Gerek LEX'in YACC ile uyum içinde Hareket etmesinin sağlanması gerek se yukarıda bahsedilen çatışmaların önlenmesi için çok fazla zaman harcanmıştır. Ancak bir programlama dilinin derleyicisi ne şekilde yazılırsa yazılsın, önce bu tip bir çalışmayla kullanılacak söz dizimi kurallarının test edilmesinin faydalı olacağı kanaatine varılmıştır.

Bu aşama geçildikten sonra LEX aksiyonları ve YACC aksiyonları C programlam dilinde yazılarak tanımlar içinde devreye girmeleri gereken noktalara dahil edilmiş, her eklenen aksiyonun anılan kural hatırlandığında devreye girerek gerekli işlemleri yapıp yapmadığı test edilmiştir. Bu aksiyonlardan bazıları, hedef olan C dilindeki programın üretiminde, derleyici tarafından yapılan işlemlerin düzenlenmesi, sembol tablosu yönetimi, değişkenlerin geçerli olduğu aralıkların ayırt edilmesi gibi, bazıları ise hedef programın anlamlı ve doğru bir C programı metni olarak üretilmesine yönelik yazılmıştır. Sonuç olarak bu derleyicinin Anlamsal Analiz

("Semantic Analyzer") ve Kod Üretim ("Code Generator") bölümleri "Parser" bölümü içine bütünleştirilmiştir. Derleyicinin yukarıda belirtilen anlamsal aksiyonları yapabilmesi için belirli bir T anına kadar Parser tarafından hatırlanan gramer kuralları (geçilen yol/tarihçe) belli mantıksal anahtarların 1 veya 0 konumuna getirilmesiyle belirlenmiş ve bu anda yapılan işlemin şekli bu mantıksal anahtarların konumuna göre belirlenmiştir. Yacc tarafından üretilen tablo içeriği ile yönlendirilen sözdizimsel çözümleyici, kuralların hatırlanması esnasında belli noktalardaki ileri bakışındaki (Look Ahead) değişimler için özel tedbirler alınmış ve bunlar için özel aksiyonlar yazılmıştır.

Üretilen programın sonuç olarak, kaynak RELP programının bire bir karşılığı olmaktan ziyade, "kaynak programın kabullendiği paralel bilgisayar mimarisindeki sanal işlemcilerle yönelik işlemler yapan bir çalışma zamanı icracısı" olması, ve bu modelin üretilen programa uygulanması, anlamsal yapıyı daha karmaşık hale getirmiştir. Aynı bağlamda, dinamik sanal işlemci ve kuyruk atama özelliğinin de hedef C programı içine bütünleştirilmesi gerekmektedir. Bu maksatla çok miktarda iletişim amaçlı değişken kullanılmış ve mantıksal deyimlerle takip edilmiştir.

Sembol tablosu işlemlerinde dağıtım kodlaması ("hashing") tekniği uygulanmış ve değişkenlerin tipleri ile tanımlı oldukları aralıklar (kuyruk tanım ortamları) takip edilmiştir. Derleyici yanlış kullanım ve bildirimlerde yeterli ikazları ve hata mesajlarını vermektedir.

6.2.5.2 Derleyicide kullanılan önemli veri yapıları ve değişkenler
Derleyicide aşağıdaki veri yapıları yanlarında belirtilen amaçlar için kullanılmıştır.

İlişki İsmi (rngond): Lex tarafından okunan ilişki ismini Yacc içine taşımak.

Değişken İsmi (vrgond): Lex tarafından okunan değişken ismini Yacc içine taşımak.

Saha ismi (atgond): Lex tarafından okunan saha ismini Yacc içine taşımak.

Karakter dizin ismi (stgond): Lex tarafından okunan karakter dizisi sabitini Yacc içine taşımak.

Etiket ismi (lbgond) : Lex tarafından okunan etiket ismini (label) Yacc içine taşımak.

bloknm : Kaynak programda ilerleme esnasında, halen üzerinde çalışma yapılan blok numarasını depolar.

ikuyrnm : O anda üzerinde çalışma yapılan bloğun, derleyici tarafından takip edilen giriş kuyruk ortam (Context) numarası.

okuyrnmD : O anda üzerinde çalışma yapılan bloğun, derleyici tarafından takip edilen çıkış kuyruk ortam (Context) numarası.

sontl : En son üzerinde çalışmanın bitirilmiş olduğu bloğun, çıkış kuyruğu için geçerli olan Hareket birimi uzunluğu.

blokm : Blok ilişkilendirme veri yapısı. Her blok için bir satır mevcuttur. Her satırın birinci girdisinde, söz konusu bloğun kaç numaralı bloktan giriş yaptığı, girişin okunduğu kuyruk numarası, takip eden her girdide ise kaç numaralı bloğa çıkış yaptığı ve hangi kuyruk numarası üzerinden çıkış yaptığı yer alır. (Bazı bloklar birden fazla bloğa çıkış yapabilirler)

kuyuz : Her kuyruk için bir girdiden oluşan bir veri yapısıdır.

Her girdide derleyici tarafından bu kuyruk için belirlenen Hareket genişliği (Transaction Lenght) depolanır.

bloktip : Her blok için bir girdiden oluşan bir veri yapısıdır. Her girdide ait olduğu bloğun tipi, derleyici tarafından depolanır.

nlist : Sembol tablosu dağıtım kodlama (hash) yapısında, her değişken ismi veya isim için tutulan girdidir. Aşağıdaki elemanlara sahiptir.

- Bir sonraki girdiye işaretçi.
- tip (Sayısal-n, karakter-c)
- sıra (Bulunduğu kuyruk ortamında kaçınıcı değişken)
- Kuyruk ortam numarası. (Değişkenin geçerli olduğu kuyruk ortam numarası.)
- Etiket numarası (Etiketler sembol tablosuna işlendiğinde kullanılır.)

hashtab : Sembol tablosu, dağıtım kodlama (hash) veri yapısı.

6.2.5.3 Derleyici algoritması

Bu bölümde mümkün olduğunca genellemeler yaparak derleyicinin genel algoritması verilmektedir.

ALGORİTMA BAŞI

Başla,

Her karşılaşılan blok için,

Blok Anahtar kelimesini al ve bloğun tipini belirle.

- 1 Bu blok etiket ile markalanmış ise, bu etiketi referanslıyarak çıkış yapan blok numarasını, etiket ile markalanmamış ise bir önceki bloğun numarasın giriş yapılan blok numarası olarak blokmp yapısına işle.
- 2 Halen geçerli giriş kuyruk ortam numarasını (ikuyrnm) blokmp yapısına giriş yapılan kuyruk numarası olarak işle.
- 3 Eğer blok, standart olarak bir sonraki bloğa çıkış veren bir

- blok ise, blokmp yapısının bir sonraki girdisine bir sonraki blok numarasını çıkışın gönderildiği blok numarası ve halen geçerli çıkış kuyruk ortam yapısını (okuyrnm) göndermenin yapıldığı kuyruk numarası olarak işle.
- 4 Blok parametresi olarak okunacak girdilerin sayımında kullanılacak sayaçlara başlangıç değerlerini ver.
 - 5 Daha sonra girilecek, diğer blok tipleri ile ortak kullanılan gramer yapılarında, bu bloğa özgü işlemler yapmada ayırım için kullanılacak "toggle" değişkenine, bu blok için geçerli harfi ata.
 - 6 Blok parametresi olarak verilen , (blok tipine göre değişen) ilişki ismi, saha ismi, değişken ismi gibi parametreleri oku. Çıkışta kullanılacaksa , bu parametrelerin sayılarını sayaçları kullanarak tesbit et.
 - 7 Bloğa giriş parametresi olarak yer alan giriş Hareket değişken isimlerini, giriş kuyruk ortam numarasını esas alarak, giriş kuyruk ortamında (context) sembol tablosunda ara. Bulunamazsa hata mesajı ver.
 - 8 Çıkışta oluşturulan C kaynak programı içine include direktifi ile dahil edilecek olan, bloğa özgü sabit C kaynak program parçaları için gerekli include direktiflerini, çıkışta oluşturulan C kaynak programı içine yerleştir.
 - 9 Blok gövdesi ile ilgili kaynak programda belirtilen ifadeleri okuyup gramer kurallarından kurallarına atlayarak pars ağacını oluştururken, her pars ağacı dalında çıkışa yazılması gereken C dilindeki deyimleri yaz. C dilindeki bu deyimler, mesaj blokları aracılığıyla sanal işlemcilere yapacakları işlemler için gerekli, ilişki ismi, saha ismi, giriş/çıkış Hareket uzunlukları, yarı kod komutları(pseudo instructions) gibi girdileri sağlamaktadır.
 - 10 Bu adımlar esnasına karşılaşılan çıkış Hareket değişkenlerini, çıkış kuyruk ortamı numarasını (okuyrnm) kullanarak , çıkış ortamına (context) sembol tablosuna işle.

- 11 Eğer blok, bir sonraki bloğa gönderi yapan tipte değil de, etiketler kullanarak, bu etiketler ile markalanmış bloklara gönderi yapan türde ise, ve bu adımlar esnasında bu anlamda etiketler ile karşılaşılırsa, blokmp yapısının çıkış ile ilgili girdilerini "etiket yer doldurucu" sayacından alınacak değerleri yerleştir.
- 12 Eğer blok giriş kuyruk ortamını çıkış kuyruk ortamına aynen geçiren tipte değilse, Bu adımlar esnasında çıkış Hareket genişliği olarak tesbit edilen sayıyı sontl (son Hareket uzunluğu) adlı değişkene depola, kullanılagelen çıkış kuyruk numarasını (okuyrnm), giriş kuyruk numarası olarak (ikuyrnm) kullanıma sok. (ikuyrnm=okuyrnm). Bu numarayı kullanarak, kuyuz yapısının ikuyrnm numaralı girdisine eriş, ve son Hareket uzunluğunu bu girdiye işle. Çıkış kuyruk numarasını bir arttır. (Giriş kuyruk ortamını aynen çıkış kuyruğuna geçiren tipteki bloklar bu değişkenleri değiştirmeden bırakırlar.) program sonu karakteri ile karşılaşıldıysa dur. Bir sonraki blok anahtar kelimesi ile karşılaşıldıysa başa dön.

ALGORİTMA SONU

Örnek :

Aşağıda "APPLY" bloğuna ait bu algoritmaya uygun olarak yazılmış, derleyicinin Yacc tanım metnindeki hali, yer almaktadır.

applybaslik : TAPPLY

```
{
    blokmp[bloknm][0]=bloknm-1;
    blokmp[bloknm][1]=ikuyrnm;
    /* BİR ÖNCEKİ BLOKTAN GİRİŞ
       YAPILIRSA BLOKMP YAPISINA BİR
       ÖNCEKİ BLOK NO VE GİRİŞ KUYRUK
       NUMARASI İŞLENİR */
    | genellabel
    /* GENELLABEL KURALINA GİDİLDİĞİNDE
       YAPILAN İŞLEMLER AŞAĞIDA BU
       KURALIN AÇIKLAMASINDA YER ALMAKTADIR */
```



```

TAPPLY :intf("\n Ikaz : aplikasyon saha sayisi ile giris \n");
        printf("\n /* APPLY ANAHTAR KELİMESİ LEX'DEN
        ALINDI */ ;

applyblok : applybaslik {
        blokmp[bloknm][2]=bloknm+1;
        blokmp[bloknm][3]=okuyrnm;
        /* BU BLOK STANDART OLARAK BİR
        SONRAKİ BLOĞA ÇIKIŞ VERİR , BUNA
        GÖRE BİR SONRAKİ BLOĞUN
        NUMARASI VE ÇIKIŞ KUYRUK NUMARASI
        BLOKNM
        YAPISININ BİR SONRAKİ GÖZÜNE
        YERLEŞTİRİLİR */
        bloktip[bloknm]=4;
        /* BU BLOĞUN TİPİ 4 */
        l=1; /* çıkış girdi sayacı */
        n=1; /* bypass var atr sayacı (10 dan büyük
        olmaz) */
        gas=1; /* giriş Hareket değişken sayacı */
        cas=1; /* çıkış saha sayacı */
        togle='a';
        /* GİDİLECEK KURALLARDA TOGGLE
        DEĞERİ KONTROL EDİLİP "A" OLDUĞU
        GÖRÜLDÜĞÜNDE BU BLOK İŞLEMİNİN
        YAPILMAKTA OLDUĞU
        ANLAŞILACAKTIR*/
        fprintf(cp,"#include </usr/ccs/bin/appinc.c>\n");
        /* BU BLOĞA ÖZGÜ SABİT C PROGRAM
        PARÇASI APPINC.C ÇIKIŞTAKİ C
        KAYNAK PROGRAMINA EKLENDİ */
    }
    giratrlst TBOLOP
    { fprintf(cp,"\n adrp->applygas[yer]=%d ;\n",gas );
    if (gas>kuyuz[ikuyrnm]){

```



```

printf("\n Ikaz : applikasyon saha sayisi ile giris \n");
printf("\n kuyruk uzunlugu uyusmuyor \n");
}
fprintf(cp,"\n adrp->applytlg[yer]=%d ;\n",kuyuz[ikuyrnm]);
}
RELNAME
{fprintf(cp,"\nsprintf(adrp->applyrel[yer],\"%s\",
        \"%s\");\n",formt,rgond);
}
TERSBOL {l=1;n=1;} cikis
TNOKVIRG {
    fprintf(cp,"\n adrp->applycas[yer]=%d ; \n",cas);
    fprintf(cp,"\n adrp->applytlc[yer]=%d ; \n",l);
    /* MESAJ BLOĞU ÜZERİNDEN SANAL APPLY
       İŞLEMCİSİNE ÇIKIŞ SAHA SAYISINI cas
       DEĞİŞKENİNDE BELİRTİLEN DEĞER,
       ÇIKIŞ HAREKET GENİŞLİĞİNİ İSE l
       DEĞİŞKENİNDEKİ DEĞER OLARAK İLETECEK C
       KAYNAK PROGRAM DEYİMLERİ ÇIKIŞTA
       OLUŞTURULAN C KAYNAK PROGRAMINA EKLENDİ.
       */
    ikuyrnm=okuyrnm;
    sontl=1;
    kuyuz[ikuyrnm]=sontl;
    okuyrnm+=1;
    /* ÇIKIŞ ORTAMI YARATILDI */
};
giratrlst : giratrent | giratrlst TVIRG {gas+=1;} giratrent;
giratrent : VARNAME
    { if( (np=bul(vrgond,ikuyrnm))==NULL) {
        printf("\n Degisken sembol tablosunda yok
        \n");
        exit(-1);
    } }

```



```

/* GİRİŞ Hareket DEĞİŞKENİ İSMİ
ALGILANDIĞINDA BU DEĞİŞKEN İSMİNİ
SEMBOL TABLOSUNDA GİRİŞ KUYRUK ORTAMI
BAZINDA ARA, BULUNAMAZSA HATA */
TPIPE
ATRNAME
{
    if (toggle=='a'){
        fprintf(cp, "\nsprintf(adrp->applyga[yer][%d],
        fprintf(cp, \"%s\\\", \"%s\\\")\n\", gasl, format, atgond);
    /* EĞER BU NOKTAYA APPLY BLOĞUNDAN
    GELİNDİYSE LEX'DEN OKUNAN GİRİŞ SAHA
    İSMİ, applyga MESAJ BLOK GİRDİSİ
    ÜZERİNDEN SANAL APPLY İŞLEMCİSİNE
    GÖNDERİLECEK ŞEKİLDE, UYGUN
    C DEYİMİ YARATIP ÇIKIŞTAKİ C KAYNAK
    PROGRAMINA İŞLE */
    }
    if (toggle=='m'){
        fprintf(cp, "\nsprintf(adrp->memga[yer][%d],
        fprintf(cp, \"%s\\\", \"%s\\\")\n\", gas-1, format, atgond);
        install(atgond, np->tip, np->sira, ikuyrnm, 0);
    }
    /* member yok girilecek hareketi eleman değilse aynıan dışarı
    çıkarılır */
    |
    ATRNAME
    {
        if( (np=bul(atgond, ikuyrnm))==NULL) {
            printf("\n Degisken sembol tablosunda yok
            \n");
            exit(-1); }
    }
}

```



```

/* GİRİŞ Hareket DEĞİŞKENİ İSMİ
   ALGILANDIĞINDA BU DEĞİŞKEN İSMİNİ
   SEMBOL TABLOSUNDA GİRİŞ KUYRUK ORTAMI
   BAZINDA ARA, BULUNAMAZSA HATA */

```

```

TPIPE ip=bul(vergond,ikuyrm)==NULL; {
    printf("a Degisten sembol tablosunda yok

```

```

ATRNAME

```

```

{
    if (togle=='a'){
        fprintf(cp,"\nsprintf(adrp->applyga[yer][%d],
            \"%s\\\", \"%s\\\")\n\",gasl,formt,atgond);
        /* EĞER BU NOKTAYA APPLY BLOĞUNDAN
           GELİNDİYSE LEX'DEN OKUNAN GİRİŞ SAHA
           İSMİ,applyga MESAJ BLOK GİRDİSİ
           ÜZERİNDEN SANAL APPLY İŞLEMCİSİNE
           GÖNDERİLECEK ŞEKİLDE, UYGUN
           C DEYİMİ YARATIP ÇIKIŞTAKİ C KAYNAK
           PROGRAMINA İŞLE */

```

```

    }

```

```

    if (togle=='m'){
        fprintf(cp,"\nsprintf(adrp->memga[yer][%d],
            \"%s\\\", \"%s\\\")\n\",gasl,formt,atgond);
        install(atgond,np->tip,np->sira,ikuyrm,0);

```

```

/* member blok giristeki Hareketi eleman degilse aynen disari
   cikarir */

```

```

    }

```

```

;

```

```

cikis : cikatrlist
        | cikatrlist TINOKTA cikvarlist;

```

```

cikvarlist: VARNAME

```

```

    {l+=1;

```

```

    n+=1;

```



```

if (n>10) {
    printf("\n bypass edilen degisken ve saha
    sayisi 10 dan buyuk olamaz \n");
    exit(-1);
}
if( (np=bul(vrgond,ikuyrnm))==NULL) {
    printf("\n Degisken sembol tablosunda yok
    \n");
    exit(-1);
}
/* ÇIKIŞA GÖNDERİLEN DEĞİŞKEN GİRİŞTE
YOKSA HATA */
install(vrgond,np->tip,l,okuyrnm,0);
/* GİRİŞTE VARSA ÇIKIŞ KUYRUK ORTAMINDA
(CONTEXT) SEMBOL TABLOSUNA İŞLE */
fprintf(cp,"\n adrp->applytxix[yer][ %d][0]=%d;\n",
        n-1,np->sira);
fprintf(cp,"\n adrp->applytxix[yer][ %d][1]=%d;
        \n",n-1,1);
/* APPLYTXIX GİRDİSİ ÜZERİNDEN ÇIKIŞ
HAREKET DEĞİŞKENLERİNİN HAREKET
İÇİNDEKİ GÖZ NUMARALARINI SANAL
APPLY İŞLEMCİSİNE ULAŞTIRACAK
ŞEKİLDE HAZIRLANMIŞ C DEYİMLERİNİ
ÇIKIŞTA OLUŞTURULAN C KAYNAK
PROGRAMINA İŞLE */
}

cikatrlist :  ATRNAME
{
    install(atgond,'c',l,okuyrnm,0);
    fprintf(cp,"\nsprintf(adrp->applyca[yer][ %d],\"%s\\\", \"%s\\");\n",cas-
    1,formt,atgond);

```



```

/* ÇIKIŞ SAHA İSİMLERİNİ ÇIKIŞ KUYRUK
   ORTAMINA (CONTEX) İŞLE. */
}
| cikatrlist TVIRG {cas+=1;l+=1;} ATRNAME
{
    install(atgond,'c',l,okuyrm,0);
    fprintf(cp,"\nsprintf(adrp->applyca[yer][%d],
        \"%s\",\"%s\");\n",casl,formt,atgond);
}

;
genellabel: TLABEL
{
    if( (np=bul(lbgond,-1))==NULL) {
        printf("\n Label sembol tablosunda yok
            \n");
        exit(-1);
    }
    /* -1 İLE NUMARALANMIŞ İSİM ORTAMI
       ETİKETLER İSİM ORTAMIDIR*/
    /* ETİKET SEMBOL TABLOSUNDA -1
       NUMARALI ETİKETLER TANIM
       ORTAMINDA (CONTEXT) ARANIP
       BULUNDU TANIMLI OLDUĞU ANLAŞILDI
       */
    blokmp[bloknm][0]=np->sira;
    cikbloki=2;
    /* TRANSFER BLOĞUNUN ETİKET İSİM YERİ
       TUTUCU İLE DOLDURDUĞU (O AN BELLİ
       OLMAYAN) ÇIKIŞ BLOK GÖZLERİNİ
       İNCELE VE BU GÖZLERDEN İLK
       RASTLADIĞIN FİLLER İLE DOLU
       OLANA KENDİ BLOK NUMARANI
       DOLDUR */

```



```

printf("\n blokmp=%d \n", blokmp[np->sira][cikbloki]);
while(blokmp[np->sira][cikbloki] != np->lbltan &&
      blokmp[np->sira][cikbloki] != 0)
    cikbloki += 2;
if(blokmp[np->sira][cikbloki] != np->lbltan){
    printf("\n Buyuk derleme hatasi label
    uyusmuyor \n");
    exit(-1);
}
blokmp[np->sira][cikbloki] = bloknm;
blokmp[bloknm][1] = blokmp[np->sira][cikbloki+1];
ikuyrnm = blokmp[np->sira][cikbloki+1];
} ;

```

7. DENEY ORTAMI VE DENEY SONUÇLARI

7.1 Genel

REL P dilinin performansını ve paralel işleme'den yararlanma derecesini ölçmek üzere bir deney ortamı tasarlanmıştır.

7.2 Deneyin amacı

Deney'in amacı, programlama dilinin tek işlemcili ortamdaki performansı ile çok işlemcili ortamdaki performansını, karşılaştırmaktır.

Ticari Veri Tabanı Yönetim Sistemleri ile detaylı bir karşılaştırma çalışmanın amacı dışındadır. Bunun nedenleri şu şekilde sıralanabilir:

1. Tasarladığımız ve uyguladığımız prototip programlama dili, ticari olarak pazarlanan bir çok ürüne göre, daha kapsamsız olduğundan,

sorgulamada sağladığı hız yönünden aşırı avantaja sahip olabilir. Nitekim Progress VTYS ile tek işlemcili ortamda yapılan bazı karşılaştırmalarda aynı uygulamalar için 6 ile 15 misli hızlanma elde edilmiştir. Bu husus, prototip dilin tam ticari uygulaması yapıldığında değişiklik gösterebilir.

2. Ticari olarak pazarlanan bazı ürünler, onlarca adam/yıl gayret harcanarak, bazı sorgulamalar için donanım kaynakları kullanımı yönünden özel olarak optimize edilmiş olabilirler. Yapılan çalışma ise, Veri Tabanı yapısına dönük bu çeşit optimizasyonlar yapmaktan çok, paralel işlemcilerden yararlanarak performans avantajı sağlamak üzerinde yoğunlaşmıştır.

3. Ticari Veri Tabanı Yönetim sistemlerinin çok çeşitli olması, kendi aralarında performans yönünden yapılan karşılaştırmaların dahi net olmaması, uygulamadan uygulamaya ve platformdan platforma değişen farklı sonuçlar ortaya çıkması, bu konuda referans seçimini de güçleştirmektedir. Aslında bu Veri Tabanlarının göreceli performans ölçümleri konusu ayrı bir Tez konusu olabilir.

Özetle Ticari Veri Tabanı Yönetim sistemleri ile yapılacak bir karşılaştırma, lehte ve aleyhte yanıltıcı ve somut olmayan sonuçlara götürebilecektir. Bu çalışma içinde Ticari Veri Tabanı yönetim sistemleri ile performans yönünden bazı karşılaştırmalar yapılmıştır ancak bu karşılaştırmalar tezin amacına yönelik değildir, sadece ortaya koyulan programlama dilinin ticari uygulaması yapıldığında kazandırabileceği performans hakkında potansiyel uygulayıcıların fikir edinmesine yöneliktir.

Yine çok kullanıcıli bilgisayar ortamının deneye katabileceği ilave karmaşıklıktan sakınmak amacıyla, deney, çok kullanıcıli bilgisayar sisteminde yapılmasına rağmen, tek kullanıcıli ortama indirgenmiştir. Diğer bir deyimle, deney yapılırken sistem üzerinde başka bir kullanıcı çalışmamıştır.

Yukarıdaki düşünceler altında, deneyin amacı ve yapılan kabullenmeler bir soru ifadesi ile şu şekilde özetlenebilir.

"Sağladığı diğer avantajlar da dikkate alınarak, RELP programlama dili, tek kullanıcı bir bilgisayar platformunda uygulanmak istenmektedir. Bu platformun tek işlemci yerine, çok işlemcili bir platform olması halinde sağlanabilecek avantajlar nelerdir. Diğer bir deyimle, RELP programlama dilinin paralel işlemekten yararlanabilirliği ne düzeydedir."

7.3 Deney şartları

7.3.1 Deneyin gerçekleştirildiği donanım platformu

Motorola M8840 Bilgisayar Sistemi.

İşlemci Motorola MVME188 :

- 4 Adet MC88100, 20 MHZ RISC mikro işlemci
(CPU Chip Mask Oxa ((82N-G.3)
- Ön bellek yönetim birimi : MC88200
- Her İşlemci için ön bellek yönetim birimi sayısı: 2
- Ön bellek büyüklüğü: Her işlemci için
16 Kbyte veri, 16 Kbyte komut.
- Saat hızı : 20 MHZ.

Bellek : 16 MB.

Disk alt yapısı :

Disk kontrol birimi : Motorola MVME328 SCSI kontrol birimi.

SCSI Cihazlar : 330MB SCSI Disk sürücü
(Fujitsu M2622F-512 Rev M404 V 1)
3.5" SCSI Disket sürücü 1.44 MB.
(TEAC FC-1 JHF 11 Rev J V1)

7.3.2 Prototip sistem yazılım geliştirme ve deney ortamı

İşletim Sistemi : Motorola UNIX System V/88 Release 4.0 Vers 4.0

Derleyici : GNU C.

7.4 Ölçümlerin daha hassas olması için alınan önlem

Ölçümler UNIX işletim sistemi çalıştıran platform üzerinde yapıldığından, çalışması zorunlu UNIX sistem süreçleri dışında sistem süreçlerinin tümü, tek kullanıcı (Single User) konumuna geçilerek devre dışı bırakılmıştır.

7.5 Ölçülecek büyüklük

Ölçülecek büyüklük olarak, RELP derleyicisinin derlediği RELP programının başlatılmasından sonuçlanmasına kadar geçen zaman (Elapsed Time) seçilmiştir. Bu büyüklüğün, Çeşitli Hızlanma Faktörü (Speed Up) ölçütlerine göre daha iyi bir ölçüt olduğu düşünülmektedir (Crowl,1994).

7.6 Ölçüm birimi

Ölçüm birimi UNIX Temel Zaman Birimi olan Zaman Artımı("Time Tick") dir. Her Zaman Artımı 10 Milisaniyedir. Bazı şekillerde milisaniye cinsinden geçen zaman da gösterilmiştir.

7.7 Ölçüm için seçilen uygulamalar

Tipik sorgulama programları yazılarak, bu programlar üzerinde ölçüm yapılmıştır. (Date,1981)'de yer alan örnek bir stok kontrol veri tabanı esas alınarak toplam 26 değişik sorgulama programı yazılmış ve bu programlar üzerinde ölçüm yapılmıştır. (Date,1981)'de yer alan bu programlar Veri Tabanı öğretimine dönük ve çok sayıda sorgulama alternatiflerini içeren kapsamlı sorgulama

programlarıdır. Bu uygulamadaki stok kontrol veri tabanı yapısı ve ilgili açıklamalar EK-2'de yer almaktadır.

Bu programların denenmesi, hem dilin değişik sorgulamaları yapabilme kabiliyetini göstermek hem de gerçek uygulamalarda genel olarak nasıl bir performans sağlayacağı konusunda fikir edinmek amacını taşımaktadır.

Bunun dışında dilin temel yapısını oluşturan blok tipleri kullanılarak yazılan deneysel programlar, daha büyük veri kümeleri üzerinde çalıştırılarak, değişik işlemci sayıları söz konusu olduğunda geçen zamanlar ölçülmüştür.

7.8 Deneylerin ve sonuçlarının sergilenmesi

Deneylerin ve sonuçlarının gösterimi Şekil 7.8'deki örnekte gösterilen formatta yapılmaktadır.

Bu formattaki bölümlerin açıklaması şu şekildedir:

PROGRAM AÇIKLAMASI : Ölçüm yapılacak R programı açıklaması bu bölümde yer alır.

PROGRAM : Ölçüm yapılan R programı kaynak kodu bu bölümde yer alır.

İŞLEMCİ SAYISI : Deneyin yapıldığı işlemci sayısını gösterir.

GEÇEN ZAMAN : Çalışma zamanında R programının başladığı an ile bittiği an arasında geçen zamanı "UNIX Time Tick" olarak içerir. Bu değerlerin gerçek zaman karşılığını milisaniye cinsinden bulmak için değerlerin 10 ile çarpılması yeterli olur. Bu hanede en az iki değer bulunur, bunlardan ilki programın ilk çalıştırılmasında bulunan "ilk çalıştırma" değeridir. İkincisi ise ard arda yapılan denemelerde elde edilen sonuçların minimumudur.

DEĞERLENDİRME : Bu bölümde deney sonuçları

PROGRAM AÇIKLAMASI :

Get supplier numbers for suppliers with status value less the current maximum status value in the S table.

(S tablosundaki maksimum değerden düşük statü değerine sahip üreticilerin numaralarını listele.)

PROGRAM-10 :

```
GENERATE R_S _stat.NUM ;
APPLY MAX ON _stat ;
TRANSFER AA;
GENERATE R_S _stat.NUM, _snum;
TRANSFER BB;
CONDBLOK AA BB REF._Sonuc>INP._stat;
REPORT ?[10,15] SAY "sno CHAR _snum,
        ?[11,15] SAY "statu NUM _stat;
#
```

	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	33	21	17	9
GEÇEN ZAMAN MİNİMUM	20	19	15	9
HIZLANMA KAT- SAYISI İLK	1	1.57	1.94	3.47
HIZLANMA KAT- SAYISI MİN	1	1.05	1.33	2.1

DEĞERLENDİRME :

Minimum zamanlar cinsinden iki ve üç işlemcide elde edilen hızlanma çok belirgin değildir. Bunun nedeni, programdaki blok sayısının fazlalığı nedeniyle, her paralel işlemcide birden fazla sanal işlemcinin aynı anda aktif olması olarak yorumlanmaktadır. İlk çalıştırma anında sanal işlemcilerin daha uzun süreli çalışması, paralel işleme avantajını daha belirgin hale getirmektedir.

Şekil-7.8 Deney sonuçlarının gösterimi.

HIZLANMA KATSAYISI : Belirli bir işlemci sayısı ile elde edilen "geçen zaman" değerinin, tek işlemcide aynı programın çalışması ile elde edilen "geçen zaman" değerine oranıdır. Hem ilk çalıştırma zamanları hem de minimum çalıştırma zamanları esas alınarak hızlanma katsayıları verilmiştir.

DEĞERLENDİRME : Bu bölümde deney sonuçları ile ilgili

değerlendirme ve yorumlar yer almaktadır. Deneyin neden bu sonuçları verdiği konusunda açıklama getirilmektedir. Üzerinde deney yapılan 26 program ile ilgili sonuçlar EK-2'de yer almaktadır.

7.9 Diğer deneyler

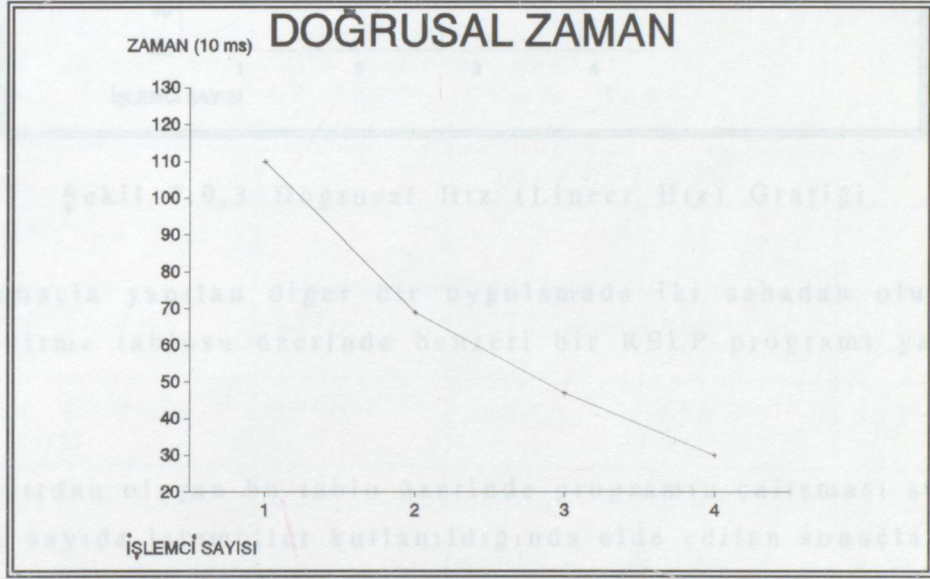
Paralel çalışmanın daha büyük veri hacimleri söz konusu olduğundaki etkisini görmek amacıyla, 174 kayıtdan oluşan, "Baba-Oğul" ilişkisini belirleyen, dolayısıyla, BABA ve OGUL adlı iki sahaya sahip R_BABA tablosu yaratılmıştır.

PROGRAM AÇIKLAMASI : BABA olduğu bilinen kimselerin isimlerinden oluşan bir küme verildiğinde, bu kimselerin erkek torunlarından oluşan bir kümeyi R_BABA tablosunda belirlenen ilişkilerden yararlanarak yaratınız.				
GENERATE R_BABA _baba; APPLY _baba _baba / R_BABA \ _ogul; APPLY _ogul _baba / R_BABA \ _ogul; REPORT ?[10,15] SAY "ogul CHAR _ogul;				
	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	120	75	52	32
GEÇEN ZAMAN MİNİMUM	110	69	47	30
HIZLANMA KAT- SAYISI İLK	1	1.6	2.30	3.75
HIZLANMA KAT- SAYISI MİN	1	1.59	2.34	3.66
DEĞERLENDİRME : Veri yükü arttıkça, sanal işlemcilerin çalışma süreleri uzamakta ve paralel işlemenin getirdiği avantaj daha belirgin hızlanma katsayıları halinde gözükmemektedir. 174 kayıtdan oluşan R_BABA tablosu kullanılarak oluşturulan 106 elemanlı "Torunlar" kümesi, dört işlemci ile saniyenin üçte birinden az bir süre içinde (300ms) oluşturulmaktadır.				

Şekil 7.9-1 R_BABA ilişkisi üzerinde yapılan deney.

Bu tablo üzerinde Şekil-7.9-1'de yer alan deney yapılmıştır. Deneyde RELP dilinde en çok rastlanan sorgulama tipinde bir sorgulama yapılmıştır. Sorgulamada bir BABA kümesi verildiğinde bu kümede yer alan kişilerin oğullarının oğullarının, yani erkek torunlarının bulunması amaçlanmaktadır.

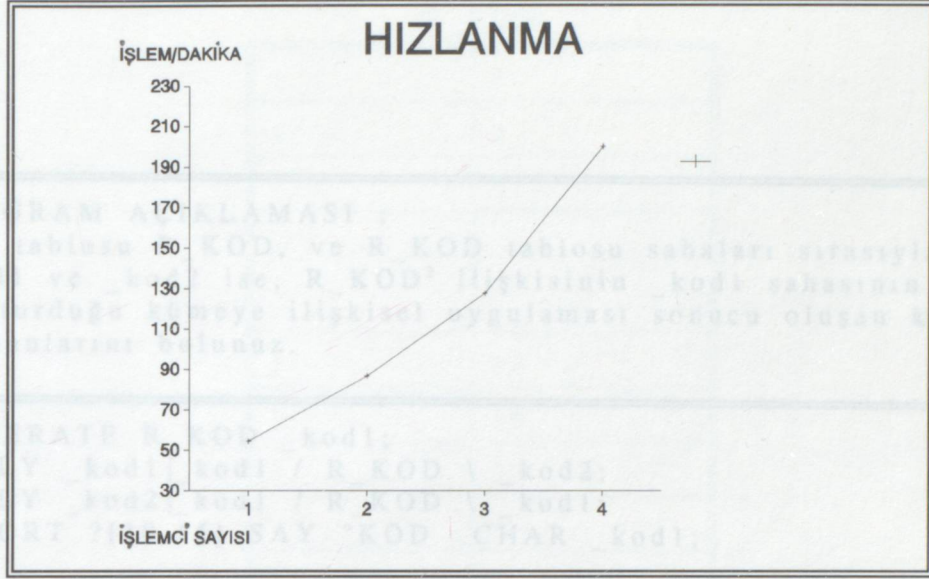
Deney sonuçları esas alınarak, işlemci sayısı bağımsız değişken, "zaman" ise bağımlı değişken seçilerek çizilen grafik şekil 7.9-2' dendir.



Şekil 7.9-2 Doğrusal Zaman (Lineer Time) Grafiği.

Yine bağımsız değişken "işlemci sayısı" olmak üzere, Y ekseninde yer alan bağımlı değişkenin, dakikada tamamlanan işlem sayısı olarak seçilmesi ile oluşturulan "Doğrusal Hız" (Lineer Speed) grafiği şekil 7.9-3'de yer almaktadır.

Görüldüğü gibi veri hacmi arttıkça, sanal işlemci süreçlerinin icra zamanları uzamakta ve paralel işleme etkisi daha belirgin olarak ortaya çıkmaktadır.



Şekil 7.9.3 Doğrusal Hız (Lineer Hız) Grafiği.

Aynı amaçla yapılan diğer bir uygulamada iki sahadan oluşan bir kod çevirme tablosu üzerinde benzeri bir RELP programı yazılmıştır.

400 kayıtdan oluşan bu tablo üzerinde programın çalışması sonunda, değişik sayıda işlemciler kullanıldığında elde edilen sonuçlar şekil-7.9-4'de yer almaktadır.

Tablonun oluşturduğu ilişkilendirme düzeninde `_kod1` sahasında birden yirmiye kadar sayılar yer almaktadır. `_kod1` sahasındaki her sayı `_kod2` sahasında birden yirmiye kadar tüm sayılar ile ilişkilendirilmiştir. Böylece 400 kayıt oluşturulmuştur.

R_KOD tablosunun düzeni şekil-7.9-5'de gösterilmiştir.

PROGRAM AÇIKLAMASI : Kod tablosu R_KOD, ve R_KOD tablosu sahaları sırasıyla _kod1 ve _kod2 ise, R_KOD ² ilişkisinin _kod1 sahasının oluşturduğu kümeye ilişkisel uygulaması sonucu oluşan küme elemanlarını bulunuz.				
GENERATE R_KOD _kod1; APPLY _kod1[_kod1 / R_KOD \ _kod2; APPLY _kod2[_kod1 / R_KOD \ _kod1; REPORT ?[10,15] SAY "KOD CHAR _kod1;				
	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI ----	1	2	3	4
GEÇEN ZAMAN İLK	1810	1072	862	540
GEÇEN ZAMAN MİNİMUM	1800	1070	855	535
HIZLANMA KAT- SAYISI İLK	1	1.688	2.09	3.35
HIZLANMA KAT- SAYISI MİN	1	1.682	2.14	3.36
DEĞERLENDİRME : Veri yükü arttıkça, sanal işlemcilerin çalışma süreleri uzamakta ve paralel işleminin getirdiği avantaj daha belirgin hızlanma katsayıları halinde gözükmemektedir. 400 kayıtdan oluşan R_KOD tablosu kullanılarak oluşturulan 8000 elemanlı sonuç kümesi, 5.3 saniyede oluşturulmaktadır.				

Şekil 7.9-4 R_KOD ilişkisi üzerinde yapılan deney.

R_KOD

1	1
1	2
1	3
.	.
1	18
1	19
1	20
2	1
2	2
2	3
.	.
2	17
2	18
2	19
2	20
.	.
20	17
20	18
20	19
20	20

Şekil 7.9-5 R_KOD İlişki Tablosu İçeriği.

Bu uygulamada, sistemde 5.4 saniyede 8400 Hareket aktığı saptanmıştır. Her iki deneyde program çıkışı ekrana değil diske yazdırılmıştır. Sonuçların ekrana yazdırılmasının bu kadar büyük bir çıkış için anlamlı olmaması yanında, Unix ekran performansının

düşük olması ekran yazma süresini çok uzatarak, paralel işlem ile kazanılan avantajın harcanan bu zaman yanında ihmal edilebilir bir düzeyde kalması sonucunu doğurmaktadır.

REL P programlama dilinin ticari veri tabanı sorgulama dilleri ile karşılaştırılması amaçlanmasa da, bir fikir edinmek maksadıyla PROGRESS Veri Tabanı Yönetim Sistemi ve dördüncü nesil programlama dili kullanılarak bazı örnek programlar yazılmış ve aynı içerikte tabloları kullanılarak, aynı platformda tek işlemci üzerinde çalıştırılarak ölçüm yapılmıştır. Bu denemelerde tek işlemci üzerinde çalışan REL P programlarının karşılık gelen PROGRESS programlarından altı ile onbeş misli daha hızlı çalıştığı gözlenmiştir. Ancak daha önce belirtildiği üzere, bu konu tez'in amaçları dışında tutulmakta ve tam skala ticari REL P uygulaması yapıldıktan sonra incelenmesi gerekli bir husus olarak görülmektedir.

7.10 Deneylerdeki saptamalar

Deneyler sırasında aşağıdaki gözlemler yapılmıştır:

1. Tek kullanıcı (Single User) konumuna indirgenen deney şartlarında bile, belirli UNIX sistem süreçlerinin çalışması engellenememekte, bu nedenle de tek işlemci üzerinde yapılan ölçümlerde değişken sonuçlar elde edilmektedir.

2. Normal halde uyuma konumunda (Sleeping State), Uyuma Kuyruğunda (Sleeping Queue) bekleyen, Sanal İşlemciler (Blokleri temsil eden süreçler) ana program tarafından mesaj gönderilerek görevlerini yapmak üzere uyarılmaktadır. Uyandırılan sanal işlemci, uyuma konumundan, çalışabilir konuma geçerek, Uyuma Kuyruğundan (Sleep Queue) Çalışma Kuyruğuna (Run Queue) aktarılmakta, bu andan itibaren, İşletim Sistemi Zamanlayıcı (Scheduler) süreci tarafından tahsis edilen zaman dilimlerini almaya başlamaktadır.

Bu durumda tek işlemci deneylerinde, eğer blokların RELP programı içinde belirlenen akıştaki pozisyonlarına uygun bir uyandırma sırası gerçekleşirse daha iyi bir performans elde edilmektedir. Bu sıra, uyandırma mesajlarının süreçler arasındaki iletiliş süresine bağlı olarak değişebilmektedir. Bu sıra bazı süreçler için bozulursa, olması gereken sıradan önce aktive olup, gerçekte henüz aktif olmamış ve diğer bir sanal işlemciden çıkış bekleyen sanal işlemciler, gereksiz işlemci zamanı aldığından performans gerilemektedir. Aynı etki, çok işlemcili deneylerde, fiziki işlemci sayısından fazla sayıda sanal işlemci çalıştırma, yani her fiziki işlemcide birden fazla sanal işlemci çalıştırma söz konusu olduğunda da görülmektedir. Bu etki, tamamiyle işletim sisteminin süreçler arası mesaj iletişimi mekanizmasındaki(Inter Process Communication Message Facility) uygulamaya bağlı olan ve uygulayıcı olarak bizim müdahale edemediğimiz bir etkidir.

3. Sanal İşlemciler arasında kurulan kuyruk bağlantılarının yapılarının çok etkin olduğu ve Sanal İşlemcilerin işe başlama, bitiş zamanları esas alınarak yapılan gözlemlerde, ayrı fiziki işlemciler üzerinde çalışan sanal işlemcilerin en çok bir zaman birimi (tick) farkla aynı zaman dilimlerinde birbirine paralel işlem yaptıkları saptanmıştır.

4. Bazı programlarda, ekrana çok büyük hacimde veri yazma, veya bazı sanal işlemcilerin görevleri gereği diske yoğun olarak erişmesi söz konusu olduğunda, hem seri hem de paralel uygulama için belirli bir işlemde, çok büyük bir zaman harcaması söz konusu olabilmektedir. Bu durumda paralel işlemeden kazanılan avantaj, bu büyük zaman yanında ihmal edilebilir düzeyde kalmaktadır.

5. Kullanılan Veri Tabanı yapısı erişim metodu olarak Dağıtım Kodlaması ("Hashing"), "Yoğun İndeks" ("Dense Index") ve diske direk erişim kullanmaktadır. Bu nedenle bir çok sorgulama programının çalışma hızı veri büyüklüğünden büyük ölçüde etkilen-

memektedir. Bu programlar çoğunlukla tek bir sabite dayalı bir kriter ile ilişki tabloları içinde arama yapan tip programlardır. Bu tip programlarda veri tabanının hacminin programın çalışma hızını etkilemediği saptanmıştır. Bir ilişkinin tüm kayıtlarını akışa uğratan tipteki programlar ise tabii olarak veri tabanı büyüklüğü ile doğru orantılı olarak artan bir zaman harcamasına sebep olmaktadır. Paralel işlemenin etkisi bu ikinci tür programlar üzerinde daha iyi görülmektedir. Özellikle veri hacmi arttıkça, sanal işlemcilerin çalışma süreleri artmakta, ve küçük hacimli veri tabanı üzerinde net gözlenemeyen paralel işlem etkisi daha iyi gözlenebilmektedir. Zira, küçük hacimli veri üzerinde, tek işlemcili bir sistemde, sanal işlemciler çok yüksek hızla arka arkaya çalışarak işlemlerini bitirmekte, her birinin harcadığı zaman dilimi çok küçük olmaktadır. Aynı işlemi paralel sanal işlemciler ile yaptığımızda ise en az bir sanal işlemcinin harcadığı kadar zaman harcanacağı için, özellikle RELP programındaki blok sayısı da azsa, seri işlem sonucu ile paralel işlem sonucu birbirine çok yakın çıkmaktadır.

6. Seri ve paralel programların ilk çalıştırılmaları sonucu elde edilen değerler göreceli olarak daha sonraki çalışmaları sonucu elde edilen değerler ortalamasından daha yüksek değerler olarak bulunmaktadır. Bunun nedeni ise ilk çalıştırmada üstlenilen görev ile ilgili dosya açmalar nedeniyle, dosyaların disk cep belleğe aktarılması, sonraki çalıştırmalarda bu dosyaların disk erişimi yerine disk cep bellekte bulunarak işleme girmesidir (Crowl,1994). Programların paralel ve seri çalıştırma zamanlarının karşılaştırılmasında ilk çalışmada bulunan değerleri esas alınırsa daha anlamlı sonuçlar çıkmaktadır. Bunun nedeni ise, yine programların çalışma zamanının uzaması ve bunun sonucu olarak da paralel işlemenin kazancının ön plana çıkmasıdır. Bir çok program derlendikten sonra sadece bir kez çalıştığına göre, programın kullanıcı gözündeki çalışma hızı açısından değerinin ölçümünde ilk çalıştırma zamanının önemi açıktır. Deney sonuçlarının sunulmasında, ilk çalışma değerleri ayrıca gösterilmiştir. Tek işlemci denemelerinde, yukarıda

madde 1 ve 2'de açıklanan nedenler ile bazı durumlarda, tek işlemci üzerinde ilk seri çalıştırma sonucunun, paralel işlemcilerde ilk paralel çalıştırma sonucundan beklenmedik şekilde çok yukarıda çıkabildiği gözlenmiş, ve madde 1 ve 2'de belirtilen etkiyi yok etmek üzere, seri ilk çalıştırma zamanları, değişik zamanlarda yapılan deneyler ile tekrar tekrar tesbit edilerek minimum değer esas alınmıştır. İlk çalıştırma zamanının ölçümü yanında, ilk çalıştırmayı takibeden çalıştırmalar yapılmış ve elde edilen "geçen zaman" (elapsed time) değerlerinin minimumları her işlemci sayısı için ayrı ayrı listelenmiştir.

8. TARTIŞMA, SONUÇ VE ÖNERİLER

7.11 Deney sonuçlarının değerlendirilmesi

Bu bölümde deneylerde yapılan saptamalar esas alınarak varılan kanılar aşağıda kısaca açıklanmıştır.

1.RELP programlama dili paralel işlemcilerden yararlanabilirlik düzeyi yüksek bir dildir.

a. Paralel işleme avantajı, Paralel fiziki işlemci başına aktive olan sanal işlemci sayısı azaldıkça daha belirginleşmektedir.

b. Sanal işlemcilerin çalışma süresi üstlendikleri görev niteliğine bağlı olarak homojen bir şekilde uzadıkça paralel işleme avantajı belirginleşmektedir.

c. Programların ilk çalışma zamanları esas alındığında paralel işleme avantajı daha belirgindir.

2. Paralel işleme avantajı, bir veya daha fazla sanal işlemci aşırı giriş/çıkış yoğun işlemler ile meşgul olmak zorunda kaldığında büyük oranda azalmaktadır. Bu tür uygulamalarda Ekran yerine diske çıkış verme, Yarı İletken Disk kullanma gibi teknikler ile Giriş/Çıkış yüklenme etkisinin azaltılması paralel işleme avantajının daha iyi kullanılması için gereklidir.

3. Herhangi bir uygulama için, her fiziki işlem birimi başına, bir

sanal işlemci aktive olduğu takdirde, ölçülen "geçen zaman" değeri deneyden deneye değişkenlik göstermemekte, sabit kalmaktadır.

4. RELP programlama dili tüm diğer programlama dilleri gibi, işletim sisteminden büyük ölçüde etkilenmektedir. Paralel işleme avantajlarının daha iyi kullanılabilmesi için işletim sistemi düzeyinde görev zamanlayıcı (scheduler) ve süreçler arası iletişim (Inter Process Communication) gibi mekanizmaları üzerinde değişiklikler yapılması faydalı olacaktır.

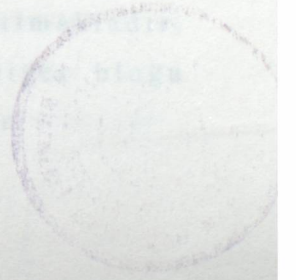
8. TARTIŞMA, SONUÇ VE ÖNERİLER

Donanım platformlarındaki gelişmeler, kısa zaman içinde ana bellek kapasitesinin Gigabyte düzeylerine ekonomik olarak çıkartılabileceğinin, kullanılan paralel işlemci sayısının ve bilgisayar iletişim hızlarının artacağına işaretlerini açık bir şekilde vermektedir.

Bu kaynakların etkin bir şekilde kullanımı için gereken yazılım alt tabanının da, uygulama gereksinimleri doğrultusunda, hızla geliştirilmesine ihtiyaç vardır.

Karar verme sürecinde, karar destek sistemlerinde hızla veri işleyerek karar sürecini çabuklaştıracak programların yazılabileceği programlama dillerinin geliştirilmesi, bu konudaki potansiyel uygulama konularından biridir.

RELP programlama dili, bu uygulama alanına yönelik tasarlanmıştır. Dil, anılan kaynakların genişliğine dayanarak, sonuç verme süresini kısaltma amacını taşımaktadır. Bu amaca eşlik eden diğer bir amaç ise dilin mümkün olduğunca basit, programcının doğal düşünme şekline uyan, soyutlamalara sahip ve sonuçta programlamayı kolaylaştıran bir dil olmasıdır.



Makro Veri Akış modeli (Almasi et al, 1989) ve büyük parçalı (course grain) paralelleştirme mantığı kullanılan RELP programlama dilinin tasarımı, bir çok değişik tip programlama dilinin özelliklerinden yararlanılmıştır.

Dilin kontrol yapısı, paralel çalışmadan en fazla yararlanma düzeyini sağlayan veri akış yapısını kullanmaktadır (Almasi et al, 1989).

RELP dil blokları arasında kullanılan kuyruk (FIFO) yapıları ile paralel bloklar arası zamanuyumsuz eş zamanlama ve iletişim yapılmaktadır. Zamanuyumsuz eş zamanlama paralel işlemede iletişim maliyetini en aza indiren yapıdır. Paralel çalışan girdiler arasında eş zamanlama ve veri iletişimi için kullanılan bu teknik ve aynı bağlamda, kullanılan "Veri Akış" tipi dil kontrol yapısı, birlikte "Pipeline Etkisi" adını verdiğimiz etkiyi yaratarak paralel işleme platformu olanaklarından yeterince yararlanılmasını sağlamıştır. Her RELP bloğu bir "Pipeline" kademesi olarak düşünülürse bu etkinin varlığı kolayca görülebilir.

Her blok paralel işlemeye aday bir paralel işlem girdisi olarak açık bir biçimde bilindiğinden, programcının paralel işlem düzeneği kurma konusunda ayrıca gayret sarfetmesi gerekmemektedir. Paralel işleme düzeneği derleyici ve dil çalışma ortamı tarafından otomatik olarak kurulup işletilmektedir.

RELP programlama dili, yapı taşları olan blokların, akan verilere ilişki ve veri transformasyon fonksiyonlarını uygulaması nedeniyle "uygulama tabanlı" (applicative) diller sınıfına girer. Her blokta yapılan işlemler girişteki veri kümesini çıkışta bir diğer veri kümesine dönüştürme veya akış yönlendirme şeklindedir. Dil bloklarındaki işlemler tamamıyla blok içinde yerel olarak yapılmaktadır. Veri Tabanına yapılan günlemeler dışında, bir bloğun diğer bloğu dolaylı olarak etkilemesi, yani yan etki olasılığı yoktur.

Mantıksal programlama dillerindeki "Gerçekler" (Facts) ve kurallar (RULES), RELP dili ilişki tablolarında tutulduğu takdirde, mantıksal hedef taramaları RELP dilinde de yapılabildiği görülmüştür. Patern uyumuna dayanan RELP ilişki uygulama yapısı kuralların içinde yer alan gerçek paternlerinin ilişki tablolarında hızla aranmasını sağlar. Bu taramalar RELP yapısına bağlı olarak paralel olarak icra edebilir. Böylece, Mantıksal Paralel Programlama Dillerindeki AND/OR paralel çalıştırma RELP dilinde bir başka biçimde otomatikman gerçekleştirilmiş olur. Önemli bir özellik olarak, RELP dili, dağıtılmış veri yapılarına dayanan LINDA programlama dilinin "Kayıt Uyuşturma-Okuma" özelliğini "İlişki Uygulama" operasyonunun yapısında kullanmaktadır.

RELP dilinin bloklarına, verilerden oluşan mesajları alan, içindeki metodlar ile işleyen ve verilerden oluşan yeni mesajları diğer nesnelere gönderen, nesneler olarak bakabiliriz. Bu durumda, RELP dilinin, programcının program modelini canlandırması açısından, nesne temelli yapıya sahip olduğunu söyleyebiliriz.

Çoklu Komut Çoklu Veri (MIMD-Multiple Instruction Multiple Data) Paylaşılan Bellek bilgisayar mimarisinde çalışması öngörülen RELP programlama dilinin gerçekleştirilebileceğini göstermek amacıyla, bir prototipi geliştirilmiş ve dört işlemcili bir donanım platformunda, UNIX işletim sistemi altında çalıştırılmıştır.

Prototipin geliştirilmesinin diğer bir nedeni ise, dilin paralel işlemcilerden yararlanma derecesini pratikte görüp deneyler yapmak olmuştur. Prototip, dilin gerçekleştirilebileceğinin görülebileceği ve deneylerin yapılabileceği noktaya kadar geliştirilmiştir. Bu tür dillerin ticari olarak geliştirilmesinde onlarca programcı-yıl gayret harcanmaktadır. Dolayısıyla, dilin, ticari olarak uygulanabilmesi ve gizli kalmış olabilecek hatalarından arındırılması için daha uzun süreye yayılmış ilave emek sarfedilmesi gerektiği düşünülmektedir.

Deney bir stok kontrol veri tabanı üzerinde gerçekleştirilebilecek sorgulama programlarının, RELP programlama dilinde yazılması yoluyla yapılmıştır. Bu programlar prototip RELP derleyicisi ile derlenmiş, seri ve paralel performans, çalışma zamanı cinsinden ölçülmüştür. Değişik uygulamalarda dört işlemci ile paralel işleminin 3.83'e ulaşan hızlanma katsayıları sağlayabildiği saptanmıştır. Daha az sayıda fiziki işlemci söz konusu olduğunda ise, değişik uygulamalar sonunda, ulaşılabilen en yüksek hızlanma katsayıları üç işlemci ile 2.7, 2 işlemci ile 1.8 olarak bulunmuştur. Özellikle programların ilk çalıştırılma zamanları ele alınıp karşılaştırma yapıldığında, paralel işleminin avantajı daha belirgin olarak göze çarpmaktadır. Yine, işlenen veri hacmi arttıkça paralel işleme avantajının daha net olarak görülebildiği saptanmıştır.

Dilin bu uygulamasında, aktive edilen sanal işlemci sayısı ile mevcut fiziki işlemci sayısı eşit olduğunda paralel işlemcilerden etkin ve istikrarlı bir biçimde yararlanıldığı görülmüştür. Herhangi bir program içinde yer alan blokların sayısı gereği, çok sayıda sanal işlemci, az sayıda fiziki işlemci üzerinde aktif hale getirildiğinde, düzeneğin, sanal işlemcilerin fiziki işlemcilere atanma şemasındaki değişikliklere hassas hale geldiği, ve daha istikrarsız sonuçlar elde edildiği görülmüştür.

Dörtten daha fazla sayıda işlemci bulunan platformlarda az sayıda bloğa sahip küçük programların tüm fiziki işlemcilerden yararlanma yolu da açıktır. Örneğin dört bloktan oluşan bir program 36 fiziki işlemciye sahip bir platformda çok geniş çaplı veri üzerinde çalışacaksa, her blok sanal işlemcilerinden dokuz kopya olmak üzere sanal işlemcilerin fiziki işlemciler üzerine dağıtılması ve aynı bloğu temsil eden sanal işlemcilerin, aynı kuyruktan giriş yapıp, aynı kuyruğa çıkış yapmaları, bu yapı içinde mümkündür. Bu şekilde bir bloğun iş yükü 9 süreç tarafından paylaşılarak yapılmış olacaktır.

Yazılım konusunda ölçüm kriterleri oluşturmak ve sübjektif olmayan sonuçlara varmak oldukça iddialı bir konudur. Bu konuda, seçilen bir yazılım metriği kullanılarak (Orman,1991) RELP dilinin çeşitli karmaşıklık değerleri FDL (Functional Data Language) referans dili temel alınarak bulunmuştur.

Dilin ilk öğrenilmesindeki kolaylık derecesini saptayan "İlk Karmaşıklık" SQL ve İlişkisel Cebirden yüksek, NDL'den düşük olarak bulunmuştur.

Bir grup sorgulama örneği seçilerek yapılan "Sorgulama Karmaşıklığı" saptamasında, RELP dilindeki sorgulamaların karmaşıklıkları, SQL ve İlişkisel Cebirdeki sorgulamaların karmaşıklıklarına çok yakın değerler olarak bulunmuştur.

Programlama Dili Karmaşıklığı için yapılan karşılaştırmada, seçilen örnek uygulama esas alındığında, RELP dilinin referans dil FDL ile lineer karmaşıklık ilişkisinde olduğu sonucuna varılmıştır.

Programlama dillerinin yaygın kullanımı bir çok faktöre bağlı olarak mümkün olmaktadır. RELP dili bu haliyle sadece akademik bir ürün olarak geliştirilmiş olup, dilden çok kullanılan dil özelliklerinin sağladığı avantajların ortaya çıkması önem kazanmaktadır.

RELP, paralel işleme olanaklarından yararlanma düzeyi yüksek ve karmaşıklığı aynı kategorideki dillerden SQL ve İlişkisel Cebir gibi kullanımı kolay dillere denk bir dil olarak değerlendirilmiştir.

Bu tez çalışması esnasında RELP dili ile ilgili elde edilen sonuçlardan bağımsız olarak, paylaşılan bellek çok işlemci bilgisayar mimarisi üzerinde paralel işleyen süreçlerin veri paylaşımı, eş zamanlaması ve davranışı konusunda önemli tecrübeler elde edilmiştir.



F. Valejo ve arkadaşları tarafından önerilen "Genişletilmiş Petri Net modeli ile Paylaşılan Bellek Çok mikro işlemci İşletim Sistemi" ("Shared Memory Multimicroprocessor Operating System with an Extended Petri Net Model") (Valejo et al, 1994) benzeri yaklaşımlar kullanarak bir İşletim Sistemi uygulamasını hedeflemektedir. Bu çalışmada olay güdümlü (event driven) sistemler hedef uygulama olarak alınmış ve bir "Bus" yapısına bağlı işlemci kartları ile çok işlemcili paralel uygulamalar yapılmıştır. Bu çalışma ortamında, tek bir işlemci tarafından icra edilecek seri programlar klasik geliştirme araçlarıyla geliştirilmektedir. Bu programların içine yerleştirilen sistem program referanslarıyla, iletişim ve eş zamanlama sağlanmaktadır. Özel bir programlama dili ile yazılan paralel icra komutları, derleyici tarafından Genişletilmiş Petri Net (Extended Petri Net) temsil biçimine dönüştürülmektedir. İşletim sistemindeki icra edici (executive) program, bu yapıyı izleyerek klasik olarak geliştirilmiş programların Petri Net'deki düzende icrasını ve Simge alışverişi yapmalarını sağlamaktadır. Çalışma, Paralel İşlem için Makro Veri Akış Modelini kullanması açısından bu çalışmaya benzemektedir.

Bu çalışma sırasında, özellikle deney safhasında, maddi değeri yüksek olan, dört işlemcili bir donanım platformu kısa süreli olarak kullanılabilmiştir. Deneydeki ölçümlerin hassas olması için bu platformun tek kullanıcı düzende kullanılması gerekmiş, bu da deney süresini daha kısa zamana sığdırma zorunluluğunu doğurmuştur. Program çalışma zamanı olarak elde edilen her sonucun, sistemin harici etkilerinden mümkün olduğunca arındırılması için arka arkaya yapılan denemeler sonucu minimum değerinin bulunması amaçlanmış, bu nedenle, denenen her programın en az yirmi kere çalıştırılması ve bulunan değerlerin minimumunun alınması gerekmiştir. Yine, değişik işlemci yükleme şemalarının, program çalışma performansına etkisini görmek üzere, denemelerin değişik işlemci atama şemaları altında tekrar tekrar yapılması da zaman yönünden ilave baskı yaratmıştır. Sonuç olarak deney platformunun kullanım

zamanı kısıtı araştırmının kapsamını limitleyici asıl faktör olarak değerlendirilmiştir.

Bu tezin kapsamı dışında tutulan ancak RELP dilinin bu noktadan sonraki gelişimine yön verebilecek hususlar ve geleceğe dönük öneriler aşağıda kısaca özetlenmiştir.

Gelecekte RELP dili Yarı İletken Diskler üzerinde çalışmalıdır. Zira paralel işlemcilerden ne kadar iyi yararlanılsa da, çok düşük hıza sahip olan disk kaynağı performansta dar boğaz oluşturacaktır.

RELP dili Görüntüsel Programlamaya (Visiual Programming) uygun bir dildir. Her bir tip blok uygun bir simge ile temsil edildiğinde, grafik ekran üzerinde bu simgelerin alt alta dizilmesi ve grafik oklarla birleştirilmesi ile program yapılabilir. Programın icrasında veri akışı görüntü olarak izlenerek hem ara sonuçlar yönünden bilgi sağlanabilir hem de hata ayıklama operasyonlarına olanak tanınabilir.

RELP dili, bilgisayar Giriş / Çıkış yolu hızları arttıkça, veri yolu üzerine yerleştirilmiş tek kart bilgisayarlardan oluşmuş platformlarda kullanılabilir. Yerel Ağ iletişim hızları arttıkça yerel ağ düzeni içinde, Geniş Alan Ağ iletişim hızları arttıkça Geniş Alanda bağlı bilgisayarlar üzerinde RELP dilinin çalışması sağlanabilir.

Ağ üzerinde çalışma durumunda, Global Simge Deposunun değişik platformlara dağıtılması söz konusu olacaktır. Özellikle bellek kapasiteleri çok arttığında bu yapının içerik olarak çok zenginleştirilmesi ve her platformda tutulması söz konusu olabilecektir. Simgelerin gerçek veri karşılıkları çok az sayıda blok için gerekli olmaktadır. Gerekli olduğunda, gerekli olduğu noktada bu karşılık bulunamazsa sadece bu Simge karşılığı olan veri, merkezi bir veri rezervuarından talep edilerek getirilebilecektir. Veri tabloları sadece

gerekli oldukları noktalarda depolanabilecektir. RELP programının bir noktada derlenmesi sonucu her platformdaki sanal işlemcilere görev dağıtımı bu platformlara mesajlar göndermek suretiyle yapılabilecektir. Bu şekilde ağ çapında icra eden dağıtık RELP programları geliştirilebilecektir. RELP dilindeki bu potansiyel, dağıtık veri tabanları açısından detaylı olarak incelenmelidir.

RELP dilinin sorgulama formülasyonunda programcıya sağladığı kolaylık derecesi ve öğrenme kolaylığı uzun dönemde deneysel olarak ölçülebilir.

Bu çalışma esnasında 4 işlemcili bir platform deneyler için kısa süreli olarak kullanılabilmıştır. RELP dilinin daha çok işlemciye sahip bir platformda daha uzun süreli ölçümler ile irdelenmesi faydalı olacaktır. Özellikle fiziki işlemcilere görev ataması ve yük dağıtımı için yeni tekniklerin geliştirilmesi yararlı olacaktır.

Bu çalışmanın, bu konudaki diğer çalışmalara ve gelecekte yapılacak çalışmalara katkısı olacağını umuyorum.

Arhan William C., Seitz Charles L. 1988. Multicomputers: Message-Passing Concurrent Computers. Computer August 1988, Pages 9-24

Backus John. 1978. Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs. Communications of the ACM, August 1978, Volume 21, Number 8.

Bal, Henri, E., Steiner Jennifer G., Tanenbaum Andrew S. 1989. Programming Languages for Distributed Computing Systems. ACM Computing Surveys, Vol 21, No 3, September 1989 Pages 261-322.

Barnes J.G. 1982. Programming in ADA. Addison-Wesley Publishers Limited, 1982. ISBN 0-201-13793-3.

Brogi Antonio, Ciancarini Paolo. 1991. The Concurrent Language. Shared Prolog. ACM Transactions on Programming Languages and Systems, Vol. 13, No. 1, January 1991, Pages 99-123.

Carl Stefano, Pelagatti Giuseppe. 1984. Distributed Databases: Principles and Systems. Mc.Graw Hill Book Company. 1984.

KAYNAKLAR

Abiteboul Serge , Grumbach, Stephane. 1991. A Rule Based Language with Functions and Sets. ACM Transactions on Database Systems, Vol.16, No 1, March 1991 Pages 1-30.

Akl Selim G.1989. The Design and Analysis of Parallel Algorithms. Prentice -Hall International Editions 1989. ISBN 0-13-200073-3.

Agha Gul, Frolund Svend, Kim Young Woo, Panwar Rajenda, Patterson Anna, Sturman Daniel.1993. Abstraction and Modularity Mechanisms for Concurrent Computing. IEEE Parallel & Distributed Technology. May 1993.

Almasi G.S., Gottlieb Allan.1989. Highly Parallel Computing. The Benjamin/Cummings Publishing Company Inc. 1989. ISBN 0-8053-0177-1.

Andrews Gregory R.1991. Paradigms for Process Interaction in Distributed Programs. ACM Computing Surveys, Vol.23, No. 1, March 1991, Pages 49-89.

Andrews Gregory R., Olsson Ronald A., Coffin Michael, Elshoff Irving, Nilsen Kelvin, Purdin Titus, Townsend Gregg. 1988. An Overview of the SR Language and Implementation. ACM Transactions on Programming Languages and Systems. Vol 10, No 1, January 1988, pp 51-86.

Arvind Rishiyur S. Nikhl, Keshav K. Pingali. 1989. I-Structures: Data Structures for Parallel Computing. ACM Transactions on Programming Languages and Systems, Vol. 11, No. 4, October 1989.

Athas William C., Seitz Charles L. 1988. Multicomputers: Message-Passing Concurrent Computers. Computer August 1988, Pages 9-24

Backus John. 1978. Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs. Communications of the ACM, August 1978, Volume 21, Number 8.

Bal, Henri, E., Steiner Jennifer G., Tanenbaum Andrew S. 1989. Programming Languages for Distributed Computing Systems, ACM Computing Surveys, Vol.21, No.3, September 1989 Pages 261-322.

Barnes J.G.1982. Programming in ADA. Addison-Wesley Publishers Limited. 1982. ISBN 0-201-13793-3.

Brogi Antonio, Ciancarini Paolo.1991. The Concurrent Language, Shared Prolog. ACM Transactions on Programming Languages and Systems, Vol. 13, No. 1, January 1991, Pages 99-123.

Ceri Stefano, Pelagatti Giuseppe.1984. Distributed Databases Principles and Systems. Mc.Graw Hill Book Company. 1984.

Chen Marina , Choo Young-Il , Li Jingke. 1991. Crystal: Theory and Pragmatics of Generating Efficient Parallel Code. Parallel Computing Seminar 15-19 July 1991 IBM Europe Institute Oberlech Austria.

Chen Marina. 1991. Crystal: Programming Approach and Compilation Techniques for Massively Parallel Machines. Parallel Computing Seminar 15-19 July 1991 IBM Europe Institute Oberlech Austria.

Chin Roger S. , Chanson Samuel T. 1991. Distributed Object-Based Programming Systems. ACM Computing Surveys, Vol.23, No. 1, March 1991, Pages 91-123.

Chiola Giovanni., Ferscha, Alois. 1993. Distributed Simulation of Petri Nets. IEEE Parallel & Distributed Technology. August 1993.

Clark Keith, Gregory Steve. 1986. Parlog : Parallel Programming in Logic. ACM Transactions on Programming Languages and Systems, Vol. 8, No. 1, January 1986, Pages 1-49.

Codd E.F. 1972. Relational Completeness of Data Base Sublanguages. Database Systems, Courant Computer Science Symposia Series, Vol 6. Englewood Cliffs, N.J.: Prentice-Hall (1972)

Crowl Lawrence A. 1994. How to Measure Present and Compare Parallel Performance. IEEE Parallel & Distributed Technology, Spring 1994. Pg 9-25.

Dally William J. 1991. MIMD Message Passing Architectures, Parallel Computing Seminar 15-19 July 1991 IBM Europe Institute Oberlech Austria.

Danforth S., Tomlinson C. 1988. Type theories and Object Oriented Programming. ACM Computing Surveys, Vol 20, No. 1, March 1988. Pages 31-72.

Danforth S., Valduriez Patrick. 1992. A FAD for Data Intensive Applications. IEEE Transactions on Knowledge and Data Engineering. February 1992, Pages 34-51.

Date C.J. 1981. An Introduction to Database Systems. Addison-Wesley Publishing Company. 1981.

Demers R.A., Fisher J.D. 1992. Gaitonde S.S., Sanders R.R. 1992. Inside IBM's Distributed Data Management architecture. IBM Systems Journal Vol 31, No. 3, 1992.

Dennis Ackerman, W.B. and Dennis J.B. 1979. " VAL-a value oriented algorithmic language: Preliminary reference manual" Laboratory for Computer Science, MIT, Cambridge, Mass. 1979.

Fiadeiro Jose, Sernadas Amilcar. 1986. The Infolog Linear Tense Propositional Logic of Events and Transactions. Information Systems Vol. 11, No.1, pp 61-85, 1986.

Fox Geoffrey. 1991. Parallel Applications Some Experience and Some Lessons, Parallel Computing Seminar 15-19 July 1991 IBM Europe Institute Oberlech Austria.

Fradet Pascal, Metayer Daniel. 1991. Compilation of Functional Languages by Program Transformation. ACM Transactions on Programming Languages and Systems, Vol. 13, No.1, January 1991, Pages 21-51.

Gabb R.G. 1984. Parallel Processing with Large Grain Dataflow Techniques. Computer, Vol.17, No.7. July 1984. pp 55-61.

Gehani N.H. 1993. Capsules: A shared Memory Access Mechanism for Concurrent C/C++, IEEE Transactions on Parallel and Distributed Systems, Vol 4, No 7, July 1993.

Gordon Michael J.C. 1979. The Denotational Description of Programming Languages An Introduction. Springer Verlag-1979. ISBN 0-387-90433

Gottlieb Allan. 1991. Shared Memory MIMD, Parallel Computing Seminar 15-19 July 1991 IBM Europe Institute Oberlech Austria.

Gottlieb Allan. 1991. Parallel Computing Overview System Software, Parallel Computing Seminar 15-19 July 1991 IBM Europe Institute Oberlech Austria.

Grimshaw Andrew S., Strayer Timothy, Narayan Padmimi. 1993. Dynamic Object-Oriented Parallel Processing. IEEE Parallel & Distributed Technology. May 1993.

Haltstead Robert H.Jr. 1985. Multilisp : A Language for Concurrent Symbolic Computation. Acm Transactions on Programming Languages and Systems. Vol. 7, No. 4, October 1985, Pages 501-538.

Hoare, C.A.R. 1981. The Emperor's Old Clothes. Communication of the ACM, February 1981, Volume 24, Number 2, pp 75-83.

Holt R.C., Lazowska E.D., Graham G.S., Scott M.A. 1978. Structured Concurrent Programming with Operating Systems Applications. Addison -Wesley Publishing Company, 1978. ISBN 0-201-02937-5.

Horowitz Ellis.1987. Fundamentals of Programming Languages. Computer Science Press 1987. ISBN 3-540-11768-7.

Hudak Paul.1989. Conception Evolution and Application of Functional Programming Languages. ACM Computing Surveys, Vol. 21, No. 3, September 1989.

Hudak Paul. 1986. Para-Functional Programming. Computer, August 1986,Pages 60-70.

IEEE Spectrum.1994. Kit helps with design of fast VMEbus systems. IEEE Spectrum September 1994. pg 61.

INFORMIX-OnLine Dynamic Server Version 6.0 For the UNIX Operating System, Technical Brief, 1994, 000-20563-77 INFORMIX.

Jones Geraint.1987. Programming in OCCAM. Prentice Hall International. 1987.

Kaiser Gail E., Hailpern Brent.1992. An Object Based Programming Model for Shared Data. ACM Transactions on Programming Languages and Systems, Vol.14, No.2, April 1992 Pages 201-264.

Kesselman Carl.1991. Program Composition Notation, Parallel Computing Seminar 15-19 July 1991 IBM Europe Institute Oberlech Austria.

Kuhns J.L. 1967. Answering Questions by Computer; A logical Study. Report RM-5428-PR, Rand Corp., Santa Monica Calif. (1967)

Kumar Manoj.1991. Application Experience on Parallel Computers at IBM Research. Parallel Computing Seminar 15-19 July 1991 IBM Europe Institute Oberlech Austria.

Lakshmivarahan S.1990. Dhall Sudarshan K.1990. Analysis and Design of Parallel Algorithms, Arithmetic and Matrix Problems. McGraw-Hill Publishing Company-1990. ISBN 0-07-036139-8.

Lewis Ted, El-Rewini Hesham. 1993. Parallax: A Tool for Parallel Program Scheduling. IEEE Parallel & Distributed Technology. May 1993.

Li Jingke , Chen Marina. 1991. Compiling Communication-Efficient Programs for Massively Parallel Machines. Parallel Computing Seminar 15-19 July 1991 IBM Europe Institute Oberlech Austria.

Lieberman H. 1987. Concurrent object oriented programming in Act 1. In Object Oriented Concurrent Programming. A Yonezawa and M. Tokoro, Eds. MIT Press, Cambridge, Mass., pp 9-36.

MacLennan B.J.1981. Elements of Programming Linguistics Part I, The Lambda Calculus and its Implementation. Technical Report NPS52-81-012, Naval Postgraduate School.

MacLennan B.J.1981. Programming with a Relational Calculus. Technical Report NPS52-81-013. Naval Postgraduate School.

MacLennan B.J.1983. Relational Programming. Technical Report NPS52-83-012. Naval Postgraduate School.

Maier David , Rozenshtein David , Salveter Sharon, Stein Jacob, Warren David S. 1987. PIQUE: A Relational Query Language Without Relations. Information Systems Vol.12, No.3, pp 317-335, 1987.

Michael Burke.1991. Compiling for Parallelism, Parallel Computing Seminar 15-19 July 1991 IBM Europe Institute Oberlech Austria.

Montry Gary R.1991. Aspects of Parallel Applications, Parallel Computing Seminar 15-19 July 1991 IBM Europe Institute Oberlech Austria.

Occam Programming Manual. Inmos Limited. Prentice Hall International. 1984.

Orman Levent.1991. Complexity of Database Languages. Information Systems. Vol.16, No.2,pp 169-184, 1991.

Ozsu M. Tamer, Valduriez Patrick. 1991. Distributed Database Systems Where Are We Now. Computer, 1991. Page 68-78.

Pagan Frank G.1981. Formal Specification of Programming Languages a Panoramic Primer. Prentice Hall, Inc. 1981.

Parsaye Kamran, Chignell Mark, Khoshafian Setrag ,Wong Harry.1989. Intelligent Databases. John Wiley & Sons Inc. 1989. ISBN 0-471-50436-0.

Pritchard David. 1991. Transputers and Their Applications, Parallel Computing Seminar 15-19 July 1991 IBM Europe Institute Oberlech Austria.

Reference Manual for the ADA Programming Language. United States Department of Defense. July 1980.

Riezenman Michael J.1994. Special Report : Communications. IEEE Spectrum. June 1994.

Salzberg Betty.1988. File Structures an Analytical Approach. Prentice -Hall International Inc. 1988. ISBN 0-13-314550-6.

Shapiro Ehud.1989. The Family of Concurrent Logic Programming Languages. ACM Computing Surveys, Vol 21, No 3, September 1989, Pg 414-510.

Saunders Stephan.1993. Fibre Channel Outpaces ATM, Data Communications, May 1993.

Saydam Tuncay.1980. Discrete Stochastic System Dynamics an Event Simulation Approach. Boğaziçi University Publications No 222.1980.

Shapiro Ehud.1989. The Family of Concurrent Logic Programming Languages. ACM Computing Surveys, Vol.21, No.3, September 1989, Pages 414-510.

Snir Marc.1991. Software for Parallelism Goals & Means, Parallel Computing Seminar 15-19 July 1991 IBM Europe Institute Oberlech Austria.

Solworth John A. 1992. Ephocs. ACM Transactions on Programming Languages and Systems. Vol 14, No 1, January 1992, pp 28-53.

Spooner C.R. 1986. The ML approach to the Readable All-Purpose Language. ACM Transactions on Programming Languages and Systems, Vol.8, No. 2, April 1986, Pages 215-243.

Stallings William.1984. Local Networks an Introduction. McMillan Publishing Company. 1984.

Strom R.E., Yemini,S.1983, NIL: An Integrated Language and System for Distributed Programming. SIGPLAN Notices. (ACM)18,6.

Tanenbaum Andrew S.1987. Operating Systems Design and Implementation. Prentice Hall Inc.1987. ISBN 0-13-637406-9.

Transputer Development System. Inmos Limited. Prentice Hall. 1990. ISBN 0-13-929068-0.

Tucker Lewis W., Robertson George G.1988. Architecture and Applications of the Connection Machine, Computer, August 1988, Pages 26-38.

Vallejo F., Gregorio, J.A., Harbour, Gonzalez, M., Drake, J.M. 1994. Shared Memory Multiprocessor Operating System with and Extended Petri Net Model, IEEE Transactions on Parallel and Distributed Systems, Vol 5, No 7, July 1994.

Yonezawa A., Briot J.P., Shibayama E. 1986. Object oriented concurrent programming in ABCL/1. In Proceedings of Object-Oriented Programming Systems, Languages and Applications 1986. SIGPLAN Not.(ACM) 21,11(Nov) pp 258-268.

Yonezawa A., Matsuoka Satoshi, Yasugi Masahiro, Taura Kenjiro.
1993. Implementing Concurrent Object-Oriented Languages on
Multicomputers. IEEE Parallel & Distributed Technology. May 1993.

EK-1

**REL P PROGRAMLAMA DİLİ SÖZ DİZİMİ YAPISININ
BNF(BACKUS NAUR FORM), EBNF (EXTENDED BACKUS NAUR
FORM) VE SÖZ DİZİMİ DİYAGRAMLARI KULLANILARAK
AÇIKLANMASI**

BNF FORMUNDA REL P SÖZ DİZİMİ KURALLARI

<program> ::= <bloklar> #

**<bloklar> ::= <blok>
 | <bloklar> ; <blok>**

/* ----- */

**<blok> ::= <inputblok>
 | <applyblok>
 | <generblok>
 | <genertblok>
 | <genereblok>
 | <reportblok>
 | <computeblok>
 | <comfixblok>
 | <sortblok>
 | <transblok>
 | <countblok>
 | <filtblok>
 | <sinkblok>
 | <updtblok>
 | <testblok>
 | <delblok>
 | <mergeblok>
 | <applycumblok>
 | <txcblok>
 | <condblok>
 | <memblok>**

<inputblok> ::= INPUT <getblok>

**<getblok> ::= @ <koordinat> GET <karakterdizin>
 NUM
 <degismi>
 |
 @ <koordinat> GET <karakterdizin>
 CHAR
 <degismi>
 |
 <getblok> , @ <giris_koordinat> GET
 <karakterdizin> NUM <degismi>**


```

|
<getblok> , @ <koordinat> GET
<karakterdizin> CHAR <degismi>

```

```

<koordinat> ::= ( <positiftamsayi>
                  , <positiftamsayi>
                  )

```

```

<applycumbas> ::= <applybaslik> <cumop> ON
                  <uygulamael>

```

```

<applycumblok> ::= <applycumbas>
                  | <applycumbas>
                  BY <uygulamalist>

```

```

<cumop> ::= SUM
          | MAX
          | MIN
          | AVG
          | PROD

```

```

<applybaslik> ::= APPLY
               | <geneletiket>

```

```

APPLY

```

```

<applyblok> ::= <applybaslik>
               <girsahalist> /
               <relismi>
               \ <cikis>

```

```

<girsahalist> ::= <girsahagird> | <girsahalist> , <girsahagird>

```

```

<girsahagird> ::= <degismi>

```

```

|
<degi> <sahaismi>

```


$\langle \text{cikis_saha_listisi} \rangle ::= \langle \text{cikis_saha_girdisi} \rangle \mid$
 $\langle \text{cikis_saha_listisi} \rangle, \langle \text{cikis_saha_girdisi} \rangle$
 $\langle \text{sahaismi} \rangle$

$\langle \text{cikis_saha_girdisi} \rangle \mid \langle \text{sahaismi} \rangle$
 $\langle \text{sahaismi} \rangle \cdot \text{NUM}$
 $\langle \text{sahaismi} \rangle \cdot \text{CHAR}$

$\langle \text{cikis} \rangle ::= \langle \text{ciksahalist} \rangle$
 $\mid \langle \text{ciksahalist} \rangle : \langle \text{cikdeglist} \rangle$

MEMBER

$\langle \text{cikdeglist} \rangle ::= \langle \text{degismi} \rangle$
 $\mid \langle \text{sahaismi} \rangle$
 $\mid \langle \text{cikdeglist} \rangle, \langle \text{degismi} \rangle$
 $\mid \langle \text{cikdeglist} \rangle, \langle \text{sahaismi} \rangle$

$\langle \text{ciksahalist} \rangle ::= \langle \text{sahaismi} \rangle$
 $\mid \langle \text{ciksahalist} \rangle, \langle \text{sahaismi} \rangle$

$\langle \text{cik_saha_list} \rangle ::= \langle \text{cik_saha_gird} \rangle$
 $\mid \langle \text{cik_saha_list} \rangle$
 $, \langle \text{cik_saha_gird} \rangle$

$\langle \text{cik_saha_gird} \rangle ::= \langle \text{sahaismi} \rangle$
 $\mid$
 $\langle \text{degismi} \rangle$
 $\mid \langle \text{sahaismi} \rangle \cdot \text{NUM}$
 $\mid$
 $\langle \text{degismi} \rangle \cdot \text{NUM}$
 $\mid \langle \text{sahaismi} \rangle \cdot \text{CHAR}$
 $\mid$
 $\langle \text{degismi} \rangle$

$\langle \text{cikis_saha_listesi} \rangle ::= \langle \text{cikis_saha_girdisi} \rangle \mid$
 $\langle \text{cikis_saha_listesi} \rangle, \langle \text{cikis_saha_girdisi} \rangle$

$\langle \text{cikis_saha_girdisi} \rangle ::= \langle \text{sahaismi} \rangle$
 $\mid \langle \text{sahaismi} \rangle . \text{NUM}$
 $\mid \langle \text{sahaismi} \rangle . \text{CHAR}$

$\langle \text{membaslik} \rangle ::= \text{MEMBER}$

$\mid \langle \text{geneletiket} \rangle$

MEMBER

$\langle \text{memblok} \rangle ::= \langle \text{membaslik} \rangle$

$\langle \text{girsahalist} \rangle /$

$\langle \text{relismi} \rangle$

$\langle \text{cartesbaslik} \rangle ::= \text{CARTESIANP}$

$\mid \langle \text{geneletiket} \rangle$

CARTESIANP

$\langle \text{cartesblok} \rangle ::= \langle \text{cartesbaslik} \rangle$

$\langle \text{relismi} \rangle$

$\langle \text{cik_saha_list} \rangle$

$\langle \text{generblok} \rangle ::= \text{GENERATE}$

$\langle \text{relismi} \rangle$

$\langle \text{sahagird} \rangle$

$\langle \text{sahagird} \rangle ::= \langle \text{sahaismi} \rangle$

$\mid \langle \text{sahaismi} \rangle . \text{NUM}$

$\mid \langle \text{sahaismi} \rangle . \text{CHAR}$

$\langle \text{genertblok} \rangle ::= \text{GENERATET}$

<relismi>

<cikis_saha_listesi>

<generebaslik> ::= **GENERATEE**

<genereblok> ::= <generebaslik>

<generelist>

<generelist> ::= <generegirdi> | <generelist> , <generegirdi>

<generegirdi> ::= <degismi> =

<gercel>

| <degismi> =

<karakterdizin>

<reportblok> ::= <repbaslik>

<sayblok>

<repbaslik> ::= **REPORT**

| <geneletiket>

REPORT

<geneletiket> ::= <etiket>

<deg_saha> ::= <degismi>
| <sahaismi>

<sayblok> ::= ? <cikis_koordinat> **SAY** <karakterdizin>

NUM

<deg_saha>

|

? <cikis_koordinat> **SAY** <karakterdizin>

CHAR


```

<degshah>
|
<sayblok> , ? <cikis_koordinat> SAY
                                <karakterdizin>

NUM
<degshah>
|
<sayblok> , ? <cikis_koordinat> SAY
                                <karakterdizin>

CHAR
<degshah>

<cikis_koordinat> ::= [ <positiftamsayi>
                        , <positiftamsayi>
                        ]

<testbaslik> ::= TEST

| <geneletiket>

TEST

<testblok> ::= <testbaslik>
              <mantgrup>
              <hedefetiket>

<hedefetiket> ::= <etiket>
                ,
                <etiket>
                | <etiket>

```


<comfixbaslik> ::= **COMFIX**

| <geneletiket>

COMFIX

<comfixblok> ::= <comfixbaslik>

<gruplar>

<computebaslik> ::= **COMPUTE**

| <geneletiket>

COMPUTE

<computeblok> ::= <computebaslik>

<gruplar>

<mantgrup> ::= <iliskilendirme> <aramagrubu> <mantiksata>

| <iliskilendirme> <mantiksata>

<gruplar> ::= <iliskilendirme> <ortgrup> <cikgrup>

| <iliskilendirme> <ortgrup>

<ortgrup> ::= <aramagrubu> <atamagrup>

| <atamagrup>

<iliskilendirme> ::= <iliski> | <iliskilendirme> , <iliski>

<iliski> ::= <sahaismi>

|

<degismi>

|

<degismi>


```

<aramagrubu> ::= ASSIGN <degismi>
|
<degismi>
<aramagrubu> ::= LOCATE <relismi>
|
<aramagrubu> , <aramagrubu> , <aramagrubu>
<aramagrubu> ::= <sahaismi>
=
<sahaismi>
|
<sahaismi>
=
<degismi>
|
<sahaismi>
=
<degismi>
<gercel>
|
<sahaismi>
=
<aramagrubu> & <aramagrubu>
<aramagrubu> ::= <aramagrubu> , <aramagrubu> , <aramagrubu>
|
<sahaismi>
<mantiksata> ::= ASSIGN <degismi>
=
<mantifade>

```


$\langle \text{atamagrup} \rangle ::= \text{ASSIGN } \langle \text{degismi} \rangle$

$= \langle \text{ifade} \rangle$

| $\text{ASSIGN } \langle \text{sahaismi} \rangle$

$= \langle \text{ifade} \rangle$

| $\langle \text{atamagrup} \rangle ,$

$\langle \text{degismi} \rangle$

$= \langle \text{ifade} \rangle$

| $\langle \text{atamagrup} \rangle ,$

$\langle \text{sahaismi} \rangle$

$= \langle \text{ifade} \rangle$

$\langle \text{cikgrup} \rangle ::= \text{OUTPUT}$

$\langle \text{degismi} \rangle$

| $\langle \text{cikgrup} \rangle ,$

$\langle \text{degismi} \rangle$

$\langle \text{mantifade} \rangle ::= ! \langle \text{andorifade} \rangle$

| $\langle \text{andorifade} \rangle$

$\langle \text{andorifade} \rangle ::= \langle \text{andorifade} \rangle \ \&\& \ \langle \text{karsifadesi} \rangle$

| $\langle \text{andorifade} \rangle \ || \ \langle \text{karsifadesi} \rangle$

| $\langle \text{andorifade} \rangle \ \&\& \ \langle \text{kardizinif} \rangle$

| $\langle \text{andorifade} \rangle \ || \ \langle \text{kardizinif} \rangle$

| $(\langle \text{kardizinif} \rangle)$

| $(\langle \text{karsifadesi} \rangle)$

| $\langle \text{kardizinif} \rangle$

| $\langle \text{karsifadesi} \rangle$

$\langle \text{kardizininif} \rangle ::= \langle \text{karakterdizin} \rangle == \langle \text{degismi} \rangle$
 $| \langle \text{karakterdizin} \rangle != \langle \text{degismi} \rangle$

$\langle \text{karsifadesi} \rangle ::= \langle \text{ifade} \rangle > \langle \text{ifade} \rangle$

$| \langle \text{ifade} \rangle < \langle \text{ifade} \rangle$

$| \langle \text{ifade} \rangle >= \langle \text{ifade} \rangle$

$| \langle \text{ifade} \rangle <= \langle \text{ifade} \rangle$

$| :: \langle \text{ifade} \rangle != \langle \text{ifade} \rangle$

$| \langle \text{ifade} \rangle == \langle \text{ifade} \rangle$

$\langle \text{ifade} \rangle ::= \langle \text{ifade} \rangle +$

$\langle \text{carpifadesi} \rangle$

$| \langle \text{ifade} \rangle -$

$\langle \text{carpifadesi} \rangle$

$| \langle \text{carpifadesi} \rangle$

$\langle \text{carpifadesi} \rangle ::= \langle \text{carpifadesi} \rangle *$

$\langle \text{ilk} \rangle$

$| \langle \text{carpifadesi} \rangle /$

$\langle \text{ilk} \rangle$

$| \langle \text{ilk} \rangle$

$\langle \text{ilk} \rangle ::= (\langle \text{ifade} \rangle)$

$| - \langle \text{ilk} \rangle$

$| \langle \text{degismi} \rangle$

<sahaismi>

<posgercel>

FILTER

<sinkbaslik> ::= SINK

| <geneletiket>

SINK

<uygulamalist> ::= <uygulamasi>

<sinkblok> ::= <sinkbaslik>

<uygulamalist>, <uygulamasi>

<sortbaslik> ::= SORT

| <geneletiket>

SORT

<transferbaslik> ::= TRANSFER

<sortblok> ::= <sortbaslik>

<uygulamalist>

TRANSFER

<countbaslik> ::= COUNT

<transferblok> ::= <transferbaslik>

| <geneletiket>

<etiketlist>

COUNT

<condblok> ::= CONDBLOK

<countblok> ::= <countbaslik>

<uygulamalist>

<etiket>

<txsbaslik> ::= TXSAY

| <geneletiket>

<transferbaslik> ::= TRANSFER

TXSAY

<txcblok> ::= <txsbaslik>

<etiket>

INF



$\langle \text{filtbaslik} \rangle ::= \text{FILTER}$

| $\langle \text{geneletiket} \rangle$

$\langle \text{filtbaslik} \rangle ::= \text{FILTER}$

$\langle \text{filtblok} \rangle ::= \langle \text{filtbaslik} \rangle$

$\langle \text{uygulamalist} \rangle$

$\langle \text{uygulamalist} \rangle ::= \langle \text{uygulamael} \rangle$

| $\langle \text{uygulamalist} \rangle, \langle \text{uygulamael} \rangle$

$\langle \text{uygulamael} \rangle ::= \langle \text{sahaismi} \rangle$

| $\langle \text{degismi} \rangle$

$\langle \text{transbaslik} \rangle ::= \text{TRANSFER}$

| $\langle \text{geneletiket} \rangle$

TRANSFER

$\text{transblok} ::= \langle \text{transbaslik} \rangle$

$\langle \text{etiketlist} \rangle$

$\langle \text{condblok} \rangle ::= \text{CONDBLOK}$

$\langle \text{etiket} \rangle$

$\langle \text{etiket} \rangle$

$\langle \text{sartifade} \rangle$

$\langle \text{sartifade} \rangle ::= \text{REF}$

$\langle \text{degsahalist} \rangle$

$\langle \text{sart} \rangle$

INP

$\langle \text{degsahalist} \rangle ::= \langle \text{degismi} \rangle$

| $\langle \text{sahaismi} \rangle$

$\langle \text{sart} \rangle ::= >$

| $<$

| $>=$

| $<=$

| $\neq$

| $=$

$\langle \text{mergeblok} \rangle ::= \text{MERGE}$

$\langle \text{etiketlist} \rangle$

$\langle \text{etiketlist} \rangle ::= \langle \text{etiket} \rangle$

| $\langle \text{etiketlist} \rangle, \langle \text{etiket} \rangle$

$\langle \text{updtbaslik} \rangle ::= \text{APPEND}$

| $\langle \text{geneletiket} \rangle$

APPEND

$\langle \text{updtblok} \rangle ::= \langle \text{updtbaslik} \rangle$

$\langle \text{aramagrubu} \rangle$

$\langle \text{delbaslik} \rangle ::= \text{DELETE}$

| $\langle \text{geneletiket} \rangle$

DELETE

$\langle \text{delblok} \rangle ::= \langle \text{delbaslik} \rangle$

$\langle \text{aramagrubu} \rangle$

$\langle \text{positiftamsayi} \rangle ::= \langle \text{dijit} \rangle$
 $\quad \quad \quad | \quad \langle \text{positiftamsayi} \rangle \quad \langle \text{dijit} \rangle$

$\langle \text{negatiftamsayi} \rangle ::= - \quad \langle \text{positiftamsayi} \rangle$

$\langle \text{posgercel} \rangle ::= \langle \text{positiftamsayi} \rangle .$

$\quad \quad \quad \langle \text{positiftamsayi} \rangle$

$\langle \text{neggercel} \rangle ::= \langle \text{negatiftamsayi} \rangle .$

$\quad \quad \quad \langle \text{positiftamsayi} \rangle$

$\langle \text{gercel} \rangle ::= \langle \text{posgercel} \rangle \quad | \quad \langle \text{neggercel} \rangle$

(EBNF Notasyonunda)

$\langle \text{relismi} \rangle ::= R_ (A \mid B \mid C \mid \dots \mid Z)$
 $\quad \quad \quad \{ A \mid B \mid C \mid \dots \mid Z \mid 0 \mid 1 \mid \dots \mid 9 \}$

$\langle \text{degismi} \rangle ::= _ (A \mid B \mid C \mid \dots \mid Z)$
 $\quad \quad \quad \{ a \mid b \mid c \mid \dots \mid z \mid 0 \mid 1 \mid \dots \mid 9 \}$

$\langle \text{sahaismi} \rangle ::= _ (a \mid b \mid c \mid \dots \mid z)$
 $\quad \quad \quad \{ a \mid b \mid c \mid \dots \mid z \mid 0 \mid 1 \mid \dots \mid 9 \}$

$\langle \text{karakterdizin} \rangle ::= "(A \mid B \mid C \mid \dots \mid Z$
 $\quad \quad \quad | a \mid b \mid c \mid \dots \mid z \mid 0 \mid 1 \mid \dots \mid 9)$
 $\quad \quad \quad \{ A \mid B \mid C \mid \dots \mid Z$
 $\quad \quad \quad | a \mid b \mid c \mid \dots \mid z \mid 0 \mid 1 \mid \dots \mid 9 \}$

$\langle \text{etiket} \rangle ::= (A \mid B \mid C \mid \dots \mid Z)$
 $\quad \quad \quad \{ A \mid B \mid C \mid \dots \mid Z \}$



SÖZ DİZİMİ KURALLARININ EXTENDED BACKUS NAUR FORMDA TANIMI

```
program ::= blok {; blok } #
```

```
/* ----- */
```

```
blok ::= inputblok
```

```
      applyblok
```

```
      generblok
```

```
      genertblok
```

```
      genereblok
```

```
      reportblok
```

```
      computeblok
```

```
      comfixblok
```

```
      sortblok
```

```
      transblok
```

```
      countblok
```

```
      filtblok
```

```
      sinkblok
```

```
      updtblok
```

```
      testblok
```

```
      delblok
```

```
      mergeblok
```

```
      applycumblok
```

```
      txcblok
```

```
      condblok
```

```
      memblok
```

```
geneletiket ::= etiket
```

```
inputblok  ::= INPUT
```

```
      @  giris_koordinat  GET
```

```
          baslik [NUM | CHAR | degismi
```

```
      {, @  giris_koordinat  GET
```

```
          baslik [NUM | CHAR | degismi}
```

```
reportblok ::= ( REPORT | geneletiket  REPORT )
```

```
      ?  cikis_koordinat  SAY  baslik
```

```
          (NUM | CHAR) (degismi | sahaismi)
```

```
      {, ?  cikis_koordinat  SAY  baslik
```

```
          (NUM | CHAR) (degismi | sahaismi) }
```

```
baslik      ::= karakterdizin
```



```

cikis_koordinat ::= [ satirno
                        , sutunno
                    ]

giris_koordinat ::= ( satirno .CHAR | degismi |
                        , sutunno
                      )

satirno ::= positiftamsayi

sutunno ::= positiftamsayi

applycumblok ::= (APPLY | geneletiket APPLY)
                (SUM | MAX | MIN | AVG | PROD ) ON
                (sahaismi | degismi)
                [ BY
                  (sahaismi | degismi) {, (sahaismi | degismi
                )}]

applyblok ::= (APPLY | geneletiket APPLY)
              ( degismi | sahaismi ) | sahaismi
              {, ( degismi | sahaismi ) | sahaismi}
              /
              relismi

              \ sahaismi {, sahaismi }
              [:( sahaismi | degismi | sahaismi .NUM |
                  degismi
                  | sahaismi .CHAR | degismi )
              {, ( sahaismi | degismi |
                  sahaismi .NUM | degismi
                  | sahaismi .CHAR | degismi )
              }

memblok ::= (MEMBER | geneletiket MEMBER )
            (degismi | sahaismi) | sahaismi
            {, (degismi | sahaismi) | sahaismi}
            /
            relismi

```


cartesblok ::= (CARTESIANP | geneletiket CARTESIANP)

relismi
 (sahaismi | degismi | sahaismi .NUM |
 degismi
 | sahaismi .CHAR | degismi |
 {, (sahaismi | degismi |
 sahaismi .NUM | degismi
 | sahaismi .CHAR | degismi)
 }
)

generblok ::= GENERATE relismi

(sahaismi | sahaismi.NUM |
 sahaismi.CHAR)

genertblok ::= GENERATET relismi

(sahaismi | sahaismi.NUM | sahaismi.CHAR)
 {, (sahaismi | sahaismi.NUM |
 sahaismi.CHAR)}

genereblok ::= GENERATEE

(degismi = gercel | degismi = karakterdizin)
 {, (degismi = gercel | degismi =
 karakterdizin)
 }

updtblok ::= (APPEND | geneletiket APPEND)
 aramagrup

delblok ::= (DELETE | geneletiket DELETE)
 aramagrup

testblok ::= (TEST | geneletiket TEST)

iliskilendirme

[aramagrup]

ASSIGN degismi = mantifade

(etiket , etiket | etiket)

comfixblok ::= (COMFIX | geneletiket COMFIX)

iliskilendirme

[aramagrup]

ASSIGN (degismi | sahaismi)

= ifade

{, (degismi | sahaismi) = ifade) }

[OUTPUT

degismi

{, degismi }

]

computeblok ::= (COMPUTE | geneletiket COMPUTE)

iliskilendirme

[aramagrup]

ASSIGN degismi = ifade

{, degismi = ifade) }

[OUTPUT

degismi

{, degismi }

]

iliskilendirme ::= (sahaismi | degismi | degismi | degismi)

{, (sahaismi | degismi | degismi | degismi)}

aramagrup ::= LOCATE relismi (sahaismi = sahaismi |

sahaismi = degismi

| sahaismi = karakterdizin

| sahaismi = gercel

| sahaismi)

{, (sahaismi = sahaismi |

sahaismi = degismi

| sahaismi = karakterdizin

| sahaismi = gercel

| sahaismi)

}

{, LOCATE relismi (sahaismi = sahaismi |

sahaismi = degismi

| sahaismi = karakterdizin


```

sahaismi = gercel
sahaismi ) degismi ) ( , (sahaismi | degismi) )

{ , (sahaismi = sahaismi
  sahaismi = degismi
  sahaismi = karakterdizin
  sahaismi = gercel
  sahaismi )
}
} CONDBLOK

```

mantifade ::= [!] andorifade

```

andorifade ::= (karsifadesi | kardizinifr | ( karsifadesi ) |
  ( kardizinif ) )
  { ( && | || )
    (karsifadesi | kardizinif |
      ( karsifadesi ) | ( kardizinif ))}

```

kardizinif ::= karakterdizin (== | !=) degismi

karsifadesi ::= ifade (> | < | >= | <= | != | ==) ifade

ifade ::= carpifadesi (+ | -) carpifadesi

carpifadesi ::= temelifade (* | /) temelifade

temelifade ::= ((ifade) | - ilk | degismi | sahaismi
| posgercel)

sinkblok ::= (SINK | geneletiket SINK)

sortblok ::= (SORT | geneletiket SORT)
(sahaismi | degismi) { , (sahaismi|degismi) }

countblok ::= (COUNT | geneletiket COUNT)
(sahaismi | degismi) { , (sahaismi|degismi) }

txcblok ::= (TXSAY | geneletiket TXSAY)

filtblok ::= (FILTER | geneletiket FILTER)
(sahaismi | degismi) {, (sahaismi | degismi) }

transblok ::= (TRANSFER | geneletiket TRANSFER)
etiket {, etiket }

condblok ::= CONDBLOK
etiket
etiket
REF (degismi | sahaismi)
(> | < | >= | <= | != | ==)
INP

mergeblok ::= MERGE etiket {, etiket }

gercel ::= posgercel | neggercel

posgercel ::= positiftamsayi . positiftamsayi

neggercel ::= negatiftamsayi . positiftamsayi

positiftamsayi ::= dijit {dijit}

negatiftamsayi ::= - positiftamsayi

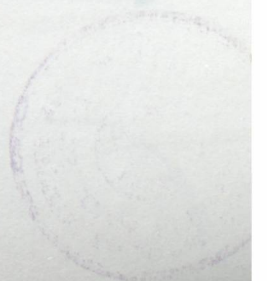
relismi ::= R_ (A | B | C | | Z)
{ A | B | C | | Z | 0 | 1 | | 9 }

degismi ::= _ (A | B | C | | Z)
{ a | b | c | | z | 0 | 1 | | 9 }

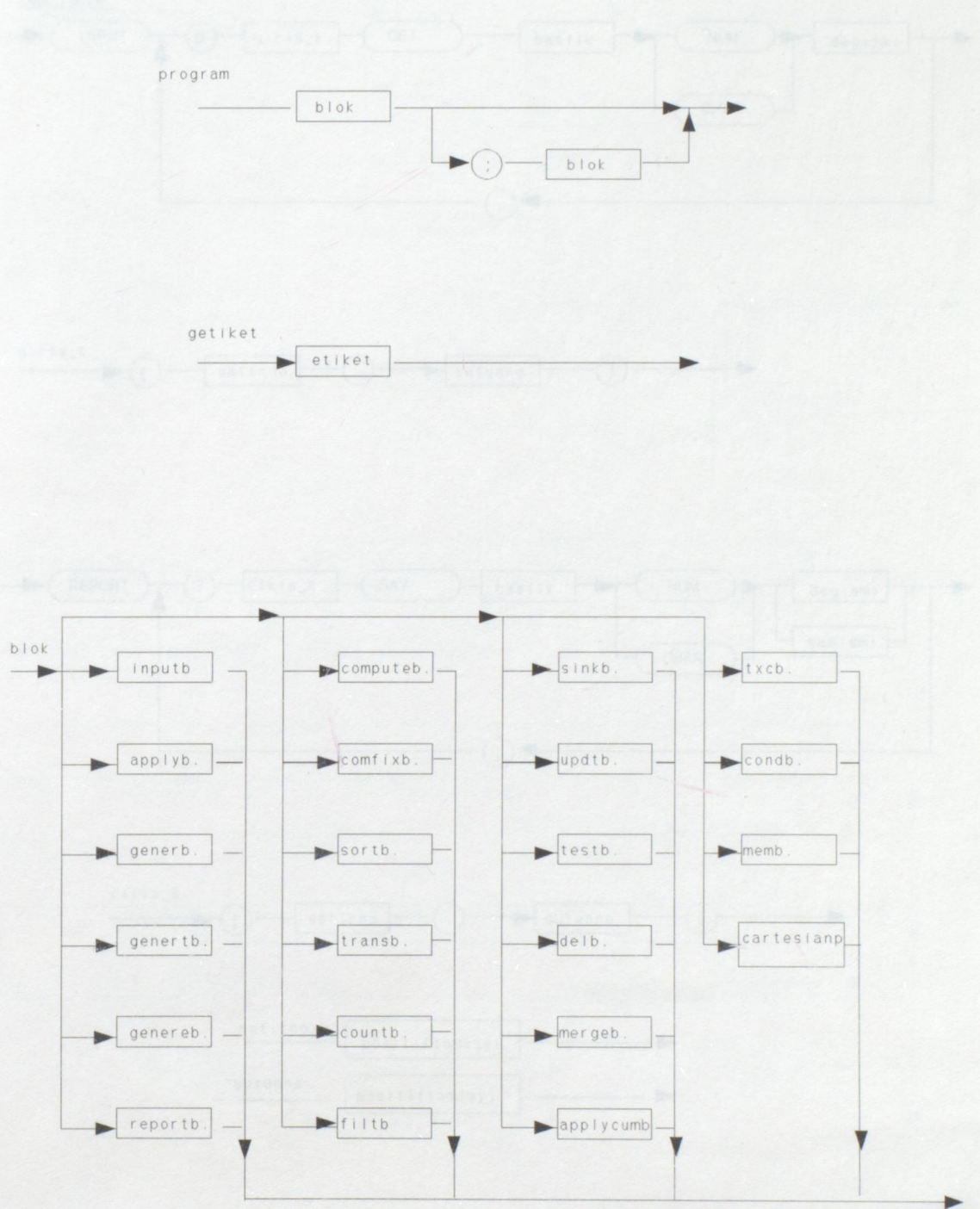
sahaismi ::= _ (a | b | c | | z)
{ a | b | c | | z | 0 | 1 | | 9 }

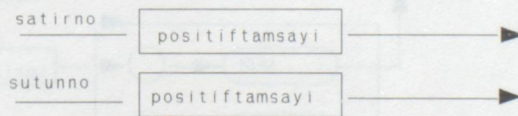
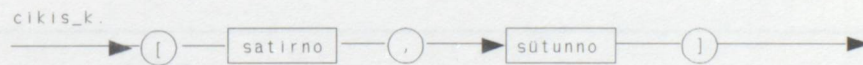
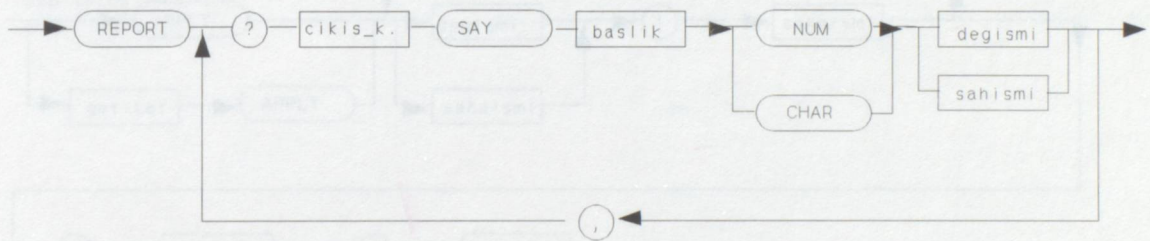
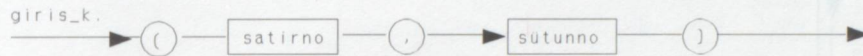
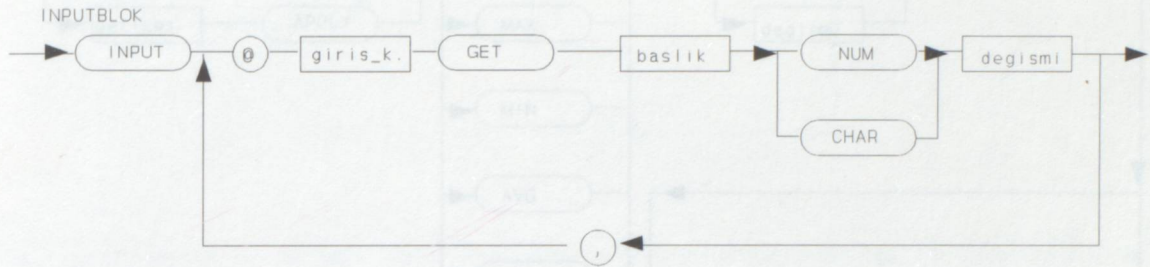
karakterdizin ::= "(A | B | C | | Z
| a | b | c | | z | 0 | 1 | | 9)
{ A | B | C | | Z
| a | b | c | | z | 0 | 1 | | 9 }

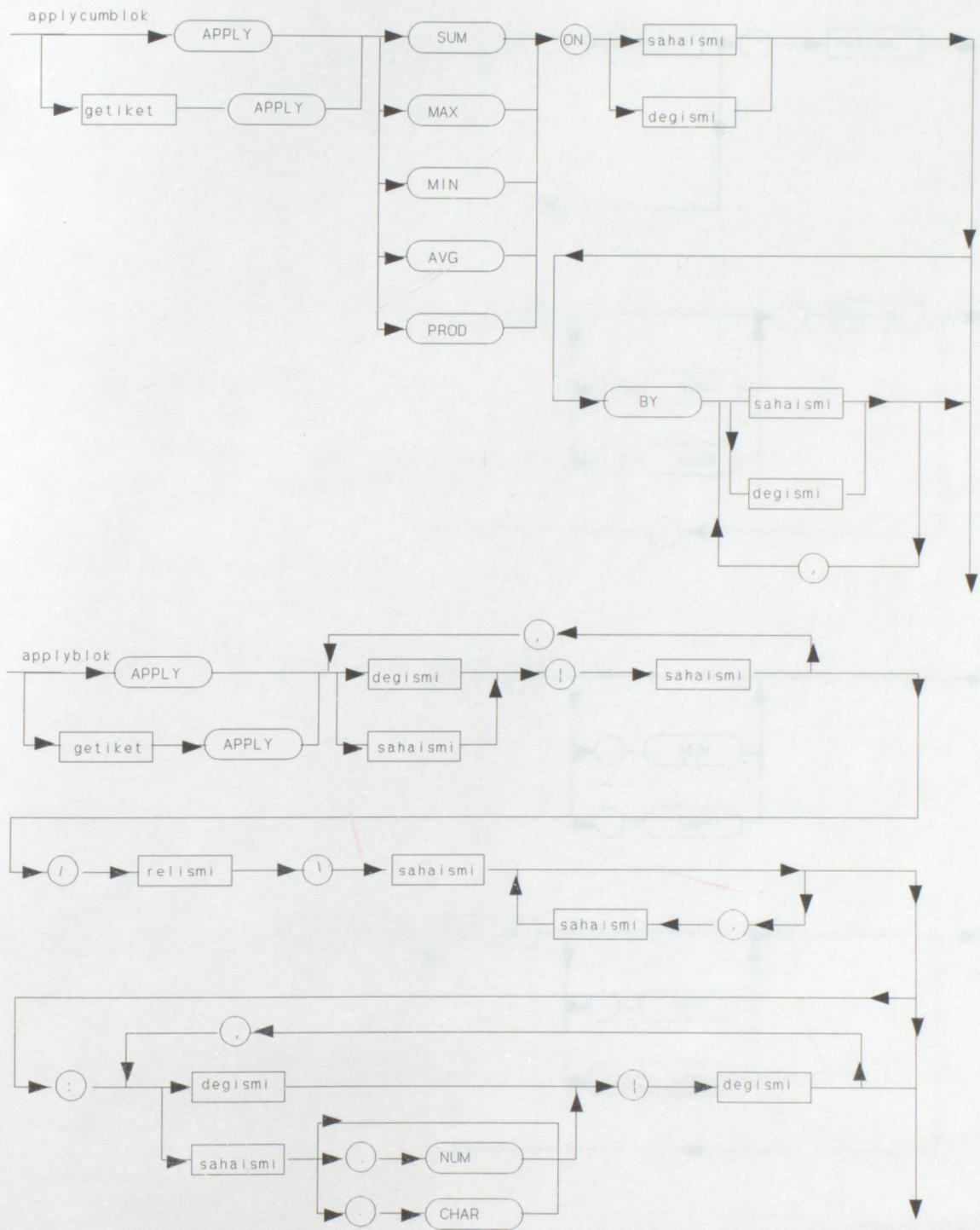
etiket ::= (A | B | C | | Z)
 { A | B | C | | Z }



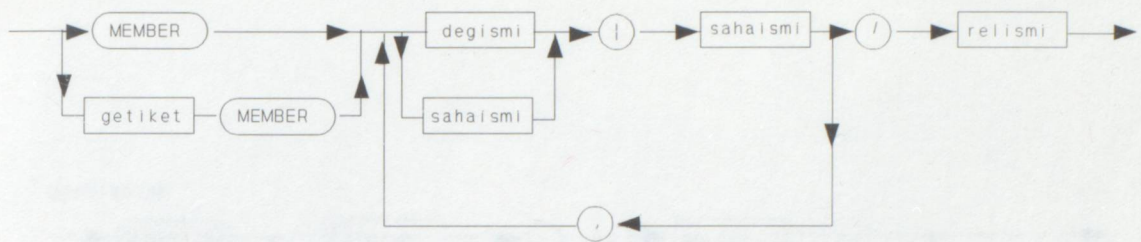
SÖZ DİZİMİ KURALLARININ DİAGRAM OLARAK GÖSTERİLİŞİ



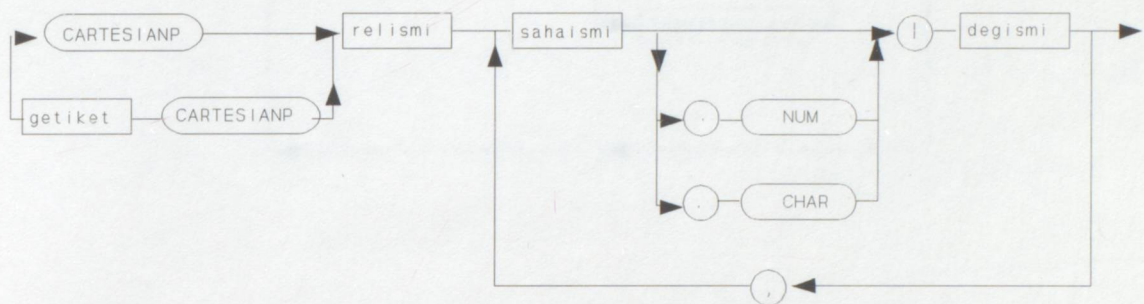




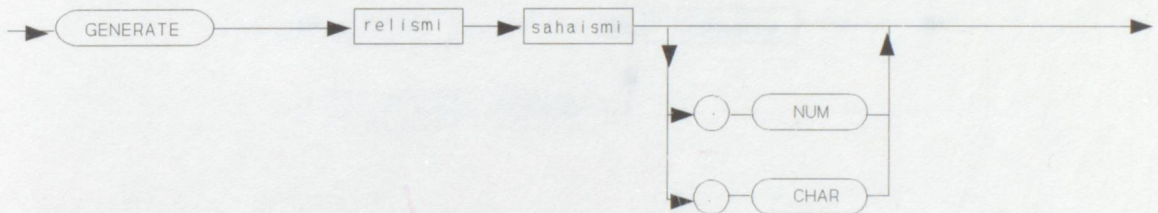
memblok



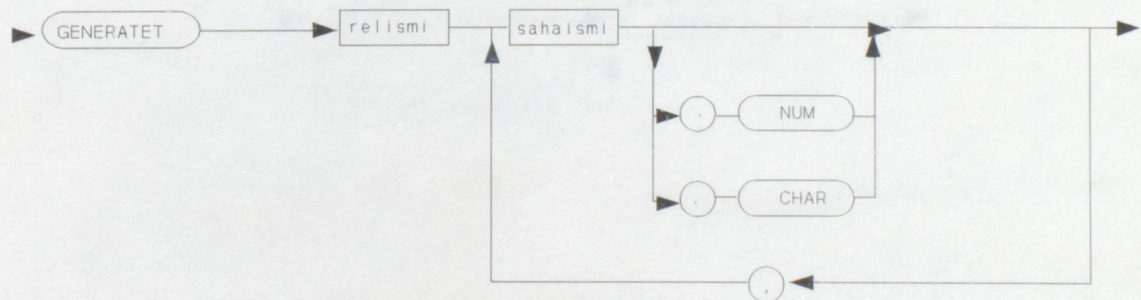
cartesblok

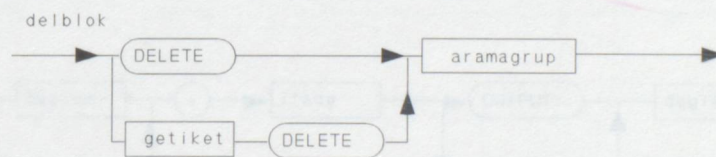
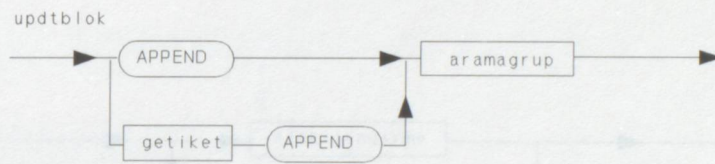
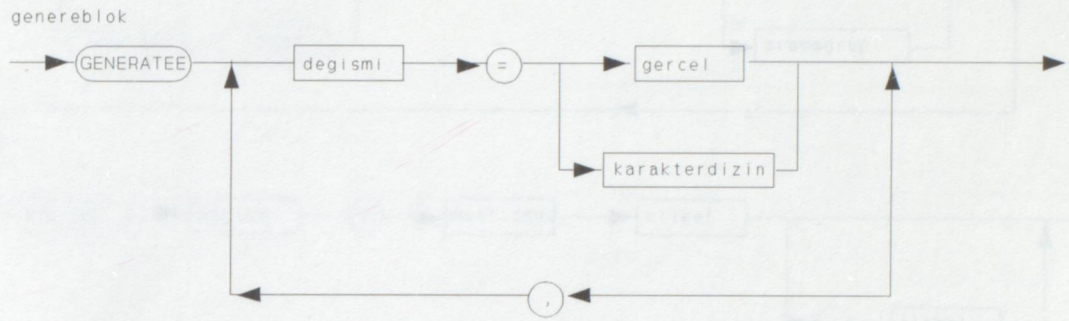


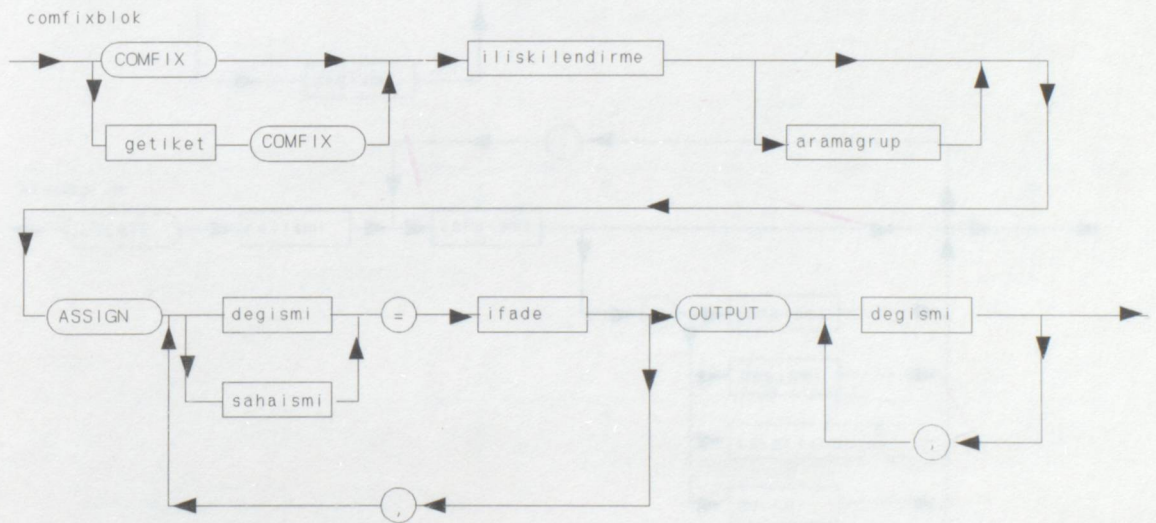
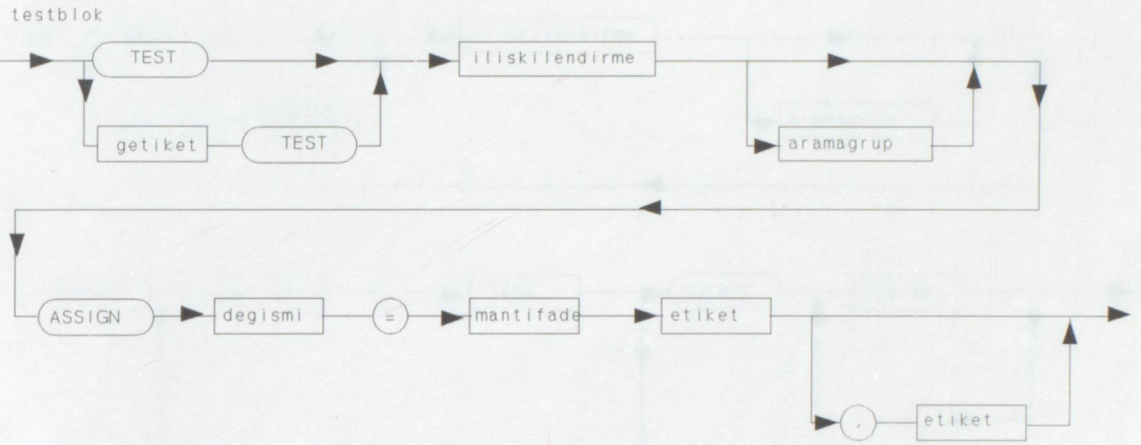
generblok

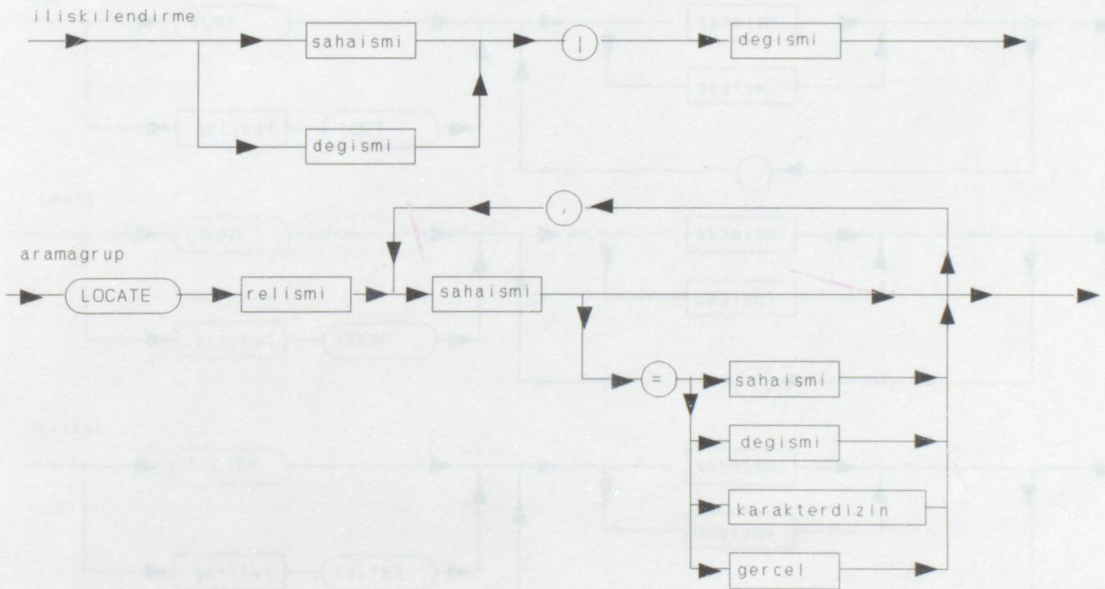
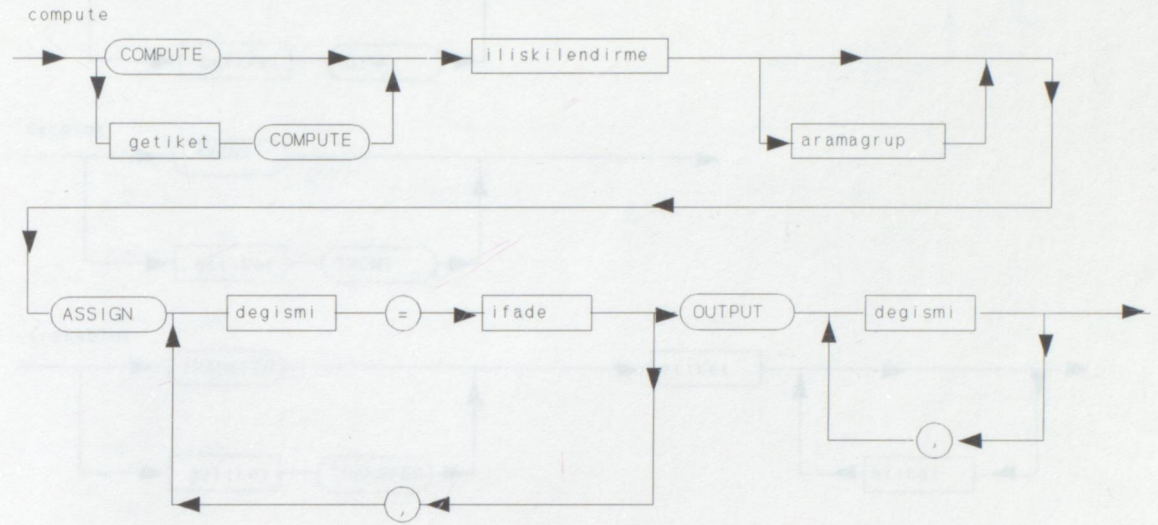


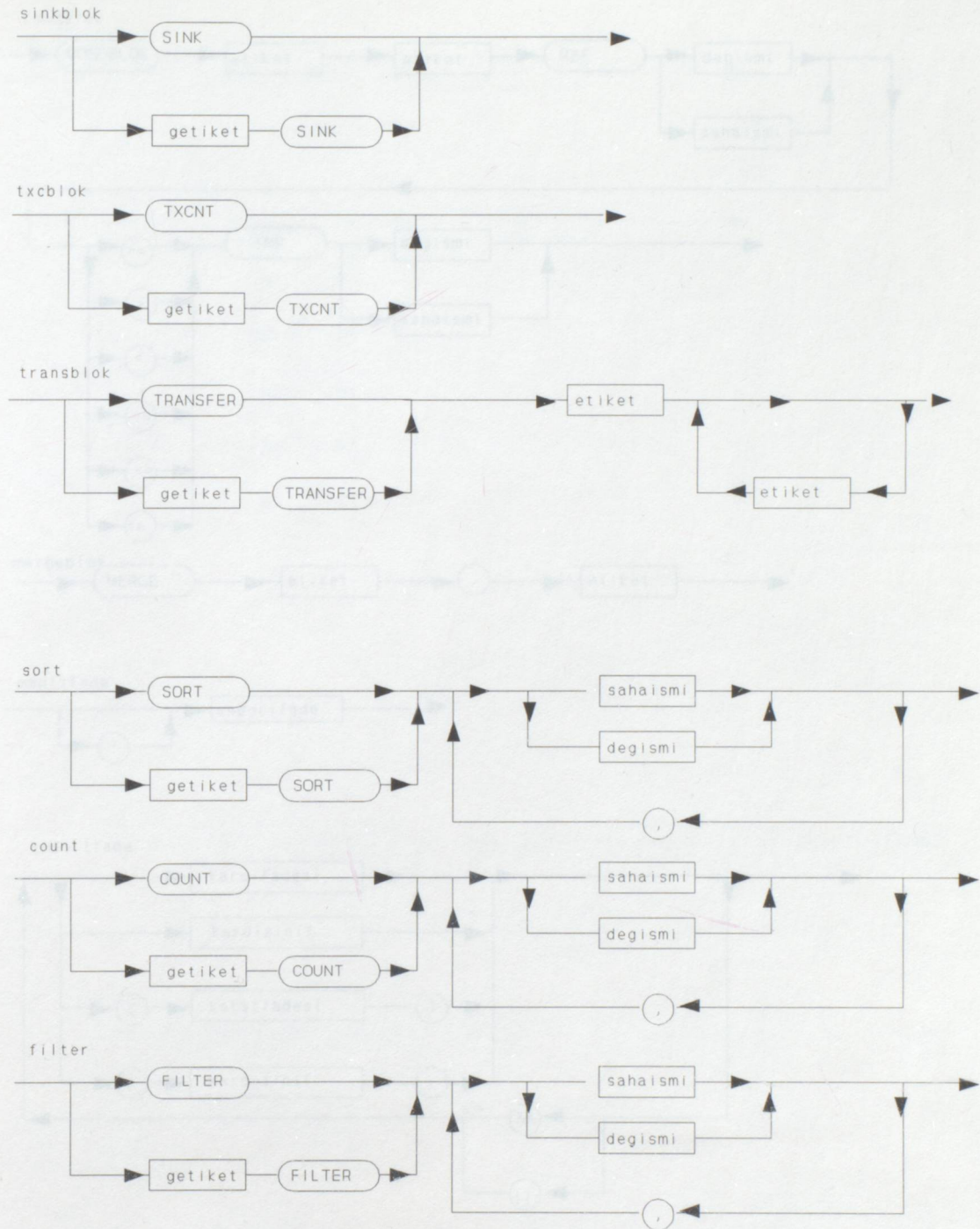
genertblok

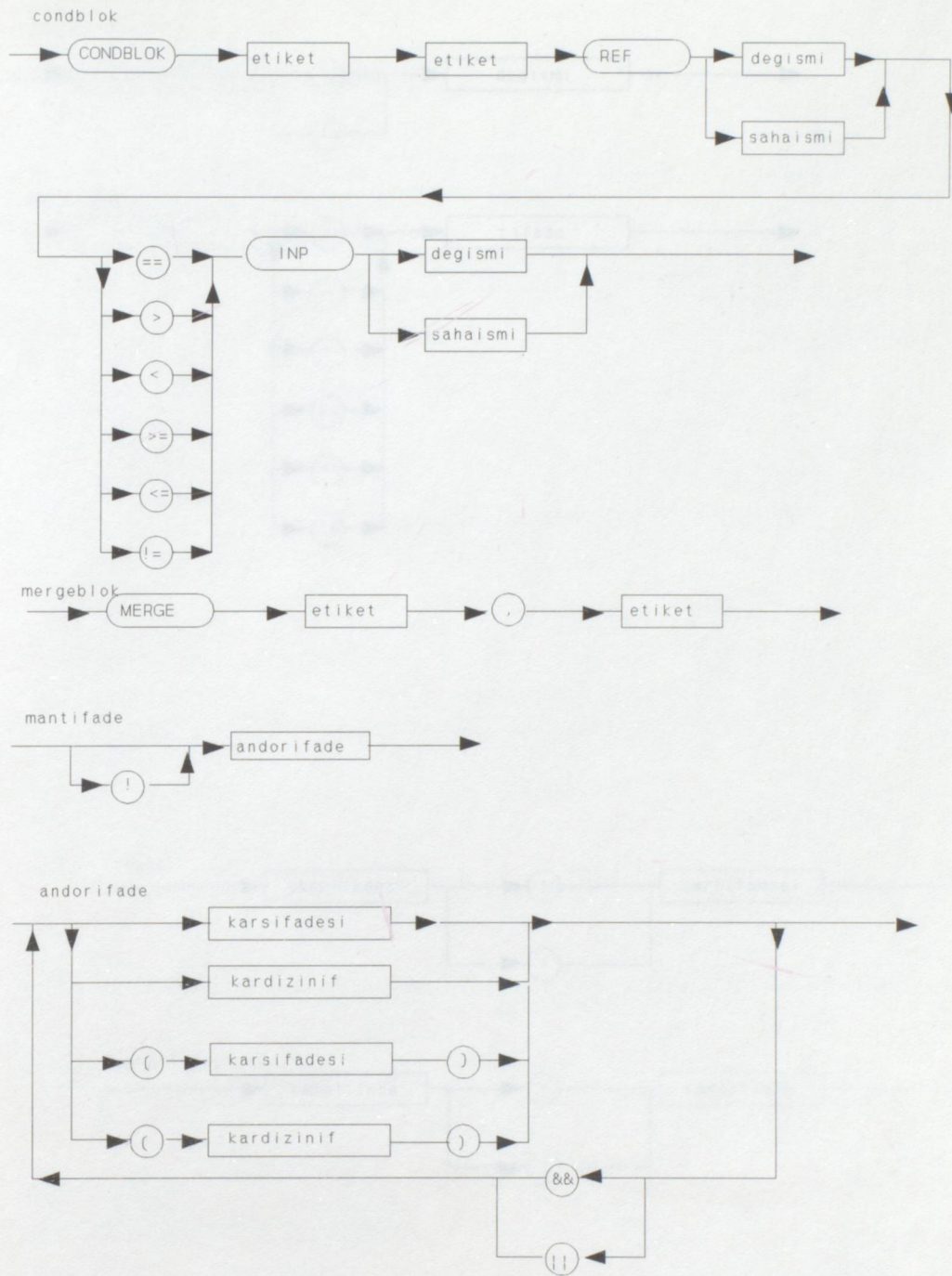


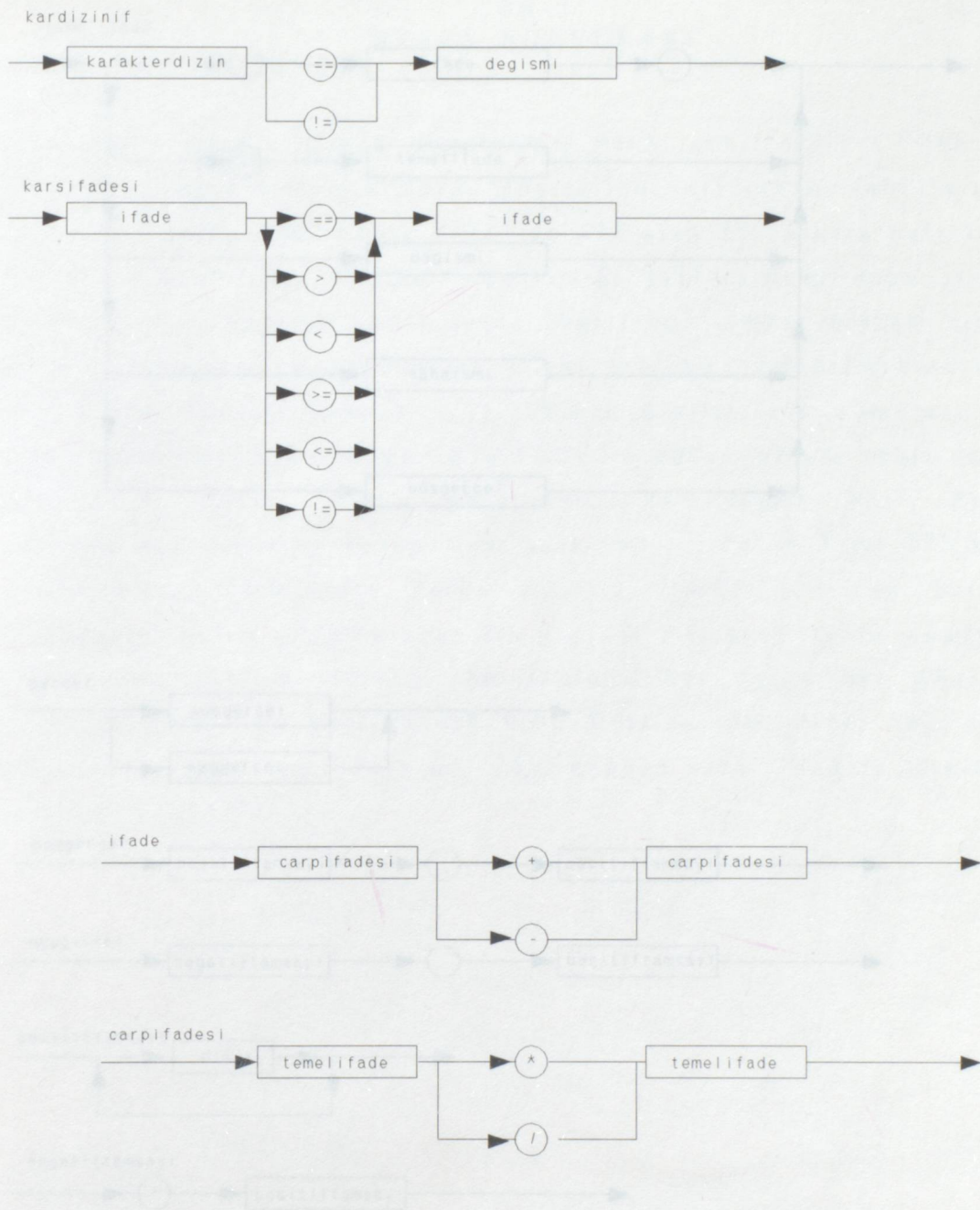






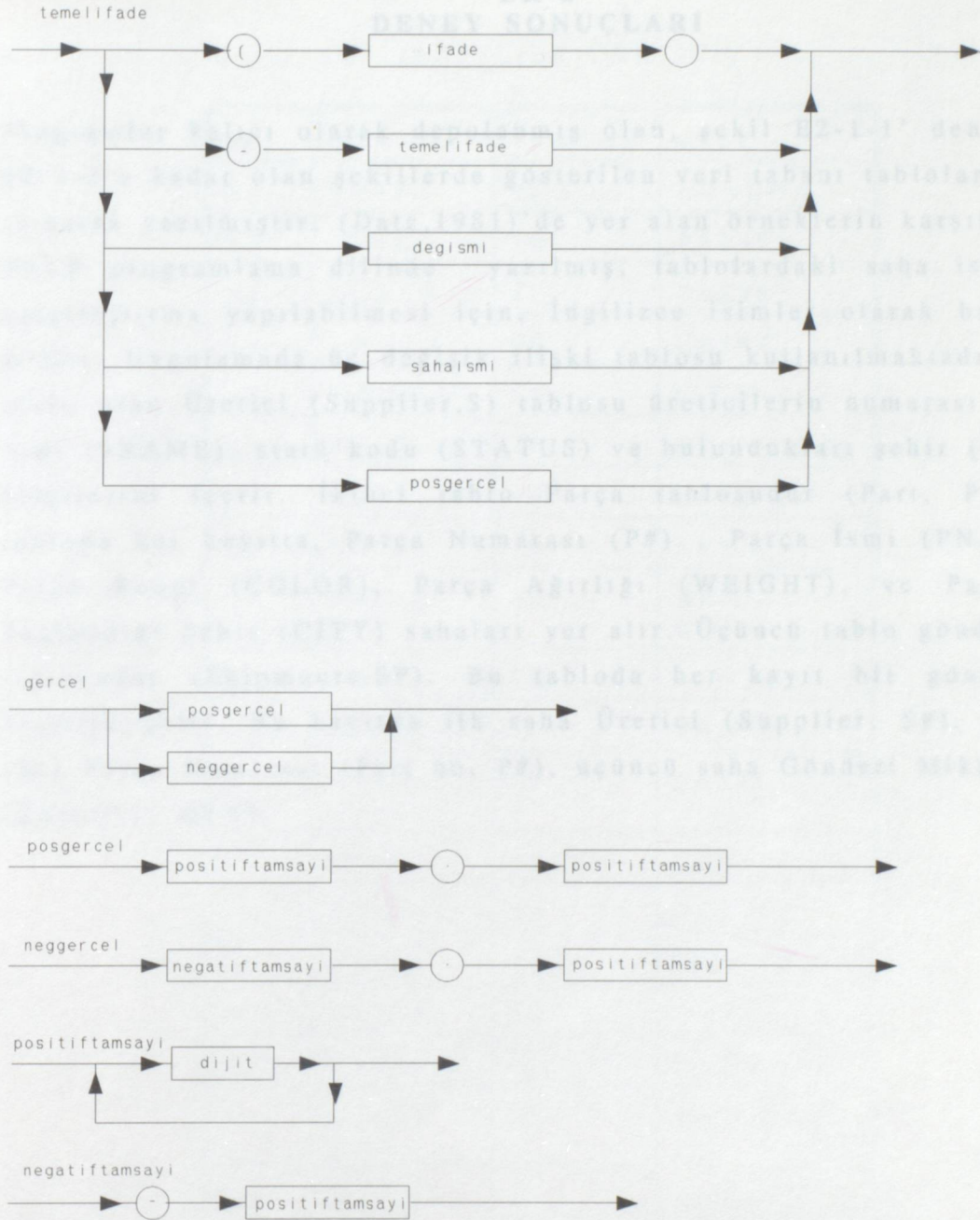






EK-3

DENEY SONUÇLARI



EK-2 DENEY SONUÇLARI

Programlar kalıcı olarak depolanmış olan, şekil E2-1-1' den şekil E2-1-3'e kadar olan şekillerde gösterilen veri tabanı tabloları esas alınarak yazılmıştır. (Date,1981)'de yer alan örneklerin karşılıkları RELP programlama dilinde yazılmış, tablolardaki saha isimleri karşılaştırma yapılabilmesi için, İngilizce isimler olarak bırakılmıştır. Uygulamada üç değişik ilişki tablosu kullanılmaktadır. İlk tablo olan Üretici (Supplier,S) tablosu üreticilerin numarası (S#), ismi (SNAME), statü kodu (STATUS) ve bulundukları şehir (CITY) bilgilerini içerir. İkinci tablo Parça tablosudur (Part, P). Bu tabloda her kayıta, Parça Numarası (P#) , Parça İsmi (PNAME), Parça Rengi (COLOR), Parça Ağırlığı (WEIGHT), ve Parçanın Sağlandığı Şehir (CITY) sahaları yer alır. Üçüncü tablo gönderiler tablosudur (Shipments,SP). Bu tabloda her kayıt bir gönderiye karşılık gelir. Bu kayıtda ilk saha Üretici (Supplier, S#), ikinci saha Parça Numarası (Part no, P#), üçüncü saha Gönderi Miktarıdır (Quantity, QTY).

S#	SNAME	STATUS	CITY
S13	JEVNE	25	BRUSSEL
S14	NOMA	27	PARIS
S15	ORUC	20	ISTANBUL
S16	BRSOY	30	ANTALYA
S17	KALMAN	40	HAMBURG
S18	BART- MAN	35	MUNICH
S19	AHMET	35	ANKARA
S20	ABBAS	25	BAGHDAT

Şekil E2.1 Üreticiler ilişkisi tablosu.

S (SUPPLIER-ÜRETİCİ)

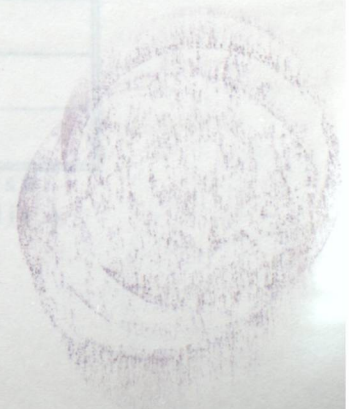
S#	SNAME	STATUS	CITY
S1	SMITH	20	LONDON
S2	JONES	10	PARIS
S3	BLAKE	30	PARIS
S4	AHMET	20	CAIRO
S5	MEHMET	10	ISTANBUL
S6	BOOM	30	NEWYORK
S7	GLYN	20	LONDON
S8	LOUIS	10	PARIS
S9	SMITH	30	HONG-KONG
S10	ANN	20	LONDON
S11	JASON	10	TAIPEI
S12	FRAN-COIS	30	PARIS
S13	JEVNE	25	BRUSSEL
S14	NOMA	22	PARIS
S15	ORUC	20	ISTANBUL
S16	ERSOY	30	ANTALYA
S17	KALMAN	40	HAMBURG
S18	HART-MAN	35	MUNICH
S19	AHMET	35	ANKARA
S20	ABBAS	25	BAGHDAT

Şekil E2.1 Üreticiler ilişkisi tablosu.

P (PARÇA - PART)

P#	PNAME	COLOR	WEIGHT	CITY
P1	NUT	RED	12	LONDON
P2	BOLT	GREEN	17	PARIS
P3	SCREW	BLUE	17	ROME
P4	SCREW	RED	14	LONDON
P5	NUT	RED	12	LONDON
P6	BOLT	GREEN	17	PARIS
P7	SCREW	BLUE	17	ROME
P8	SCREW	RED	14	LONDON
P9	NUT	RED	12	LONDON
P10	BOLT	GREEN	17	PARIS
P11	SCREW	BLUE	17	ROME
P12	SCREW	RED	14	LONDON
P13	NUT	RED	12	LONDON
P14	BOLT	GREEN	17	PARIS
P15	SCREW	BLUE	17	ROME
P16	SCREW	RED	14	LONDON
P17	NUT	RED	12	LONDON
P18	BOLT	GREEN	17	PARIS
P19	SCREW	BLUE	17	ROME
P20	SCREW	RED	14	LONDON

Şekil E2.2 Parçalar ilişki tablosu.



SP (SHIPMENTS-GÖNDERİLER)

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S2	P1	300
S2	P2	400
S3	P2	200
S4	P4	300
S4	P5	200
S4	P6	400
S5	P4	300
S5	P5	400
S6	P5	200
S7	P7	300
S7	P8	200
S7	P9	400
S8	P7	300
S8	P8	400
S9	P8	200
S10	P10	300
S10	P11	200
S10	P12	400
S11	P10	300
S11	P11	400
S12	P11	200
S13	P13	300
S13	P14	200
S13	P15	400

Şekil E2.3 "Gönderiler" ilişki tablosu.
(Bir sonraki syafada devam etmektedir.)

SP (SHIPMENTS-GÖNDERİLER)

S14	P13	300
S14	P14	400
S15	P14	200
S16	P10	300
S16	P11	400
S17	P11	200
S17	P13	300
S18	P14	200
S19	P15	400

Şekil E2.3 "Gönderiler" ilişki tablosu (Devam)

PROGRAM AÇIKLAMASI : suppliers in PARIS.

(Pariste bulunan üreticilerin üretici numarası(S1) ve

Find city for supplier s1)

(Üretici "S1"in yerleşik olduğu şehri bul)

PROGRAM-1 : City="PARIS;

APPLY City|_city /R_S_snum, _city;

GENERATEE _Sno="S1; _snum CHAR _snum;

APPLY _Sno|_snum /R_S_snum, _city;

REPORT ?[10,15] SAY "sno CHAR _snum,

?[11,15] SAY "city CHAR _city;

#

	İŞLEMÇİ SAYISI			
İŞLEMÇİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	12	8	5	4
GEÇEN ZAMAN MİNİMUM	7	6	4	3
HIZLANMA KATSAYISI İLK	1	1.5	2.4	3
HIZLANMA KATSAYISI MİN	1	1.16	1.75	2.33

DEĞERLENDİRME : bloktan oluşmaktadır. Yani her fiziki

Program sadece üç bloktan oluşmaktadır. Yani her fiziki

işlemci başına bir sanal işlemci aktive edilerek icra

edebilmektedir. Sanal işlemcilere görev mesajlarını gönderen

ana program da dördüncü işlemci üzerinde paralel

çalıştırılabilir. Bu durumda program üç bloktan meydana

gelse dahi, üç işlemci ve dört işlemci performansı aynı

olmayacak , dört işlemci performansı 3 işlemci

performansından daha iyi olacaktır.

Şekil E2.4 Program-1 Deneyi.

PROGRAM AÇIKLAMASI :

Find S# and STATUS for suppliers in PARIS.

(Pariste bulunan üreticilerin üretici numarası(S#) ve statülerini(STATUS) listele)

PROGRAM-2 :

GENERATEE _City="PARIS;

APPLY _City|_city /R_S_snum,_stat,_city;

REPORT ?[10,15] SAY "numara CHAR _snum,

?[11,15] SAY "statu NUM _stat,

?[12,15] SAY "city CHAR _city;

#

	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	12	8	5	4
GEÇEN ZAMAN MİNİMUM	7	6	4	3
HIZLANMA KATSAYISI İLK	1	1.5	2.4	3
HIZLANMA KATSAYISI MİN	1	1.16	1.75	2.33

DEĞERLENDİRME :

Program sadece üç bloktan oluşmaktadır. Yani her fiziki işlemci başına bir sanal işlemci aktive edilerek icra edebilmektedir. Dördüncü işlemci üzerinde çalışan ana program'da paralele girerek dört işlemcili çalıştırmada ilave avantaj sağlamaktadır.

Şekil E2.5 Program-2 deneyi.

PROGRAM AÇIKLAMASI :

Find PNAME for parts supplied by supplier S1.
(S1 üreticisi tarafından sağlanan parçaların Parça İsimlerini (PNAME) listele)

PROGRAM-3 :

```
GENERATEE _Sno="S1;
APPLY _Sno|_snum /R_SP\_pnum;
APPLY _pnum|_pnum /R_P\_pnum,_pname;
REPORT ?[10,15] SAY "parcno CHAR _pnum,
        ?[11,15] SAY "parcismi CHAR _pname;
#
```

	İŞLEMÇİ SAYISI			
İŞLEMÇİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	13	10	6	4
GEÇEN ZAMAN MİNİMUM	6	5	4	3
HIZLANMA KATSAYISI İLK	1	1.3	2.16	3.25
HIZLANMA KATSAYISI MİN	1	1.2	1.5	2

DEĞERLENDİRME :

İlk çalıştırma denemesinde bulunan değerler ile takip eden çalıştırma denemelerinde bulunan değerler farklıdır. Bunun sebebi UNIX System V Release 4. İşletim sisteminin dosya düzeyinde yaptığı disk buffer yönetimi rol oynamaktadır. İlk çalıştırmada , her sanal işlemcinin icra zamanı uzadığında paralel işleme etkisinin daha belirgin olduğu görülmektedir.

Şekil E2.6 Program-3 deneyi.

PROGRAM AÇIKLAMASI :

For each part supplied, find P# and names of all cities supplying the part.
(Sağlanan her bir parça için, parça numarası ve bu parçanın sağlandığı şehirleri listele)

PROGRAM-4 :

```
GENERATE R_SP _pnum ;
FILTER _pnum;
APPLY _pnum|_pnum /R_P\ _pnum, _city;
REPORT ?[10,15] SAY "parcno CHAR _pnum,
        ?[11,15] SAY "parcismi CHAR _city;
#
```

	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	16	13	11	8
GEÇEN ZAMAN MİNİMUM	13	11	10	7
HIZLANMA KATSAYISI İLK	1	1.23	1.45	2
HIZLANMA KATSAYISI MİN	1	1.18	1.3	1.85
DEĞERLENDİRME : GENERATE ve FILTER blokları çok kısa zaman harcamaktadır. Paralel işlemenin avantajı sadece APPLY bloğu ve REPORT bloğunun paralele girmesiyle yaşanmaktadır. Bu nedenle elde edilen maksimum hızlanma katsayısı 2 civarında oluşmaktadır.				

Şekil E2.7 Program-4 deneyi.

PROGRAM AÇIKLAMASI :

Get part numbers for all parts supplied by at least three different suppliers.

(En az üç değişik üretici tarafından sağlanan parçaların numarasını listele)

PROGRAM-5

```
GENERATET R_SP _pnum, _snum ;
```

```
COUNTBY _pnum;
```

```
TEST _pnum|_Pnum, _Cnt|_Cnt
```

```
ASSIGN _R1=_Cnt>=3.0
```

```
AA,BB;
```

```
BB REPORT ?[10,15] SAY "parcno CHAR _Pnum,
```

```
?[11,15] SAY "sayisi NUM _Cnt;
```

```
#
```

	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	36	26	17	12
GEÇEN ZAMAN MİNİMUM	17	15	12	10
HIZLANMA KATSAYISI İLK	1	1.38	2.11	3
HIZLANMA KATSAYISI MİN	1	1.13	1.41	1.7

DEĞERLENDİRME : Tek işlemcide ilk çalıştırma zamanı çok yüksek çıkmakta daha sonraki çalıştırmalarda bu zaman büyük düşüş göstermektedir.

Şekil E2.8 Program-5 deneyi.

PROGRAM AÇIKLAMASI : Get supplier numbers who supply part P2 (Parça P2'yi sağlayan üreticileri listele)				
PROGRAM-6 GENERATEE _Pnum="P2; APPLY _Pnum _pnum/R_SP_snum; FILTER _snum; APPLY _snum _snum/R_S_sname; REPORT ?[10,15] SAY "name CHAR _sname; #				
	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	16	12	10	8
GEÇEN ZAMAN MİNİMUM	9	9	8	6
HIZLANMA KATSAYISI İLK	1	1.33	1.6	2
HIZLANMA KATSAYISI MİN	1	1	1.12	1.5
DEĞERLENDİRME : GENERATEE, FILTER ve ilk APPLY bloklarının icra zamanları çok düşüktür. İkinci APPLY bloğu ve REPORT bloğu paralel çalışmadan fayda temin etmektedir. Dolayısıyla bu programda paralel işlemenin avantajı az görülmüştür.				

Şekil E2.9 Program-6 deneyi.

PROGRAM AÇIKLAMASI :

Get supplier numbers for suppliers with status value less the current maximum status value in the S table.

(S tablosundaki maksimum değerden düşük statü değerine sahip üreticilerin numaralarını listele.)

PROGRAM-7

GENERATE R_S _stat.NUM ;

APPLY MAX ON _stat ;

TRANSFER AA;

GENERATET R_S _stat.NUM,_snum;

TRANSFER BB;

CONDBLOK AA BB REF._Sonuc>INP._stat;

REPORT ?[10,15] SAY "sno CHAR _snum,

?[11,15] SAY "statu NUM _stat;

#

	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	33	21	15	10
GEÇEN ZAMAN MİNİMUM	20	19	13	9
HIZLANMA KATSAYISI İLK	1	1.57	2.2	3.3
HIZLANMA KATSAYISI MİN	1	1.05	1.538	2.22

DEĞERLENDİRME : Bu program diğer programlara oranla daha çok sayıda Hareket üretmektedir. Bloklardan her birinin icra süresi göreceli olarak uzundur. Paralel işleme etkisi 3 ve daha fazla işlemcide belirgin bir şekilde görülmektedir.

Şekil E2.10 Program 7 deneyi.

PROGRAM AÇIKLAMASI :

Get supplier names for suppliers who supply at least one red part.

(En az bir kırmızı parça sağlayan üreticilerin isimlerini listele.)

PROGRAM-8 :

```
GENERATEE _Color="RED;
APPLY _Color|_color /R_P\_pnum;
APPLY _pnum|_pnum /R_SP\_snum;
APPLY _snum|_snum /R_S\_sname;
REPORT ?[10,15] SAY "sno CHAR _sname;
#
```

	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	30	20	12	8
GEÇEN ZAMAN MİNİMUM	12	11	8	7
HIZLANMA KATSAYISI İLK	1	1.5	2.5	3.75
HIZLANMA KATSAYISI MİN	1	1.09	1.5	1.71

DEĞERLENDİRME : Bu programda ilk üç blok çok kısa sürede icra etmektedir. Son iki blok paralel işlemeden gerçekte yararlanan bloklardır. İlk çalıştırmada ise APPLY bloklarından üçüde uzun zaman aldıklarından paralel işleme etkisi daha belirgindir. İlk çalıştırmayı takip eden çalıştırmalarda etki süratle azalmaktadır.

Şekil E2.11 Program-8 deneyi.

PROGRAM AÇIKLAMASI :

Get supplier numbers for suppliers who supply at least one part supplied by supplier S2
(Üretici S2 tarafından sağlanan parçalardan en az birini sağlayan üreticilerin numaralarını listele.)

PROGRAM-9 :

```
GENERATEE _Snum="S2 ;
APPLY _Snum|_snum /R_SP\_pnum;
APPLY _pnum|_pnum /R_SP\_snum;
TEST _snum|_Snum
  ASSIGN _R1="S2==_Snum
  AA,BB ;
```

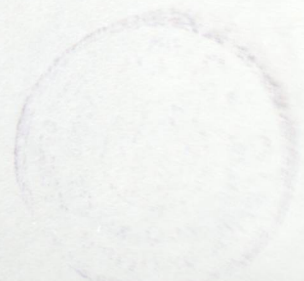
```
AA REPORT ?[10,15] SAY "parcno CHAR _Snum;
BB REPORT ?[10,15] SAY "parcno CHAR _Snum;
#
```

	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	22	20	15	10
GEÇEN ZAMAN MİNİMUM	10	10	8	6
HIZLANMA KATSAYISI İLK	1	1.1	1.46	2.2
HIZLANMA KATSAYISI MİN	1	1	1.25	1.66

DEĞERLENDİRME : Program içindeki GENERATEE ve APPLY blokları az zaman harcamaktadır. Bu nedenle seri ve paralel çalışma arasında ilk çalıştırma haricinde büyük farklılık görülmemektedir.

Şekil E2.12 Program-9 deneyi.

Şekil E2.13 Program-10 deneyi.



PROGRAM AÇIKLAMASI : Get part numbers for all part numbers supplied by more than two supplier. (İki değişik üretici tarafından sağlanan parça numaralarını listele.)				
PROGRAM-10 : GENERATE R_P _pnum; APPLY _pnum _pnum/R_SP_pnum,_snum; FILTER _pnum,_snum; COUNTBY _pnum; TEST _pnum _Pnum ,_Cnt _Cnt ASSIGN _R1=_Cnt>2.0 AA,BB; AA REPORT ?[10,15] SAY "pnum CHAR _Pnum, ?[10,15] SAY "cnt NUM _Cnt; BB REPORT ?[10,15] SAY "pnum CHAR _Pnum, ?[10,15] SAY "cnt NUM _Cnt; #				
	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	40	27	17	13
GEÇEN ZAMAN MİNİMUM	22	18	13	12
HIZLANMA KATSAYISI İLK	1	1.48	2.35	3.07
HIZLANMA KATSAYISI MİN	1	1.22	1.69	1.83
DEĞERLENDİRME : COUNTBY ve REPORT blokları uzun diğer bloklar az zaman harcamaktadır. Paralel çalışma etkisi orta düzeydedir.				

Şekil E2.13 Program-10 deneyi.

PROGRAM AÇIKLAMASI :

Get supplier numbers for suppliers who are located in the same city as supplier S2
(S2 üreticisi ile aynı şehirde bulunan üreticilerin numaralarını listele.)

PROGRAM-11 :

```

GENERATEE _Sno="S2;
APPLY _Sno|_snum /R_S\ _snum, _city;
TRANSFER AA;
GENERATEE R_S _snum, _city;
TRANSFER BB;
CONDBLOK AA BB REF._city ==INP._city;
TEST _snum|_Snum ,_city|_City
ASSIGN _R1="S2!=_Snum
CC,DD;
DD REPORT ?[10,15] SAY "snum CHAR _Snum,
REFUR ?[11,15] SAY "city CHAR _City;
#

```

	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	45	32	19	13
GEÇEN ZAMAN MİNİMUM	18	15	12	9
HIZLANMA KATSAYISI İLK	1	1.4	2.36	3.46
HIZLANMA KATSAYISI MİN	1	1.2	1.5	2

DEĞERLENDİRME : Blok sayısının çokluğu nedeniyle seri uygulama ile paralel uygulama arasında belirgin fark vardır.

Şekil E2.14 Program-11 deneyi.

Şekil E2.15 Program-12 deneyi.



PROGRAM AÇIKLAMASI :

Get supplier numbers for suppliers who supply at least all those parts supplied by supplier S2.

(S2 üreticisi tarafından sağlanan tüm parçaları sağlayan üreticilerin numarasını listele.)

PROGRAM-12 :

```
GENERATEE _Snum="S2;
APPLY _Snum|_snum/R_SP\_pnum;
APPLY _pnum|_pnum /R_SP\_snum,_pnum;
FILTER _snum,_pnum;
COUNTBY _snum;
TEST _snum|_Snum,_pnum|_Pnum ,_Cnt|_Cnt
ASSIGN _R1="S2!=_Snum
AA,BB;
CONDBLOK AA BB REF._Cnt==INP._Cnt;
REPORT ?[10,15] SAY "supplier CHAR _Snum,
?[11,15] SAY "pnum CHAR _Pnum,
?[12,15] SAY "cnt NUM _Cnt;
#
```

	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	54	33	20	15
GEÇEN ZAMAN MİNİMUM	16	14	11	8
HIZLANMA KATSAYISI İLK	1	1.63	2.7	3.6
HIZLANMA KATSAYISI MİN	1	1.14	1.45	2
DEĞERLENDİRME : Blok sayısının çokluğu nedeniyle seri uygulama ile paralel uygulama arasında belirgin fark vardır.				

Şekil E2.15 Program-12 deneyi.

PROGRAM AÇIKLAMASI :

Get part numbers for parts that either weigh more than 14 pounds or are currently supplied by supplier S2 or both.
(S2 tarafından üretilen ve/veya 18 libreden fazla ağırlıkta olan parçaların numaralarını listele.)

PROGRAM-13 :

```

GENERATET R_P _pnum,_weig;
TEST _pnum|_Pnum ,_weig|_Weig
ASSIGN _R1=_Weig>14.0
CC,DD;
GENERATET R_SP _pnum,_snum;
TEST _pnum|_Pnum ,_snum|_Snum
ASSIGN _R1="S2==_Snum
EE,FF;
MERGE DD,FF;
FILTER _Pnum;
REPORT ?[10,15] SAY "pnum CHAR _Pnum;
CC REPORT ?[10,15] SAY "pnum CHAR _Pnum;
EE REPORT ?[10,15] SAY "pnum CHAR _Pnum;
#

```

	İŞLEMÇİ SAYISI			
İŞLEMÇİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	70	44	31	19
GEÇEN ZAMAN MİNİMUM	30	22	17	13
HIZLANMA KATSAYISI İLK	1	1.59	2.25	3.68
HIZLANMA KATSAYISI MİN	1	1.36	1.76	2.3

DEĞERLENDİRME :

Blok sayısının çoktur. Çok sayıda Hareketin akışa uğraması nedeniyle blokların icra zamanları uzundur. Bu nedenle paralel işlemcilerin rolü daha belirgin olarak ortaya çıkmıştır.

Şekil E2.16 Program-13 deneyi.

PROGRAM AÇIKLAMASI :

For all parts get the part number and the weight of the part in grams .

(Bütün parçaların numarasını ve ağırlığını gram olarak listele.)

PROGRAM-14 :

```
GENERATET R_P _pnum, _weig;
COMPUTE _weig | _Weig , _pnum | _Pnum
ASSIGN _A1 = _Weig * 454.0
OUTPUT _A1, _Pnum;
REPORT ?[10,15] SAY "parcano CHAR _Pnum,
        ?[11,15] SAY "gramaj NUM _A1;
#
```

	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	26	15	11	7
GEÇEN ZAMAN MİNİMUM	15	12	9	6
HIZLANMA KATSAYISI İLK	1	1.73	2.36	3.71
HIZLANMA KATSAYISI MİN	1	1.25	1.66	2.5
DEĞERLENDİRME :				
Her üç blokta çok zaman harcayan tiptedir. GENERATET tüm R_P kayıtlarını akışa uğratmaktadır. Bu yoğun işlemde paralel işlemenin avantajı ön plana çıkmaktadır.				

Şekil E2.17 Program-14 deneyi.

PROGRAM AÇIKLAMASI :

Get the total number of suppliers.

(Üreticilerin toplam sayısını listele.)

PROGRAM-15 :

GENERATE R_S _snum;

TXSAY;

REPORT ?[10,15] SAY "sayi NUM _Cnt;

#

	İŞLEMÇİ SAYISI			
İŞLEMÇİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	20	13	8	7
GEÇEN ZAMAN MİNİMUM	6	5	3	3
HIZLANMA KATSAYISI İLK	1	1.53	2.5	2.85
HIZLANMA KATSAYISI MİN	1	1.2	2	2
DEĞERLENDİRME : Program üç bloktan ibarettir ve 3 işlemci ile elde edilen sonuç ile 4 işlemci ile elde edilen sonuç yakındır. Ana programın dördüncü işlemci üzerinde çalışması dört işlemci ile çalışmada üç işlemciye göre avantaj sağlamaktadır.				

Şekil E2.18 Program-15 deneyi.

PROGRAM AÇIKLAMASI :

Get the total number of suppliers currently supplying parts.
(Halen parça sağlayan tüm üreticilerin toplam sayısını
listele.)

PROGRAM-16 :

```
GENERATE R_SP_snum;
TXSAY;
REPORT ?[10,15] SAY "sayi NUM _Cnt;
#
```

	İŞLEMÇİ SAYISI			
İŞLEMÇİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	17	11	7	6
GEÇEN ZAMAN MİNİMUM	6	5	3	3
HIZLANMA KATSAYISI İLK	1	1.54	2.42	2.83
HIZLANMA KATSAYISI MİN	1	1.2	2	2
DEĞERLENDİRME : Program üç bloktan ibarettir ve 3 işlemci ile elde edilen sonuç ile 4 işlemci ile elde edilen sonuç yakındır. Ana programın dördüncü işlemci üzerinde çalışması dört işlemci ile çalışmada üç işlemciye göre avantaj sağlamaktadır.				

Şekil E2.19 Program-16 deneyi.

PROGRAM AÇIKLAMASI :

Get the number of shipments for part P1
(P1 gönderi sayısını listele.)

PROGRAM-17 :

```
GENERATEE _Pnum="P1;
APPLY _Pnum|_pnum/R_SP\_pnum;
TXSAY;
REPORT ?[10,15] SAY "sayi NUM _Cnt;
#REPORT ?[10,15] SAY "sayi NUM _Cnt;
```

	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	19	13	8	6
GEÇEN ZAMAN MİNİMUM	6	5	3	3
HIZLANMA KATSAYISI İLK	1	1.46	2.37	3.16
HIZLANMA KATSAYISI MİN	1	1.2	2	2.25
DEĞERLENDİRME :				
APPLY bloğundan itibaren blokların harcadıkları zaman artmaktadır. Paralel işlemde yararlanma düzeyi ortadır.				

Şekil E2.20 Program-17 deneyi.



PROGRAM AÇIKLAMASI :
(Gönderilen toplam P2 parça sayısını listele.)
Get the total quantity of part P2 supplied

PROGRAM-18 :

```
GENERATEE _Pnum="P2 ;
APPLY _Pnum|_pnum/R_SP\_qty;
APPLY SUM ON _qty ;
REPORT ?[11,15] SAY "sayisi NUM _Sonuc;
#
```

	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	23	14	9	6
GEÇEN ZAMAN MİNİMUM	9	7	5	4
HIZLANMA KATSAYISI İLK	1	1.64	2.55	3.83
HIZLANMA KATSAYISI MİN	1	1.28	1.8	2.25

DEĞERLENDİRME :
R_SP tablosu büyük bir tablodur. Dolayısıyla blokların çalışma süreleri de bu nisbette uzamaktadır. APPLY SUM bloğunun ayrı bir işlemci üzerinde çalışması paralel programa büyük avantaj sağlamaktadır. Programın paralel işlemeyi yararlanma düzeyi yüksektir.

Şekil E2.21 Program-18 deneyi.

PROGRAM AÇIKLAMASI :

For each part supplied get the part number and the total quantity supplied of that part.

(Her sağlanan parçanın parça numarası ve gönderilen toplam sayısını listele.)

PROGRAM-19 :

```
GENERATET R_SP _pnum, _qty ;
APPLY SUM ON _qty BY _pnum;
REPORT ?[10,15] SAY "parcno CHAR _pnum,
COUNT ?[11,15] SAY "sayisi NUM _Sonuc;
#
```

	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	41	22	15	12
GEÇEN ZAMAN MİNİMUM	32	18	13	10
HIZLANMA KATSAYISI İLK	1	1.86	2.61	3.41
HIZLANMA KATSAYISI MİN	1	1.77	2.46	3.2

DEĞERLENDİRME :

R_SP tablosu büyük bir tablodur ve tüm kayıtları akışa uğramaktadır. Dolayısıyla blokların çalışma süreleri de bu nisbette uzamaktadır. APPLY SUM bloğunun ayrı bir işlemci üzerinde çalışması paralel programa büyük avantaj sağlamaktadır. Programın paralel işlemekten yararlanma düzeyi yüksektir.

Şekil E2.22 Program-19 deneyi.

PROGRAM AÇIKLAMASI :

Birden fazla üretici tarafından sağlanan tüm parçaların parça numarasını listele.

(Get part numbers for all parts supplied by more than one supplier)

PROGRAM-20 :

```
GENERATE R_P _pnum;
APPLY _pnum|_pnum/R_SP\_pnum,_snum;
FILTER _pnum,_snum;
COUNTBY _pnum;
TEST _pnum|_Pnum,_Cnt|_Cnt
ASSIGN _R1=_Cnt>1.0
AA,BB;
BB REPORT ?[10,15] SAY "parcno CHAR _Pnum;
#
```

	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	95	60	41	28
GEÇEN ZAMAN MİNİMUM	23	17	12	10
HIZLANMA KATSAYISI İLK	1	1.58	2.3	3.39
HIZLANMA KATSAYISI MİN	1	1.35	1.91	2.3

DEĞERLENDİRME :

COUNTBY, TEST ve REPORT blokları uzun süre harcamaktadır. R_P ilişkisinin tüm _pnum sahası içeriği akışa uğramaktadır. Netice olarak iş yükü belirli bir düzeyin üzerine çıktığından paralel işlemenin sağladığı avantaj belirgindir.

Şekil E2.23 Program-20 deneyi.

KATSAYISI İLK	1.57	2.42	3.43
HIZLANMA	1.4	2.08	3.96
KATSAYISI MİN			
DEĞERLENDİRME			
Programın ve işlenen verinin büyüklüğü açısından paralel işleme katkısı yüksek düzeyde görülmektedir.			

Şekil E2.24 Program-31 deneyi.

PROGRAM AÇIKLAMASI :

Gönderilen miktarı, 200 den az gönderiler hariç tutularak alınan toplamlar bazında 300 ve daha büyük olan parçaların numarasını ve sağlanan maksimum miktarını parça numarasının azalan sırada sıralanmasıyla listele.

For all parts such that the total quantity supplied is greater than 300 (excluding from the total, all shipments for which the quantity is less than or equal to 200) get the part number and the maximum quantity of the parts supplied and order the result by descending part number with those maximum quantity values.

PROGRAM-21 :

```

GENERATET R_SP _pnum, _qty;
TEST _qty|_Qty,_pnum|_Pnum
ASSIGN _R1=_Qty>200.0
AA,BB;
BB APPLY SUM ON _Qty BY _Pnum;
TEST _Pnum|_Pnum,_Qty|_Qty,_Sonuc|_S1
ASSIGN _R1=_S1>300.0
CC,DD;
DD APPLY _Pnum|_pnum/R_SP\_pnum,_qty:_S1;
APPLY MAX ON _qty BY _pnum;
SORT _pnum;
REPORT ?[10,15] SAY "parcano CHAR _pnum,
        ?[11,15] SAY "maxmikt NUM _Sonuc;
#

```

	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	184	117	74	48
GEÇEN ZAMAN MİNİMUM	77	55	37	26
HIZLANMA KATSAYISI İLK	1	1.57	2.48	3.83
HIZLANMA KATSAYISI MİN	1	1.4	2.08	2.96

DEĞERLENDİRME :

Programın ve işlenen verinin büyüklüğü nedeniyle paralel işlemenin katkısı yüksek düzeyde görülmektedir.

Şekil E2.23 Program-21 deneyi.

PROGRAM AÇIKLAMASI :

Change the color of part P2 to yellow increase its weight by 5 and set its city to unknown
(Parça P2'nin rengini sarıya değiştir ve ağırlığını 5 arttır ve sağlandığı şehri "bilinmeyen" olarak ata.)

PROGRAM-22 :

```
GENERATEE _Pnum="P2,_Color="YELLOW ;
COMFIX _Pnum|_Pnum,_Color|_Color
LOCATE R_P _pnum=_Pnum,_weig,_color
ASSIGN _weig=_weig+5.0,
       _color=_Color
OUTPUT _Pnum;
REPORT ?[10,15] SAY "parcno CHAR _Pnum;
#
```

	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	17	16	12	11
GEÇEN ZAMAN MİNİMUM	9	9	8	7
HIZLANMA KATSAYISI İLK	1	1.06	1.41	1.54
HIZLANMA KATSAYISI MİN	1	1	1.12	1.28
DEĞERLENDİRME : Sadece Compute Fix bloğu 6 tick sürmektedir. Diğer bloklar ise çok kısa süreli çalışmaktadır. Bu durumda sadece bir bloğun çalışması tüm programın çalışması gibi düşünülebilir ve tabii olarak seri çalışma ile paralel çalışma arasında büyük bir zaman farkı gözlenmez.				

Şekil E2.24 Program-22 deneyi.

PROGRAM AÇIKLAMASI :

Double the status of all suppliers in London
(Londradaki üreticilerin statü değerlerini iki ile çarp.)

PROGRAM-23 :

```
GENERATEE _City="LONDON ;
APPLY _City|_city /R_S\_snum;
COMFIX _snum|_Snum
LOCATE R_S _snum=_Snum,_stat
ASSIGN _stat=_stat*2.0,
      _Stat=_stat
OUTPUT _Stat;
REPORT ?[10,15] SAY "statu NUM _Stat;
#
```

	İŞLEMÇİ SAYISI			
İŞLEMÇİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	18	17	10	8
GEÇEN ZAMAN MİNİMUM	9	9	8	7
HIZLANMA KATSAYISI İLK	1	1.05	1.8	2.25
HIZLANMA KATSAYISI MİN	1	1	1.12	1.28
DEĞERLENDİRME : Sadece Compute Fix bloğu 6 tick sürmektedir. Diğer bloklar ise çok kısa süreli çalışmaktadır. Bu durumda sadece bir bloğun çalışması tüm programın çalışması gibi düşünülebilir ve tabii olarak seri çalışma ile paralel çalışma arasında büyük bir zaman farkı gözlenmez.				

Şekil E2.25 Program-23 deneyi.

PROGRAM AÇIKLAMASI :

Add part P7 (Name 'Washer', color 'Grey', weight 2, city 'Athens') to table P.
(P7 parçasını tablo P'ye ekle. Name 'Washer', color 'Grey', weight 2, city 'Athens')

PROGRAM-24 :

GENERATEE

```
_Pnum="P7,_Color="YELLOW,_Weig=2.0,_City="Athens ;
APPEND LOCATE R_P _pnum=_Pnum, _color=_Color,
_weig=_Weig,_city=_City;
REPORT ?[10,15] SAY "parcno CHAR _Pnum,
?[10,15] SAY "color CHAR _Color,
?[10,15] SAY "weight NUM _Weig,
?[10,15] SAY "city CHAR _City;
```

#

	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	22	16	11	11
GEÇEN ZAMAN MİNİMUM	19	15	10	10
HIZLANMA KATSAYISI İLK	1	1.375	2	2
HIZLANMA KATSAYISI MİN	1	1.26	1.9	1.9

DEĞERLENDİRME :

REPORT ve APPEND blokları zaman harcayan asıl bloklar olup, paralel işlemin avantajı sadece bu bloklar paralel çalıştığında görülmektedir. Bu etkide maksimum 2 kat hızlanma şeklinde kendini göstermektedir.

Şekil E2.26 Program-24 deneyi.

PROGRAM AÇIKLAMASI :

Table TEMP has one column called p#. Enter into TEMP part numbers for all parts supplied by supplier S2
(TEMP isimli tabloya üretici S2 tarafından sağlanan parçaları ekle.)

PROGRAM-25 :

```
GENERATEE _Snum="S2 ;
APPLY _Snum|_snum/R_SP\_pnum;
FILTER _pnum;
APPEND LOCATE R_TEMP _pnum=_pnum;
REPORT ?[10,15] SAY "parcno CHAR _pnum;
#
```

	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	41	30	17	12
GEÇEN ZAMAN MİNİMUM	12	11	10	8
HIZLANMA KATSAYISI İLK	1	1.36	2.41	3.41
HIZLANMA KATSAYISI MİN	1	1.09	1.2	1.5

DEĞERLENDİRME :

APPEND blok söz konusu ilişkinin tüm sahalarıyla ilişkili dosyaları açan ve kullanan bir bloktur. İlk çalıştırmada açılan bu dosyalar, kaşe edildiğinden, ardından gelen çalıştırmalarda çok kısa sürede açılabilir. Bu nedenle ilk çalıştırmayla takip eden çalıştırmalar arasında büyük zaman farkı oluşmaktadır. Takip eden çalıştırmalarda her blok çok hızlı icra ettiğinden paralel ve seri çalıştırma arasında belirgin bir fark gözlenmemektedir.

Şekil E2.27 Program-25 deneyi.

PROGRAM AÇIKLAMASI :
Delete suppliers located in London
(Londrada bulunan üreticileri iptal et.)

PROGRAM-26 :

```
GENERATEE _City="LONDON;
APPLY _City|_city/R_S\_snum;
DELETE LOCATE R_S _snum=_snum;
REPORT ?[10,15] SAY "snum CHAR _snum;
#
```

	İŞLEMCİ SAYISI			
İŞLEMCİ SAYISI	1	2	3	4
GEÇEN ZAMAN İLK	19	17	15	11
GEÇEN ZAMAN MİNİMUM	11	11	10	9
HIZLANMA KATSAYISI İLK	1	1.11	1.26	1.72
HIZLANMA KATSAYISI MİN	1	1	1.1	1.5
DEĞERLENDİRME : Tüm bloklar çok kısa süreli icra etmekte , seri çalışma çok kısa bir sürede bitmekte, bu nedenle paralel ve seri çalışma arasında büyük bir zaman farkı gözlenmemektedir.				

Şekil E2.28 Program-26 deneyi.

EK-3

YACC VE LEX ARAÇLARI İÇİN YAPILAN TANIMLAMALAR

LEX TANIMLAMASI

```

% %
[0-9] {yyval=yytext[yylength-1]-'0';
      return(TDIJIT);
}
# {return(TERMIN);}
\ . {return(TNOKTA);}
\ + {return(TARTIOP);}
% {return(TATSGN);}
\ { {return(TLBRACE);}
\ } {return(TRBRACE);}
\ - {return(TEKSOP);}
\ * {return(TCARPOP);}
\ / {return(TBOLOP);}
: {return(TINOKTA);}
= {return(TESOP);}
; {return(TNOKVIRG);}
\ , {return(TVIRG);}
\ [ {return(TLBRACK);}
\ ] {return(TRBRACK);}
\ > {return(TBUYUK);}
\ < {return(TKUCUK);}
\\ {return(TERSBOL);}
\& {return(TAMPS);}
\? {return(TSORU);}
\ ( {return(TLPARA);}
\ ) {return(TRPARA);}
\ < = {return(TKUCUKES);}
\ > = {return(TBUYUKES);}
! = {return(TESDEGIL);}
! {return(TUNLEM);}
\\ {return(TPIPE);}
E {return(TEXP);}
NUM {return(TNUM);}
CHAR {return(TCHAR);}
DCL {return(TDCL);}
MAX {return(TMAX);}
MIN {return(TMIN);}
SUM {return(TSUM);}
AVG {return(TAVG);}
GET {return(TGET);}
ON {return(TON);}
BY {return(TBY);}
REF. {return(TREF);}
INP. {return(TINP);}
PRODUCT {return(TPROD);}
CONDBLOK {return(TCONDBLOK);}
TXCNT {return(TTXSAY);}

```



```

STRASGN {return(TSTRASGN);}
GENERATE {return(TGENER);}
GENERATET {return(TGENERET);}
CARTESIANP {return(TCARTES);}
GENERATEE {return(TGENERE);}
NOTMEMBER {return(TMEMBER);}
SAY {return(TSAY);}
TEST {return(TTEST);}
APPLY {return(TAPPLY);}
COMPUTE {return(TCOMPUTE);}
LOCATE {return(TLOCATE);}
COMFIX {return(TCOMFIX);}
SPLIT {return(TSPLIT);}
INPUT {return(TINPUT);}
REPORT {return(TREPORT);}
SORT {return(TSORT);}
COUNTBY {return(TCOUNTB);}
APPCUM {return(TAPPCUM);}
TRANSFER {return(TTRANSF);}
SINK {return(TSINK);}
FILTER {return(TFILTER);}
UNIQUE {return(TUNIQ);}
APPEND {return(TUPDAT);}
DELETE {return(TDEL);}
MERGE {return(TMERGE);}
ASSIGN {return(TASSIGN);}
OUTPUT {return(TOUTPUT);}
R_[A-Z][A-Z0-9]* {
    for (i=0;i<=prevyyl1-1;i++){
        rngond1[i]=rngond[i]; }
    if (prevyyl1-1<200) rngond1[i]='\0';
    for (i=0;i<=yyleng-1;i++){
        rngond[i]=yytext[i]; }
    if (yyleng-1<200)rngond[i]='\0';
    prevyyl1=yyleng;
    return(RELISMI);
}
_[A-Z][a-z0-9]+ {
    for (i=0;i<=prevyyl2-1;i++){
        vrgond1[i]=vrgond[i]; }
    if (prevyyl2-1<200)vrgond1[i]='\0';
    for (i=0;i<=yyleng-1;i++){
        vrgond[i]=yytext[i]; }
    if (yyleng-1<200)vrgond[i]='\0';
    prevyyl2=yyleng;
    return(DEGISMI);
}
_[a-z][a-z0-9]* {
    for (i=0;i<=prevyyl3-1;i++){
        atgond1[i]=atgond[i]; }

```



```

    if (prevyyl3-1 < 200) atgond1[i] = '\0';
    for (i=0; i <= yytext[i]; i++){
        atgond[i] = yytext[i];
    }
    if (yytext[i] < 200) atgond[i] = '\0';
    prevyyl3 = yytext[i];
    return(SAHAISMI);
}

\"[A-Za-z0-9_]\" {
    for (i=0; i <= prevyyl4-1; i++){
        stgond1[i] = stgond[i];
    }
    if (prevyyl4-1 < 200) stgond1[i] = '\0';
    for (i=1; i <= yytext[i]; i++){
        stgond[i-1] = yytext[i];
    }
    if (yytext[i] < 200) stgond[i-1] = '\0';
    prevyyl4 = yytext[i];
    return(KARAKTERDIZIN);
}

[A-Z][A-Z]* {
    for (i=0; i <= prevyyl5-1; i++){
        lbgond1[i] = lbgond[i];
    }
    if (prevyyl5-1 < 200) lbgond1[i] = '\0';
    for (i=1; i <= yytext[i]; i++){
        lbgond[i-1] = yytext[i];
    }
    if (yytext[i] < 200) lbgond[i-1] = '\0';
    prevyyl5 = yytext[i];
    return(TETIKET);
}

```

YACC TANIMLAMASI

```

%{
#include <stdio.h>
#include <ctype.h>
%}

%start program

%token TERMIN TNOKTA TARTIOP TLBRACE TRBRACE TEKSOP TCARPOP TBOLOP
%token TINOKTA TESOP TQUOTE TNOKVIRG TVIRG TLBRACK TRBRACK
%token TBUYUK TKUCUK TERSBOL TAMPs TATSGN TSORU TLPARA
TRPARA TNOT TKUCUKES

%token TBUYUKES TESDEGIL TPIPE TEXP TNUM TCHAR TDCL TMAX TMIN TON TBY
%token TSUM TAVG TPROD TGENER TGENER T SAY TTEST TAPPLY TCOMPUTE TCOMFIX
%token TSPLIT TINPUT TREPORT TSORT TCOUNTB TAPPCUM TTRANSF TSINK TCARTES

```



```
%token TFILTER TUNIQ TUPDAT TDEL TASSIGN TOUTPUT TETIKET TGENERE TMEMBER
%token KARAKTERDIZIN DEGISMI RELISMI SAHAISMI
%token TGET TDIJIT TLOCATE TMERGE TSTRASGN
%left TBUYUK TKUCUK TKUCUKES TBUYUKES TESDEGIL TUNLEM TTXSAY
%left TARTIOP TEKSOP TCONDBLOK TINP TREF
%left TCARPOP TBOLOP
%%
```

```
program: bloklar TERMIN
```

```
{ bp=fopen("bloks.c","w");
  for (i=1;i<=bloknm;i++){
    for (j=0;j<=36;j+=2){
      if(blokmp[i][j]!=0){
        fprintf(bp,"blokmap[ %d][ %d] = %d;\n",i,j/2,blokmp[i][j]);
        sonbno=blokmp[i][j];
      }
    }
    if (blokmp[i][0]!=0&&blokmp[i][2]==0){
      fprintf(bp,"blokmap[ %d][1] = %d; \n",i,blokmp[i][0]);
    }
  }
  fprintf(bp,"progbloks= %d;",bloknm);
  fclose(bp);
```

```
printf("\n PROGRAM TANINDI \n");
printf("\n derleme bitti CIKIS CIKTI.c
      Adli dosyaya yazildi \n");
return(0);};
```

```
bloklar : blok
```

```
{
  fprintf(cp,"\n adrp->procact[pno]=1;\n");
  fprintf(cp,"\n s_entry.mtype=(long)pno;\n");
  fprintf(cp," if(msgsnd(queue_id,&s_entry,5,0)==-1) ");
  fprintf(cp,"\n      perror(\"msgsnd failed\"); \n");
}
| bloklar {
  bloknm += 1;
} blok
{
  fprintf(cp,"\n adrp->procact[pno]=1;\n");
  fprintf(cp,"\n s_entry.mtype=(long)pno;\n");
  fprintf(cp," if(msgsnd(queue_id,&s_entry,5,0)==-1) ");
  fprintf(cp,"\n      perror(\"msgsnd failed\"); \n");
}
;

```

```
/* SOLA OZDEVINIM KULLANILARAK BLOKLAR NONTERMINALININ BIR VEYA
   BIRDEN FAZLA BLOK NONTERMINALINDEN OLUSTUGU BELIRTIKTE */
```

```
/* Temel girdiler */
```

```
: TDIJIT { $$=TDIJIT;
          fp=fopen("gecici","w");
```



```

        if (eksitgl) { /* eger negint den gelindiyse
                        isareti isle ve toggle'i
                        eski konumuna getir */
                fprintf(fp, "-");
                eksitgl=0;
        }
        fprintf(fp, "%d", $1) ;
    }|
    positiftamsayi TDIJIT { fprintf(fp, "%d", $2);
    };
negatiftamsayi : TEKSOP { eksitgl=1; } positiftamsayi;

```

```

posgercel      : positiftamsayi TNOKTA {

```

```

        fclose(fp);

```

```

        fp=fopen("gecici", "r");

```

```

        fscanf(fp, "%d", &k) ;

```

```

    }

```

```

    positiftamsayi {

```

```

        fclose(fp);

```

```

        fp=fopen("gecici", "r");

```

```

        fscanf(fp, "%d", &m);

```

```

        rp=fopen("gecici2", "w");

```

```

        fprintf(rp, "%d.%d", k, m);

```

```

    } ;

```

```

neggercel      : negatiftamsayi TNOKTA {

```

```

        fclose(fp);

```

```

        fp=fopen("gecici", "r");

```

```

        fscanf(fp, "%d", &k) ;

```

```

    positiftamsayi {

```

```

        fclose(fp);

```

```

        fp=fopen("gecici", "r");

```

```

        fscanf(fp, "%d", &m);

```

```

        rp=fopen("gecici2", "w");

```

```

        fprintf(rp, "%d.%d", k, m);

```

```

    } ;

```

```

gercel          : posgercel | neggercel;

```

```

/* ----- */

```

```

blok : inputblok

```

```

| applyblok

```

```

| generblok

```

```

| genertblok

```

```

| genereblok

```

```

| reportblok

```

```

| computeblok

```

```

| comfixblok

```

```

| sortblok

```

```

| transblok

```

```

| countblok

```



```

filtblok
sinkblok
updtblok
testblok
delblok
mergeblok
applycumblok
txcblok
condblok
memblok
cartesblok

```

```
;
```

```

inputblok : TINPUT {
    blokmp[bloknm][2]=bloknm+1;
    blokmp[bloknm][3]=okuyrm;
    bloktip[bloknm]=7;
    fprintf(cp,"#include </usr/ccs/bin/inpinc.c>\n");
    l=1; /* okunacak degişken sayacı ve tx'daki pozisyon */
}

```

```

getblok
{fprintf(cp,"\\n adrp->inputmat[yer]=%d;\\n",l-2);
 fprintf(cp,"\\n adrp->inputtl[yer]=%d;\\n",l-1);
} TNOKVIRG {
    ikuyrm=okuyrm;
    sontl=l-1;
    okuyrm+=1;
    kuyuz[ikuyrm]=sontl;
}

```

```
getblok : TATSGN giriş_koordinat TGET KARAKTERDIZIN
```

```

{
    fprintf(cp,"\\nsprintf(adrp->inputfb[yer][%d],
        \\\" %s\\\",\\\" %s\\\" );\\n",l,format,stgond);
    fprintf(cp,"\\nadrp->inputent[yer][2][%d]=%d;\\n",l,l);
}

```

```

TNUM
{fprintf(cp,"\\n adrp->inputent[yer][3][%d]=1;\\n",l);}
DEGISMI {
    install(vrgond,'n',l,okuyrm,0);
    l+=1;
} |

```

```
TATSGN giriş_koordinat TGET KARAKTERDIZIN
```

```

{fprintf(cp,"\\nsprintf(adrp->inputfb[yer][%d],
    \\\" %s\\\",\\\" %s\\\" );\\n",l,format,stgond);
    fprintf(cp,"\\nadrp->inputent[yer][2][%d]=%d;\\n",l,l);
}

```

```

TCHAR
{fprintf(cp,"\\n adrp->inputent[yer][3][%d]=0;\\n",l);}
DEGISMI {
    install(vrgond,'c',l,okuyrm,0);
}

```



```

l += 1;
}|
getblok TVIRG TATSGN giris_koordinat TGET KARAKTERDIZIN
{fprintf(cp, "\nsprintf(adrp->inputfb[yer][ %d],
    \"%s\\\", \"%s\\\")\n\", l, format, stgond);
fprintf(cp, "\nadrp->inputent[yer][2][ %d] = %d;\n\", l, l);
}
TNUM
{fprintf(cp, "\n adrp->inputent[yer][3][ %d] = 1;\n\", l);}
DEGISMI {
install(vrgond, 'n', l, okuyrnm, 0);
l += 1;
}|
getblok TVIRG TATSGN giris_koordinat TGET KARAKTERDIZIN
{fprintf(cp, "\nsprintf(adrp->inputfb[yer][ %d],
    \"%s\\\", \"%s\\\")\n\", l, format, stgond);
fprintf(cp, "\nadrp->inputent[yer][2][ %d] = %d;\n\", l, l);
}
TCHAR
{fprintf(cp, "\n adrp->inputent[yer][3][ %d] = 0;\n\", l);}
DEGISMI
{
install(vrgond, 'c', l, okuyrnm, 0);
l += 1;
}
;

```

```

giris_koordinat : TLPARA positiftamsayi
    {fclose(fp);
    fp = fopen("gecici", "r");
    fscanf(fp, "%d", &k);
    fprintf(cp, "\nadrp->inputent[yer][0][ %d] = %d;\n\", l, k);
    }
TVIRG positiftamsayi
    {fclose(fp);
    fp = fopen("gecici", "r");
    fscanf(fp, "%d", &k);
    fprintf(cp, "\nadrp->inputent[yer][1][ %d] = %d;\n\", l, k);
    }
TRPARA;

```

```

applycumbas: applybaslik cumop TON {
    girdegisay = 1;
    blokmp[bloknm][2] = bloknm + 1;
    blokmp[bloknm][3] = okuyrnm;
    bloktip[bloknm] = 17;
    togle = 'u';
} ilis2 { togle = 'm';
};

```

```

applycumblok : applycumbas

```



```

{
    fprintf(cp, "\nadrp-> cumtxix[yer][0] = -1; \n");
    fprintf(cp, "\nadrp-> cumtsay[yer] = 1; \n");
    sprintf(vrgond, "_Sonuc");
    install(vrgond, 'n', kuyuz[ikuyrnm] + 1, okuyrnm, 0);
    fprintf(cp, "\nadrp-> cumtloc[yer] = %d; \n", kuyuz[ikuyrnm] + 1);
    fprintf(cp, "\nadrp-> cumtl[yer] = %d; \n", kuyuz[ikuyrnm]);
    sontl = kuyuz[ikuyrnm] + 1;
    ikuyrnm = okuyrnm;
    kuyuz[ikuyrnm] = sontl;
    okuyrnm += 1;
}
TNOKVIRG
| applycumbas
TBY iliskil2
TNOKVIRG
{
    fprintf(cp, "\nadrp-> cumtsay[yer] = %d; \n", girdegssay);
    sprintf(vrgond, "_Sonuc");
    install(vrgond, 'n', kuyuz[ikuyrnm] + 1, okuyrnm, 0);
    fprintf(cp, "\nadrp-> cumtloc[yer] = %d; \n", kuyuz[ikuyrnm] + 1);
    fprintf(cp, "\nadrp-> cumtl[yer] = %d; \n", kuyuz[ikuyrnm]);
    sontl = kuyuz[ikuyrnm] + 1;
    ikuyrnm = okuyrnm;
    kuyuz[ikuyrnm] = sontl;
    okuyrnm += 1;
};
cumop : TSUM
{
    fprintf(cp, "#include </usr/ccs/bin/appcinc.c> \n");
    fprintf(cp, "\nadrp-> cumop[yer] = '+'; \n");
}
| TMAX
{
    fprintf(cp, "#include </usr/ccs/bin/appcinc.c> \n");
    fprintf(cp, "\nadrp-> cumop[yer] = 'x'; \n");
}
| TMIN
{
    fprintf(cp, "#include </usr/ccs/bin/appcinc.c> \n");
    fprintf(cp, "\nadrp-> cumop[yer] = 'm'; \n");
}
| TAVG
{
    fprintf(cp, "#include </usr/ccs/bin/appcinc.c> \n");
    fprintf(cp, "\nadrp-> cumop[yer] = 'a'; \n");
}
| TPROD

```



```

{
    fprintf(cp, "#include </usr/ccs/bin/appcinc.c>\n");
    fprintf(cp, "\nadrp->cumop[yer]='*';\n");
}

;
applybaslik : TAPPLY
{
    blokmp[bloknm][0]=bloknm-1;
    blokmp[bloknm][1]=ikuyrnm;
}
| geneletiket
TAPPLY
;

applyblok : applybaslik {
    blokmp[bloknm][2]=bloknm+1;
    blokmp[bloknm][3]=okuyrnm;
    bloktip[bloknm]=4;
    l=1; /* çıkış girdi sayacı */
    n=1; /* bypass deg saha sayacı (10 dan büyük olmaz) */
    gas=1; /* giriş saha sayacı */
    cas=1; /* çıkış saha sayacı */
    togle='a';
    fprintf(cp, "#include </usr/ccs/bin/appinc.c>\n");
}
girsahlist TBOLOP
{ fprintf(cp, "\n adrp->applygas[yer]=%d ;\n", gas );
  if (gas > kuyuz[ikuyrnm]){
    printf("\n Blokno=%d Ikaz :
          aplikasyon saha sayisi ile giris \n", bloknm);
    printf("\n kuyruk uzunlugu uyusmuyor \n");
  }
  fprintf(cp, "\n adrp->applytlg[yer]=%d ;\n", kuyuz[ikuyrnm]);
}
RELISMI
{ fprintf(cp, "\nsprintf(adrp->applyrel[yer],
  \"%s\", \"%s\");\n", fmt, rngond);
}
TERSBOL {l=1;n=1;} cikis
TNOKVIRG {
    fprintf(cp, "\n adrp->applycas[yer]=%d ; \n", cas);
    fprintf(cp, "\n adrp->applytlc[yer]=%d ; \n", l);
    ikuyrnm=okuyrnm;
    sontl=1;
    kuyuz[ikuyrnm]=sontl;
    okuyrnm+=1;
};
girsahlist : girsahgird | girsahlist TVIRG {gas+=1;} girsahgird;

girsahgird : DEGISMI
{

```



```

if( (np=bul(vrgond,ikuyrnm))==NULL) {
    printf("\n Blokno= %d Hata :
    Degisken %s sembol tablosunda yok \n",bloknm,vrgond);
    exit(-1);
}
}
}

```

TPIPE

SAHAISMI

```

{
    if (togle=='a'){
        fprintf(cp, "\nsprintf(adrp->applyga[yer][ %d],
        \"%s\", \"%s\");\n",gas-1,formt,atgond);
    }
    if (togle=='m'){
        fprintf(cp, "\nsprintf(adrp->memga[yer][ %d],
        \"%s\", \"%s\");\n",gas-1,formt,atgond);
        install(atgond,np->tip,np->sira,ikuyrnm,0);
    }
    /* giriş kuyruğunu aynen dışarı çıkarır programcının bu bloğu takip eden
    blokta kazaen ilişkilendirmenin sağındaki saha ismini kullanması
    halinde bir problem çıkmasın diye aliasing yapılıyor yani pipe'ın
    sağındaki ile solundaki değişken aynı data lokasyonuna kumanda ediyor */
}
}
}

```

SAHAISMI

```

{
    if( (np=bul(atgond,ikuyrnm))==NULL) {
        printf("\n Blokno= %d Hata :
        Degisken %s sembol tablosunda yok \n",bloknm,atgond);
        exit(-1);
    }
}
}

```

TPIPE

SAHAISMI

```

{
    if (togle=='a'){
        fprintf(cp, "\nsprintf(adrp->applyga[yer][ %d],
        \"%s\", \"%s\");\n",gas-1,formt,atgond);
    }
    if (togle=='m'){
        fprintf(cp, "\nsprintf(adrp->memga[yer][ %d],
        \"%s\", \"%s\");\n",gas-1,formt,atgond);
        install(atgond,np->tip,np->sira,ikuyrnm,0);
    }
    /* member blok girişteki transaction eleman değilse aynen dışarı çıkarır */
}
}
;

```



```

cikis : ciksahtlist
      | ciksahtlist TINOKTA cikdeglist;

cikdeglist: DEGISMI
{
  l += 1;
  n += 1;
  if (n > 10) {
    printf("\n Blokno = %d Hata :
    bypass edilen degisken ve saha sayisi
    10 dan buyuk olamaz \n", bloknm);
    exit(-1);
  }
  if ( (np = bul(vrgond, ikuyrnm)) == NULL) {
    printf("\n Blokno = %d Hata :
    Degisken %s sembol tablosunda yok \n",
    bloknm, vrgond);
    exit(-1);
  }
  install(vrgond, np->tip, l, okuyrnm, 0);
  fprintf(cp, "\n adrp->applytxix[yer][%d][0] = %d;\n",
    n-1, np->sira);
  fprintf(cp, "\n adrp->applytxix[yer][%d][1] = %d;\n", n-1, l);
}
| SAHAISMI
{
  l += 1;
  n += 1;
  if (n > 10) {
    printf("\n Blokno = %d Hata :
    bypass edilen degisken ve
    saha sayisi 10 dan buyuk olamaz \n",
    bloknm);
    exit(-1);
  }
  if ( (np = bul(atgond, ikuyrnm)) == NULL) {
    printf("\n Blokno = %d Hata :
    Degisken %s sembol tablosunda yok \n", bloknm, atgond);
    exit(-1);
  }
  install(atgond, np->tip, l, okuyrnm, 0);
  fprintf(cp, "\n adrp->applytxix[yer][%d][0] = %d;\n",
    n-1, np->sira);
  fprintf(cp, "\n adrp->applytxix[yer][%d][1] = %d;\n",
    n-1, l);
}
| cikdeglist TVIRG DEGISMI
{
  l += 1;
  n += 1;
  if (n > 10) {
    printf("\n Blokno = %d Hata :
    bypass edilen degisken ve saha sayisi 10 dan
    buyuk olamaz \n", bloknm);

```



```

        exit(-1);
    }
    if (np=bul(vrgond,ikuyrnm))==NULL) {
        printf("\n Blokno=%d Hata :
            Degisken %s sembol tablosunda yok \n",bloknm,vrgond);
        exit(-1);
    }
    install(vrgond,np->tip,l,okuyrnm,0);
    fprintf(cp,"\n adrp->applytxix[yer][%d][0]=%d;\n",
        n-1,np->sira);
    fprintf(cp,"\n adrp->applytxix[yer][%d][1]=%d;\n",
        n-1,l);
}
| cikdeglislist TVIRG SAHAISMI
{l+=1;
n+=1;
if (n>10) {
    printf("\n Blokno=%d Hata :
        bypass edilen degisken ve saha sayisi 10
        dan buyuk olamaz \n",bloknm);
    exit(-1);
}
if (np=bul(atgond,ikuyrnm))==NULL) {
    printf("\n Blokno=%d Hata :
        Degisken %s sembol tablosunda yok \n",
        bloknm,atgond);
    exit(-1);
}
install(atgond,np->tip,l,okuyrnm,0);
fprintf(cp,"\n adrp->applytxix[yer][%d][0]=%d;\n",
    n-1,np->sira);
fprintf(cp,"\n adrp->applytxix[yer][%d][1]=%d;\n",
    n-1,l);
}
;

```

ciksahlist : SAHAISMI

```

{
    install(atgond,'c',l,okuyrnm,0);
    fprintf(cp,"\nsprintf(adrp->applyca[yer][%d],
        \"%s\\\", \"%s\\\");\n",cas-1,fmt,atgond);
}
| ciksahlist TVIRG {cas+=1;l+=1;} SAHAISMI
{
    install(atgond,'c',l,okuyrnm,0);
    fprintf(cp,"\nsprintf(adrp->applyca[yer][%d],
        \"%s\\\", \"%s\\\");\n",cas-1,fmt,atgond);
}
;

```



```
ciksahlist3: ciksahgird3 | ciksahlist3 {l+=1;} TVIRG ciksahgird3;
ciksahgird3 : SAHAISMI
```

```
{
  fprintf(cp, "\nsprintf(adrp->cpistb[yer][ %d],
    \"%s\", \"%s\");\n", l, formt, atgond);
}
```

```
TPIPE
```

```
DEGISMI
```

```
{
  install(vrgond, 'c', l+kuyuz[ikuyrnm], ikuyrnm, 0);
  /* Giriş kuyruk contex'ti aynen dışarı çıkar
  bu nedenle yeni deklare edilen değişkenler
  çıkış kuyruğu yerine giriş kuyruk context'ine
  install edilirler */
}
```

```
| SAHAISMI TNOKTA TNUM
```

```
{
  fprintf(cp, "\nsprintf(adrp->cpistb[yer][ %d],
    \"%s\", \"%s\");\n", l, formt, atgond);
}
```

```
TPIPE
```

```
DEGISMI
```

```
{
  install(vrgond, 'n', l+kuyuz[ikuyrnm], ikuyrnm, 0);
}
```

```
| SAHAISMI TNOKTA TCHAR
```

```
{
  fprintf(cp, "\nsprintf(adrp->cpistb[yer][ %d],
    \"%s\", \"%s\");\n", l, formt, atgond);
}
```

```
TPIPE
```

```
DEGISMI
```

```
{
  install(vrgond, 'c', l+kuyuz[ikuyrnm], ikuyrnm, 0);
}
```

```
;
ciksahlist2: ciksahgird | ciksahlist2 {l+=1;} TVIRG ciksahgird;
```

```
ciksahgird : SAHAISMI
```

```
{
  install(atgond, 'c', l, okuyrnm, 0);
  fprintf(cp, "\nsprintf(adrp->gtistb[yer][ %d],
    \"%s\", \"%s\");\n", l, formt, atgond);
}
```

```
| SAHAISMI TNOKTA TNUM
```

```
{
  install(atgond, 'n', l, okuyrnm, 0);
  fprintf(cp, "\nsprintf(adrp->gtistb[yer][ %d],
    \"%s\", \"%s\");\n", l, formt, atgond);
}
```

```
| SAHAISMI TNOKTA TCHAR
```

```
{
```



```

install(atgond, 'c', l, okuyrnm, 0);
fprintf(cp, "\nsprintf(adrp->gtistb[yer][ %d],
        \"%s\", \"%s\");\n", l, formt, atgond);
}

```

```
;
```

membaslik : T MEMBER

```

{
    blokmp[bloknm][0] = bloknm - 1;
    blokmp[bloknm][1] = ikuyrnm;
}
| geneletiket
T MEMBER
;

```

memblok : membaslik {

```

    blokmp[bloknm][2] = bloknm + 1;
    blokmp[bloknm][3] = okuyrnm;
    bloktip[bloknm] = 4;
    l = 1; /* çıkış girdi sayacı */
    n = 1; /* bypass deg saha sayacı (10 dan büyük olmaz) */
    gas = 1; /* giriş saha sayacı */
    cas = 1; /* çıkış saha sayacı */
    toggle = 'm';
    fprintf(cp, "#include </usr/ccs/bin/meminc.c>\n");
}
girsahlist TBOLOP
{ fprintf(cp, "\n adrp->memgas[yer] = %d ;\n", gas );
  if (gas > kuyuz[ikuyrnm]){
    printf("\n Blokno = %d Ikaz :
           aplikasyon saha sayisi ile giris \n", bloknm);
    printf("\n kuyruk uzunlugu uyusmuyor \n");
  }
  fprintf(cp, "\n adrp->memtlg[yer] = %d ;\n", kuyuz[ikuyrnm]);
}
RELISMI
{ fprintf(cp, "\nsprintf(adrp->memrel[yer],
        \"%s\", \"%s\");\n", formt, rngond);
}
TNOKVIRG {
    /* giris contextini aynen disari gecirir */
};

```

generblok : TGENER {

```

    blokmp[bloknm][2] = bloknm + 1;
    blokmp[bloknm][3] = okuyrnm;
    bloktip[bloknm] = 1;
    fprintf(cp, "#include </usr/ccs/bin/geninc.c>\n");
    l = 1; /* okunacak degişken sayacı ve tx'daki pozisyon */
}

```


RELISMI

```
{fprintf(cp, "\nsprintf(adrp->genistb[yer][0],
    \"%s\\\", \"%s\\\")\n", formt, rngond);
}
```

sahgird

TNOKVIRG

```
{
    ikuyrnm=okuyrnm;
    sontl=1;
    kuyuz[ikuyrnm]=sontl;
    okuyrnm+=1;
};
```

sahgird

: SAHAISMI

```
{fprintf(cp, "\nsprintf(adrp->genistb[yer][1],
    \"%s\\\", \"%s\\\")\n", formt, atgond);
    install(atgond, 'c', 1, okuyrnm, 0);
}
```

| SAHAISMI TNOKTA TNUM

```
{fprintf(cp, "\nsprintf(adrp->genistb[yer][1],
    \"%s\\\", \"%s\\\")\n", formt, atgond);
    install(atgond, 'n', 1, okuyrnm, 0);
}
```

| SAHAISMI TNOKTA TCHAR

```
{fprintf(cp, "\nsprintf(adrp->genistb[yer][1],
    \"%s\\\", \"%s\\\")\n", formt, atgond);
    install(atgond, 'c', 1, okuyrnm, 0);
}
```

;

cartesbaslik : TCARTES {

```
    blokmp[bloknm][0]=bloknm-1;
    blokmp[bloknm][1]=ikuyrnm;
}
```

| geneletiket

TCARTES;

cartesblok : cartesbaslik {

```
    blokmp[bloknm][2]=bloknm+1;
    blokmp[bloknm][3]=okuyrnm;
    bloktip[bloknm]=25;
    fprintf(cp, "#include </usr/ccs/bin/cartesinc.c>\n");
    l=1;
}
```

RELISMI

```
{fprintf(cp, "\nsprintf(adrp->cpistb[yer][0],
    \"%s\\\", \"%s\\\")\n", formt, rngond);
}
```

ciksahlist3


```

TNOKVIRG
{
    ilktl = kuyuz[ikuyrnm];
    fprintf(cp, "\n adrp-> cplg[yer] = %d ; \n", ilktl);
    fprintf(cp, "\n adrp-> cpcas[yer] = %d ; \n", l);
    sontl = l + ilktl;
    kuyuz[ikuyrnm] = sontl;
    /* Giriş kuyruk context'i aynen çıkışa geçer */
};

genertblok : TGENERT {
    blokmp[bloknm][2] = bloknm + 1;
    blokmp[bloknm][3] = okuyrnm;
    bloktip[bloknm] = 2;
    fprintf(cp, "#include </usr/ccs/bin/gentinc.c>\n");
    l = 1; /* okunacak değişken sayacı ve tx'daki pozisyon */
    togle = 'g';
}
RELISMI
{ fprintf(cp, "\nsprintf(adrp-> gtistb[yer][0],
    \"%s\\\", \"%s\\\") ; \n", formt, rngond);
}
ciksahlist2
TNOKVIRG
{
    fprintf(cp, "\n adrp-> gtcas[yer] = %d ; \n", l);
    ikuyrnm = okuyrnm;
    sontl = l;
    kuyuz[ikuyrnm] = l;
    okuyrnm += 1;
};

generebaslik : TGENERE /* generate element ile input sabit eş anlamlı */
{
    blokmp[bloknm][2] = bloknm + 1;
    blokmp[bloknm][3] = okuyrnm;
    bloktip[bloknm] = 2;
    fprintf(cp, "#include </usr/ccs/bin/inpsbinc.c>\n");
};

genereblok : generebaslik
{l = 1;}
generelist
TNOKVIRG
{
    ikuyrnm = okuyrnm;
    sontl = l - 1;
    if (sontl > 10){
        printf("\nBlokno = %d Ikaz :
            GENERE giris sabit sayisi 10'dan buyuk olamaz \n",
                bloknm);
    }
    fprintf(cp, "\n adrp-> inputsbtl[yer] = %d ; \n", l - 1);
}

```



```

        kuyuz[ikuyrnm]=sontl;
        okuyrnm += 1;
    };

generelist : generegirdi | generelist TVIRG generegirdi;

generegirdi : DEGISMI TESOP
    gercel {
        fclose(rp);
        rp=fopen("gecici2","r");
        fscanf(rp,"%f",&cc);
        fprintf(cp,"\n adrp->inputsbf[yer][%d]=%f;\n",l,cc);
        fprintf(cp,"\n adrp->inputsbent[yer][%d]=1;\n",l);
        install(vrgond,'n',l,okuyrnm,0);
        l++;
    }
    | DEGISMI TESOP
    KARAKTERDIZIN
    {
        fprintf(cp,"\n sprintf(adrp->inputsbst[yer][%d],
            \"%s\",\"%s\");\n",l,format,stgond);
        fprintf(cp,"\n adrp->inputsbent[yer][%d]=0;\n",l);
        install(vrgond,'c',l,okuyrnm,0);
        l++;
    }
};

reportblok : repbaslik {
    fprintf(cp,"#include </usr/ccs/bin/repinc.c>\n");
    l=1; /* okunacak degişken sayacı ve tx'daki pozisyon */
}
sayblok
{
    fprintf(cp,"\n adrp->outmat[yer]=%d;\n",l-2);
    if ((l-1)!=kuyuz[ikuyrnm]) {
        printf("\n Blokno=%d Ikaz :
            report blogu giris kuyruk\n",bloknm);
        printf("\n uzunlugu ile kaynak kuyruk \n");
        printf("\n uzunlugu uyusmuyor \n");
    }
    fprintf(cp,"\n adrp->outputtl[yer]=%d;\n",
        kuyuz[ikuyrnm]);

} TNOKVIRG
;

repbaslik : TREPORT
{
    blokmp[bloknm][0]=bloknm-1;
    blokmp[bloknm][1]=ikuyrnm;
    bloktip[bloknm]=8;
    togle='r';

```



```

}
| geneletiket
{   bloktip[bloknm]=8; }
TREPORT ;

```

geneletiket: TETIKET

```

{
    if( (np=bul(lbgond,-1))==NULL) {
        printf("\n Blokno= %d Hata : Etiket sembol tablosunda yok \n",bloknm);
        exit(-1);
    }
    /* -1 context'i etiket'ler context'idir */
    blokmp[bloknm][0]=np->sira;
    cikbloki=2;
    /* transfer bloğunun etiket filler'lar ile doldurduğu
    (o an belli olmayan) çıkış blok gözlerini incele
    ve bu gözlerden ilk rastladığın "filler" ile dolu
    olana kendi blok numaranı doldur */
    while(blokmp[np->sira][cikbloki]!=np->lbtan&&
        blokmp[np->sira][cikbloki]!=0) cikbloki += 2;
    if(blokmp[np->sira][cikbloki]!=np->lbtan){
        printf("\n Blokno= %d Hata :
        Buyuk derleme hatasi etiket uyusmuyor \n",bloknm);
        exit(-1);
    }
    blokmp[np->sira][cikbloki]=bloknm;
    blokmp[bloknm][1]=blokmp[np->sira][cikbloki+1];
    ikuyrnm=blokmp[np->sira][cikbloki+1];
} ;

```

```

degsah : DEGismi {
    if( (np=bul(vrgond,ikuyrnm))==NULL) {
        printf("\n Blokno= %d Hata :
        Degisken %s sembol tablosunda yok \n",bloknm,vrgond);
        exit(-1);
    }
    fprintf(cp, "\naddrp->outputent[yer][2][ %d]= %d;\n",
        l,np->sira);
}

```

```

| SAHAismi {
    if( (np=bul(atgond,ikuyrnm))==NULL) {
        printf("\n Blokno= %d Hata :
        Saha sembol tablosunda yok \n",bloknm);
        exit(-1);
    }
    fprintf(cp, "\naddrp->outputent[yer][2][ %d]= %d;\n",
        l,np->sira);
}

```

} TTEST


```

;
sayblok : TSORU cikis_koordinat TSAY KARAKTERDIZIN
{fprintf(cp, "\nsprintf(adrp->outputfb[yer][ %d],
  \" %s\\\", \" %s\\");\n", l, formt, stgond);
}
TNUM
{fprintf(cp, "\n adrp->outputent[yer][3][ %d] = 1;
  \n", l);}
degsh {l += 1;
} |
TSORU cikis_koordinat TSAY KARAKTERDIZIN
{fprintf(cp, "\nsprintf(adrp->outputfb[yer][ %d],
  \" %s\\\", \" %s\\");\n", l, formt, stgond);
}
TCHAR
{fprintf(cp, "\n adrp->outputent[yer][3][ %d] = 0;\n", l);}
degsh {l += 1;
} |
sayblok TVIRG TSORU cikis_koordinat TSAY KARAKTERDIZIN
{fprintf(cp, "\nsprintf(adrp->outputfb[yer][ %d],
  \" %s\\\", \" %s\\");\n", l, formt, stgond);
}
TNUM
{fprintf(cp, "\n adrp->outputent[yer][3][ %d] = 1;\n", l);}
degsh {l += 1;
} |
sayblok TVIRG TSORU cikis_koordinat TSAY KARAKTERDIZIN
{fprintf(cp, "\nsprintf(adrp->outputfb[yer][ %d],
  \" %s\\\", \" %s\\");\n", l, formt, stgond);
}
TCHAR
{fprintf(cp, "\n adrp->outputent[yer][3][ %d] = 0;\n", l);}
degsh {l += 1;
};
cikis_koordinat : TLBRACK positiftamsayi
{fclose(fp);
  fp = fopen("gecici", "r");
  fscanf(fp, "%d", &k);
  fprintf(cp, "\nadrp->outputent[yer][0][ %d] = %d;\n", l, k);
}
TVIRG positiftamsayi
{fclose(fp);
  fp = fopen("gecici", "r");
  fscanf(fp, "%d", &k);
  fprintf(cp, "\nadrp->outputent[yer][1][ %d] = %d;\n", l, k);
}
TRBRACK;
testbaslik : TTEST

```



```

{   girsahsay=0;
    togle='t';
    blokmp[bloknm][0]=bloknm-1;
    blokmp[bloknm][1]=ikuyrnm;
    cikbloki += 2;
}
| geneletiket
TTEST ;

testblok : testbaslik {
    ksay=0;
    tmptsbent=62;
    tmptsent=32;
    strcnt=92;
    cikbloki=2;
    stp=0; /* stack pointer */
    bloktip[bloknm]=3;
    fprintf(cp, "#include </usr/ccs/bin/testinc.c>\n");
}
mantgrup

hedeflbl
TNOKVIRG

{ /* bitimde sayacıları da çıktıya işle */
    fprintf(cp, "\nadrp->testksay[yer]=%d;\n", ksay);
    fprintf(cp, "\nadrp->testtlc[yer]=%d;\n", kuyuz[ikuyrnm]);

    if (girdegssay!=kuyuz[ikuyrnm]){
        printf("\n Blokno=%d Ikaz :
        Test blogu transaction uzunluk uyusmazligi\n", bloknm);
    }

    fprintf(cp, "\nadrp->testtlg[yer]=%d;\n", kuyuz[ikuyrnm]);
    fprintf(cp, "\nadrp->testgas[yer]=%d;\n", girsahsay);
    sontl=kuyuz[ikuyrnm];
    ikuyrnm=okuyrnm;
    kuyuz[ikuyrnm]=sontl;
    okuyrnm += 1;
}

hedeflbl : TETIKET
{
    install(lbgond, 'n', bloknm, -1, lblfills);
    blokmp[bloknm][cikbloki]=lblfills ;
    blokmp[bloknm][cikbloki+1]=okuyrnm;
    cikbloki += 2;
    lblfills=-1;
}
TVIRG
TETIKET

```



```

{
    install(lbgond,'n',bloknm,-1,lblfills);
    blokmp[bloknm][cikbloki]=lblfills ;
    blokmp[bloknm][cikbloki+1]=okuyrnm;
    cikbloki+=2;
    lblfills-=1;
}
|TETIKET
{
    install(lbgond,'n',bloknm,-1,lblfills);
    blokmp[bloknm][cikbloki]=lblfills ;
    blokmp[bloknm][cikbloki+1]=okuyrnm;
    cikbloki+=2;
    lblfills-=1;
}
;
comfixbaslik : TCOMFIX
{
    blokmp[bloknm][0]=bloknm-1;
    blokmp[bloknm][1]=ikuyrnm;
}
|geneletiket
TCOMFIX
;

comfixblok : comfixbaslik {
    girsahtsay=0;
    ksay=0;
    tmprcnt=32;
    stp=0;
    togle='f';
    blokmp[bloknm][2]=bloknm+1;
    blokmp[bloknm][3]=okuyrnm;
    bloktip[bloknm]=6;
    fprintf(cp,"#include </usr/ccs/bin/compfinc.c>\n");
}

gruplar
TNOKVIRG

{ /* bitimde sayaçları da çıktıya işle */
    fprintf(cp,"nadrp->compfksay[yer]=%d;\n",ksay);
    fprintf(cp,"nadrp->compftlc[yer]=%d;\n",cikdegsay);

    if (girdegsay!=kuyuz[ikuyrnm]){
        printf("\n Blokno=%d Ikaz :
        Comfix blogu transaction uzunluk uyumsuzlugu\n",bloknm);
    }

    fprintf(cp,"nadrp->compftlg[yer]=%d;\n",kuyuz[ikuyrnm]);
    fprintf(cp,"nadrp->compfgas[yer]=%d;\n",girsahtsay);
    sontl=cikdegsay;

```



```

| iliskil ortgrup ;
/* LOCATE hiç olmayabilir */
/* LOCATE saha hangisi ise onu markalayacak
bu degiskeni ise bu sahada LOCATE sahasi olarak kullanacak
ortgrup : aramagrubu atamagrup
{
| atamagrup;
}

iliskil : ilis {girdegsey=1;} | iliskil TVIRG {girdegsey+=1;} ilis ;
/* giriş değişken sayısı girsahsay+değişken sayısıdır */
/* girişte yer alan sahalara blok içinde kullanılmaz
değerlerini bu ilişkilendirmede değişkenlere geçirirler */
ilis : SAHAISMI
{
if( (np=bul(atgond,ikuyrnm))==NULL) {
printf("\n Blokno=%d Hata :
Degisken %s sembol tablosunda yok \n",bloknm,atgond);
exit(-1);
}
}
/* Giriş kuyruğu context'inde bu saha (attribute) varmı bak */
TPIPE
DEGISMI
{ l= np->sira;
install(vrgond,np->tip,l,200+bloknm,0);
install(vrgond,np->tip,l,okuyrnm,0);
}

/* Bu saha ile ilişkilendirilen değişken (variable) blok içi context'de
sembol tablosuna işlenir ve daha önce bulunan sahanın address
space'de daha önce bulunduğu sıraya yerleşir */
| DEGISMI
{
if( (np=bul(vrgond,ikuyrnm))==NULL) {
printf("\n Blokno=%d Hata :
Degisken %s sembol tablosunda yok \n",bloknm,vrgond);
exit(-1);
}
}
TPIPE
DEGISMI
{ l= np->sira;
install(vrgond,np->tip,l,200+bloknm,0);
install(vrgond,np->tip,l,okuyrnm,0);
/* blok içi context numarası
max kuyruk sayısı + bloğun numarası */
}

```



```

;
aramagrubu : TLOCATE {m=0;locmark=-1;} RELISMI
/* locmark locate saha hangisi ise onu markalayacak
m degişkeni ise bu sahada locate sahası olsun olmasın
tüm sahaları belirli bir sıra numarası ile takip eden sayacıdır */
{
if (togle=='c')
    fprintf(cp,"\\nsprintf(adrp->compuse[yer][%d].relname,
        \\\"%s\\\",\\\"%s\\\")\\n",m,fmt,ngond);
if (togle=='f')
    fprintf(cp,"\\nsprintf(adrp->compufse[yer][%d].relname,
        \\\"%s\\\",\\\"%s\\\")\\n",m,fmt,ngond);
if (togle=='t')
    fprintf(cp,"\\nsprintf(adrp->testuse[yer][%d].relname,
        \\\"%s\\\",\\\"%s\\\")\\n",m,fmt,ngond);
if (togle=='u')
    {fprintf(cp,"\\nsprintf(adrp->updtreln[yer],
        \\\"%s\\\",\\\"%s\\\")\\n",fmt,ngond);
    }
if (togle=='d')
    {fprintf(cp,"\\nsprintf(adrp->delent[yer][0].relname,
        \\\"%s\\\",\\\"%s\\\")\\n",fmt,ngond);
    }
}

arag |

aramagrubu TVIRG arag;

arag: SAHAISMI
    TESOP
    {
/* LOCATE deyiminde yer alan sahalar ifadelerde sağ tarafta
yer alabilirler bu nedenle 62'den itibaren address space'de sıra
alacak şekilde sembol tablosuna blok içi context'de yerleştirilirler */
    if (togle=='c'){
        printf("\\n Blokno=%d Hata : Compute blogunda
            kriter sag tarafında saha olamaz \\n",bloknm);
        exit(-1);
    }
    if (togle=='f'){
        printf("\\n Blokno=%d Hata : C
            Compute blogunda kriter sag tarafında saha olamaz \\n",bloknm);
        exit(-1);
    }
    if (togle=='t'){
        printf("\\n Blokno=%d Hata :
            Test blogunda kriter sag tarafında saha olamaz \\n",bloknm);
        exit(-1);
    }
    if (togle=='u'){

```



```

fprintf(cp, "\nsprintf(adrp->updtent[yer][ %d].atname,
        \"%s\\\", \"%s\\\")";\n", m, formt, atgond1);
}
if (togle == 'd'){
    fprintf(cp, "\nsprintf(adrp->delent[yer][ %d].atname,
        \"%s\\\", \"%s\\\")";\n", m, formt, atgond1);
}
girsahsay += 1;
}
SAHAISMI
{
    if (togle == 'u'){
        if( (np=bul(atgond, ikuyrnm)) == NULL) {
            printf("\n Blokno = %d Hata :
                Degisken %s sembol tablosunda yok \n", bloknm, atgond);
            exit(-1);
        }
        fprintf(cp, "\nadrp->updtent[yer][ %d].ind = %d;\n",
            m, np->sira*(-1));
    }
    if (togle == 'd'){
        if( (np=bul(atgond, ikuyrnm)) == NULL) {
            printf("\n Blokno = %d Hata :
                Degisken %s sembol tablosunda yok \n", bloknm, atgond);
            exit(-1);
        }
        fprintf(cp, "\nadrp->delent[yer][ %d].ind = %d;\n",
            m, np->sira*(-1));
    }
    m += 1;
}
}
| SAHAISMI
TESOP
{

```

/* LOCATE deyiminde yer alan sahalar expression'larda sağ tarafta yer alabilirler bu nedenle 62'den itibaren address space de sıra alacak şekilde sembol tablosuna blok içi context'de yerleştirilirler */

```

if (togle == 'c'){
    fprintf(cp, "\nsprintf(adrp->compuse[yer][ %d].atname,
        \"%s\\\", \"%s\\\")";\n", m, formt, atgond);
    install(atgond, 'n', 62 + girsahsay, 200 + bloknm, 0);
}
if (togle == 'f'){
    fprintf(cp, "\nsprintf(adrp->compufse[yer][ %d].atname,
        \"%s\\\", \"%s\\\")";\n", m, formt, atgond);
    install(atgond, 'n', 62 + girsahsay, 200 + bloknm, 0);
}

```



```

if (togle == 't'){
    fprintf(cp, "\nsprintf(adrp->testuse[yer][ %d].atrname,
        \"%s\\\", \"%s\\\")";\n", m, formt, atgond);
    install(atgond, 'n', 82 + girsahsay, 200 + bloknm, 0);
}
if (togle == 'u'){
    fprintf(cp, "\nsprintf(adrp->updtent[yer][ %d].atrname,
        \"%s\\\", \"%s\\\")";\n", m, formt, atgond);
}
if (togle == 'd'){
    fprintf(cp, "\nsprintf(adrp->delent[yer][ %d].atrname,
        \"%s\\\", \"%s\\\")";\n", m, formt, atgond);
}
girsahsay += 1;
}

DEGISMI
{
    if (togle == 'c' || togle == 'f' || togle == 't'){
        if( (np=bul(vrgond, 200 + bloknm)) == NULL) {
            printf("\n Blokno = %d Hata :
                Degisken %s sembol tablosunda yok \n", bloknm, vrgond);
            exit(-1);
        }
        if (np->sira > 31)
            printf("\n Blokno = %d Hata :
                Bu degisken ref olarak kullanilamaz \n", bloknm);
        /* eğer eşitlik şartında ilişkilendirme grubu dışında bir değişken
        kullanıldıysa reddedilir */
    }
    if (togle == 'u' || togle == 'd'){
        if( (np=bul(vrgond, ikuyrnm)) == NULL) {
            printf("\n Blokno = %d Hata :
                Degisken %s sembol tablosunda yok \n", bloknm, vrgond);
            exit(-1);
        }
    }

    if(togle == 'c'){
        fprintf(cp, "\nadrp->complt[yer][ %d] = %d;\n", m, np->sira*(-1));
        fprintf(cp, "\nadrp->compuse[yer][ %d].ind=0;\n", m);
    }
    if(togle == 'f'){
        fprintf(cp, "\nadrp->compflt[yer][ %d] = %d;\n", m, np->sira*(-1));
        fprintf(cp, "\nadrp->compufse[yer][ %d].ind=0;\n", m);
    }
    if(togle == 't'){
        fprintf(cp, "\nadrp->testlt[yer][ %d] = %d;\n", m, np->sira*(-1));
        fprintf(cp, "\nadrp->testuse[yer][ %d].ind=0;\n", m);
    }
    if(togle == 'u'){
        fprintf(cp, "\nadrp->updtent[yer][ %d].ind = %d;\n", m, np->sira*(-1));
    }
}

```



```

    }
    if(togle == 'd'){
        fprintf(cp, "\nadrp->delent[yer][ %d].ind = %d;\n", m, np->sira*(-1));
    }
    if (locmark == -1) locmark = m;
/* locate'e esas olacak veri variable'da ise bu variable'in sıra nosunu
negatif olarak lt içine yerleşecek şekilde complt'ye depola, ind
değişkenine de bu sahanın locate sahası olduğunu belli edecek
şekilde 0 yerleştir */
    m += 1;
}
| SAHAISMI
TESOP gercel
{
if (togle == 'c'){
    fprintf(cp, "\nsprintf(adrp->compuse[yer][ %d].atname,
        \" %s\", \" %s\");\n", m, format, atgond);
    install(atgond, 'n', 62 + girsahsay, 200 + bloknm, 0);
}
if (togle == 'f'){
    fprintf(cp, "\nsprintf(adrp->compuse[yer][ %d].atname,
        \" %s\", \" %s\");\n", m, format, atgond);
    install(atgond, 'n', 62 + girsahsay, 200 + bloknm, 0);
}
if (togle == 't'){
    fprintf(cp, "\nsprintf(adrp->testuse[yer][ %d].atname,
        \" %s\", \" %s\");\n", m, format, atgond);
    install(atgond, 'n', 82 + girsahsay, 200 + bloknm, 0);
}
if (togle == 'u'){
    fprintf(cp, "\nsprintf(adrp->updtent[yer][ %d].atname,
        \" %s\", \" %s\");\n", m, format, atgond);
}
if (togle == 'u'){
    fprintf(cp, "\nsprintf(adrp->delent[yer][ %d].atname,
        \" %s\", \" %s\");\n", m + 1, format, atgond);
}
}
fclose(rp);
rp = fopen("gecici2", "r");
fscanf(rp, "%f", &cc);

fprintf(cp, "cc = %f;\n", cc);
fprintf(cp, "paketd(tempbuf, (char *)&cc);\n");
fprintf(cp, "sident = altyapi(ght_buf, gct_buf, tempbuf);\n");
fprintf(cp, "if(sident == 0){\n");
fprintf(cp, "    printf(\"CALISMA ZAMANI HATASI \");\n");
fprintf(cp, "    exit(-1) ; \n");
fprintf(cp, "};\n");
if (togle == 'c'){
    fprintf(cp, "\nadrp->complt[yer][ %d] = sident;\n", m);
    fprintf(cp, "\nadrp->compuse[yer][ %d].ind = 0;\n", m);
}

```



```

}
if (toggle == 'f'){
    fprintf(cp, "\nadrp->compflt[yer][ %d] = sident;\n", m);
    fprintf(cp, "\nadrp->compufse[yer][ %d].ind = 0;\n", m);
}
if (toggle == 't'){
    fprintf(cp, "\nadrp->testlt[yer][ %d] = sident;\n", m);
    fprintf(cp, "\nadrp->testuse[yer][ %d].ind = 0;\n", m);
}
if (toggle == 'u'){
    fprintf(cp, "\nadrp->updtent[yer][ %d].ind = sident;\n", m);
}
if (toggle == 'd'){
    fprintf(cp, "\nadrp->delent[yer][ %d].ind = sident;\n", m);
}
if (locmark == -1) locmark = m;
girsahsay += 1;
m += 1;

/* saha ismini ilgili shared memory yapısına işleyecek deyimleri
c çıktı dosyasına yaz, bu sahayı mevcut blok içi context'de
62'den başlıyarak sıra no vererek sembol tablosuna yerleştir */
}

```

```

/* burada belirtilen sayıyı ght'de ve gct'de
bulup sident değerini complt'e pozitif
değer olarak yerleştirecek kod parçasını
yazacak c kodu yerlesiyor*/

```

| SAHAISMI TESOP KARAKTERDIZIN

```

{
if (toggle == 'c'){
    fprintf(cp, "\nsprintf(adrp->compuse[yer][ %d].atrname,
        \"%s\", \"%s\");\n", m, format, atgond);
    install(atgond, 'n', 62 + girsahsay, 200 + bloknm, 0);
}
if (toggle == 'f'){
    fprintf(cp, "\nsprintf(adrp->compufse[yer][ %d].atrname,
        \"%s\", \"%s\");\n", m, format, atgond);
    install(atgond, 'n', 62 + girsahsay, 200 + bloknm, 0);
}
if (toggle == 't'){
    fprintf(cp, "\nsprintf(adrp->testuse[yer][ %d].atrname,
        \"%s\", \"%s\");\n", m, format, atgond);
    install(atgond, 'n', 82 + girsahsay, 200 + bloknm, 0);
}
if (toggle == 'u'){
    fprintf(cp, "\nsprintf(adrp->updtent[yer][ %d].atrname,
        \"%s\", \"%s\");\n", m, format, atgond);
}
if (toggle == 'd'){

```

```

fprintf(cp, "\nsprintf(adrp->delent[yer][ %d].atname,
                \"%s\", \"%s\");\n", m, format, atgond);
}
fprintf(cp, "\nsprintf(tempbuf, \"%s\", \"%s\");\n", format, stgond);
fprintf(cp, "sident=altyapi(ght_buf, gct_buf, tempbuf);\n");
fprintf(cp, "if(sident==0){\n");
fprintf(cp, "    printf(\"CALISMA ZAMANI HATASI \");\n");
fprintf(cp, "    exit(-1) ; \n");
fprintf(cp, "}\n");
if (togle=='c'){
    fprintf(cp, "\nadrp->complt[yer][ %d]=sident;\n", m);
    fprintf(cp, "\nadrp->compuse[yer][ %d].ind=0;\n", m);
}
if (togle=='f'){
    fprintf(cp, "\nadrp->compflt[yer][ %d]=sident;\n", m);
    fprintf(cp, "\nadrp->compufse[yer][ %d].ind=0;\n", m);
}
if (togle=='t'){
    fprintf(cp, "\nadrp->testlt[yer][ %d]=sident;\n", m);
    fprintf(cp, "\nadrp->testuse[yer][ %d].ind=0;\n", m);
}
if (togle=='u'){
    fprintf(cp, "\nadrp->updtent[yer][ %d].ind=sident;\n", m);
}
if (togle=='d'){
    fprintf(cp, "\nadrp->delent[yer][ %d].ind=sident;\n", m);
}
if (locmark==-1) locmark=m;
m+=1;
girsahsay+=1;
}
| SAHAISMI
{
if (togle=='c'){
    fprintf(cp, "\nsprintf(adrp->compuse[yer][ %d].atname,
                \"%s\", \"%s\");\n", m, format, atgond);
    fprintf(cp, "\nadrp->complt[yer][ %d]=0;\n", m);
/* Bu saha locate sahası değil bu nedenle tnumind'e karşılık gelen ind
   girdisi locmark ile belirlenen locate girdisinin 1 fazlası,
   compute bu değerden 1 çıkartarak tnumind'de locate sahasını bulur .
   yine bu sahanın lt karşılığı locate sahası don't care anlamında 0'a
   eşitlenir */
    fprintf(cp, "\nadrp->compuse[yer][ %d].ind= %d;\n", m, locmark+1);
}
if (togle=='f'){
    fprintf(cp, "\nsprintf(adrp->compufse[yer][ %d].atname,
                \"%s\", \"%s\");\n", m, format, atgond);
    fprintf(cp, "\nadrp->compflt[yer][ %d]=0;\n", m);
    fprintf(cp, "\nadrp->compufse[yer][ %d].ind= %d;\n",
            m, locmark+1);
    install(atgond, 'n', 62 + girsahsay, 200 + bloknm, 0);
}

```



```

}
if (togle=='t'){
    fprintf(cp, "\nsprintf(adrp->testuse[yer][ %d].atrname,
        \"%s\\\", \"%s\\\")\n", m, formt, atgond);
    fprintf(cp, "\nadrp->testlt[yer][ %d]=0;\n", m);
    fprintf(cp, "\nadrp->testuse[yer][ %d].ind = %d;\n", m, locmark + 1);
    install(atgond, 'n', 82 + girsahsay, 200 + bloknm, 0);
}
if (togle=='u'){
    printf("\n Blokno = %d Hata : Soz dizimi kurali hatasi
        Update blogu tek basina saha ismi
        iceremez \n", bloknm);
}
if (togle=='d'){
    printf("\n Blokno = %d Hata :
        Soz dizimi kurali hatasi Delete
        blogu tek basina saha ismi iceremez \n", bloknm);
}
m += 1;
girsahsay += 1;
}
;
mantiksata : TASSIGN DEGISMI
{
    if( (np=bul(vrgond, 200 + bloknm)) == NULL) {
        install(vrgond, 'n', tmptsbent, 200 + bloknm, 0);
        temp = tmptsbent;
        if (tmptsbent == 1;
        } else { temp = np->sira; }
        /* atama sol taraf degiskeni sembol tablosuna isle
        bu ayni zamanda bir oto deklarasyon olup temps
        veri sahasından veri alanı kullanır */
        /* eğer sembol tablosunda daha önce yer alıyorsa
        bu eski sıra numarasını kullanır */
        /* bu degiskenin sıra numarasını daha aşağıda kullanılmak
        uzere sakla */
    }
    TESOP
    mantifade
    {
        ll = 0;
        pop1(&kk, &stp); /* stack de son kalan ifade
            sonucunun atandığı geçici değişken */
        fprintf(cp, "\nadrp->testinst[yer][ %d][1] = %d;\n", ksay, kk);
        fprintf(cp, "\nadrp->testinst[yer][ %d][2] = %d;\n", ksay, ll);
        fprintf(cp, "\nadrp->testinst[yer][ %d][0] = %d;\n", ksay, temp);
        fprintf(cp, "\nadrp->testop[yer][ %d] = '=';\n", ksay);
        fprintf(cp, "\nadrp->testicrk[yer][ %d] = %d %02d;\n", ksay, kk, ll);
        ksay += 1;
    }
    /* atama işleminin sonu :
        stack'de kalan son geçici değişken numarası aritmetik

```

ifadenin sonucunun atandığı geçici değişkeni gösterir
 burada yapılacak iş bu değişkenin değerini atama sol taraf
 değişkenine atayacak bir yarı komutu (pseudo komutu)
 komut dizisinin sonuna eklemektir */

```

    }
    ;
atamagrup : TASSIGN DEGismi
{
    if( (np=bul(vrgond,200+bloknm))==NULL) {
        install(vrgond,'n',tmpscnt,200+bloknm,0);
        temp=tmpscnt;
        tmpscnt += 1;
    } else { temp=np->sira; }
    /* atama sol taraf değişkenini sembol tablosuna işle
       bu aynı zamanda bir oto deklarasyon olup temps
       veri sahasından veri alanı kullanır */
    /* eğer sembol tablosunda daha önce yer alıyorsa
       bu eski sıra numarasını kullanır */
    /* bu değişkenin sıra numarasını daha aşağıda kullanılmak
       üzere sakla */
}
TESOP ifade
{
    ll=0;
    popl(&kk,&stp); /* stack'de son kalan ifade
                      sonucunun atandığı geçici değişken */
    if (togle=='c'){
        fprintf(cp,"\nadrp->compinst[yer][%d][1]=%d;\n",ksay,kk);
        fprintf(cp,"\nadrp->compinst[yer][%d][2]=%d;\n",ksay,ll);
        fprintf(cp,"\nadrp->compinst[yer][%d][0]=%d;\n",ksay,temp);
        fprintf(cp,"\nadrp->compop[yer][%d]='';\n",ksay);
        fprintf(cp,"\nadrp->compicrk[yer][%d]=%d%02d;\n",ksay,kk,ll);
    }
    if (togle=='f'){
        fprintf(cp,"\nadrp->compfinst[yer][%d][1]=%d;\n",ksay,kk);
        fprintf(cp,"\nadrp->compfinst[yer][%d][2]=%d;\n",ksay,ll);
        fprintf(cp,"\nadrp->compfinst[yer][%d][0]=%d;\n",ksay,temp);
        fprintf(cp,"\nadrp->compfop[yer][%d]='';\n",ksay);
        fprintf(cp,"\nadrp->compficrk[yer][%d]=%d%02d;\n",ksay,kk,ll);
    }
    tmpscnt += 1;
    ksay += 1;
/* atama işleminin sonu :
   stack'de kalan son geçici değişken numarası aritmetik
   ifadenin sonucunun atandığı geçici değişkeni gösterir
   burada yapılacak iş bu değişkenin değerini atama sol taraf
   değişkenine atayacak bir pseudo komutu komut dizisinin
   sonuna eklemektir */
}

```


| TASSIGN SAHAISMI

```
{ if( (np=bul(atgond,200+bloknm))==NULL) {
    printf("\n Blokno= %d Hata :
    Bu saha tanimli degil derleme durdu \n",bloknm);
} else { temp=np->sira ; }
```

```
} /* bu degiskenin sira numarasini data asagida tutulmus
```

TESOP ifade

```
{
    ll=0;
    popl(&kk,&stp); /* stack'de son kalan ifade
                        sonucunun atandigi gecici degisken */
    if (togle=='c'){
```

```
        printf("\nBlokno= %d Hata :
        Compute blogu ifadelerinde sol tarafta saha\n",bloknm);
        printf("\nyer alamaz\n");
    }
```

```
    if (togle=='f'){
        if (SONVARTIP=='n'){
            fprintf(cp,"\nadrp->compfinst[yer][ %d][1]= %d;\n",ksay,kk);
            fprintf(cp,"\nadrp->compfinst[yer][ %d][2]= %d;\n",ksay,ll);
            fprintf(cp,"\nadrp->compfinst[yer][ %d][0]= %d;\n",ksay,temp);
            fprintf(cp,"\nadrp->compfop[yer][ %d]= ';\n",ksay);
            fprintf(cp,"\nadrp->compficrk[yer][ %d]= %d%02d;\n",ksay,kk,ll);
        }
```

```
        if (SONVARTIP=='c'){
            /* Eger sahaya atamada SONVARTIP character tipinde ise
            ifade aslında tek char tipi degiskenden ibarettir ve bu degisken
            ile tasınan sident deęeri & operasyonu ile direk comfix
            blogu ile belirtilen sahaya atanmaktadır */
```

```
            fprintf(cp,"\nadrp->compfinst[yer][ %d][1]= %d;\n",ksay,kk);
            fprintf(cp,"\nadrp->compfinst[yer][ %d][2]= %d;\n",ksay,ll);
            fprintf(cp,"\nadrp->compfinst[yer][ %d][0]= %d;\n",ksay,temp);
            fprintf(cp,"\nadrp->compfop[yer][ %d]= '&';\n",ksay);
            fprintf(cp,"\nadrp->compficrk[yer][ %d]= %d%02d;\n",ksay,kk,ll);
        }
```

```
    }
```

```
    tmpscnt += 1;
```

```
    ksay += 1;
```

/* atama işleminin sonu :

stack'de kalan son geçici değişken numarası aritmetik ifadenin sonucunun atandığı geçici değişkeni gösterir burada yapılacak iş bu değişkenin değerini atama sol taraf sahaya atayacak bir pseudo komutu komut dizisinin sonuna eklemektir */

```
}
```

```
printf("\nBlokno= %d Hata :
```

```
Compute blogu ifadelerinde sol tarafta saha\n",bloknm);
```

```
printf("\nyer alamaz\n");
```

| atamagrup TVIRG

DEGISMI

```

{if( (np=bul(vrgond,200+bloknm))==NULL) {
install(vrgond,'n',tmpscst,200+bloknm,0);
temp=tmpscst;
tmpscst+=1;
}else { temp=np->sira; }
/* bu deęişkenin sıra numarasını daha aşağıda kullanılmak
üzere sakla */
}
TESOP ifade
{
ll=0;
pop1(&kk,&stp); /* stack'de son kalan ifade
sonucunun atandığı geçici deęişken */
if (togle=='c'){
fprintf(cp,"\\nadrp->compinst[yer][ %d][1] = %d;\\n",ksay,kk);
fprintf(cp,"\\nadrp->compinst[yer][ %d][2] = %d;\\n",ksay,ll);
fprintf(cp,"\\nadrp->compinst[yer][ %d][0] = %d;\\n",ksay,temp);
fprintf(cp,"\\nadrp->compop[yer][ %d] = '=';\\n",ksay);
fprintf(cp,"\\nadrp->compicrk[yer][ %d] = %d %02d;\\n",ksay,kk,ll);
}
if (togle=='f'){
fprintf(cp,"\\nadrp->compfinst[yer][ %d][1] = %d;\\n",ksay,kk);
fprintf(cp,"\\nadrp->compfinst[yer][ %d][2] = %d;\\n",ksay,ll);
fprintf(cp,"\\nadrp->compfinst[yer][ %d][0] = %d;\\n",ksay,temp);
fprintf(cp,"\\nadrp->compfop[yer][ %d] = '=';\\n",ksay);
fprintf(cp,"\\nadrp->compficrk[yer][ %d] = %d %02d;\\n",ksay,kk,ll);
}
tmpscst+=1;
ksay+=1;
}
| atamagrup TVIRG
SAHAISMI
{if( (np=bul(atgond,200+bloknm))==NULL) {
printf("\\n Blokno = %d Hata :
Bu saha bulunamadi derleme durduruldu \\n",bloknm);
}else { temp=np->sira; }
/* bu deęişkenin sıra numarasını daha aşağıda kullanılmak
üzere sakla */
}
TESOP ifade
{
ll=0;
pop1(&kk,&stp); /* stack'de son kalan ifade
sonucunun atandığı geçici deęişken */
if (togle=='c'){
printf("\\nBlokno = %d Hata :
Compute blogu ifadelerinde sol tarafta saha\\n",bloknm);
printf("\\nyer alamaz\\n");
}
if (togle=='f'){

```



```

if (SONVARTIP == 'n'){
    fprintf(cp, "\nadrp->compfinst[yer][ %d][1] = %d;\n", ksay, kk);
    fprintf(cp, "\nadrp->compfinst[yer][ %d][2] = %d;\n", ksay, ll);
    fprintf(cp, "\nadrp->compfinst[yer][ %d][0] = %d;\n", ksay, temp);
    fprintf(cp, "\nadrp->compfop[yer][ %d] = '=';\n", ksay);
    fprintf(cp, "\nadrp->compficrk[yer][ %d] = %d %02d;\n", ksay, kk, ll);
}
if (SONVARTIP == 'c'){
    fprintf(cp, "\nadrp->compfinst[yer][ %d][1] = %d;\n", ksay, kk);
    fprintf(cp, "\nadrp->compfinst[yer][ %d][2] = %d;\n", ksay, ll);
    fprintf(cp, "\nadrp->compfinst[yer][ %d][0] = %d;\n", ksay, temp);
    fprintf(cp, "\nadrp->compfop[yer][ %d] = '&';\n", ksay);
    fprintf(cp, "\nadrp->compficrk[yer][ %d] = %d %02d;\n", ksay, kk, ll);
}
}
}
tmpscnt += 1;
ksay += 1;
}
;

cikgrup : TOUTPUT
DEGISMI
{
    cikdegsey = 1;
    if( (np = bul(vrgond, 200 + bloknm)) == NULL) {
        printf("\n Blokno = %d Hata :
        Degisken %s sembol tablosunda yok \n", bloknm, vrgond);
        exit(-1);
    }
    install(vrgond, 'n', cikdegsey, okuyrnm, 0);
    ll = 98;
    if (togle == 'c'){
        fprintf(cp, "\nadrp->compinst[yer][ %d][1] = %d;\n", ksay, np->sira);
        fprintf(cp, "\nadrp->compinst[yer][ %d][2] = %d;\n", ksay, ll);
        fprintf(cp, "\nadrp->compinst[yer][ %d][0] = %d;\n",
            ksay, cikdegsey + 71);
        /* hedef variable'lar çıkış variable data alanı için ayrılan alan
        olan 72+... da yer alan variable'lardır */
        /* cikis variable'ları listede gözüktükleri sıra ile sembol tablosuna
        işlenirler, bu variable'lar output kuyruk context'inde yer alırlar
        bu variable'ların data alanına dahili variable'ların data alanalarından
        kopyalama yapan atama mikro komutları komut dizisinin sonuna
        ilave edilir. */

        fprintf(cp, "\nadrp->compop[yer][ %d] = '=';\n", ksay);
        fprintf(cp, "\nadrp->compicrk[yer][ %d] = %d %02d;\n",
            ksay, np->sira, ll);
    }
    if (togle == 'f'){
        fprintf(cp, "\nadrp->compfinst[yer][ %d][1] = %d;\n",
            ksay, np->sira);
    }
}

```

```

fprintf(cp, "\nadrp->compfinst[yer][ %d][2] = %d;\n",
        ksay,ll);
fprintf(cp, "\nadrp->compfinst[yer][ %d][0] = %d;\n",
        ksay,cikdegsay + 71);
fprintf(cp, "\nadrp->compfop[yer][ %d] = '=';\n",
        ksay);
fprintf(cp, "\nadrp->compficrk[yer][ %d] = %d %02d;\n",
        ksay,np->sira,ll);
}
ksay += 1;
}
| cikgrup TVIRG
DEGISMI
{ cikdegsay += 1;
  if( (np=bul(vrgond,200+bloknm)) == NULL) {
    printf("\n Blokno = %d Hata :
    Degisken %s sembol tablosunda yok \n",bloknm,vrgond);
    exit(-1);
  }
  install(vrgond,'n',cikdegsay,okuyrnm,0);
  ll = 98;
  if (togle == 'c'){
    fprintf(cp, "\nadrp->compinst[yer][ %d][1] = %d;\n",
            ksay,np->sira);
    fprintf(cp, "\nadrp->compinst[yer][ %d][2] = %d;\n",
            ksay,ll);
    fprintf(cp, "\nadrp->compinst[yer][ %d][0] = %d;\n",
            ksay,cikdegsay + 71);

```

/* hedef variable'lar çıkış variable data alanı için ayrılan alan olan 72+... da yer alan variable'lardır */
 /* cikis variable'ları listede gözüktükleri sıra ile sembol tablosuna işlenirler, bu variable'lar output kuyruk context'inde yer alırlar bu variable'ların data alanına dahili variable'ların data alanalarından kopyalama yapan atama mikro komutları komut dizisinin sonuna ilave edilir. */

```

fprintf(cp, "\nadrp->compop[yer][ %d] = '=';\n",ksay);
fprintf(cp, "\nadrp->compicrk[yer][ %d] = %d %02d;\n",
        ksay,np->sira,ll);
}
if (togle == 'f'){
  fprintf(cp, "\nadrp->compfinst[yer][ %d][1] = %d;\n",
          ksay,np->sira);
  fprintf(cp, "\nadrp->compfinst[yer][ %d][2] = %d;\n",ksay,ll);
  fprintf(cp, "\nadrp->compfinst[yer][ %d][0] = %d;\n",
          ksay,cikdegsay + 71);
  fprintf(cp, "\nadrp->compfop[yer][ %d] = '=';\n",ksay);
  fprintf(cp, "\nadrp->compficrk[yer][ %d] = %d %02d;\n",
          ksay,np->sira,ll);

```



```

    }
    ksay += 1;
}
;

mantifade : TUNLEM andorifade
{
    ll=0;
    pop1(&kk,&stp);
    fprintf(cp, "\nadrp->testinst[yer][%d][1] = %d;\n", ksay, kk);
    fprintf(cp, "\nadrp->testinst[yer][%d][2] = %d;\n", ksay, ll);
    fprintf(cp, "\nadrp->testinst[yer][%d][0] = %d;\n", ksay, tmpsbcnt);
    fprintf(cp, "\nadrp->testop[yer][%d] = '!\n', ksay);
    fprintf(cp, "\nadrp->testicrk[yer][%d] = %d%02d;\n", ksay, kk, ll);
    push(tmpsbcnt, &stp);
    tmpsbcnt += 1;
    ksay += 1;
}
| andorifade;

andorifade : andorifade TAMPS TAMPS karsifadesi
{
    pop(&kk, &ll, &stp);
    fprintf(cp, "\nadrp->testinst[yer][%d][1] = %d;\n", ksay, kk);
    fprintf(cp, "\nadrp->testinst[yer][%d][2] = %d;\n", ksay, ll);
    fprintf(cp, "\nadrp->testinst[yer][%d][0] = %d;\n",
            ksay, tmpsbcnt);
    fprintf(cp, "\nadrp->testop[yer][%d] = '&'\n", ksay);
    fprintf(cp, "\nadrp->testicrk[yer][%d] = %d%02d;\n",
            ksay, kk, ll);
    push(tmpsbcnt, &stp);
    tmpsbcnt += 1;
    ksay += 1;
}
| andorifade TPIPE TPIPE karsifadesi
{
    pop(&kk, &ll, &stp);
    fprintf(cp, "\nadrp->testinst[yer][%d][1] = %d;\n", ksay, kk);
    fprintf(cp, "\nadrp->testinst[yer][%d][2] = %d;\n", ksay, ll);
    fprintf(cp, "\nadrp->testinst[yer][%d][0] = %d;\n", ksay, tmpsbcnt);
    fprintf(cp, "\nadrp->testop[yer][%d] = '|'\n", ksay);
    fprintf(cp, "\nadrp->testicrk[yer][%d] = %d%02d;\n", ksay, kk, ll);
    push(tmpsbcnt, &stp);
    tmpsbcnt += 1;
    ksay += 1;
}
| andorifade TAMPS TAMPS kardizini
{
    pop(&kk, &ll, &stp);
    fprintf(cp, "\nadrp->testinst[yer][%d][1] = %d;\n", ksay, kk);

```

```

fprintf(cp, "\nadrp->testinst[yer][ %d][2] = %d;\n", ksay, ll);
fprintf(cp, "\nadrp->testinst[yer][ %d][0] = %d;\n", ksay, tmpsbcnt);
fprintf(cp, "\nadrp->testop[yer][ %d] = '&';\n", ksay);
fprintf(cp, "\nadrp->testicrk[yer][ %d] = %d %02d;\n", ksay, kk, ll);
push(tmpsbcnt, &stp);
tmplsbcnt + = 1;
ksay + = 1;
}
| andorifade TPIPE TPIPE kardizinizif
{
    pop(&kk, &ll, &stp);
    fprintf(cp, "\nadrp->testinst[yer][ %d][1] = %d;\n", ksay, kk);
    fprintf(cp, "\nadrp->testinst[yer][ %d][2] = %d;\n", ksay, ll);
    fprintf(cp, "\nadrp->testinst[yer][ %d][0] = %d;\n", ksay, tmpsbcnt);
    fprintf(cp, "\nadrp->testop[yer][ %d] = ' '|';\n", ksay);
    fprintf(cp, "\nadrp->testicrk[yer][ %d] = %d %02d;\n", ksay, kk, ll);
    push(tmpsbcnt, &stp);
    tmpsbcnt + = 1;
    ksay + = 1;
}
| TLPARA kardizinizif TRPARA
| TLPARA karsifadesi TRPARA
| kardizinizif
| karsifadesi;

```

kardizinizif: KARAKTERDIZIN TESOP TESOP DEGISMI

```

{
    if( (np=bul(vrgond, 200+bloknm)) == NULL) {
        printf("\n Blokno = %d Hata :
            Degisken %s sembol tablosunda yok \n", bloknm, vrgond);
        exit(-1);
    }
    fprintf(cp, "\nsprintf(adrp->teststr[yer][ %d],
        \"%s\\\", \"%s\\\")\n", strcnt-92, formt, stgond);
    fprintf(cp, "\nadrp->testinst[yer][ %d][1] = %d;\n",
        ksay, strcnt);
    kk = strcnt;
    ll = np->sira;
    strcnt + = 1;
    if (np->sira > 97)
        printf("\n Blokno = %d Hata :
            %s degiskeninin transaction daki
            pozisyonu sorunlu\n", bloknm, vrgond);
    else
        fprintf(cp, "\nadrp->testinst[yer][ %d][2] = %d;\n",
            ksay, np->sira);
    fprintf(cp, "\nadrp->testinst[yer][ %d][0] = %d;\n",
        ksay, tmpsbcnt);
    fprintf(cp, "\nadrp->testicrk[yer][ %d] = %d %02d;\n",

```



```

                                ksay, kk, ll);
fprintf(cp, "\nadrp->testop[yer][%d]='>';\n",
                                ksay);
push(tmpsbcnt, &stp);
tmpsbcnt += 1;
ksay += 1;
}
| KARAKTERDIZIN TESDEGIL DEGISMI
{
if( (np=bul(vrgond, 200 + bloknm)) == NULL) {
    printf("\n Blokno = %d Hata :
        Degisken %s sembol tablosunda yok \n", bloknm, vrgond);
    exit(-1);
}
fprintf(cp, "\nfprintf(adrp->teststr[yer][%d],
    \" %s\\\", \" %s\\\" );\n", strcnt-92, format, stgond);
fprintf(cp, "\nadrp->testinst[yer][%d][1] = %d;\n",
                                ksay, strcnt);
kk = strcnt;
ll = np->sira;
strcnt += 1;
if (np->sira > 97)
    printf("\n Blokno = %d Hata :
        %s degiskeninin transaction daki pozisyonu sorunlu\n",
        bloknm, vrgond);
else
    fprintf(cp, "\nadrp->testinst[yer][%d][2] = %d;\n",
                                ksay, np->sira);
fprintf(cp, "\nadrp->testinst[yer][%d][0] = %d;\n",
                                ksay, tmpsbcnt);
fprintf(cp, "\nadrp->testicrk[yer][%d] = %d %02d;\n",
                                ksay, kk, ll);
fprintf(cp, "\nadrp->testop[yer][%d]='>';\n", ksay);
push(tmpsbcnt, &stp);
tmpsbcnt += 1;
ksay += 1;
}
;

```

karsifadesi : ifade TBUYUK ifade

```

{
    pop(&kk, &ll, &stp);
    fprintf(cp, "\nadrp->testinst[yer][%d][1] = %d;\n", ksay, kk);
    fprintf(cp, "\nadrp->testinst[yer][%d][2] = %d;\n", ksay, ll);
    fprintf(cp, "\nadrp->testinst[yer][%d][0] = %d;\n", ksay, tmpsbcnt);
    fprintf(cp, "\nadrp->testop[yer][%d]='>';\n", ksay);
    fprintf(cp, "\nadrp->testicrk[yer][%d] = %d %02d;\n", ksay, kk, ll);
    push(tmpsbcnt, &stp);
    tmpsbcnt += 1;
    ksay += 1;
}

```

```

|ifade TKUCUK ifade
{
    pop(&kk,&ll,&stp);
    fprintf(cp, "\nadrp->testinst[yer][ %d][1] = %d;\n", ksay, kk);
    fprintf(cp, "\nadrp->testinst[yer][ %d][2] = %d;\n", ksay, ll);
    fprintf(cp, "\nadrp->testinst[yer][ %d][0] = %d;\n", ksay, tmpsbcnt);
    fprintf(cp, "\nadrp->testop[yer][ %d] = ' < ';\n", ksay);
    fprintf(cp, "\nadrp->testicrk[yer][ %d] = %d %02d;\n", ksay, kk, ll);
    push(tmpsbcnt, &stp);
    tmpsbcnt += 1;
    ksay += 1;
}

|ifade {} TBUYUKES ifade
{
    pop(&kk,&ll,&stp);
    fprintf(cp, "\nadrp->testinst[yer][ %d][1] = %d;\n", ksay, kk);
    fprintf(cp, "\nadrp->testinst[yer][ %d][2] = %d;\n", ksay, ll);
    fprintf(cp, "\nadrp->testinst[yer][ %d][0] = %d;\n", ksay, tmpsbcnt);
    fprintf(cp, "\nadrp->testop[yer][ %d] = 'b';\n", ksay);
    fprintf(cp, "\nadrp->testicrk[yer][ %d] = %d %02d;\n", ksay, kk, ll);
    push(tmpsbcnt, &stp);
    tmpsbcnt += 1;
    ksay += 1;
}

|ifade TKUCUKES ifade
{
    pop(&kk,&ll,&stp);
    fprintf(cp, "\nadrp->testinst[yer][ %d][1] = %d;\n", ksay, kk);
    fprintf(cp, "\nadrp->testinst[yer][ %d][2] = %d;\n", ksay, ll);
    fprintf(cp, "\nadrp->testinst[yer][ %d][0] = %d;\n", ksay, tmpsbcnt);
    fprintf(cp, "\nadrp->testop[yer][ %d] = 'k';\n", ksay);
    fprintf(cp, "\nadrp->testicrk[yer][ %d] = %d %02d;\n", ksay, kk, ll);
    push(tmpsbcnt, &stp);
    tmpsbcnt += 1;
    ksay += 1;
}

|ifade TESDEGIL ifade
{
    pop(&kk,&ll,&stp);
    fprintf(cp, "\nadrp->testinst[yer][ %d][1] = %d;\n", ksay, kk);
    fprintf(cp, "\nadrp->testinst[yer][ %d][2] = %d;\n", ksay, ll);
    fprintf(cp, "\nadrp->testinst[yer][ %d][0] = %d;\n", ksay, tmpsbcnt);
    fprintf(cp, "\nadrp->testop[yer][ %d] = 'n';\n", ksay);
    fprintf(cp, "\nadrp->testicrk[yer][ %d] = %d %02d;\n", ksay, kk, ll);
    push(tmpsbcnt, &stp);
    tmpsbcnt += 1;
    ksay += 1;
}

|ifade TESOP TESOP ifade
{
    pop(&kk,&ll,&stp);

```



```

fprintf(cp, "\nadrp->testinst[yer][ %d][1] = %d;\n", ksay, kk);
fprintf(cp, "\nadrp->testinst[yer][ %d][2] = %d;\n", ksay, ll);
fprintf(cp, "\nadrp->testinst[yer][ %d][0] = %d;\n", ksay, tmpsbcnt);
fprintf(cp, "\nadrp->testop[yer][ %d] = 'e';\n", ksay);
fprintf(cp, "\nadrp->testicrk[yer][ %d] = %d %02d;\n", ksay, kk, ll);
push(tmpsbcnt, &stp);
tmpsbcnt += 1;
ksay += 1;
};

```

ifade: ifade TARTIOP

carpifadesi

```

{
    pop(&kk, &ll, &stp);
    if (togle == 'c'){
        fprintf(cp, "\nadrp->compinst[yer][ %d][1] = %d;\n", ksay, kk);
        fprintf(cp, "\nadrp->compinst[yer][ %d][2] = %d;\n", ksay, ll);
        fprintf(cp, "\nadrp->compinst[yer][ %d][0] = %d;\n", ksay, tmpscnt);
        fprintf(cp, "\nadrp->compop[yer][ %d] = ' + ';\n", ksay);
        fprintf(cp, "\nadrp->compicrk[yer][ %d] = %d %02d;\n", ksay, kk, ll);
    }
    if (togle == 'f'){
        fprintf(cp, "\nadrp->compfinst[yer][ %d][1] = %d;\n", ksay, kk);
        fprintf(cp, "\nadrp->compfinst[yer][ %d][2] = %d;\n", ksay, ll);
        fprintf(cp, "\nadrp->compfinst[yer][ %d][0] = %d;\n", ksay, tmpscnt);
        fprintf(cp, "\nadrp->compfop[yer][ %d] = ' + ';\n", ksay);
        fprintf(cp, "\nadrp->compficrk[yer][ %d] = %d %02d;\n", ksay, kk, ll);
    }
    push(tmpscnt, &stp);
    tmpscnt += 1;
    ksay += 1;
}

```

| ifade TEKSOP

carpifadesi

```

{
    pop(&kk, &ll, &stp);
    if (togle == 'c'){
        fprintf(cp, "\nadrp->compinst[yer][ %d][1] = %d;\n", ksay, ll);
        fprintf(cp, "\nadrp->compinst[yer][ %d][2] = %d;\n", ksay, kk);
        fprintf(cp, "\nadrp->compinst[yer][ %d][0] = %d;\n", ksay, tmpscnt);
        fprintf(cp, "\nadrp->compicrk[yer][ %d] = %d %02d;\n", ksay, ll, kk);
        fprintf(cp, "\nadrp->compop[yer][ %d] = ' - ';\n", ksay);
    }
    if (togle == 'f'){
        fprintf(cp, "\nadrp->compfinst[yer][ %d][1] = %d;\n", ksay, ll);
        fprintf(cp, "\nadrp->compfinst[yer][ %d][2] = %d;\n", ksay, kk);
        fprintf(cp, "\nadrp->compfinst[yer][ %d][0] = %d;\n", ksay, tmpscnt);
        fprintf(cp, "\nadrp->compfop[yer][ %d] = ' - ';\n", ksay);
        fprintf(cp, "\nadrp->compficrk[yer][ %d] = %d %02d;\n", ksay, ll, kk);
    }
}

```

```

push(tmpscnt,&stp);
tmpscnt += 1;
ksay += 1;
}
| carpifadesi {};

```

carpifadesi : carpifadesi TCARPOP

```

ilk
{
  pop(&kk,&ll,&stp);
  if (togle == 'c'){
    fprintf(cp, "\nadrp->compinst[yer][ %d][1] = %d;\n",ksay,kk);
    fprintf(cp, "\nadrp->compinst[yer][ %d][2] = %d;\n",ksay,ll);
    fprintf(cp, "\nadrp->compinst[yer][ %d][0] = %d;\n",ksay,tmpscnt);
    fprintf(cp, "\nadrp->compicrk[yer][ %d] = %d %02d;\n",ksay,kk,ll);
    fprintf(cp, "\nadrp->compop[yer][ %d] = '*';\n",ksay);
  }
  if (togle == 'f'){
    fprintf(cp, "\nadrp->compfinst[yer][ %d][1] = %d;\n",ksay,kk);
    fprintf(cp, "\nadrp->compfinst[yer][ %d][2] = %d;\n",ksay,ll);
    fprintf(cp, "\nadrp->compfinst[yer][ %d][0] = %d;\n",ksay,tmpscnt);
    fprintf(cp, "\nadrp->compfop[yer][ %d] = '*';\n",ksay);
    fprintf(cp, "\nadrp->compficrk[yer][ %d] = %d %02d;\n",ksay,kk,ll);
  }
  push(tmpscnt,&stp);
  tmpscnt += 1;
  ksay += 1;
}
/* stack'den son iki entry'yi pop ederek mikro komut yarat, bu sonuç
ara bir değişkeni gerektirdiği için tmpscnt arttırarak bir geçici
değişkeni atama yapılacak hedef değişken olarak belirle , komut sayısını
bir arttır kk=pos2 ll=pos4 olarak compute blok tarafından kullanılır
kkl birliktte komut olarak komut dizisine eklenirler */
}

```

| carpifadesi TBOLOP

```

ilk
{
  pop(&kk,&ll, &stp);
  if (togle == 'c'){
    fprintf(cp, "\nadrp->compinst[yer][ %d][1] = %d;\n",ksay,ll);
    fprintf(cp, "\nadrp->compinst[yer][ %d][2] = %d;\n",ksay,kk);
    fprintf(cp, "\nadrp->compinst[yer][ %d][0] = %d;\n",ksay,tmpscnt);
    fprintf(cp, "\nadrp->compicrk[yer][ %d] = %d %02d;\n",ksay,ll,kk);
    fprintf(cp, "\nadrp->compop[yer][ %d] = '/';\n",ksay);
  }
  if (togle == 'f'){
    fprintf(cp, "\nadrp->compfinst[yer][ %d][1] = %d;\n",ksay,ll);
    fprintf(cp, "\nadrp->compfinst[yer][ %d][2] = %d;\n",ksay,kk);
    fprintf(cp, "\nadrp->compfinst[yer][ %d][0] = %d;\n",ksay,tmpscnt);
    fprintf(cp, "\nadrp->compfop[yer][ %d] = '/';\n",ksay);
    fprintf(cp, "\nadrp->compficrk[yer][ %d] = %d %02d;\n",ksay,ll,kk);
  }
}

```



```

push(tmpscnt,&stp);
tmpscnt += 1;
ksay += 1;
}
| ilk { }
;

ilk : TLPARA ifade TRPARA
| TEKSOP ilk
{
    pop1(&kk,&stp);
    ll=99;
    if (togle == 'c'){
        fprintf(cp, "\nadrp->compinst[yer][ %d][1] = %d;\n",ksay,kk);
        fprintf(cp, "\nadrp->compinst[yer][ %d][2] = %d;\n",ksay,ll);
        fprintf(cp, "\nadrp->compinst[yer][ %d][0] = %d;\n",ksay,tmpscnt);
        fprintf(cp, "\nadrp->compicrk[yer][ %d] = %d %02d;\n",ksay,kk,ll);
        fprintf(cp, "\nadrp->compop[yer][ %d] = '!';\n",ksay);
    }
    if (togle == 'f'){
        fprintf(cp, "\nadrp->compfinst[yer][ %d][1] = %d;\n",ksay,kk);
        fprintf(cp, "\nadrp->compfinst[yer][ %d][2] = %d;\n",ksay,ll);
        fprintf(cp, "\nadrp->compfinst[yer][ %d][0] = %d;\n",ksay,tmpscnt);
        fprintf(cp, "\nadrp->compficrk[yer][ %d] = %d %02d;\n",ksay,kk,ll);
        fprintf(cp, "\nadrp->compfop[yer][ %d] = '!';\n",ksay);
    }
    if (togle == 't'){
        fprintf(cp, "\nadrp->testinst[yer][ %d][1] = %d;\n",ksay,kk);
        fprintf(cp, "\nadrp->testinst[yer][ %d][2] = %d;\n",ksay,ll);
        fprintf(cp, "\nadrp->testinst[yer][ %d][0] = %d;\n",ksay,tmpscnt);
        fprintf(cp, "\nadrp->testicrk[yer][ %d] = %d %02d;\n",ksay,kk,ll);
        fprintf(cp, "\nadrp->testop[yer][ %d] = '!';\n",ksay);
    }
    push(tmpscnt,&stp);
    tmpscnt += 1;
    ksay += 1;
}

```

/* Monadic - operasyonu, stack'de en son kalan degisken numarasını pop edip operator olarak ! kullanarak *(-1) operasyonunun bu deęişkene uygulanmasını sağlayacak bir pseudo komutu araya sıkıştırır */

```
| DEGISMI {printf("%s",vrgond);
```

```

if( (np=bul(vrgond,200+bloknm))==NULL) {
    printf("\n Blokno = %d Hata :
    Degisken %s sembol tablosunda yok \n",bloknm,vrgond);
    exit(-1);
}
/* ifade içinde kullanılan deęişkenlerin
daha önceden sembol tablosunda yer almış olması
gerekir - deklarasyon

```

```

- ya ilişkilendirme ile
- ya da bir atamanın sol tarafında yer almasıyla
olur */
push(np->sira,&stp);
SONVARTIP=np->tip;
}
| SAHAISMI {printf("%s",atgond);

if( (np=bul(atgond,200+bloknm))==NULL) {
    printf("\n Blokno= %d Hata :
    Degisken %s sembol tablosunda yok \n",bloknm,atgond);
    exit(-1);
}
/* ifade icinde kullanılan degiskenlerin
daha onceden sembol tablosunda yer almış olması
gerekir - deklarasyon
- ya ilişkilendirme ile
- ya da bir atamanın sol tarafında yer almasıyla
olur */
push(np->sira,&stp);
SONVARTIP=np->tip;
}
| posgercel
{printf("Gercel (gercel) SABIT\n");
fclose(rp);
rp=fopen("gecici2", "r");
ll=99;
fscanf(rp, "%f",&cc);
kk=tmpscmt;
if (toggle=='c'){
    fprintf(cp,"\naddrp->compsbt[yer][ %d]= %f;\n",tmpscmt-32,cc);
    fprintf(cp,"\naddrp->compinst[yer][ %d][1]= %d;\n",ksay,tmpscmt);
    tmpscmt+=1;
    fprintf(cp,"\naddrp->compinst[yer][ %d][2]= %d;\n",ksay,ll);
    fprintf(cp,"\naddrp->compinst[yer][ %d][0]= %d;\n",ksay,tmpscmt);
    fprintf(cp,"\naddrp->compicrk[yer][ %d]= %d %02d;\n",ksay,kk,ll);
    fprintf(cp,"\naddrp->compop[yer][ %d]='$';\n",ksay);
}
if (toggle=='f'){
    fprintf(cp,"\naddrp->compsbt[yer][ %d]= %f;\n",tmpscmt-32,cc);
    fprintf(cp,"\naddrp->compfinst[yer][ %d][1]= %d;\n",ksay,tmpscmt);
    tmpscmt+=1;
    fprintf(cp,"\naddrp->compfinst[yer][ %d][2]= %d;\n",ksay,ll);
    fprintf(cp,"\naddrp->compfinst[yer][ %d][0]= %d;\n",ksay,tmpscmt);
    fprintf(cp,"\naddrp->compficrk[yer][ %d]= %d %02d;\n",ksay,kk,ll);
    fprintf(cp,"\naddrp->compfop[yer][ %d]='$';\n",ksay);
}
if (toggle=='t'){
    fprintf(cp,"\naddrp->testsbt[yer][ %d]= %f;\n",tmpscmt-32,cc);
    fprintf(cp,"\naddrp->testinst[yer][ %d][1]= %d;\n",ksay,tmpscmt);

```



```

    tmpscnt += 1;
    fprintf(cp, "\nadrp->testinst[yer][ %d][2] = %d;\n", ksay, ll);
    fprintf(cp, "\nadrp->testinst[yer][ %d][0] = %d;\n", ksay, tmpscnt);
    fprintf(cp, "\nadrp->testicrk[yer][ %d] = %d %02d;\n", ksay, kk, ll);
    fprintf(cp, "\nadrp->testop[yer][ %d] = '$';\n", ksay);
}
    push(tmpscnt, &stp);
    tmpscnt += 1;
    ksay += 1;
}
;
sinkbaslik : TSINK {
    blokmp[bloknm][0] = bloknm-1;
    blokmp[bloknm][1] = ikuyrnm;
}
    | geneletiket
    TSINK
;
sinkblok : sinkbaslik{
    bloktip[bloknm] = 12;
    fprintf(cp, "#include </usr/ccs/bin/sinkinc.c>\n");
    fprintf(cp, "\nadrp->sinktl[yer] = %d;\n", kuyuz[ikuyrnm]);
}
    TNOKVIRG
;

sortbaslik : TSORT {
    blokmp[bloknm][0] = bloknm-1;
    blokmp[bloknm][1] = ikuyrnm;
}
    | geneletiket
    TSORT
;

sortblok : sortbaslik {
    blokmp[bloknm][2] = bloknm + 1;
    blokmp[bloknm][3] = okuyrnm;
    bloktip[bloknm] = 10;
    fprintf(cp, "#include </usr/ccs/bin/sortinc.c>\n");
    togle = 's';
}
iliskil2

TNOKVIRG
{
    fprintf(cp, "\nadrp->sortsay[yer] = %d;\n", girdegsey);
    if( girdegsey != kuyuz[ikuyrnm]){
        printf("\n Blokno = %d Ikaz:
        Sort blok transaction uzunluk uyumsuzligi\n", bloknm);
    }
}

```

```

fprintf(cp, "\nadrp->sorttl[yer] = %d;\n", kuyuz[ikuyrnm]);
/* giris kuyruk contextini aynen cikisa yansit */
};

countbaslik : TCOUNTB {
    blokmp[bloknm][0] = bloknm-1;
    blokmp[bloknm][1] = ikuyrnm;
}
| geneletiket
TCOUNTB
;

countblok : countbaslik{
    girdegssay = 1;
    blokmp[bloknm][2] = bloknm + 1;
    blokmp[bloknm][3] = okuyrnm;
    fprintf(cp, "#include </usr/ccs/bin/countinc.c>\n");
    togle = 'c';
}
iliskil2

TNOKVIRG
{
    fprintf(cp, "\nadrp->cntsay[yer] = %d;\n", girdegssay);
    sprintf(vrgond, "_Cnt");
    install(vrgond, 'n', kuyuz[ikuyrnm] + 1, ikuyrnm, 0);

    /* _Cnt deęişkeni girişten aynen dışarı yansıtılan
    transaction'ın bitiminden bir sonraki göze yerleşir */

    fprintf(cp, "\nadrp->cntloc[yer] = %d;\n", kuyuz[ikuyrnm] + 1);
    if (girdegssay > kuyuz[ikuyrnm]){
        printf("\n Blokno = %d Ikaz :
        Count blok transaction uzunluk uyusmazligi \n", bloknm);
    }
    fprintf(cp, "\nadrp->cnttl[yer] = %d;\n", kuyuz[ikuyrnm]);
    kuyuz[ikuyrnm] + = 1;
    /* count deę eklenmesi ile bu noktadan
    itibaren kuyruk uzunluęu bir artar bu kuyruk sadece
    count bloęuna gelen kuyruk olduęundan sorun çıkmaz */

};

txcbaslik : TTXSAY {
    blokmp[bloknm][0] = bloknm-1;
    blokmp[bloknm][1] = ikuyrnm;
}
| geneletiket
TTXSAY
;

txcblok : txcbaslik{

```



```

        girdegsey = 1;
        blokmp[bloknm][2] = bloknm + 1;
        blokmp[bloknm][3] = okuyrnm;
        fprintf(cp, "#include </usr/ccs/bin/txsinc.c>\n");
    }
    TNOKVIRG
    {
        sprintf(vrgond, "_Cnt");
        install(vrgond, 'n', 1, okuyrnm, 0);
        fprintf(cp, "\nadrp->txstl[yer] = %d;\n", kuyuz[ikuyrnm]);
        sontl = 1;
        ikuyrnm = okuyrnm;
        kuyuz[ikuyrnm] = sontl;
        okuyrnm += 1;
    };

filtbaslik : TFILTER {
    blokmp[bloknm][0] = bloknm - 1;
    blokmp[bloknm][1] = ikuyrnm;
}
| geneletiket
TFILTER
;

filtblok : filtbaslik {
    girdegsey = 1;
    blokmp[bloknm][2] = bloknm + 1;
    blokmp[bloknm][3] = okuyrnm;
    bloktip[bloknm] = 15;
    fprintf(cp, "#include </usr/ccs/bin/ftinc.c>\n");
    togle = 'f';
}

iliskil2
TNOKVIRG
{
    if (girdegsey != kuyuz[ikuyrnm]){
        printf("\n Blokno = %d Ikaz : Update blok transaction uzunluk uyusmazligi\n", bloknm);
    }
    fprintf(cp, "\nadrp->fltsay[yer] = %d;\n", girdegsey);
    fprintf(cp, "\nadrp->flttxl[yer] = %d;\n", kuyuz[ikuyrnm]);
};

iliskil2 : ilis2 {girdegsey = 1;} | iliskil2 TVIRG {girdegsey += 1;} ilis2 ;

ilis2 : SAHAISMI
{
    if( (np=bul(atgond, ikuyrnm)) == NULL) {
        printf("\n Blokno = %d Hata :
            Degisken %s sembol tablosunda yok \n", bloknm, atgond);
        exit(-1);
    }
    install(atgond, np->tip, np->sira, okuyrnm, 0);
}
/*sort,count ve filter bloklarında giriş değişken/sahaları olduğu

```

gibi çıkış kuyruk contextinde de gözükürler, count bloğunda en son değişken olarak sistem tanımlı değişken _CNT çıkış kuyruk contextine dahil edilir */

```

    if( togle == 's' ){
        fprintf(cp, "\naddrp->sorttxix[yer][ %d] = %d;\n",
            girdegsay, np->sira);
        if (np->tip == 'n') {
            fprintf(cp, "\naddrp->sorttyp[yer][ %d] = 1;\n", girdegsay);
        } else {
            fprintf(cp, "\naddrp->sorttyp[yer][ %d] = 0;\n", girdegsay);
        }
    }
    if( togle == 'c' ){ /* count blok transaction index */
        fprintf(cp, "\naddrp->cnttxix[yer][ %d] = %d;\n",
            girdegsay, np->sira);
    }
    if( togle == 'f' ){ /* filter blok transaction index */
        fprintf(cp, "\naddrp->flttixix[yer][ %d] = %d;\n",
            girdegsay, np->sira);
    }
    if( togle == 'u' ){ /* apply cumulative ON yapısı */
        fprintf(cp, "\naddrp->cumfld[yer] = %d;\n", np->sira);
    }
    if( togle == 'm' ){ /* apply cumulative BY yapısı */
        fprintf(cp, "\naddrp->cumtxix[yer][ %d] = %d;\n",
            girdegsay, np->sira);
    }
}

/* Giriş kuyruğu context'inde bu saha varmı bak */
| DEGISMI
{
    if( (np=bul(vrgond, ikuyrnm)) == NULL) {
        printf("\n Blokno = %d Hata :
            Degisken %s sembol tablosunda yok \n", bloknm, vrgond);
        exit(-1);
    }
    install(vrgond, np->tip, np->sira, okuyrnm, 0);
}

/*sort,count ve filter bloklarında giriş değişken/sahaları olduğu
gibi çıkış kuyruk contextinde de gözükürler, count bloğunda en son
değişken olarak sistem tanımlı değişken _CNT çıkış kuyruk context'ine
dahil edilir */
    if( togle == 's' ){
        fprintf(cp, "\naddrp->sorttxix[yer][ %d] = %d;\n",
            girdegsay, np->sira);
        if (np->tip == 'n') {
            fprintf(cp, "\naddrp->sorttyp[yer][ %d] = 1;\n",
                girdegsay);
        } else {
            fprintf(cp, "\naddrp->sorttyp[yer][ %d] = 0;\n", girdegsay);
        }
    }
}

```



```

if( togle == 'c' ){
    fprintf(cp, "\nadrp->cnttxix[yer][ %d] = %d;\n",
            girdegsay, np->sira);
}
if( togle == 'f' ){
    fprintf(cp, "\nadrp->flttixix[yer][ %d] = %d;\n",
            girdegsay, np->sira);
}
if( togle == 'u' ){ /* apply cumulative ON yapısı */
    fprintf(cp, "\nadrp->cumfld[yer] = %d;\n", np->sira);
}
if( togle == 'm' ){ /* apply cumulative BY yapısı */
    fprintf(cp, "\nadrp->cumtxix[yer][ %d] = %d;\n",
            girdegsay, np->sira);
}
}

;

transbaslik : TTRANSF {
    blokmp[bloknm][0] = bloknm-1;
    blokmp[bloknm][1] = ikuyrnm;
}
| geneletiket
TTRANSF
;

transblok : transbaslik{
    bloktp[bloknm] = 9;
    cikbloki = 2;
    fprintf(cp, "#include </usr/ccs/bin/transinc.c>\n");
    fprintf(cp, "\n adrp->splttl[yer] = %d;\n", kuyuz[ikuyrnm]);
    togle = 'r';
}
etiketlist
TNOKVIRG
{
};

condblok : TCONDBLOK {
    blokmp[bloknm][2] = bloknm + 1;
    blokmp[bloknm][3] = okuyrnm;
    bloktp[bloknm] = 18;
    cikbloki = 2;
    girbloki = 4;
    fprintf(cp, "#include </usr/ccs/bin/condinc.c>\n");
    togle = 'n';
}
TETIKET
{
    if ((np = bul(lbgond, -1)) == NULL){
        printf("\n Blokno = %d Hata :

```

```

    etiket bulunamadi hata \n",bloknm);
    exit(-1);
} else{
    blokmp[bloknm][girbloki]=np->sira*(-1);
    cikbloki=2;
    /* transfer bloğunun etiket filler'lar ile doldurduğu
    (o an belli olmayan) çıkış blok gözlerini incele
    ve bu gözlerden ilk rastladığın filler ile dolu
    olana kendi blok numaranı doldur */
    while(blokmp[np->sira][cikbloki]!=np->lbtan&&
        blokmp[np->sira][cikbloki]!=0) cikbloki+=2;
    if(blokmp[np->sira][cikbloki]!=np->lbtan){
        printf("\n Blokno=%d Hata :
        Büyük derleme hatası etiket uyusmuyor \n",bloknm);
        exit(-1);
    }
    blokmp[np->sira][cikbloki]=bloknm;
    blokmp[bloknm][girbloki+1]=blokmp[np->sira][cikbloki+1];
    refkuyrnm=blokmp[np->sira][cikbloki+1];

    girbloki+=2;
}
}
TETIKET
{
    if ((np=bul(lbgond,-1))==NULL){
        printf("\n Blokno=%d Hata :
        etiket bulunamadi \n",bloknm);
        exit(-1);
    } else{
        blokmp[bloknm][girbloki]=np->sira*(-1);
        cikbloki=2;
        /* transfer bloğunun etiket filler'lar ile doldurduğu
        (o an belli olmayan) çıkış blok gözlerini incele
        ve bu gözlerden ilk rastladığın filler ile dolu
        olana kendi blok numaranı doldur */
        while(blokmp[np->sira][cikbloki]!=np->lbtan&&
            blokmp[np->sira][cikbloki]!=0) cikbloki+=2;
        if(blokmp[np->sira][cikbloki]!=np->lbtan){
            printf("\n Blokno=%d Hata :
            Buyuk derleme hatasi etiket uyusmuyor \n",bloknm);
            exit(-1);
        }
        blokmp[np->sira][cikbloki]=bloknm;
        blokmp[bloknm][girbloki+1]=blokmp[np->sira][cikbloki+1];
        ikuyrnm=blokmp[np->sira][cikbloki+1];
    }
}

```



```

TNOKVIRG
{
};
sartifade : TREF
{
    kuyrnm=refkuyrnm;
    fprintf(cp, "\n adrp-> cndrefl[yer] = %d;\n", kuyuz[refkuyrnm]);
    togle='1';
}
degsaha2
sart
TINP
{ kuyrnm=ikuyrnm;
  fprintf(cp, "\n adrp-> cndinptl[yer] = %d;\n", kuyuz[ikuyrnm]);
  togle='2';}
degsaha2
;
degsaha2 : DEGISMI {
    if( (np=bul(vrgond, kuyrnm))==NULL) {
        printf("\n Blokno= %d Hata :
        Degisken %s sembol tablosunda yok \n", bloknm, vrgond);
        exit(-1);
    }
    if (togle=='1'){
        fprintf(cp, "\n adrp-> cndrefl[yer] = %d;\n", np-> sira);
        if (np-> tip=='n')
            fprintf(cp, "\n adrp-> cndreft[yer] = 1;\n");
        else
            fprintf(cp, "\n adrp-> cndreft[yer] = 0;\n");
    }else{
        fprintf(cp, "\n adrp-> cndinpl[yer] = %d;\n", np-> sira);
        if (np-> tip=='n')
            fprintf(cp, "\n adrp-> cndinpt[yer] = 1;\n");
        else
            fprintf(cp, "\n adrp-> cndinpt[yer] = 0;\n");
    }
}
| SAHAISMI {
    if( (np=bul(atgond, kuyrnm))==NULL) {
        printf("\n Blokno= %d Hata :
        Saha sembol tablosunda yok \n", bloknm);
        exit(-1);
    }
    if (togle=='1'){
        fprintf(cp, "\n adrp-> cndrefl[yer] = %d;\n", np-> sira);
        if (np-> tip=='n')
            fprintf(cp, "\n adrp-> cndreft[yer] = 1;\n");
        else
            fprintf(cp, "\n adrp-> cndreft[yer] = 0;\n");
    }else{

```

```

        fprintf(cp, "\n adrp->cndinpl[yer] = %d;\n", np->sira);
        if (np->tip == 'n')
            fprintf(cp, "\n adrp->cndinpt[yer] = 1;\n");
        else
            fprintf(cp, "\n adrp->cndinpt[yer] = 0;\n");
    }
;

sart : TBUYUK {
    fprintf(cp, "\n adrp->cndsart[yer] = '>';\n");
}
| TKUCUK {
    fprintf(cp, "\n adrp->cndsart[yer] = '<';\n");
}
| TBUYUKES {
    fprintf(cp, "\n adrp->cndsart[yer] = 'b';\n");
}
| TKUCUKES {
    fprintf(cp, "\n adrp->cndsart[yer] = 'k';\n");
}
| TESDEGIL {
    fprintf(cp, "\n adrp->cndsart[yer] = 'n';\n");
}
| TESOP TESOP {
    fprintf(cp, "\n adrp->cndsart[yer] = 'e';\n");
}
;

mergeblok : TMERGE {
    blokmp[bloknm][2] = bloknm + 1;
    blokmp[bloknm][3] = okuyrnm;
    bloktip[bloknm] = 16;
    cikbloki = 2;
    girbloki = 4;
    fprintf(cp, "#include </usr/ccs/bin/mergeinc.c>\n");
    fprintf(cp, "\n adrp->mergetl[yer] = %d;\n", kuyuz[ikuyrnm]);
    togle = 'm';
}
etiketlist
TNOKVIRG
{
};

etiketlist : TETIKET
{ if (togle == 't'){
    install(lbgond, 'n', bloknm, -1, lblfills);
    blokmp[bloknm][cikbloki] = lblfills ;
    blokmp[bloknm][cikbloki + 1] = ikuyrnm;
    cikbloki += 2;
    lblfills = 1;
}
}

```



```

if (toggle == 'm'){
    if ((np=bul(lbgond,-1)) == NULL){
        printf("\n Blokno = %d Hata : \n", blokno);
        etiket bulunamadi \n", blokno);
        exit(-1);
    }else{
        blokmp[blokno][girkloki] = np->sira*(-1);
        cikbloki = 2;

        /* transfer bloğunun etiket filler'lar ile doldurduğu
        (o an belli olmayan) çıkış blok gözlerini incele
        ve bu gözlerden ilk rastladığın filler ile dolu
        olana kendi blok numaranı doldur */

        while(blokmp[np->sira][cikbloki] != np->lbtan &&
            blokmp[np->sira][cikbloki] != 0) cikbloki += 2;
        if (blokmp[np->sira][cikbloki] != np->lbtan){
            printf("\n Blokno = %d Hata : \n", blokno);
            Buyuk derleme hatasi etiket uyusmuyor \n", blokno);
            exit(-1);
        }
        blokmp[np->sira][cikbloki] = blokno;
        blokmp[blokno][girkloki + 1] = blokmp[np->sira][cikbloki + 1];
        if (girkloki == 4)
            ikuyrnm = blokmp[np->sira][cikbloki + 1];

        /* ilk karşılaşılın etiket ile oluşan kuyruk context
        tüm diğer kuyruklar içinde aynı kabul edilir ve
        Bu değişken isimleri aynı sıra ile diğer kuyruklardan
        da geliyormuş kabul edilir */
        girkloki += 2;
    }
}

| etiketlist TVIRG TETIKET
{ if (toggle == 't'){
    install(lbgond, 'n', blokno, -1, lblfills);
    blokmp[blokno][cikbloki] = lblfills ;
    blokmp[blokno][cikbloki + 1] = ikuyrnm;
    cikbloki += 2;
    lblfills = 1;
}
}

if (toggle == 'm'){
    if ((np=bul(lbgond,-1)) == NULL){
        printf("\n Blokno = %d Hata : \n", blokno);
        etiket bulunamadi hata \n", blokno);
        exit(-1);
    }else{
        blokmp[blokno][girkloki] = np->sira*(-1);
        cikbloki = 2;
    }
}

```



```

{
    blokmp[bloknm][0]=bloknm-1;
    blokmp[bloknm][1]=ikuyrnm;
}
| geneletiket
TDEL
;
delblok : delbaslik {
    girsahsay=0;
    blokmp[bloknm][2]=bloknm + 1;
    blokmp[bloknm][3]=okuyrnm;
    bloktip[bloknm]=14;
    fprintf(cp,"#include </usr/ccs/bin/delinc.c>\n");
    togle='d';
}
aramagrubu
TNOKVIRG
{
    if (girsahsay!=kuyuz[ikuyrnm]){
        printf("\n Blokno = %d Ikaz :
            Delete blok transaction uzunluk uyusmazligi\n",bloknm);
    }
    fprintf(cp,"\\nadrp->deltl[yer] = %d;\n",kuyuz[ikuyrnm]);
    fprintf(cp,"\\nadrp->delmat[yer] = %d;\n",girsahsay);
};
%%
/* DERLEYICININ KULLANDIGI DEGİSKENLER */
int prevyyl1,prevyyl2,prevyyl3,prevyyl4,prevyyl5,i,j,l;
char rngond[200],rngond1[200],vrgond[200],vrgond1[200],atgond[200],atgond1[200];
char stgond1[200],stgond[200],lbgond1[200],lbgond[200];
char *formt = "%s";
int bloknm,ikuyrnm,okuyrnm,cikbloki=2,girbloki=4,cik,lblfills=-700000000,sontl;
int ilktl,refkuyrnm,kuyrnm,sonbno;

/* apply icin eklenenler */
int cas,gas;
/* bloknm siradan blok numaralarini takip eden sayac
    ikuyrnm,okuyrnm   kuyruk context numaralarının takibi için
                    kullanılan değişkenler
    cikbloki         transfer bloğu birden fazla split yapıyorsa
                    bu sayaç ile aynı blokmap tablo satırındaki
                    birden fazla blok referansı takip edilir
    cik              loop'tan çıkma mantıksal değişkeni
    lblfills         etiket filler sayacı transfer geçici olarak
                    çıkış yaptığı blok numaralarının gerçekleri
                    belli olana kadar bu sayaçdan aldığı değerler
                    ile blokmap tablosuna yer doldurur.
    sontl           en son blokun çıkış transaction uzunluğu
                    bu değeri transfer blok gibi parametrelerinden
                    giriş transaction uzunluğunu bulamayan bloklar

```

```

        bu değeri bulmak için kullanır bu değer daima
        her blok tarafından update edilir */
int blokmp [200] [40]; /* satır formatı
        girişin geldiği blok#, giriş geçici kuyno,
        çıkışın gittiği blok# , çıkış geçici kuyno,
        çıkışın gittiği blok# , çıkış geçici kuyno,
        */
int kuyuz [200]; /* kuyruk transaction uzunlukları */
int bloktip [200]; /* her bir bloğun tipi
        1=Generate 2=Generatet 3=test 4=apply
        5= compute 6= computefix 7= input
        8= report 9= transfer 10= sort 11= count
        12= sink 13= update 14= delete 15= filter
        */
int shmid[6];
int semid[6];

/* bu değişkenler shared memory ve simafor id leri transfer
dosyasından okuyup cikti.c 'ye atama olarak geçirilmesinde
ara değişkenlerdir */

FILE *fp,*cp,*bp,*dp,*rp;
int k,m,n,eksitgl;
/* k ve m gerçel sayının tam ve kesirli kısımlarını okumada kullanılan
geçici değişkenlerdir. k'ya geçici dosyasından bir önce okunan
integer değeri tam sayı kısmı olarak, m ye geçici
dosyasından bir önce okunan integer değeri kesirli kısım olarak okunur.
bu değerlerden gerçel sayı oluşturulup gecici2
dosyasına yazılır eksitgl sayının eksi olup olmadığının
haberleşmesinde kullanılır */

/* COMPUTE İÇİN İLAVE EDİLEN DEĞİŞKENLER */
int tmprcnt=32,vrcnt=0,stp=0,tmprbcnt=62,srcnt=92;
int stack[100],kk,ll,ksay=0,temp,cikdegsay,girdegsay,girsahsay,locmark ;
float cc;
char togle;
/* COMPUTE FIX ICIN OZEL */
char SONVARTIP; /* ATTRIBUTE=CHAR KARAKTERDIZIN VAR tipi özel atamalar için
        son var tipini yakalayıp çek etmek ve character tipinde
        ise string sident direkt atama (&) komutu üretmek
        için kullanılan iletişim değişkeni */
#include </usr/ccs/bin/lex.yy.c>
int yyparse();

#define NULL 0

struct nlist {
    struct nlist *next;
    char *name;
    char tip; /* numerik=n char = c */
    int sıra; /* context kuyruk transaction tipine göre

```



```

transactionda kaç numaralı eleman
olarak temsil ediliyor */
int kuyno; /* bu değişkenin hangi kuyruk context'inde
geçerli olduğu */
int lbktan; /* etiketler sembol tablosuna işlenirken kullanılıyor */
};

#define hssize 500

static struct nlist *hashtab[hssize];
static struct nlist dynstack[1000];
static struct nlist *np;

pop(int *kk,int *ll,int *stp){
*kk=stack[*stp-1];
*ll=stack[*stp-2];
*stp-=2;
}
pop1(int *kk,int *stp){
*kk=stack[*stp-1];
*stp-=1;
}
push(int ps, int *stp){
stack[*stp]=ps;
*stp+=1;
}

unsigned shash(s)
char *s;
{unsigned hashval,temp;
for (hashval=0;*s!='\0';s++)
hashval=*s+31*hashval;
return hashval % hssize;
}

struct nlist *bul(s,l)
char *s;
int l;
{
struct nlist *np;

for (np=hashtab[hash(s)]; np!=NULL;np=np->next){
if (!strcmp(s,np->name)&&(np->kuyno==l))
return np;
}
return NULL;
}

struct nlist *install(name,tip1,poz,kuyno,lbktan)
char *name;
char tip1;

```

```

int poz,kuyno,lbltan;
{
    struct nlist *np,*l;
    char *f;
    unsigned hashval;
    if ((np=bul(name,kuyno))==NULL){
        np=(struct nlist *)malloc(sizeof(*np));
        f=(char *)malloc(strlen(name)+1);
        strcpy(f,name);
        np->name=f;
        if (np==NULL || (np->name==NULL) )
            return NULL;
        hashval=shash(name) ;
        if (hashtab[hashval]==NULL){
            np->next=NULL;
            hashtab[hashval]=np;
            np->tip=tip1;
            np->sira=poz;
            np->kuyno= kuyno;
            np->lbltan= lbltan;
            return np;
        }else{
            for (l=hashtab[hashval];l->next!=NULL;l=l->next);
            l->next=np;
            np->next=NULL;
            np->tip=tip1;
            np->sira=poz;
            np->kuyno=kuyno;
            np->lbltan=lbltan;
            return np;
        }
    } else {printf("IKAZ> ayni taniticinin ikinci kez deklarasyonu \n");
            np->tip=tip1;
            np->sira=poz;
            np->kuyno=kuyno;
            np->lbltan=lbltan;
            return np; }
return NULL;
}
main(){
    printf("\n Derleme basliyor \n");
    cp=fopen("cikti.c", "w");
    dp=fopen("transfer", "r");
    for (i=1;i<=6;i++)
        fscanf(dp,"%d",&shmid[i]);
    for (i=1;i<=4;i++)
        fscanf(dp,"%d",&semid[i]);
    fprintf(cp,"#include </usr/ccs/bin/cikbas.c>\n");
    fprintf(cp,"shmid= %d;\n",shmid[1]);
    fprintf(cp,"semid= %d;\n",semid[1]);
    for (i=2;i<=6;i++)

```



```

    fprintf(cp, "shmid%d = %d;\n", i-1, shmid[i]);
for (i=2; i<=4; i++)
    fprintf(cp, "semid%d = %d;\n", i-1, semid[i]);
fprintf(cp, "#include </usr/ccs/bin/cikbas1.c>\n");
fprintf(cp, "#include </home/suha/final/bloks.c>\n");
fprintf(cp, "#include </usr/ccs/bin/cikbas2.c>\n");
bloknm=1;
ikuyrnm=1;
okuyrnm=2;
i=yyparse();
fprintf(cp, "}\n");
printf("\n Sonuc = %d \n", i);
}
yywrap(){
    return(0);
}
yyerror(s)
char *s;
{
    printf("\n Soz dizimi kurali hatasi \n");
}

```

EK-4

DİL ORTAMI VE SANAL İŞLEMCİLERİN KAYNAK KODLARI

```
/* APPEND BLOĞU SANAL İŞLEMCİSİ KAYNAK PROGRAM KODU */
```

```
#include <derle.h>
```

```
main(){
RELINFO rel_buf;
RTTAB rt_buf[256];
HTTAB ht_buf[256];
CTTAB ct_buf[256];
GHTTAB *ght_buf,*ght_buf2;
GCTTAB *gct_buf,*gct_buf2;

/* bufferların deklarasyonu bitti */

char isim[50],cev[2];
char temp[2],dum[2];
char htflen[256][12],rtflen[256][12],ctflen[256][12];
char *charptr;
char tempbuf[256],relatnm[50],atrnme[50];
int tempint,fp,ft,cik,ilkparc,ikiparc;
int a,sayac,sayac2,i,k,l,j,r,dec,sign,bulunam,follow,maxeqc;
int ctp[256],rtp[256],htp[256],gecici,gecici1,gecici2,gecici3;
int eqval[256],eq[256];
long son,hashval,yenirno,yenictno,lgec,curpos,pos,sident,sidentb;
char *tempptr;
double tempflt;
struct sembuf p={0,-1,SEM_UNDO};
struct sembuf v={0,1,SEM_UNDO};
ANAYAPI *adrp;
KUYRUK *adrp2;
struct q_entry s_entry;
int queue_id,rlen;
int maxatrs,shmid,shmid1,shmid2,shmid3,semid,semid1;
long trans[100];
int pnum,mm,pno,girats,instrsay,yer;
pid_t pid;
processorid_t gelen;
idtype_t p_pid;
```

```
/* variable deklarasyonları bitti */
```

```
pno=46;
pnum=2;
```



```

pid=getpid();
p_pid=getppid();
mm=processor_bind(P_PID,pid,pnum,&gelen);

if((shmid1=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((adrp=(ANAYAPI *)shmat(shmid1,(void *)0,0))==(ANAYAPI *)(-1))
    printf ("\n attach olmadı \n");

if((shmid=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((adrp2=(KUYRUK *)shmat(shmid,(void *)0,0))==(KUYRUK *)(-1))
    printf ("\n attach kkkkolmadı \n");

/* global hash tablosu ve global çarpışma tablosunu shared
memory olarak al */

if((shmid2=shmget((key_t)0x35,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((ght_buf=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
    printf ("\n attach olmadı \n");
if((shmid3=shmget((key_t)0x45,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((gct_buf=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
    printf ("\n attach olmadı \n");
if((shmid2=shmget((key_t)0x55,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((ght_buf2=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
    printf ("\n attach olmadı \n");
if((shmid3=shmget((key_t)0x65,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((gct_buf2=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
    printf ("\n attach olmadı \n");
if ((queue_id= msgget(QKEY,IPC_CREAT|QPERM))==-1)
    perror("msg get failed");

/* çıkış kuyruğu */
while(1) { /* sonsuz döngü */
if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)
    perror("msgrcv failed");
while(adrp->procact[pno]!=1)a=a+a;

yer=adrp->tablo[pno][5];
giratr=adrp->updttl[yer];
maxatr=adrp->updtmax[yer];
strcpy(relatnm,adrp->updtreln[yer]);
fp=open("relinfo",O_CREAT|O_RDWR);
lseek(fp,0L,SEEK_END);
son=(tell(fp)-sizeof(RELINFO))/sizeof(RELINFO);
pos=0;
lseek(fp,pos*sizeof(RELINFO),SEEK_SET);

```

```

while ((pos <= son) && (!(karsilas(rel_buf.relname, relatnm) == 0))) {
    if (read(fp, &rel_buf, sizeof(RELINFO)) == -1)
        printf(" read failed in relinfo\n");
    pos += 1;
    lseek(fp, pos * sizeof(RELINFO), SEEK_SET);
}
if (pos > son && (!(karsilas(rel_buf.relname, relatnm) == 0))) {
    printf("\n calisma zamani hatasi \n");
    exit(0);
}
close(fp);
for (i = 0; i <= rel_buf.atrsayi; i++) {
    htfile[i][0] = '\0';
    ctfile[i][0] = '\0';
    rtfile[i][0] = '\0';
}

for (i = 0; i <= rel_buf.atrsayi; i++) {
    strcat(htfile[i], rel_buf.relname);
    strcat(ctfile[i], rel_buf.relname);
    strcat(rtfile[i], rel_buf.relname);
}

for (i = 0; i <= rel_buf.atrsayi; i++) {
    strcat(htfile[i], rel_buf.atrname[i]);
    strcat(htfile[i], ".h");
    strcat(ctfile[i], rel_buf.atrname[i]);
    strcat(ctfile[i], ".c");
    strcat(rtfile[i], rel_buf.atrname[i]);
    strcat(rtfile[i], ".r");
}

for (i = 0; i <= rel_buf.atrsayi; i++) {
    rtp[i] = open(rtfile[i], O_CREAT | O_RDWR);
    htp[i] = open(htfile[i], O_CREAT | O_RDWR);
    ctp[i] = open(ctfile[i], O_CREAT | O_RDWR);
}

transoku(giratr, adrp2, trans, semid1, &p, &v, adrp->tablo[pno][2]);
while (trans[1] != -1) {
    k = 0;
    for (j = 0; j < maxatrs; j++) {
        strcpy(atrname, adrp->updtent[yer][j].atrname);
        cik = 0;
        for (i = 0; i <= rel_buf.atrsayi; i++) {
            if (karsilas(rel_buf.atrname[i], atrname) == 0) {
                equal[i] = adrp->updtent[yer][j].ind;
                cik = 1;
            } /* if */
        } /* for i */
        if (!cik) {
            printf("\n Çalışma zamanı hatası %s sahası bulunamadı \n", atrname);
            exit(-1);
        }
    }
}

```



```

} /* for j */

temp[0]='\0';
sidentb=altyapi(ght_buf,gct_buf,temp);
/* boş sahalar için null character token'ı tesis edilir */

for (i=0;i<=rel_buf.atrsayi;i++){
    if (eqval[i]<0) sident=trans[eqval[i]*(-1)];
    else sident=eqval[i];
    if (sident<0) { /* bu scratch hash tablosunda mevcut ama
                    global hash tablosunda mevcut değil */
        sident=sident*(-1);
        strcpy(tempbuf,&gct_buf2->ct[sident+4]);
        sident=altyapi(ght_buf,gct_buf,tempbuf);
    }
    if (sident==0) sident=sidentb;
} /* eğer sident sıfır ise bunun null char token'ına esit olması sağlanır */

paket(temp,sident);

lseek(rtp[i],0L,SEEK_END);
write(rtp[i],&rt_buf[i],sizeof(RTTAB)); /* Bir adet boş record yaz*/
/* her durumda bir adet tuple ekleniyor bu nedenle
bir adet boş kaydı rt tablosunun sonuna ekle */

yenirno=tell(rtp[i])-sizeof(RTTAB); /*neredesin bak */
yenirno=yenirno/sizeof(RTTAB);
hashval=hash(temp); /* sident ile hash yap */

/* sident ile hash yap ve bu attribute a ait hash tablosunda
bu entry yi bul */

lseek(htp[i],hashval*sizeof(HTTAB),SEEK_SET); /* ht file'da bul */
if (read(htp[i], &ht_buf[i], sizeof(HTTAB)) == -1)
    printf("problem reading HT\n");

if (ht_buf[i].ht2!=0){ /*eğer collision bucket tesis edilmiş ise
                        yani aynı zamanda bu entry boş ise */
    lgec=ht_buf[i].ht2 ;
    lseek(ctp[i], lgec*sizeof(CTTAB), SEEK_SET);
    /* collision tablosuna geç */
    if (read(ctp[i], &ct_buf[i], sizeof(CTTAB)) == -1)
        printf("problem CT\n");
    while (ct_buf[i].ct2!=0&&ct_buf[i].ct1!=sident) {
        lgec=ct_buf[i].ct2;
        lseek(ctp[i],lgec*sizeof(CTTAB),SEEK_SET);
        if (read(ctp[i], &ct_buf[i], sizeof(CTTAB)) == -1)
            printf("problem CT\n");
    } /* collision buketi içinde bulana kadar veya sonuna
        ulaşıp bulamayana kadar ilerledi */
}

```

```

if (ct_buf[i].ct1 == sident){ /* bucket icinde buldu */
    hashval = lgec; /* ct'da bulunduğun yeri markala */
    curpos = tell(ctp[i]);
    curpos = (curpos - sizeof(CTTAB))/sizeof(CTTAB);
    lgec = ct_buf[i].ct4;
    lseek(rtp[i], lgec * sizeof(RTTAB), SEEK_SET);
    /* rt tablosunda "same linkte" en son düğüme ulaş */
    if (read(rtp[i], &rt_buf[i], sizeof(RTTAB)) == -1)
        printf("problem RT\n");
    /* "same linkte" en son düğüme eriş */
    ayris(rt_buf[i].rt, &gecici, &gecici2, &gecici3);
    /* ayırıştırıp bak */
    if (!(gecici == 999 | gecici == hashval)){
        printf("\n alarm sistem entegrasyonu bozulmuş\n");
    }
    /* eğer burada bulunan değer bizim hareket
    ettiğimiz ct girdisini göstermiyorsa, uymuyorsa
    uyumsuzluk var uyar */
    gecici2 = yenirno; /* long tipini tamsayıya çevir */
    rt_buf[i].rt = birles(gecici, gecici2, gecici3);
    /* "same linkte" bir önceki düğüme pointer günlemesi */
    lseek(rtp[i], lgec * sizeof(RTTAB), SEEK_SET);
    if (write(rtp[i], &rt_buf[i], sizeof(RTTAB)) == -1)
        printf("problem RT\n");
}

/* eklenen rt kaydının günlemesi */
gecici3 = lgec;
if (gecici == 999) /* bu bağda (linkte) ulaşılan kayıt iptal
edilmiş vaziyette duruyor */
    gecici = -hashval;
rt_buf[i].rt = birles(gecici, 0, gecici3);
lseek(rtp[i], yenirno * sizeof(RTTAB), SEEK_SET);

if (write(rtp[i], &rt_buf[i], sizeof(RTTAB)) == -1)
    printf("problem RT\n");
ct_buf[i].ct4 = yenirno;
/* ct girdisinde, "same link" in sonunu gösteren
bağı günle */
lseek(ctp[i], curpos * sizeof(CTTAB), SEEK_SET);
if (write(ctp[i], &ct_buf[i], sizeof(CTTAB)) == -1)
    printf("problem RT\n");

```

/* yukarıdaki adımların algoritması:

ct_buf[i].ct4, u takiben rt ye git
 same link ini takip et, bu linkte en sondan bir evelki kayda
 geldiğinde sonlandırılmış olan ileri(forward) pointer'i
 yeni kaydı gösterecek şekilde günle
 yeni kaydın ileri pointer'ını sonlandır

geri (backward) pointer'ını bir önceki kayıdı gösterecek şekilde güncle */

```

}
else { /* ct_buf[i].ct1!=sident daha önce hiç girilmemiş
      */

```

```

    curpos=lgec ;
    lseek(ctp[i],0L,SEEK_END);
    /* en sona gidildi */

```

```

    ilkparc=tell(ctp[i]);
    ikiparc=sizeof(CTTAB);
    yenictno=((ilkparc-ikiparc)/ikiparc)+1;
    ct_buf[i].ct2=yenictno;
    lseek(ctp[i],curpos*sizeof(CTTAB),SEEK_SET);
    if (write(ctp[i], &ct_buf[i], sizeof(CTTAB)) == -1)
        printf("problem cT\n");

```

```

    ct_buf[i].ct1=sident; /* integer id yi bu girdi ye koy */
    ct_buf[i].ct2=0; /* collision bağı sonlandır */
    ct_buf[i].ct3=yenirno; /* rt tabloya pointer'ı yerleştir */
    ct_buf[i].ct4=yenirno;
    gecici=-yenictno;
    rt_buf[i].rt=birles(gecici,0,0);
    lseek(rtp[i],yenirno*sizeof(RTTAB),SEEK_SET);
    if (write(rtp[i], &rt_buf[i], sizeof(RTTAB)) == -1)
        printf("problem RT\n");
    /* ilgili ilişki tablosu girdisi günclendi */
    lseek(ctp[i],yenictno*sizeof(CTTAB),SEEK_SET);
    if (write(ctp[i], &ct_buf[i], sizeof(CTTAB)) == -1)
        printf("problem CT\n");

```

```

/* if ct_buf[i].ct1!= sident sona geldiği halde
   collision tablosunda da bulunamadı */

```

```

/* bu collision bukete yeni bir girdi ekle
   bağla ve sident i bu yeni entry ye atayıp yukarıda
   hash entry boş ise yapılan işleri yap */

```

```

} /* end else */

```

```

}/* end if ht_buf[i].ht2!=0 */

```

```

else { /* if ht_buf[i].ht2==0 bu hash entry ile ilgili
      daha önceden girilmiş bir sey yok */
    curpos=(tell(htp[i])-sizeof(HTTAB))/sizeof(HTTAB);
    lseek(ctp[i],0L,SEEK_END);
    /* coll table da en sona git */

```

```

    ilkparc=tell(ctp[i]);
    ikiparc=sizeof(CTTAB);
    yenictno=((ilkparc-ikiparc)/ikiparc)+1;
    ht_buf[i].ht2=yenictno;
    /* hash table girdisini bu coll table girdisine linkle */
    lseek(htp[i],curpos*sizeof(HTTAB),SEEK_SET);
    if (write(htp[i], &ht_buf[i], sizeof(HTTAB)) == -1)
        printf("problem HT\n");

```

```

ct_buf[i].ct1=sident; /* sident'i bu entry ye yerleştir */
ct_buf[i].ct2=0; /* collision linki toprakla */
ct_buf[i].ct3=yenirno; /* rt tabloya pointer'i yerleştir */
ct_buf[i].ct4=yenirno;

/* ilk giriş olduğundan başlangıç ve sonuç pointer'lar
rt tablosunda aynı kayıdı gösterir coll table içinde
bu hash girdisine ait coll bağı sadece bir eleman içeriyor
dolayısıyla ct2 (coll ileri pointer) 0 oluyor */

gecici=-yenictno;
rt_buf[i].rt=birles(gecici,0,0);

/* rt de yeni girdiyi hazırla , bu, bu sident değerinin
bu ilişkinin bu sahasına ilk girişi olduğundan
same linkte sadece bir eleman var ve bunun "same link"
forward(ileri) ve backward(geri) pointer'larında 0 */

lseek(rtp[i],yenirno*sizeof(RTTAB),SEEK_SET);
if (write(rtp[i], &rt_buf[i], sizeof(RTTAB)) == -1)
    printf("problem RT\n");
/* ilgili ilişki tablosunda girdi günlendi */
lseek(ctp[i],yenictno*sizeof(CTTAB),SEEK_SET);
if (write(ctp[i], &ct_buf[i], sizeof(CTTAB)) == -1)
    printf("problem CT\n");
/* collision tablosu girdisi günlendi */

} /* else */
} /* for i=0..... */
transyaz(giratr,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);
transoku(giratr,adrp2,trans,semid1,&p,&v,adrp->tablo[pno][2]);
} /* while trans[1]!=-1 */
trans[1]=-1;
transyaz(giratr,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);

adrp->procact[pno]=0;
adrp->tablo[pno][0]=0;
for (i=0;i<=rel_buf.atrsayi;i++){
    close(rtp[i]);
    close(htp[i]);
    close(ctp[i]);
}
semop(adrp->simtan[0],&p,1);
adrp->updtubit[adrp->tablo[pno][5]]=0;
semop(adrp->simtan[0],&v,1);
semop(adrp->simtan[1],&p,1);
adrp->tablo[pno][0]=0;
adrp->procact[pno]=0;
semop(adrp->simtan[1],&v,1);
semop(adrp->simtan[2],&p,1);

```



```

adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
semop(adrp->simtan[2],&v,1);
} /* sonsuz döngü sonu */
} /* main */

```

```
/* APPLY SANAL BLOĞU KAYNAK PROGRAMI */
```

```
#include <derle.h>
```

```
main(){ /*1*/
```

```
/* değişken deklarasyonları */
```

```

FILE *rp;
RTTAB rt_buf[256],rt_bufc[256];
HTTAB ht_buf[256];
CTTAB ct_buf[256],ct_bufc[256];
FBUF f_buf;
RELINFO rel_buf;
struct q_entry s_entry;
int queue_id,rln;
char atrname[50],relname[50];
char htflen[256][24],rtflen[256][24],temp[2],dum[2],ctflen[256][24];
char rtflenl[256][24],ctflenl[256][24];
char in_fifo[50];
char relatnm[50];
int ok,fp,fifop,fifop2;
int yer,ii,sayac,sayac2,sayac3,i,k,l,j,r,sident,gec,gec2,gec3;
int ctp[256],rtp[256],htp[256],ctpc[256],rtpc[256];
int cevap,gecici,gecici1,gecici2,gecici3;
int applytxix[10][2];
long curpos2,curpos,son,hashval;
long lgec2,lgec,lgec3;
int maxeqc,mastfld,eq[256],eqc[256],eqval[256],cikatr,giratr;
int translc,translg,a;
struct tms timebuf;

```

```

pid_t pid;
processorid_t gelen;
idtype_t p_pid;
long pnum,ilk,iki;
int ka;

```

```

struct sembuf p={0,-1,SEM_UNDO};
struct sembuf v={0,1,SEM_UNDO};
ANAYAPI *adrp;
KUYRUK *adrp2;
int shmids,shmids1,semids,semids1;
long trans[100],transv[100],transc[100];
int pno;

```

```

pno=13;
pnum=1;
pid=getpid();
p_pid=getppid();
k=processor_bind(P_PID,pid,pnum,&gelen);

if((shmid1=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
    printf("\n get olmadi \n");
if((adrp=(ANAYAPI *)shmat(shmid1,(void *)0,0))==(ANAYAPI *)(-1))
    printf("\n attach olmadı \n");

if((shmid=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
    printf("\n get olmadi \n");
if((adrp2=(KUYRUK *)shmat(shmid,(void *)0,0))==(KUYRUK *)(-1))
    printf("\n attach kkkkolmadı \n");
if((queue_id=msgget(QKEY,IPC_CREAT|QPERM))==-1)
    perror("msg get failed");
/* maxeqc maksimum eşitlik şart sayısı
   mastfld tupple aramada master saha numarası
   eq endirekt veri yapısı içinde match için kullanılan
   attribute lerin sıra numarasını tasir
   eqval ise match için kullanılan değerlerin
   sident kartsiliklerinin depolandığı veri yapısıdır */
/* mesajları al */
while(1){
if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)
    perror("msgrcv failed");

while(adrp->procact[pno]!=1)a=a*a;
giratr=adrp->applygas[adrp->tablo[pno][5]];
cikatr=adrp->applycas[adrp->tablo[pno][5]];
translg=adrp->appltlg[adrp->tablo[pno][5]];
translc=adrp->appltlc[adrp->tablo[pno][5]];
yer=adrp->tablo[pno][5];

ii=1;
while (adrp->appltix[yer][ii][1]!=0){
    appltix[ii][1]=adrp->appltix[yer][ii][1];
    appltix[ii][0]=adrp->appltix[yer][ii][0];
    ii+=1;
}
for (i=0;i<giratr;i++){
    rtflen[i][0]='\0';
    htflen[i][0]='\0';
    ctflen[i][0]='\0';
    strcat(rtflen[i],adrp->applyrel[adrp->tablo[pno][5]]);
    strcat(rtflen[i],adrp->applyga[adrp->tablo[pno][5]][i]);
    strcat(htflen[i],rtflen[i]);
    strcat(ctflen[i],rtflen[i]);
    strcat(rtflen[i],".r");
    strcat(htflen[i],".h");
}

```



```

    strcat(ctfilen[i], ".c");
    rtp[i] = open(rtfilen[i], O_RDONLY);
    htp[i] = open(htfilen[i], O_RDONLY);
    ctp[i] = open(ctfilen[i], O_RDONLY);
}

for (i=0; i < cikatr; i++) {
    rtfilenc[i][0] = '\0';
    ctfilenc[i][0] = '\0';
    strcat(rtfilen[i], adrp->applyrel[adrp->tablo[pno][5]]);
    strcat(rtfilen[i], adrp->applyca[adrp->tablo[pno][5]][i]);
    strcat(ctfilenc[i], rtfilen[i]);
    strcat(rtfilen[i], ".r");
    strcat(ctfilenc[i], ".c");
    rtpc[i] = open(rtfilen[i], O_RDONLY);
    ctpc[i] = open(ctfilenc[i], O_RDONLY);
}

transoku(translg, adrp2, transv, semid1, &p, &v, adrp->tablo[pno][2]);
for (ii=0; ii <= translg; ii++) {
    if (transv[ii] < 0 && transv[ii] != -1) trans[ii] = transv[ii] * (-1);
    else trans[ii] = transv[ii];
}
ii = 1;
while (applytxix[ii][1] != 0) {
    trans[applytxix[ii][1]] = trans[applytxix[ii][0]];
    ii++;
}
/* apply blokta çıkış grupta iki noktadan sonra yer alan
saha ve değişkenlerin çıkış hareketindeki yerlerine taşınması */

sident = trans[1];

/* apply operasyonunda kullanılacak en az bir attribute
trans [1] de yer alacak */

i = 0;
while (sident != -1) {
    paket(temp, sident);
    r = hash(temp);
    lgec = r;
    /* kuyruktan okunan sident değeri ile hash tablosuna
    erişilecek */
    lseek(htp[i], lgec * sizeof(HTTAB), SEEK_SET);

    if (read(htp[i], &ht_buf[i], sizeof(HTTAB)) == -1)
        printf("\n hash tablosu okumada hata \n");

    if (ht_buf[i].ht2 != 0) {
        /*3 eğer collision bucket tesis edilmiş ise

```

```

        yani aynı zamanda bu entry boş ise */
lgec=ht_buf[i].ht2 ;
lseek(ctp[i], lgec*sizeof(CTTAB), SEEK_SET);
/* collision tablosuna geç */
if (read(ctp[i], &ct_buf[i], sizeof(CTTAB)) == -1)
    printf("problem CT\n");
while (ct_buf[i].ct2!=0&&ct_buf[i].ct1!=sident) {
    /*4 collision
        tablosunda bulununcaya kadar veya collision tablosunun
        sonuna varılincaya kadar ilerle */
    lgec=ct_buf[i].ct2;
    lseek(ctp[i],lgec*sizeof(CTTAB),SEEK_SET);
    if (read(ctp[i], &ct_buf[i], sizeof(CTTAB)) == -1)
        printf("problem CT\n");
} /*4 collision buket içinde bulana kadar veya sonuna
    ulaşp bulamayana kadar ilerledi */

if (ct_buf[i].ct1 == sident){ /*4buket içinde buldu */
    curpos=tell(ctp[i]);
    curpos=(curpos-sizeof(CTTAB))/sizeof(CTTAB);
    lgec=ct_buf[i].ct3;
    /* giriş sahasının collision tablosundan
        rt deki "same linke" atlama yapıldı */
    while(lgec!=0){ /* 5*/
        lseek(rtp[i],lgec*sizeof(RTTAB),SEEK_SET);
        /* rt tablosunda "same linkte"
            en baştaki düğüme ulaş */
        if (read(rtp[i], &rt_buf[i], sizeof(RTTAB)) == -1)
            printf("problem RT\n");
        ayris(rt_buf[i].rt,&gecici,&gecici2,&gecici3);
        /* ayırıştır */
        ok=1;
        for (j=1;j<giratr;j++){
            lseek(rtp[j],lgec*sizeof(RTTAB),SEEK_SET);
            /* diğer giriş sahasının rt tablosundan aynı
                numaralı kayıdı oku */
            if (read(rtp[j], &rt_buf[j], sizeof(RTTAB)) == -1)
                printf("problem RT\n");
            ayris(rt_buf[j].rt,&gec,&gec2,&gec3);
            /* bu kayıdı da ayırıştır */
            lgec2=-gec; /* long a çevir - den kurtar */
            lseek(ctp[j],lgec2*sizeof(CTTAB),SEEK_SET);
            if (read(ctp[j],&ct_buf[j],sizeof(CTTAB))== -1)
                printf("\n ct okumada problem \n");
            /* giriş sahası coll tablosundan sident
                değerini oku */
            if(ct_buf[j].ct1!=trans[j+1]){
                ok=0; /* uyum bozuldu */
            }
        } /* for j */
        if (ok){ /* eğer tam bir uyum varsa bu tuple

```



```

        çıkış sahaları output edilecek */
for (k=0; k<cikatr; k++){
    lseek(rtpc[k], lgec*sizeof(RTTAB), SEEK_SET);
    /* çıkış sahasının rt tablosundan aynı
       numaralı kayıdı oku */
    if (read(rtpc[k], &rt_bufc[k], sizeof(RTTAB))
        == -1)
        printf("problem çıkış RT de\n");
    ayris(rt_bufc[k].rt, &gec, &gec2, &gec3);
    /* bu kayıdı da ayırıştır */
    lgec2 = -gec; /* long a çevir - den kurtar */
    lseek(ctpc[k], lgec2*sizeof(CTTAB), SEEK_SET);
    if (read(ctpc[k],
        &ct_bufc[k], sizeof(CTTAB)) == -1)
        printf("\n çıkış ct
            okumada problem \n");
    /* çıkış sahası coll tablosunda sident
       değerini oku */
    transc[k+1] = ct_bufc[k].ct1;

    } /* for k */
    transyaz(translc, adrp2,
        transc, semid, &p, &v, adrp->tablo[pno][3]);
    } /* if ok */
    /* same linkte bir ileri hareket et */
    lgec = gecici2; /* ileri (forward) pointer atamasi */
    } /*5 while lgec!=0 */
    } /*4 if ct_buf[i].ct1==sident */
    } /*3 if ht_buf[i].ht2!=0 */

transoku(translg, adrp2, transv, semid1, &p, &v, adrp->tablo[pno][2]);
for (ii=0; ii<=translg; ii++){
    if (transv[ii]<0 && transv[ii]!=-1) trans[ii]=transv[ii]*(-1);
    else trans[ii]=transv[ii];
}
ii=1;
while (applytxix[ii][1]!=0){
    transc[applytxix[ii][1]]=trans[applytxix[ii][0]];
    ii+=1;
}
/* apply blokta çıkış iki noktadan sonra yer alan
saha ve değişkenlerin çıkış hareketindeki yerlerine taşınması */
sident=trans[1];
} /* while sident!=-1 */
transc[1]=-1;
transyaz(translc, adrp2, transc, semid, &p, &v, adrp->tablo[pno][3]);

iki=times(&timebuf);

for (i=0; i<giratr; i++){
    close(rtp[i]);

```

```

    close(htp[i]);
    close(ctp[i]);
}

for (i=0;i<cikatr;i++){
    close(rtpc[i]);
    close(ctpc[i]);
}

semop(adrp->simtan[0],&p,1);
adrp->applyubit[adrp->tablo[pno][5]]=0;
semop(adrp->simtan[0],&v,1);
semop(adrp->simtan[1],&p,1);
adrp->tablo[pno][0]=0;
adrp->procact[pno]=0;
semop(adrp->simtan[1],&v,1);
semop(adrp->simtan[2],&p,1);
adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
semop(adrp->simtan[2],&v,1);
/* fclose(rp); */
} /* while sonsuz dongu */

}/* main */

/* APPLY CUMULATIVE SANAL BLOĞU KAYNAK PROGRAMI */

#include </home/suha/final/paket.h>
#include <derle.h>

main(){ /*1*/

/* değişken bildirimleri */

GHTTAB *ght_buf,*ght_buf2,*ght_gec;
GCTTAB *gct_buf,*gct_buf2,*gct_gec;
int shmid2,shmid3;

int po[5000];
char ct[30000];
int ind[10];

/* buffer bildirimi bitti */

char isim[50],cev[2];
char temp[10],dum[2];
char *charptr,cumop;
char tempbuf[256],gec[256];
int tempint,ft,ii;
int sayac,sayac2,cumtsay,k,l,j,r,s,demir,dec,sign,bulunam, follow;
int gtp,gcp,ctp,rtp,htp,gecici,gecici1,gecici2,gecici3,ident1,ident2,ident3;

```



```

int a,ident4,ident5,cumtl    ;
long hashval,yenirno,yenictno,lgec,curpos;
char *tempptr1="\0",*tempptr2="\0";
double tempflt;
float cc,cum;
/* deęişken bildirimleri bitti */

pid_t pid;
processorid_t gelen;
idtype_t p_pid;
long sident,sident2,pnum,ilk,iki,baslan;

struct sembuf p={0,-1,SEM_UNDO};
struct sembuf v={0,1,SEM_UNDO} ;
ANAYAPI *adrp;
KUYRUK *adrp2;
struct q_entry s_entry;
int queue_id,rlen;
int i,shmid,shmid1,semid,semid1;
long trans[100];
int pno,transl,yer,fltsay;
struct tms timebuf;
FILE *fp;
char fname[10];

pnum=1;
pno=61;
pid=getpid();
p_pid=getppid();
k=processor_bind(P_PID,pid,pnum,&gelen);
paket(fname,pno);
fp=fopen("cum40","w");
k=1;

if((shmid1=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((adrp=(ANAYAPI *)shmat(shmid1,(void *)0,0))==(ANAYAPI *)(-1))
    printf ("\n attach olmadı \n");

if((shmid=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((adrp2=(KUYRUK *)shmat(shmid,(void *)0,0))==(KUYRUK *)(-1))
    printf ("\n attach kkkkolmadı \n");

if((shmid2=shmget((key_t)0x35,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((ght_buf=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
    printf ("\n attach olmadı \n");
if((shmid3=shmget((key_t)0x45,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");

```

```

if((gct_buf=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
    printf ("\n attach olmadı \n");

if((shmid2=shmget((key_t)0x55,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((ght_buf2=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
    printf ("\n attach olmadı \n");
if((shmid3=shmget((key_t)0x65,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((gct_buf2=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
    printf ("\n attach olmadı \n");
if ((queue_id= msgget(QKEY,IPC_CREAT|QPERM))==-1)
    perror("msg get failed");

/* çıkış kuyruğu */
while(1) { /* sonsuz döngü */
for(ii=0;ii<4999;ii++) po[ii]=0;
for(ii=0;ii<29999;ii++) ct[ii]=0;
if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)
    perror("msgrcv failed");

while(adrp->procact[pno]!=1)a=a+a;

yer=adrp->tablo[pno][5];
/* buraya kuyruktan okuma bölümü girecek */
transl=adrp->cumtl[yer];
cumtsay=adrp->cumtsay[yer];
for (i=0;i<cumtsay;i++){
    ind[i]=adrp->cumtxix[yer][i+1];
}
transoku(transl,adrp2,trans,semid1,&p,&v,adrp->tablo[pno][2]);
sident=trans[1];

while (sident!=-1){
    if (ind[0]>=0){/* eğer BY CLAUSE varsa */
        tempbuf[0]='\0';
        *tempptr2='\0';
        for (i=0;i<cumtsay;i++){
            paket(tempptr2,trans[ind[i]]);
            strcat(tempbuf,tempptr2);
        }
    }else { /* eğer sadece tek bir saha tasnifsiz olarak
        işleme tabi tutuluyorsa yani BY CLAUSE yoksa */
        tempbuf[0]='\0';
    }

    baslan=trans[adrp->cumfld[yer]];
    if (baslan<0) {gct_gec=gct_buf2; baslan=baslan*(-1); }
    else gct_gec=gct_buf;
    baslan+=4;
    cumop=adrp->cumop[yer];

```



```
hashval = hash(tempbuf);
```

```
/* Eğer daha önce hiç girilmemiş ise hash tablosuna coll tablosunda
bu yeni verinin yerleşeceği yeri işle ve gidip coll tablosuna bu
değeri yerleştirdikten sonra bu hash girdisinin aynı zamanda
collision bağının ilk düğümü olan bu verinin arkasına 2 byte'lık
collision bağı olarak 00 koy. */
```

```
j = hashval;
if (po[hashval] == 0){
```

```
    transyazd(transl, trans, fp);
```

```
    /* Eğer global hash tablosunun bu girdisi boş ise
    global collision tablosuna girilecek yeni değerin
    pozisyonunu belirleyen pointer değerini
    buraya yerleştir */
```

```
    po[j] = k;
    sident = k;
```

```
    /* sident çok önemli index sakla */
```

```
    l = 0;
```

```
    ct[k++] = 0;
```

```
    ct[k++] = 0;
```

```
    ct[k++] = 0;
```

```
    ct[k++] = 0;
```

```
    while (tempbuf[l] != '\0'){
        ct[k++] = tempbuf[l++];
    }
```

```
    ct[k++] = '\0';
```

```
    for (ii = 0; ii <= 7; ii++) ct[k+ii] = gct_gec->ct[baslan+ii];
```

```
    k += 8;
```

```
    /* girilen değer gct a yüklendi ve sonlandırıldı */
```

```
}
```

```
/*eğer hash entry 0'dan büyük ise daha önce buraya giriş yapılmış */
```

```
else {
```

```
    j = hashval;
```

```
    follow = 0;
```

```
    /* follow traverse esnasında bir kayıt geriden
    gelen pointer'dır */
```

```
    l = po[j]; /* Bu pointer'ı al */
```

```
    sident = l;
```

```
    bulunam = karsilas(&ct[l+4], tempbuf);
```

```
    /*
```

```
        coll tablosundaki ilk değer ile karşılaştırmak */
```

```
    while (bulunam && (l != 0)){
```

```
        temp[3] = ct[l+3];
```

```
        temp[2] = ct[l+2];
```

```
        temp[1] = ct[l+1]; /* integer pointer paket aç */
```

```
        temp[0] = ct[l];
```

```

follow=1;
l=acpaket(temp);

/* pointer'ı paketlenmiş olduğu karakter dizisi
formundan tam sayı formuna aç ve bu pointer'ı
takip ederek coll linkte bir sonraki kayıda
geç */

```

```

sident=1;
if (l>0){
    bulunam=karsilas(&ct[l+4],tempbuf) ;
}

```

```

if (l==0&&bulunam){ /* collision buketin sonuna
erişti ama bulamadı*/

```

```

    j=0;

```

```

transyazd(transl,trans,fp);

```

```

    sident=k;

```

```

    ct[k++] = 0;

```

```

    ct[k++] = 0;

```

```

    ct[k++] = 0;

```

```

    ct[k++] = 0;

```

```

    /* sident çok önemli indeks , sakla */

```

```

    while (tempbuf[j]!='\0'){

```

```

        ct[k++] = tempbuf[j++];

```

```

    ct[k++] = '\0';

```

```

    for (ii=0;ii<=7;ii++) ct[k+ii]=gct_gec->ct[baslan+ii];

```

```

    k+=8;

```

```

    /* yeni entry olarak yerleştirildi */

```

```

    paket(temp,sident);

```

```

    if (follow>0){

```

```

        paket(temp,demir);

```

```

        ct[follow]=temp[0];

```

```

        ct[follow+1]=temp[1];

```

```

        ct[follow+2]=temp[2];

```

```

        ct[follow+3]=temp[3];

```

```

        /* bir kayıt geriden gelen pointer'ı

```

```

        kullanarak bir önceki kayıda eriş

```

```

        ve bu kaydın collision forward

```

```

        bağını yeni girdiyi gösterecek

```

```

        şekilde güncle */

```

```

    }

```

```

    if (bulunam==0){

```

```

        l=sident+4;

```

```

        while (ct[l]!='\0') l++;

```

```

        l+=1;

```



```

        cum=acpaketd(ct,l);
        cc=acpaketd(gct_gec->ct,baslan);
        if (cumop=='+'){ cum=cum+cc;
        }
        if (cumop=='*') cum=cum*cc;
        if (cumop=='x') { /* maksimum */
            if (cum<cc) cum=cc;
        }
        if (cumop=='m') { /* minimum */
            if (cum>cc) cum=cc;
        }
        if (cumop=='a') { /* average */
            cum=cum*sayac+cc;
            sayac+=1;
            cum=cum/sayac;
            /* running average */
        }
        paketd(gec,(char *)&cum);
        for (ii=0;ii<=7;ii++) ct[l+ii]=gec[ii];
        k+=8;
    }

```

} /* end else == if po[hashval]>0 */
 /* şimdi girdi ya bulundu ya da yeni
 olarak yerleştirildi , bu girdiye pointer olan sident elimizde
 ve bu noktadan itibaren sadece sident kullanılacak */

```

        transoku(transl,adrp2,trans,semid1,&p,&v,adrp->tablo[pno][2]);
        sident=trans[1];
    } /* while sident!=-1 */
    transl[1]=-1;
    transyazd(transl,trans,fp);
    fclose(fp);

```

/* diske geçici olarak kaydedilen hareketlerden hareketle
 hash tablosuna erişilerek sayaç bilgisi de harekete eklenmek suretiyle
 çıkış kuyruğuna çıkış yapılır */

```

fp=fopen("cum40","r");
transokd(transl,trans,fp);
sident=trans[1];
while (sident!=-1){
    if (ind[0]>=0){
        tempbuf[0]='\0';
        for (i=0;i<cumtsay;i++){
            paket(tempptr2,trans[ind[i]]);
            strcat(tempbuf,tempptr2);
        }
    }
    else { /* eğer ind[0] negatif bir değer ise bu sadece tek bir

```

saha üzerinde BY clause olmadan yani tasnifsiz
kümülatif işlem yapılıyor demektir */

```
tempbuf[0]='\0';
}
hashval=hash(tempbuf);
```

/* eğer daha önce hiç girilmemiş ise hash tablosuna coll tablosunda
bu yeni verinin yerleşeceği yeri işle ve gidip coll tablosuna bu
değeri yerleştirdikten sonra bu hash entrynin aynı zamanda
collision link'inin ilk nodu olan bu verinin arkasına 2 byte'lık
collision pointer olarak 00 koy. */

```
j=hashval;
if (po[hashval]==0){
    printf("\n ciddi hata \n");
}
```

/*eğer hash entry 0 dan büyük ise daha önce buraya giriş yapılmış */

```
else {
    j=hashval;
    follow=0;

    /* follow traverse esnasında bir kayıt geriden
    gelen pointer'dır */

    l=po[j] ;/* Bu pointer i al */
    sident=l;
    bulunam=karsilas(&ct[l+4],tempbuf);
    /*
    coll tablosundaki ilk değer ile karşılaştırmak */

    while (bulunam && (l!=0)){
        temp[3]=ct[l+3];
        temp[2]=ct[l+2];
        temp[1]=ct[l+1]; /* integer pointer
        paket aç */

        temp[0]=ct[l];
        follow=l;
        l=acpaket(temp);

        /* pointer'ı paketlenmiş olduğu karakter dizisi
        formundan integer formuna aç ve bu pointer'ı
        takip ederek coll linkte bir sonraki kayıda
        geç */

        sident=l;
        if (l>0){
            bulunam=karsilas(&ct[l+4],tempbuf) ;
```



```

    }
    }
    if (l==0&&bulunam){
        /* collision buketin sonuna
        erişti ama bulamadı*/
        printf("\n ciddi hata \n");
    }

    if (bulunam==0){
        l=sident+4;
        while (ct[l]!='\0') l++;
        l+=1;
        temp[0]=ct[l];
        temp[1]=ct[l+1];
        temp[2]=ct[l+2];
        temp[3]=ct[l+3];
        temp[4]=ct[l+4];
        temp[5]=ct[l+5];
        temp[6]=ct[l+6];
        temp[7]=ct[l+7];
        temp[8]='\0';
        sident2=altyapi(ght_buf2,gct_buf2,temp);
        trans[adrp->cumtloc[yer]]=sident2;
        transyaz(transl+1,adrp2,trans,
            semid,&p,&v,adrp->tablo[pno][3]);
    }
} /* end else === if po[hashval]>0 */

/* simdi girdi ya bulundu ya da yeni
olarak yerleştirildi , bu girdiye pointer olan sident elimizde
ve bu noktadan itibaren sadece sident kullanılacak */
transokd(transl,trans,fp);
sident=trans[1];
} /* while sident!=-1 */
fclose(fp);
trans[1]=-1;
transyaz(transl+1,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);

/* _Cnt degiskeni eklendiğinden daima okunan transl den bir fazla
çıkış transl olacaktır */

semop(adrp->simtan[0],&p,1);
adrp->cumubit[adrp->tablo[pno][5]]=0;
semop(adrp->simtan[0],&v,1);
semop(adrp->simtan[1],&p,1);

```

```

adrp->tablo[pno][0]=0;
adrp->procact[pno]=0;
semop(adrp->simtan[1],&v,1);
semop(adrp->simtan[2],&p,1);
adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
semop(adrp->simtan[2],&v,1);
} /* sonsuz dongu sonu */

/* main */

```

/* CARTESIAN PRODUCT SANAL BLOĞU KAYNAK PROGRAMI */

```
#include <derle.h>
```

```

main() { /*1*/
RTTAB rt_buf[256],rt_bufc[256];
HTTAB ht_buf[256];
CTTAB ct_buf[256],ct_bufc[256];
FBUF f_buf;
RELINFO rel_buf;
struct q_entry s_entry;
int queue_id,rlen;

/* değişken deklarasyonlari */
struct stat buf;
int retval;
char atrname[50],relname[50];
char htflen[256][12],rtflen[256][12],temp[2],dum[2],ctflen[256][12];
char rtflenl[256][12],ctflenl[256][12];
char in_fifo[50];
char relatnm[50];
int ok,fp,fifop,fifop2;
int sayac,sayac2,sayac3,i,k,l,j,r,sident,gec,gec2,gec3,a;
int ctp[256],rtp[256],htp[256],ctpc[256],rtpc[256];
int cptlg,transl,bias,cikamaz,cevap,gecici,gecici1,gecici2,gecici3;
long curpos2,curpos,son,hashval;
long lgec2,lgec,lgec3;
int maxeqc,mastfld,eq[256],eqc[256],eqval[256],cikatrsl,giratrsl;
struct tms timebuf;

pid_t pid;
processorid_t gelen;
long pnum,ilk,iki;
idtype_t p_pid;

struct sembuf p={0,-1,SEM_UNDO};
struct sembuf v={0,1,SEM_UNDO};
ANAYAPI *adrp;
KUYRUK *adrp2;

```



```

int shmid,shmid1,semid,semid1;
long trans[100];
int pno;

pno=43;
pnum=2;
pid=getpid();
p_pid=getppid();
k=processor_bind(P_PID,pid,pnum,&gelen);
cikamaz=0;

if((shmid1=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((adrp=(ANAYAPI *)shmat(shmid1,(void *)0,0))==(ANAYAPI *)(-1))
    printf ("\n attach olmadı \n");

if((shmid=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((adrp2=(KUYRUK *)shmat(shmid,(void *)0,0))==(KUYRUK *)(-1))
    printf ("\n attach kkkkolmadı \n");
if ((queue_id= msgget(QKEY,IPC_CREAT|QPERM))==-1)
    perror("msg get failed");

/* mesajları al */
while(1) { /* sonsuz döngü */
if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)
    perror("msgrcv failed");
while(adrp->procact[pno]!=1)a=a+a;

cikatr=adrp->cpcas[adrp->tablo[pno][5]];
cptlg= adrp-> cptlg[adrp->tablo[pno][5]];
bias=transl=cptlg;

for (i=0;i<cikatr;i++){
    rtfilenc[i][0]='\0';
    ctfilenc[i][0]='\0';
    strcat(rtfilenc[i],adrp->cpistb[adrp->tablo[pno][5]][0]);
    strcat(rtfilenc[i],adrp->cpistb[adrp->tablo[pno][5]][i+1]);
    strcat(ctfilenc[i],rtfilenc[i]);
    strcat(rtfilenc[i],".r");
    strcat(ctfilenc[i],".c");
    rtpc[i]=open(rtfilenc[i], O_CREAT|O_RDWR);
    ctpc[i]=open(ctfilenc[i], O_CREAT |O_RDWR);
}

son=lseek(rtpc[0],0L,SEEK_END);
retval=fstat(rtpc[0],&buf);
son=buf.st_size;

```

```

son=(son-sizeof(RTTAB))/sizeof(RTTAB);

transoku(transl,adrp2,trans,semid,&p,&v,adrp->tablo[pno][2]);
sident=trans[1];

while(sident!=-1){
for (lgec=1;lgec<=son;lgec++){
    for (j=0;j<cikatr;j++){
        lseek(rtpc[j],lgec*sizeof(RTTAB),SEEK_SET);
        /* diğer giriş sahasının rt tablosundan aynı
        numaralı kayıdı oku */
        if (read(rtpc[j],
            &rt_buf[j], sizeof(RTTAB)) ==-1)
            printf("problem RT\n");
        ayris(rt_buf[j].rt,&gec,&gec2,&gec3);
        /* bu kayıdı da ayrıştır */
        if (gec==999) {cikamaz=1;}
        else {
            lgec2=-gec; /* long a çevir - den kurtar */
            lseek(ctpc[j],lgec2*sizeof(CTTAB),SEEK_SET);
            if (read(ctpc[j],&ct_buf[j],sizeof(CTTAB))==-1)
                printf("\n ct okumada problem \n");
            /* giriş attribute u coll table dan sident
            değerini oku */
            trans[j+bias+1]=ct_buf[j].ct1;
            cikamaz=0;
        }
    } /* for j */
} /* eğer daha önce iptal edilmiş bir tuple değil ise çıkışa
yaz toggle'ı tekrar eski konumuna al */
if (!cikamaz){
    transyaz(cikatr+cptlg,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);
    cikamaz=0;
}
} /* for lgec */
transoku(transl,adrp2,trans,semid,&p,&v,adrp->tablo[pno][2]);
sident=trans[1];
} /* while sident!=-1 */
trans[1]=-1;
transyaz(cikatr+cptlg,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);

adrp->procact[pno]=0;
adrp->tablo[pno][0]=0;
for (i=0;i<cikatr;i++){
    close(rtpc[i]);
    close(ctpc[i]);
}
semop(adrp->simtan[0],&p,1);
adrp->cpubit[adrp->tablo[pno][5]]=0;
semop(adrp->simtan[0],&v,1);

```



```

semop(adrp->simtan[1],&p,1);
adrp->tablo[pno][0]=0;
adrp->procact[pno]=0;
semop(adrp->simtan[1],&v,1);
semop(adrp->simtan[2],&p,1);
adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
semop(adrp->simtan[2],&v,1);
} /* sonsuz döngü sonu */
}/* main */

```

```
/* COUNTBY SANAL BLOĞU KAYNAK PROGRAMI */
```

```
#include <derle.h>
```

```

int karsilas(char *t,char *ilk){
/* bu fonksiyon karşılaştırma amaçlı kullanılmaktadır */
int n;
n=0;
for (;*ilk!='\0';ilk++){
if (*ilk!=*t)n=1;
t++;
}
return(n);
}

```

```
main(){ /*1*/
```

```
/* değişken bildirimleri */
```

```

struct q_entry s_entry;
int queue_id,rlen;

```

```

int po[5000];
char ct[30000];
int ind[10];

```

```

char isim[50],cev[2];
char temp[4],dum[2];
char *charptr;
char tempbuf[256],gec[256];
int tempint,ft,a;
int sayac,sayac2,cntsay,k,l,j,r,s,demir,dec,sign,bulunam,follow;
int gtp,gcp,ctp,rtp,htp,gecici,gecici1,gecici2,gecici3,ident1,ident2,ident3;
int ident4,ident5;

```



```

long hashval,yenirno,yenictno,lgec,curpos;
char *tempptr1="\0",*tempptr2="\0";
double tempflt;
float cc;

```

```

pid_t pid;
processorid_t gelen;
idtype_t p_pid;
long sident,sident2, pnum,ilk,iki;

struct sembuf p={0,-1,SEM_UNDO};
struct sembuf v={0,1,SEM_UNDO} ;
ANAYAPI *adrp;
KUYRUK *adrp2;
GCTTAB *gct_buf2;
GHTTAB *ght_buf2;
int i,shmid,shmid1,shmid2,shmid3,semid,semid1;
long trans[100];
int pno,transl,yer,fltsay;
struct tms timebuf;
FILE *fp;
char fname[10];

```

```

pnum=0;
pid=getpid();
p_pid=getppid();
k=processor_bind(P_PID,pid,pnum,&gelen);
pno=31;
paket(fname,pno);
fp=fopen("say47","w");
k=1;

```

```

if((shmid1=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((adrp=(ANAYAPI *)shmat(shmid1,(void *)0,0))==(ANAYAPI *)(-1))
    printf ("\n attach olmadı \n");

if((shmid=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((adrp2=(KUYRUK *)shmat(shmid,(void *)0,0))==(KUYRUK *)(-1))
    printf ("\n attach kkkkolmadı \n");
if((shmid2=shmget((key_t)0x55,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((ght_buf2=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
    printf ("\n attach olmadı \n");
if((shmid3=shmget((key_t)0x65,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((gct_buf2=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
    printf ("\n attach olmadı \n");
if ((queue_id= msgget(QKEY,IPC_CREAT|QPERM))==-1)

```



```

    perror("msg get failed");

/* çıkış kuyruğu */
while(1) { /* sonsuz döngü */
    for(i=0;i<=4999;i++) po[i]=0;
    for(i=0;i<=19999;i++) ct[i]=0;
    if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)
        perror("msgrcv failed");
    printf("\n mesaj alındı %ld \n",s_entry.mtype);
    while(adrp->procact[pno]!=1)a=a+a;

    yer=adrp->tablo[pno][5];

    transl=adrp->cnttl[yer];
    cntsay=adrp->cnsay[yer];
    for (i=0;i<cnsay;i++){
        ind[i]=adrp->cntxix[yer][i+1];
    }
    transoku(transl,adrp2,trans,semid1,&p,&v,adrp->tablo[pno][2]);
    sident=trans[1];

    while (sident!=-1){
        tempbuf[0]='\0';
        *tempptr2='\0';
        for (i=0;i<cnsay;i++){
            paket(tempptr2,trans[ind[i]]);
            strcat(tempbuf,tempptr2);
        }

        hashval=hash(tempbuf);

/* eğer daha önce hiç girilmemiş ise hash tablosuna coll tablosunda
bu yeni verinin yerleşeceği yeri işle ve gidip coll tablosuna bu
değeri yerleştirdikten sonra bu hash girdisinin aynı zamanda
collision bağının ilk düğümü olan bu verinin arkasına 2 bytelik
collision pointer olarak 00 koy. */

        j=hashval;
        if (po[hashval]==0){

            transyazd(transl,trans,fp);

            /* eğer global hash tablosunun bu girdisi boş ise
            global collision tablosuna girilecek yeni değerin
            pozisyonunu belirleyen pointer (k) değerini
            buraya yerleştir */

            po[j]=k;
            sident=k;

```

```

/* sident çok önemli indeks sakla */
l=0;
ct[k++] = 0;
ct[k++] = 0;
ct[k++] = 0;
ct[k++] = 0;
while (tempbuf[l] != '\0'){
    ct[k++] = tempbuf[l++];
}
ct[k++] = '\0';
paket(temp, 1);
ct[k++] = temp[0];
ct[k++] = temp[1];
ct[k++] = temp[2];
ct[k++] = temp[3];
/* girilen değer gct'a yüklendi ve sonlandırıldı */
}

/*eğer hash entry 0 dan büyük ise daha önce buraya giriş yapılmış */
else {
    j=hashval;
    follow=0;
    /* follow traverse esnasında bir kayıt geriden
    gelen pointer */
    l=po[j] ;/* Bu pointer i al */
    sident=1;
    bulunam=karsilas(&ct[l+4],tempbuf);
    /*
    coll tablosundaki ilk değer ile karşılaştır */
    while (bulunam && (l!=0)){
        temp[3]=ct[l+3];
        temp[2]=ct[l+2];
        temp[1]=ct[l+1]; /* integer pointer
        paket ac */
        temp[0]=ct[l];
        follow=1;
        l=acpaket(temp);

        /* pointer i paketlenmiş olduğu karakter dizisi
        formundan integer formuna aç ve bu pointer i
        takip ederek coll linkte bir sonraki kayıda
        gec */

        sident=1;
        if (l>0){
            bulunam=karsilas(&ct[l+4],tempbuf) ;
        }
    }

    if (l==0&&bulunam){ /* collision bucket in sonuna
    eristi ama bulamadi*/

```



```

j=0;
transyazd(transl,trans,fp);
sident=k;
ct[k++]=0;
ct[k++]=0;
ct[k++]=0;
ct[k++]=0;
/* sident çok önemli indeks , sakla */
while (tempbuf[j]!='\0'){
    ct[k++]=tempbuf[j++];
}
ct[k++]='\0';
paket(temp,1);
ct[k++]=temp[0];
ct[k++]=temp[1];
ct[k++]=temp[2];
ct[k++]=temp[3];
/* yeni girdi olarak yerleştirildi */
paket(temp,sident);
if (follow>0){
    paket(temp,demir);
    ct[follow]=temp[0];
    ct[follow+1]=temp[1];
    ct[follow+2]=temp[2];
    ct[follow+3]=temp[3];
    /* bir kayıt geriden gelen pointer'ı
    kullanarak bir önceki kayıda eriş
    ve bu kaydın collision forward
    linkini yeni girdiyi gösterecek
    şekilde güncle */
}
}
if (bulunam==0){
    l=sident+4;
    while (ct[l]!='\0') l++;
    l+=1;
    temp[0]=ct[l];
    temp[1]=ct[l+1];
    temp[2]=ct[l+2];
    temp[3]=ct[l+3];
    j=acpaket(temp);
    j+=1;
    paket(temp,j);
    ct[l]=temp[0];
    ct[l+1]=temp[1];
    ct[l+2]=temp[2];
    ct[l+3]=temp[3];
}
}
/* end else === if po[hashval]>0 */

```

/* simdi girdi ya bulundu ya da yeni
olarak yerlestirildi , bu girdiye pointer olan sident elimizde
ve bu noktadan itibaren sadece sident kullanılacak */

```
transoku(transl,adrp2,trans,semid1,&p,&v,adrp->tablo[pno][2]);
sident=trans[1];
} /* while sident!=-1 */
trans[1]=-1;
transyazd(transl,trans,fp);
fclose(fp);
```

/* diske gecici olarak kaydedilen hareketlerden hareketle
hash tablosuna erisilerek sayac bilgisi de harekete eklenmek suretiyle
cikis kuyruğuna cikis yapilir */

```
fp=fopen("say47","r");
transokd(transl,trans,fp);
sident=trans[1];
while (sident!=-1){
    tempbuf[0]='\0';
    for (i=0;i<cntsay;i++){
        paket(tempptr2,trans[ind[i]]);
        strcat(tempbuf,tempptr2);
    }
```

```
hashval=hash(tempbuf);
/* eger daha once hic girilmemis ise hash tablosuna coll tablosunda
bu yeni verinin yerlesecegi yeri isle ve gidip coll tablosuna bu
degeri yerlestirdikten sonra bu hash girdisinin ayni zamanda
collision baginin ilk dugumu olan bu verinin arkasina 2 bytelik
collision pointer olarak 00 koy. */
j=hashval;
if (po[hashval]==0){
```

```
    printf("\n ciddi hata \n");
}
```

```
/*eger hash entry 0 dan buyuk ise daha once buraya giris yapilmis */
else {
```

```
    j=hashval;
    follow=0;
    /* follow traverse esnasinda bir kayıt geriden
    gelen pointer */
    l=po[j] ;/* Bu pointer'ı al */
    sident=l;
    bulunam=karsilas(&ct[l+4],tempbuf);
    /*
```

```
    coll tablosundaki ilk deger ile karsilastir */
    while (bulunam && (!=0)){
```



```

temp[3]=ct[l+3];
temp[2]=ct[l+2];
temp[1]=ct[l+1]; /* integer pointer
                    paket aç */
temp[0]=ct[l];
follow=1;
l=acpaket(temp);

/* pointer'ı paketlenmiş olduğu karakter dizisi
formundan integer formuna aç ve bu pointer'ı
takip ederek coll linkte bir sonraki kayıda
geç */

sident=1;
if (l>0){
    bulunam=karsilas(&ct[l+4],tempbuf) ;
}

}

if (l==0&&bulunam){ /* collision bucket'in sonuna
                    erişti ama bulamadı*/
    printf("\n ciddi hata \n");
}

if (bulunam==0){
    l=sident+4;
    while (ct[l]!='\0') l++;
    l+=1;
    temp[0]=ct[l];
    temp[1]=ct[l+1];
    temp[2]=ct[l+2];
    temp[3]=ct[l+3];
    j=acpaket(temp);
    cc=j;
    paketd(gec,(char *)&cc);
    sident2=altyapi(ght_buf2,gct_buf2,gec);
    trans[adrp->cntloc[yer]]=sident2;
    transyaz(transl+1,adrp2,trans,
             semid,&p,&v,adrp->tablo[pno][3]);
}

} /* end else == if po[hashval]>0 */
/* simdi girdi ya bulundu ya da yeni
olarak yerlestirildi , bu girdiye pointer olan sident elimizde
ve bu noktadan itibaren sadece sident kullanılacak */

transokd(transl,trans,fp);
sident=trans[1];
} /* while sident!=-1 */

```

```

trans[1]=-1;
transyaz(transl+1,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);
/* _Cnt değişkeni eklendiğinden daima okunan transl den bir fazla
çıkış transl olacaktır */

```

```

semop(adrp->simtan[0],&p,1);
adrp->applyubit[adrp->tablo[pno][5]]=0;
semop(adrp->simtan[0],&v,1);
semop(adrp->simtan[1],&p,1);
adrp->tablo[pno][0]=0;
adrp->procact[pno]=0;
semop(adrp->simtan[1],&v,1);
semop(adrp->simtan[2],&p,1);
adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
semop(adrp->simtan[2],&v,1);
} /* sonsuz döngü sonu */

```

```

}/* main */

```

```

/* COMPUTE FIX SANAL BLOĞU KAYNAK PROGRAMI */

```

```

#include <derle.h>

```

```

main(){
GHTTAB *ght_buf,*ght_buf2;
GCTTAB *gct_buf,*gct_buf2;
CTTAB ct_buf[10];
RTTAB rt_buf[10],rt_buf2;
HTTAB ht_buf[10];
FILE *kp;
char dumy,a,mesaj1[2],mesaj2[2],mesaj3[1],mesaj4[2],temp1[10],temp2[10];
char temp[100],dum[2],mesaj[2],mesajh[2],rname[20],atname[20],relnam[20];
char rtflen[20],htflen[20],ctflen[20];
char *r=".r",*h=".h",*c=".c";
int inst[100],bas,q,i,gcp,gtp,trcv,trov,r1,r2,fdsay;
int tlg,tlc,fd[10][3],htp,ctp,rtp,ltsay,sayac,deleteon;
long lt[10];
float cc,temps[100],filref[10],transv[32],transcf[100] ;
long lgec,lgec2,lgec3,tnum[10],sident,sident1,sident2;
long yenirno,curpos,yenictno;
int tnumind[10],gecic,gec,gec2,gec3,gec1,gec12,gec13,blnmdi;
float oper1,oper2;
double son;
char crossc[100][100][1];
char tempbuf[100],tempbuf2[100];
char *charptr,index[10],operat,hedef;
int pos1,pos2,pos4,j,k,l,b,m,dec,sign,follow;
int hashval,bulunam,ind;

```



```

double gecici;
struct sembuf p={0,-1,SEM_UNDO};
struct sembuf v={0,1,SEM_UNDO} ;
ANAYAPI *adrp;
KUYRUK *adrp2;
struct q_entry s_entry;
int queue_id,rlen;
int shmid,shmid1,shmid2,shmid3,semid,semid1;
long trans[100],transc[100];
int pno,giratr,cikatr;
int instrsay,yer;
int pnum;
pid_t pid;
processorid_t gelen;
idtype_t p_pid;

deleteon=0;

pno=19;
pnum=2;
pid=getpid();
p_pid=getppid();
k=processor_bind(P_PID,pid,pnum,&gelen);

if((shmid1=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((adrp=(ANAYAPI *)shmat(shmid1,(void *)0,0))==(ANAYAPI *)(-1))
    printf ("\n attach olmadı \n");

if((shmid=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((adrp2=(KUYRUK *)shmat(shmid,(void *)0,0))==(KUYRUK *)(-1))
    printf ("\n attach kkkkolmadı \n");

/* global hash tablosu ve global carpisma tablosunu shared
memory olarak al */

if((shmid2=shmget((key_t)0x35,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((ght_buf=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
    printf ("\n attach olmadı \n");
if((shmid3=shmget((key_t)0x45,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((gct_buf=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
    printf ("\n attach olmadı \n");

```

```

if((shmid2=shmget((key_t)0x55,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
    printf("\n get olmadı \n");
if((ght_buf2=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
    printf("\n attach olmadı \n");
if((shmid3=shmget((key_t)0x65,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
    printf("\n get olmadı \n");
if((gct_buf2=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
    printf("\n attach olmadı \n");
if ((queue_id= msgget(QKEY,IPC_CREAT|QPERM))==-1)
    perror("msg get failed");

```

```

/* çıkış kuyruğu */
while(1) { /* sonsuz döngü */
if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)
    perror("msgrcv failed");
while(adrp->procact[pno]!=1)b=b+b;

```

```

yer=adrp->tablo[pno][5];

```

```

giratr=adrp->compfgas[yer];
cikatr=adrp->compfcas[yer];
instrsay=adrp->compfkas[yer];
tlg=adrp->compftlg[yer];
tlc=adrp->compftlc[yer];

```

```

fdsay=giratr; /* file descriptor tablosu pointer'ını sıfırla */

```

```

strcpy(rname,adrp->compufse[yer][0].relname);
for (i=0;i<giratr;i++){
    tnumind[i]=adrp->compufse[yer][i].ind;
    strcpy(atname,adrp->compufse[yer][i].atname);
    strcpy(relname,rname);
    strcat(relname,atname);
    strcpy(rtfile,relname);
    strcpy(htfile,relname);
    strcpy(ctfile,relname);
    strcat(rtfile,r);
    strcat(htfile,h);
    strcat(ctfile,c);

    fd[i][0]=open(rtfile, O_CREAT|O_RDWR);
    fd[i][1]=open(htfile, O_CREAT|O_RDWR);
    fd[i][2]=open(ctfile, O_CREAT|O_RDWR);
}

```

```

for (i=0;i<giratr;i++){
    lt[i]=adrp->compflt[yer][i];
}

```



```

}
for (i=0;i<=instrsay;i++){
    crossc[adrp->compfinst[yer][i][1]]
        [adrp->compfinst[yer][i][2]] [0]=adrp->compfop[yer][i];
    crossc[adrp->compfinst[yer][i][1]]
        [adrp->compfinst[yer][i][2]][1]=adrp->compfinst[yer][i][0];
}

```

```

for (i=0;i<instrsay;i++){
    inst[i]=adrp->compficrk[yer][i];
}

```

```

transoku(tlg,adrp2,trans,semid,&p,&v,adrp->tablo[pno][2]);
sident=trans[1];

```

```

while (sident!=-1){
    /* yerleştirme yaptı sıfırlama yapmadı */
    /* bu noktada eğer locate'ler varsa yapılacak */
    if (giratrs>0){
        sayac=0;
        while (sayac<giratrs){
            if (lt[sayac]!=0){
                if (lt[sayac]>0){ /* sident değeri direkt */
                    sident=lt[sayac];
                } else{
                    sident=trans[lt[sayac]*(-1)];
                }
                paket(temp,sident); /* sident pack yapılarak
                                     hash argument'i olarak kullanılmalı */
                htp=fd[sayac][1];
                ctp=fd[sayac][2];
                hashval=hash(temp);
                lseek(htp,hashval*sizeof(HTTAB),SEEK_SET);
                /* ht file'da git bul */
                if (read(htp, &ht_buf[sayac], sizeof(HTTAB)) ==-1)
                    printf("problem reading HT\n");

                if (ht_buf[sayac].ht2!=0){
                    /*eğer collision bucket tesis edilmiş ise
                     yani aynı zamanda bu girdi boş ise */
                    lgec=ht_buf[sayac].ht2 ;
                    lseek(ctp, lgec*sizeof(CTTAB), SEEK_SET);
                    /* collision tablosuna gec */
                    if (read(ctp, &ct_buf[sayac], sizeof(CTTAB)) ==-1)
                        printf("problem C\n");
                    while (ct_buf[sayac].ct2!=0&&ct_buf[sayac].ct1!=sident) {
                        lgec=ct_buf[sayac].ct2;
                        lseek(ctp,lgec*sizeof(CTTAB),SEEK_SET);
                        if (read(ctp, &ct_buf[sayac], sizeof(CTTAB)) ==-1)
                            printf("problem CT\n");
                    }
                }
            }
            sayac++;
        }
    }
}

```

```

    } /* collision buket içinde bulana kadar veya sonuna
        ulaşp bulamayana kadar ilerledi */
    if (ct_buf[sayac].ct1 == sident) { /* bucket içinde buldu */
        tnum[sayac] = ct_buf[sayac].ct3;
    } else tnum[sayac] = 0;
    } /* if ht_buf[sayac].ht2 != 0 */
    else tnum[sayac] = 0;
    } /* if lt[sayac] != 0 */
    sayac += 1;
    } /* while sayac <= giratrs */

} /* if giratrs > 0 */

/* komutlari sirasiyla al */
q = 0;
while (inst[q] != 0) {
    gecici = inst[q] / 100;
    pos2 = gecici;
    pos4 = inst[q] - pos2 * 100;
    /* hareket değişkenlerinden ilk mesajda belirtilenleri
    al */
    if (pos2 < 32 && pos2 > 0) {
        sident1 = trans[pos2];
        if (sident1 > 0) {
            bas = sident1 + 4;
            oper1 = acpaketd(gct_buf -> ct, bas);
        } else {
            bas = sident1 * (-1) + 4;
            oper1 = acpaketd(gct_buf2 -> ct, bas);
        }
    }

    if (pos4 < 32 && pos4 > 0) {
        sident2 = trans[pos4];
        if (sident2 > 0) {
            bas = sident2 + 4;
            oper2 = acpaketd(gct_buf -> ct, bas);
        } else {
            bas = sident2 * (-1) + 4;
            oper2 = acpaketd(gct_buf2 -> ct, bas);
        }
    }

    if (pos2 >= 32 && pos2 < 62) {
        oper1 = temps[pos2 - 32];
    }

    if (pos4 >= 32 && pos4 < 62) {
        oper2 = temps[pos4 - 32];
    }

    if (pos2 >= 62 && pos2 < 72) {
        sayac = pos2 - 62;
    }

```



```

if (tnumind[sayac]!=0){
    lseek(fd[sayac][0],
        tnum[tnumind[sayac]-1]*sizeof(RTTAB),SEEK_SET);
    /* çıkış sahasının rt tablosundan aynı
    numaralı kayıdı oku */
    if (read(fd[sayac][0], &rt_buf[sayac], sizeof(RTTAB)) == -1)
        printf("problem RTcik\n");
    ayris(rt_buf[sayac].rt,&gec,&gec2,&gec3);
    /* bu kayıdı ayrıştır */
    lgec2=-gec; /* long a çevir - den kurtar */
    lseek(fd[sayac][2],lgec2*sizeof(CTTAB),SEEK_SET);
    if (read(fd[sayac][2],&ct_buf[sayac],sizeof(CTTAB)) == -1)
        printf("\n çıkış ct okumada problem \n");
    /* çıkış sahası coll tablosundan sident
    değerini oku */
    sident=ct_buf[sayac].ct1;
} else { /* tnumind[sayac] == 0 */
    if (lt[sayac]>0){ /* sident değeri direkt */
        sident=lt[sayac];
    } else{
        sident=trans[lt[sayac]*(-1)];
    }
}

}/* else */
if (sident>0){
    bas=sident+4;
    oper1=acpaketd(gct_buf->ct,bas);
}else {
    bas=sident*(-1)+4;
    oper1=acpaketd(gct_buf2->ct,bas);
}

}
if (pos4>=62&&pos4<72){
    sayac=pos4-62;
    if (tnumind[sayac]!=0){
        lseek(fd[sayac][0],
            tnum[tnumind[sayac]-1]*sizeof(RTTAB),SEEK_SET);
        /* çıkış sahasının rt tablosundan aynı
        numaralı kayıdı oku */
        if (read(fd[sayac][0], &rt_buf[sayac], sizeof(RTTAB)) == -1)
            printf("problem RTcik\n");
        ayris(rt_buf[sayac].rt,&gec,&gec2,&gec3);
        /* bu kayıdı ayrıştır */
        lgec2=-gec; /* long a çevir - den kurtar */
        lseek(fd[sayac][2],lgec2*sizeof(CTTAB),SEEK_SET);
        if (read(fd[sayac][2],&ct_buf[sayac],sizeof(CTTAB)) == -1)
            printf("\n çıkış ct okumada problem \n");
        /* çıkış sahası coll tablosunda sident
        değerini oku */
        sident=ct_buf[sayac].ct1;
    }
}

```

```

} else { /* tnumind[sayac] == 0 */
    if (lt[sayac] > 0) { /* sident değeri direkt */
        sident = lt[sayac];
    } else {
        sident = trans[lt[sayac]*(-1)];
    }
}

/* else */
if (sident > 0) {
    bas = sident + 4;
    oper2 = acpaketd(gct_buf - > ct, bas);
} else {
    bas = sident*(-1) + 4;
    oper2 = acpaketd(gct_buf2 - > ct, bas);
}
}

operat = crossc[pos2][pos4][0];
hedef = crossc[pos2][pos4][1];
if (operat == '*') {
    if (hedef < 32)
        transv[hedef] = oper1 * oper2;
    if (hedef >= 32 && hedef < 62) {
        temps[hedef-32] = oper1 * oper2;
    }
    if (hedef >= 62 && hedef < 72)
        filref[hedef-62] = oper1 * oper2;
}
}

if (operat == '+') {
    if (hedef < 32)
        transv[hedef] = oper1 + oper2;
    if (hedef >= 32 && hedef < 62)
        temps[hedef-32] = oper1 + oper2;
    if (hedef >= 62 && hedef < 72)
        filref[hedef-62] = oper1 + oper2;
}

if (operat == '-') {
    if (hedef < 32)
        transv[hedef] = oper1 - oper2;
    if (hedef >= 32 && hedef < 62)
        temps[hedef-32] = oper1 - oper2;
    if (hedef >= 62 && hedef < 72)
        filref[hedef-62] = oper1 - oper2;
}

if (operat == '/') {
    if (hedef < 32)
        transv[hedef] = oper1 / oper2;
    if (hedef >= 32 && hedef < 62)
        temps[hedef-32] = oper1 / oper2;
    if (hedef >= 62 && hedef < 72)
        filref[hedef-62] = oper1 / oper2;
}
}

```



```

if (operat == '=' ) {
    if(hedef<32)
        transv[hedef]=oper1;
    if(hedef>=32&& hedef<62)
        temps[hedef-32]=oper1;
    if(hedef>=62&&hedef<72)
        filref[hedef-62]=oper1;
    if(hedef>=72&& hedef<102){
        if (pos2<32) transc[hedef-72+1]=trans[pos2];
        /* direkt trans'dan transc'ye gecis */
        else transcf[hedef-72]=oper1;
    }
}
if (operat == '$') {
    if (hedef>=32&&hedef<62){
        temps[hedef-32]=adrp->compsbt[yer][pos2-32];
    }
}
if (operat == '&') {
    if (hedef>=62&&hedef<72){
        sident=trans[pos2];
        if (sident<0){
            strcpy(tempbuf,&gct_buf2->ct[sident*(-1)+4]);
            sident= altyapi(ght_buf,gct_buf,tempbuf);
            /* eğer gelen sident negatif ise bunu asıl global
            ct tablosuna taşı ve bu tablodaki sident'i kullan
            çünkü bu veri artık veri tabanına giriyor */
        }
    }
}

```

if (operat != '&') { /* eğer direk sident atama operasyonu değil ise
yani string sabiti temsil eden bir sident değerini direk database'e
işleme operasyonu değilse */

```

if(hedef>=72&& hedef<102){
    if (pos2>=0){
        cc=transcf[crossc[pos2][pos4][1]-72];
    }
}
if(hedef>=62&&hedef<72){
    cc=filref[crossc[pos2][pos4][1]-62];
}
} /* endif operat != '&' */

```

if(hedef>=62&&hedef<102) { /* eğer dışarı gidiyorsa veya
database e işleniyorsa bu bölüm
devreye girer */

if (operat != '&') { /* eğer direk sident atama operasyonu değil ise
yani string sabiti temsil eden bir sident değerini direkt database'e

```

işleme operasyonu değilse */
if (!(hedef >= 72 && pos2 <= 32) || (hedef < 72)){
    /* eğer transa'dan transc ye direkt sident geçişi değil ise
       yani bu blok içinde hesaplanmış değer ise veya
       filref'den geliyorsa sident karşılığını bul yoksa
       yarat */
    paketd(tempbuf, (char *)&cc);
    /* floating point i string e çevir, decimal pozisyonu ve isareti
       virgül ile ayrılmış olarak sona yerleştir */
    hashval = hash(tempbuf);
    /* eğer daha önce hic girilmemiş ise hash tablosuna coll tablosunda
       bu yeni verinin yerleşeceği yeri işle ve gidip coll tablosuna bu
       değeri yerleştirdikten sonra bu hash girdisinin aynı zamanda
       collision bağının ilk nodu olan bu verinin arkasına 2 bytelik
       collision pointer olarak 00 koy. */
    j = hashval;
    if (ght_buf->po[hashval] == 0){
        sident = altyapi(ght_buf, gct_buf, tempbuf);
    }

    /*eğer hash entry 0 dan büyük ise daha önce buraya giriş yapılmış */
    else {
        j = hashval;
        follow = 0;
        /* follow traverse esnasında bir kayıt geriden
           gelen pointer 'dır */
        l = ght_buf->po[j]; /* Bu pointer'ı al */
        sident = l;
        bulunam = karsilas(&gct_buf->ct[l+4], tempbuf);

        /*
           coll tablosundaki ilk değer ile karşılaştır */
        while (bulunam && (l != 0)){
            temp[3] = gct_buf->ct[l+3]; /* integer pointer
            temp[2] = gct_buf->ct[l+2]; /* integer pointer
            temp[1] = gct_buf->ct[l+1]; /* integer pointer
            paket ac */
            temp[0] = gct_buf->ct[l];
            follow = l;
            l = acpaket(temp);

            /* pointer'ı paketlenmiş olduğu karakter dizisi
               formundan integer formuna ac ve bu pointer'ı
               takip ederek coll linkte bir sonraki kayıda
               geç */
            sident = l;
            if (l > 0){
                bulunam = karsilas(&gct_buf->ct[l+4], tempbuf);
            }
        }
    }
}

```



```

    }

    if (l == 0 && bulunam) { /* collision bucket'in sonuna
                                erişti ama bulamadı */
        sident = altyapi(ght_buf, gct_buf, tempbuf);
    }

    if (bulunam == 0) ; /* bulundu sident
                        dogru değere eşitlendi */
} /* end else == if ght_buf->po[hashval] > 0 */

/* simdi entry ya bulundu ya da yeni
olarak yerlestirildi , bu entry ye pointer olan sident elimizde
ve bu noktadan itibaren sadece sident kullanılacak */

m = 0;
}
if (pos2 >= 32) {
    transc[crossc[pos2][pos4][1]-72+1] = sident;
}
} /* output hareketde sident değeri yerine koyuldu */

} /* endif operasyon & (direkt sident atama (string sabit ataması) operasyonu
degil */

/* bu noktaya kadar sident elde edildi dışarı giden veya
database'e işlenecek sident hazır. Aşağıda eğer hedef 62
ile 72 arasındaysa database'e işlem söz konusudur */

if (hedef >= 62 && hedef < 72) { /* filref'e bir atama söz konusudur */
    /* bulunan değeri yerine işle */
    sayac = hedef - 62;
    if (tnumind[sayac] != 0) {
        yenirno = tnum[tnumind[sayac]-1]; /* locate edilen nokta */
        lseek(fd[sayac][0],
            tnum[tnumind[sayac]-1] * sizeof(RTTAB), SEEK_SET);
        /* çıkış sahasının rt tablosundan */
    } else { /* tnumind[sayac] == 0 */
        yenirno = tnum[sayac];
        lseek(fd[sayac][0], tnum[sayac] * sizeof(RTTAB), SEEK_SET);
        /* özel durum tnumind[sayac], eğer tnumind[sayac] = 0
        ve hala hedef olarak seçilmiş ise
        yani locate sahasını belirtecek
        şekilde 0 ise , yani eğer locate edilen sahanın kendisi
        değiştiriliyor ise tnumind[sayac] = 0 bulunacaktır
        bu durumda tnumind[sayac]-1 yerine sayac'ın kendisi
        yerleştirilir çünkü kullanılacak record# karşılık
        gelen tnum değeridir */
    }
    if (read(fd[sayac][0], &rt_buf[sayac], sizeof(RTTAB)) == -1)

```

```

printf("problem RTcik\n");
ayris(rt_buf[sayac].rt,&gec,&gec2,&gec3);
/* bu kayıdı ayırıştır */
lgec2=-gec; /* long a çevir - den kurtar */
lseek(fd[sayac][2],lgec2*sizeof(CTTAB),SEEK_SET);
if (read(fd[sayac][2],&ct_buf[sayac],sizeof(CTTAB))==-1)
    printf("\n çıkış ct okumada problem \n");
/* çıkış sahası coll tablosundan sident
    değerini oku */
sident1=ct_buf[sayac].ct1;
if (sident==sident1){;}
/* bir şey yapma, yazılacak yeni değer ile eskisi aynı */
else { /* bu kaydı mevcut "same" linkten çıkart */
    if (gec2==0&&gec3==0) { /*tek olarak mevcut */
        ct_buf[sayac].ct3=0;
        ct_buf[sayac].ct4=0;
        lseek(fd[sayac][2],lgec2*sizeof(CTTAB),SEEK_SET);
        if (write(fd[sayac][2],
            &ct_buf[sayac],sizeof(CTTAB))==-1)
            printf("\n ct yazmada problem \n");
    }else{ /* gec2 ve gec3 den biri veya her ikisi 0 değil*/
        if (gec2==0&&gec3!=0){
            ct_buf[sayac].ct4=gec3;
            lseek(fd[sayac][2],lgec2*sizeof(CTTAB),SEEK_SET);
            if (write(fd[sayac][2],
                &ct_buf[sayac],sizeof(CTTAB))==-1)
                printf("\n ct yazmada problem \n");
            lgec2=gec3;
            lseek(fd[sayac][0],
                lgec2*sizeof(RTTAB),SEEK_SET);
            if (read(fd[sayac][0],
                &rt_buf, sizeof(RTTAB)) == -1)
                printf("problem RTcik\n");
            ayris(rt_buf.rt,&gec1,&gec12,&gec13);
            gec12=0;
            rt_buf.rt=birles(gec1,gec12,gec13);
            lseek(fd[sayac][0],
                lgec2*sizeof(RTTAB),SEEK_SET);
            if (write(fd[sayac][0],
                &rt_buf, sizeof(RTTAB)) == -1)
                printf("problem RTcik\n");
        }
        if (gec3==0&&gec2!=0){
            ct_buf[sayac].ct3=gec2;
            lseek(fd[sayac][2],lgec2*sizeof(CTTAB),SEEK_SET);
            if (write(fd[sayac][2],
                &ct_buf[sayac],sizeof(CTTAB))==-1)
                printf("\n ct yazmada problem \n");
            lgec2=gec2;
            lseek(fd[sayac][0],
                lgec2*sizeof(RTTAB),SEEK_SET);

```



```

if (read(fd[sayac][0],
        &rt_buft, sizeof(RTTAB)) == -1)
    printf("problem RTcik\n");
ayris(rt_buft.rt,&gec1,&gec12,&gec13);
gec13=0;
rt_buft.rt=birles(gec1,gec12,gec13);
lseek(fd[sayac][0],
        lgec2*sizeof(RTTAB),SEEK_SET);
if (write(fd[sayac][0],
        &rt_buft, sizeof(RTTAB)) == -1)
    printf("problem RTcik\n");
}
if (gec2!=0&&gec3!=0){
    /* arada */
    lgec2=gec2;
    lseek(fd[sayac][0],
        lgec2*sizeof(RTTAB),SEEK_SET);
    if (read(fd[sayac][0],
            &rt_buft, sizeof(RTTAB)) == -1)
        printf("problem RTcik\n");
    ayris(rt_buft.rt,&gec1,&gec12,&gec13);
    gec13=gec3;
    rt_buft.rt=birles(gec1,gec12,gec13);
    lseek(fd[sayac][0],
        lgec2*sizeof(RTTAB),SEEK_SET);
    if (write(fd[sayac][0],
            &rt_buft, sizeof(RTTAB)) == -1)
        printf("problem RTcik\n");
    lgec3=gec3;
    lseek(fd[sayac][0],
        lgec3*sizeof(RTTAB),SEEK_SET);
    if (read(fd[sayac][0],
            &rt_buft, sizeof(RTTAB)) == -1)
        printf("problem RTcik\n");
    ayris(rt_buft.rt,&gec1,&gec12,&gec13);
    gec12=gec2;
    rt_buft.rt=birles(gec1,gec12,gec13);
    lseek(fd[sayac][0],
        lgec3*sizeof(RTTAB),SEEK_SET);
    if (write(fd[sayac][0],
            &rt_buft, sizeof(RTTAB)) == -1)
        printf("problem RTcik\n");
} /* gec2 ve gec3 her ikisi 0 değil */
}/* gec2 veya gec3 veya her ikisi 0 değil */
paket(temp,sident); /* sident pack yapilarak
hash parametresi olarak kullanılmalî */
htp=fd[sayac][1];
ctp=fd[sayac][2];
rtp=fd[sayac][0];
hashval=hash(temp);
lseek(htp,hashval*sizeof(HTTAB),SEEK_SET);

```

```

/* ht fileda git bul */
if (ht_buf[sayac].ht2!=0){
    /*eğer collision bucket tesis edilmiş ise
       yani aynı zamanda bu girdi boş ise */
    lgec=ht_buf[sayac].ht2 ;
    lseek(ctp, lgec*sizeof(CTTAB), SEEK_SET);
    /* collision tablosuna geç*/
    if (read(ctp, &ct_buf[sayac], sizeof(CTTAB)) == -1)
        printf("problem C\n");
    while (ct_buf[sayac].ct2!=0
        &&ct_buf[sayac].ct1!=sident) {
        lgec=ct_buf[sayac].ct2;
        lseek(ctp,lgec*sizeof(CTTAB),SEEK_SET);
        if (read(ctp, &ct_buf[sayac],
            sizeof(CTTAB)) == -1)
            printf("problem CT\n");
    } /* collision bucket içinde bulana kadar veya sonuna
       ulaşp bulamayana kadar ilerledi */
    if (ct_buf[sayac].ct1 == sident){
        /* bucket içinde buldu */
        lgec2=ct_buf[sayac].ct4;
        lseek(fd[sayac][0],
            lgec2*sizeof(RTTAB),SEEK_SET);
        if (read(fd[sayac][0],
            &rt_buf, sizeof(RTTAB)) == -1)
            printf("problem RTcik\n");
        ayris(rt_buf.rt,&gec1,&gec12,&gec13);
        gec12=yenirno;
        rt_buf.rt=birles(gec1,gec12,gec13);
        lseek(fd[sayac][0],
            lgec2*sizeof(RTTAB),SEEK_SET);
        if (write(fd[sayac][0],
            &rt_buf, sizeof(RTTAB)) == -1)
            printf("problem RTcik\n");
        gec2=0; /* forward pointer 0 */
        gec3=ct_buf[sayac].ct4;
        /*backward pointer*/
        gec=-lgec;
        rt_buf.rt=birles(gec,gec2,gec3);
        lseek(fd[sayac][0],
            yenirno*sizeof(RTTAB),SEEK_SET);
        if (write(fd[sayac][0],
            &rt_buf, sizeof(RTTAB)) == -1)
            printf("problem RTcik\n");
        ct_buf[sayac].ct4=gec12;
        lseek(ctp,lgec*sizeof(CTTAB),SEEK_SET);
        if (write(ctp,
            &ct_buf[sayac], sizeof(CTTAB)) == -1)
            printf("problem CT\n");
        /* ct icinde mevcut, same linkte en sona
           eklendi */

```



```

}else {blnmdi=1; /* bucket icinde bulunamadi bir
          adet ct girdisi ekle */
    curpos=lgec ;
    lseek(ctp,0L,SEEK_END);
    /* en sona gidildi */
    yenictno=((
        tell(ctp)-sizeof(CTTAB))/sizeof(CTTAB))+1;
    ct_buf[sayac].ct2=yenictno;
    lseek(ctp,curpos*sizeof(CTTAB),SEEK_SET);
    if (write(ctp, &ct_buf[sayac], sizeof(CTTAB)) ==-1)
        printf("problem cT\n");

    ct_buf[sayac].ct1=sident;
    /* integer id yi bu girdiye koy */
    ct_buf[sayac].ct2=0;
    /* collision linki sonlandir */
    ct_buf[sayac].ct3=yenirno;
    /* rt tabloya pointer'ı yerlestir */
    ct_buf[sayac].ct4=yenirno;
    gecic=-yenictno;
    rt_buf[sayac].rt=birles(gecic,0,0);
    lseek(rtp,yenirno*sizeof(RTTAB),SEEK_SET);
    if (write(rtp, &rt_buf[sayac], sizeof(RTTAB)) ==-1)
        printf("problem RT\n");
    /* ilgili iliski tablosu girdisi gúnlendi */
    lseek(ctp,yenictno*sizeof(CTTAB),SEEK_SET);
    if (write(ctp, &ct_buf[sayac], sizeof(CTTAB)) ==-1)
        printf("problem CT\n");

    } /* else bucket icinde bulunamadığında ct girdi ekleme*/
} /* if ht_buf[sayac].ht2!=0 */
else {blnmdi=1; /*bulunamadi bir adet ct girdisi ekle */;
    lseek(ctp,0L,SEEK_END);
    /* en sona gidildi */
    yenictno=((tell(ctp)-sizeof(CTTAB))/sizeof(CTTAB))+1;
    ht_buf[sayac].ht2=yenictno;
    lseek(htp,hashval*sizeof(HTTAB),SEEK_SET);
    /* ht dosyasında git bul */
    if (write(htp, &ht_buf[sayac], sizeof(HTTAB)) ==-1)
        printf("\n hata HT'a yazma \n");

    ct_buf[sayac].ct1=sident;
    ct_buf[sayac].ct2=0;
    /* collision linki sonlandir */
    ct_buf[sayac].ct3=yenirno;
    /* rt tablosuna pointer'ı yerlestir */
    ct_buf[sayac].ct4=yenirno;
    gecic=-yenictno;
    rt_buf[sayac].rt=birles(gecic,0,0);

```

```

lseek(rtp,yenirno*sizeof(RTTAB),SEEK_SET);
if (write(rtp, &rt_buf[sayac], sizeof(RTTAB)) == -1)
    printf("problem RT\n");
/*ilgili ilişki tablosu girdisi günlendi */
lseek(ctp,yenictno*sizeof(CTTAB),SEEK_SET);
if (write(ctp, &ct_buf[sayac], sizeof(CTTAB)) == -1)
    printf("problem CT\n");
} /*else == ht_buf[sayac].ht2==0 */
} /* else sident!=sident1 */
} /* if hedef>=62 */
} /* if hedef>=62 && hedef<102 */
q+=1;
} /*while inst[q]*/

```

/* Komutların bu hareket için icrası bitti. şimdi

bu işlemler sonunda transv ye yapılan atamalardan yararlanarak

çıkış hareketi yaratılacak ve çıkış kuyruğuna yazılacak

yine eğer filref yapısına atama yapılmışsa bu değerler fd yapısındaki

file descriptionlar ile belirlenen saha dosyalarının(r) tnum

yapisında belirtilen pozisyonlarına yeni değer olarak işlenecektir.

bu numaraya sahip kayıt, mevcut same linkten çıkartılacak (same linkteki bir

önceki ve bir sonraki kayıtların forward/backward pointer'ları

buna göre ayarlanacak, eğer önceki ve sonraki kayıt yoksa, ct yapısındaki

baş ve sona pointer'lar ayarlanacak/sıfırlanacak ve ct yapısına

yeni sident değerini içeren kayıt (yok ise) eklendikten sonra

bu kaydın same linkine, .r da bu pozisyondaki kayıt linklenecektir. */

```
transyaz(tlc,adrp2,transc,semid,&p,&v,adrp->tablo[pno][3]);
```

```
transoku(tlg,adrp2,trans,semid1,&p,&v,adrp->tablo[pno][2]);
```

```
sident=trans[1];
```

```
} /* while trcv!=-1 */
```

```
transc[1]=-1;
```

```
transyaz(tlc,adrp2,transc,semid,&p,&v,adrp->tablo[pno][3]);
```

```
adrp->procact[pno]=0;
```

```
adrp->tablo[pno][0]=0;
```

```
for (i=0;i<giratr;i++){
```

```
    close( fd[i][0] );
```

```
    close( fd[i][1] );
```

```
    close( fd[i][2] );
```

```
}
```

```
semop(adrp->simtan[0],&p,1);
```

```
adrp->compufbit[adrp->tablo[pno][5]]=0;
```

```
semop(adrp->simtan[0],&v,1);
```

```
semop(adrp->simtan[1],&p,1);
```

```
adrp->tablo[pno][0]=0;
```

```
adrp->procact[pno]=0;
```

```
semop(adrp->simtan[1],&v,1);
```

```
semop(adrp->simtan[2],&p,1);
```

```
adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
```

```
adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
```

```
semop(adrp->simtan[2],&v,1);
```



```

} /* sonsuz dongu sonu */
}

```

```

/* COMPUTE SANAL İŞLEMCİSİ KAYNAK PROGRAMI */

```

```

#include <derle.h>

```

```

main(){
  GHTTAB *ght_buf,*ght_buf2;
  GCTTAB *gct_buf,*gct_buf2;
  CTTAB ct_buf[10];
  RTTAB rt_buf[10],rt_buf2;
  HTTAB ht_buf[10];
  FILE *kp;
  char dummy,a,mesaj1[2],mesaj2[2],mesaj3[1],mesaj4[2],temp1[10],temp2[10];
  char temp[100],dum[2],mesaj[2],mesajh[2],rname[20],atname[20],relname[20];
  char rtflen[20],htflen[20],ctflen[20];
  char *r=".r",*h=".h",*c=".c";
  int inst[100],bas,q,i,gcp,gtp,trcv,trov,r1,r2,fdsay;
  int tlg,tlc,fd[10][3],htp,ctp,rtp,ltsay,sayac,deleteon;
  long lt[10];
  float cc,temps[100],filref[10],transv[32],transcf[100];
  long lgec,lgec2,lgec3,tnum[10],sident,sident1,sident2;
  long yenirno,curpos,yenictno;
  int tnumind[10],gecic,gec,gec2,gec3,gec1,gec12,gec13,blnmdi;
  float oper1,oper2;
  double son;
  char crossc[100][100][1];
  char tempbuf[100],tempbuf2[100];
  char *charptr,index[10],operat,hedef;
  int pos1,pos2,pos4,j,k,l,m,b,dec,sign,follow;
  int hashval,bulunam,ind;
  double gecici;
  struct sembuf p={0,-1,SEM_UNDO};
  struct sembuf v={0,1,SEM_UNDO};
  ANAYAPI *adrp;
  KUYRUK *adrp2;
  struct q_entry s_entry;
  int queue_id,rlen;
  int shmid,shmid1,shmid2,shmid3,semid,semid1;
  long trans[100],transc[100];
  int pno,giratr,cikatr;
  int instrsay,yer;
  int pnum;
  pid_t pid;
  processorid_t gelen;
  idtype_t p_pid;

```

```
deleteon=0;
```

```
pno=16;
pnum=2;
pid=getpid();
p_pid=getppid();
k=processor_bind(P_PID,pid,pnum,&gelen);
```

```
if((shmid1=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((adrp=(ANAYAPI *)shmat(shmid1,(void *)0,0))==(ANAYAPI *)(-1))
    printf ("\n attach olmadı \n");
```

```
if((shmid=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((adrp2=(KUYRUK *)shmat(shmid,(void *)0,0))==(KUYRUK *)(-1))
    printf ("\n attach kkkkolmadı \n");
```

```
/* global hash tablosu ve global carpisma tablosunu shared
memory olarak al */
```

```
if((shmid2=shmget((key_t)0x35,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((ght_buf=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
    printf ("\n attach olmadı \n");
if((shmid3=shmget((key_t)0x45,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((gct_buf=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
    printf ("\n attach olmadı \n");
```

```
if((shmid2=shmget((key_t)0x55,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((ght_buf2=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
    printf ("\n attach olmadı \n");
if((shmid3=shmget((key_t)0x65,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((gct_buf2=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
    printf ("\n attach olmadı \n");
if ((queue_id= msgget(QKEY,IPC_CREAT|QPERM))==-1)
    perror("msg get failed");
```

```
/* çıkış kuyruğu */
```

```
while(1) { /* sonsuz dongu */
if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)
    perror("msgrcv failed");
while(adrp->procact[pno]!=1)b=b+b;
```



```

yer=adrp->tablo[pno][5];

giratr=adrp->compgas[yer];
cikatr=adrp->compcas[yer];
instrsay=adrp->compksay[yer];
tlg=adrp->comptlg[yer];
tlc=adrp->comptlc[yer];
fdsay=giratr; /* file descriptor tablosu pointer'ını sıfırla */

strcpy(rname,adrp->compuse[yer][0].relname);
for (i=0;i<giratr;i++){
    tnumind[i]=adrp->compuse[yer][i].ind;
    strcpy(atname,adrp->compuse[yer][i].atname);
    strcpy(relname,rname);
    strcat(relname,atname);
    strcpy(rtfile,relname);
    strcpy(htfile,relname);
    strcpy(ctfile,relname);
    strcat(rtfile,r);
    strcat(htfile,h);
    strcat(ctfile,c);

    fd[i][0]=open(rtfile, O_CREAT|O_RDWR);
    fd[i][1]=open(htfile, O_CREAT|O_RDWR);
    fd[i][2]=open(ctfile, O_CREAT|O_RDWR);
}

for (i=0;i<giratr;i++){
    lt[i]=adrp->complt[yer][i];
}

for (i=0;i<=instrsay;i++){
    crossc[adrp->compinst[yer][i][1]]
        [adrp->compinst[yer][i][2]] [0]=adrp->compop[yer][i];

    crossc[adrp->compinst[yer][i][1]]
        [adrp->compinst[yer][i][2]] [1]=adrp->compinst[yer][i][0];
}

for (i=0;i<instrsay;i++){
    inst[i]=adrp->compicrk[yer][i];
}

transoku(tlg,adrp2,trans,semid,&p,&v,adrp->tablo[pno][2]);
sident=trans[1];
while (sident!=-1){
    /* yerleştirme yaptı sıfırlama yapmadı */
    /* bu noktada eğer locate'ler varsa yapılacak */
    if (giratr>0){
        sayac=0;

```

```

while (sayac < giratrs){
    if (lt[sayac] != 0){
        if (lt[sayac] > 0){ /* sident deđeri direk */
            sident = lt[sayac];
        } else{
            sident = trans[lt[sayac]*(-1)];
        }
        paket(temp, sident); /* sident pack yapılarak
                               hash parametresi olarak kullanılmalı */
        htp = fd[sayac][1];
        ctp = fd[sayac][2];
        hashval = hash(temp);
        lseek(htp, hashval*sizeof(HTTAB), SEEK_SET);
        /* ht fileda git bul */
        if (read(htp, &ht_buf[sayac], sizeof(HTTAB)) == -1)
            printf("problem reading HT\n");

        if (ht_buf[sayac].ht2 != 0){
            /*eđer collision bucket tesis edilmiř ise
              yani aynı zamanda bu entry boş ise */
            lgec = ht_buf[sayac].ht2 ;
            lseek(ctp, lgec*sizeof(CTTAB), SEEK_SET);
            /* collision tablosuna geę */
            if (read(ctp, &ct_buf[sayac], sizeof(CTTAB)) == -1)
                printf("problem C\n");
            while (ct_buf[sayac].ct2 != 0 && ct_buf[sayac].ct1 != sident) {
                lgec = ct_buf[sayac].ct2;
                lseek(ctp, lgec*sizeof(CTTAB), SEEK_SET);
                if (read(ctp, &ct_buf[sayac], sizeof(CTTAB)) == -1)
                    printf("problem CT\n");
            } /* collision bucket icinde bulana kadar veya sonuna
              ulařıp bulamayana kadar ilerledi */
            if (ct_buf[sayac].ct1 == sident){ /* bucket icinde buldu */
                tnum[sayac] = ct_buf[sayac].ct3;
            } else tnum[sayac] = 0;
        } /* if ht_buf[sayac].ht2 != 0 */
        else tnum[sayac] = 0;
    } /* if lt[sayac] != 0 */
    sayac += 1;
} /* while sayac <= giratrs */

} /* if giratrs > 0 */

/* komutları sırasıyla al */
q = 0;
while(inst[q] != 0){
    gecici = inst[q]/100;
    pos2 = gecici;
    pos4 = inst[q] - pos2*100;
    /* hareket deđiřkenlerinden ilk mesajda belirtilenleri
    al */

```



```

if(pos2 < 32 && pos2 > 0){
    sident1 = trans[pos2];
    if (sident1 > 0){
        bas = sident1 + 4;
        oper1 = acpaketd(gct_buf->ct,bas);
    }else {
        bas = sident1*(-1) + 4;
        oper1 = acpaketd(gct_buf2->ct,bas);
    }
}

if (pos4 < 32 && pos4 > 0 ){
    sident2 = trans[pos4];
    if (sident2 > 0){
        bas = sident2 + 4;
        oper2 = acpaketd(gct_buf->ct,bas);
    }else {
        bas = sident2*(-1) + 4;
        oper2 = acpaketd(gct_buf2->ct,bas);
    }
}

if (pos2 >= 32 && pos2 < 62){
    oper1 = temps[pos2-32];
}

if (pos4 >= 32 && pos4 < 62){
    oper2 = temps[pos4-32];
}

if (pos2 >= 62 && pos4 < 72){
    sayac = pos2-62;
    if (tnumind[sayac] != 0){
        lseek(fd[sayac][0],
            tnum[tnumind[sayac]-1]*sizeof(RTTAB),SEEK_SET);
        /* çıkış sahası rt tablosundan aynı
        numaralı kayıdı oku */
        if (read(fd[sayac][0], &rt_buf[sayac], sizeof(RTTAB)) == -1)
            printf("problem RTcik\n");
        ayris(rt_buf[sayac].rt,&gec,&gec2,&gec3);
        /* bu kayıdı ayırıştır */
        lgec2 = -gec; /* long a çevir - den kurtar */
        lseek(fd[sayac][2],lgec2*sizeof(CTTAB),SEEK_SET);
        if (read(fd[sayac][2],&ct_buf[sayac],sizeof(CTTAB)) == -1)
            printf("\n çıkış ct okumada problem \n");
        /* çıkış sahası coll tablosundan sident
        değerini oku */
        sident = ct_buf[sayac].ct1;
    } else { /* tnumind[sayac] == 0 */
        if (lt[sayac] > 0){ /* sident değeri direk */
            sident = lt[sayac];
        } else{
            sident = trans[lt[sayac]*(-1)];
        }
    }
}

```

```

    }

/* else */
if (sident > 0){
    bas=sident+4;
    oper1=acpaketd(gct_buf->ct,bas);
}else {
    bas=sident*(-1)+4;
    oper1=acpaketd(gct_buf2->ct,bas);
}

}

if (pos4 >= 62 && pos4 < 72){
    sayac=pos4-62;
    if (tnumind[sayac] != 0){
        lseek(fd[sayac][0],
            tnum[tnumind[sayac]-1]*sizeof(RTTAB),SEEK_SET);
        /* çıkış sahası rt tablosundan aynı
        numaralı kayıdı oku */
        if (read(fd[sayac][0], &rt_buf[sayac], sizeof(RTTAB)) == -1)
            printf("problem RTcik\n");
        ayris(rt_buf[sayac].rt,&gec,&gec2,&gec3);
        /* bu kayıdı ayırıştır */
        lgec2=-gec; /* long a çevir - den kurtar */
        lseek(fd[sayac][2],lgec2*sizeof(CTTAB),SEEK_SET);
        if (read(fd[sayac][2],&ct_buf[sayac],sizeof(CTTAB)) == -1)
            printf("\n çıkış ct okumada problem \n");
        /* çıkış sahası coll tablosundan sident
        değerini oku */
        sident=ct_buf[sayac].ct1;
    } else { /* tnumind[sayac] == 0 */
        if (lt[sayac] > 0){ /* sident değeri direkt */
            sident=lt[sayac];
        } else{
            sident=trans[lt[sayac]*(-1)];
        }
    }

/* else */
if (sident > 0){
    bas=sident+4;
    oper2=acpaketd(gct_buf->ct,bas);
}else {
    bas=sident*(-1)+4;
    oper2=acpaketd(gct_buf2->ct,bas);
}

}

operat= crosssc[pos2][pos4][0];
hedef=crosssc[pos2][pos4][1];
if (operat == '*'){
    if(hedef < 32)
        transv[hedef]=oper1*oper2;
}

```



```

    if(hedef >= 32 && hedef < 62){
        temps[hedef-32] = oper1 * oper2;
    }
}
if (operat == '+' ) {
    if(hedef < 32)
        transv[hedef] = oper1 + oper2;
    if(hedef >= 32 && hedef < 62)
        temps[hedef-32] = oper1 + oper2;
}
if (operat == '-') {
    if(hedef < 32)
        transv[hedef] = oper1 - oper2;
    if(hedef >= 32 && hedef < 62)
        temps[hedef-32] = oper1 - oper2;
}
if (operat == '/') {
    if(hedef < 32)
        transv[hedef] = oper1 / oper2;
    if(hedef >= 32 && hedef < 62)
        temps[hedef-32] = oper1 / oper2;
}
if (operat == '!') {
    if(hedef < 32)
        transv[hedef] = oper1 * (-1);
    if(hedef >= 32 && hedef < 62)
        temps[hedef-32] = oper1 * (-1);
}
if (operat == '=') {
    if(hedef < 32)
        transv[hedef] = oper1;
    if(hedef >= 32 && hedef < 62)
        temps[hedef-32] = oper1;
    if(hedef >= 72 && hedef < 102)
        if (pos2 < 32) transc[hedef-72+1] = trans[pos2];
        /* giriş hareketinden çıkışa transfer */
    else
        transcf[hedef-72] = oper1; /* çıkışa verilen floating sonuc */
        /* hesaplanmış değerden çıkışa geçiş */
}
if (operat == '$') {
    if(hedef >= 32 && hedef < 62)
        temps[hedef-32] = adrp -> compsb[yer][pos2-32];
}

if (hedef >= 72 && hedef < 102){
    if(pos2 >= 32) {
        /* hesaplanmış bir değer çıkış hareketine geçişi */
        cc = transcf[crossc[pos2][pos4][1]-72];
        /* eğer dışarı gidiyorsa geçici ght'ye işlenmeli */
        paketd(tempbuf, (char *)&cc);
    }
}

```

```
hashval=hash(tempbuf);
```

```
/* eğer daha önce hiç girilmemiş ise hash tablosuna coll tablosunda
bu yeni verinin yerleşeceği yeri işle ve gidip coll tablosuna bu
değeri yerleştirdikten sonra bu hash girdisinin aynı zamanda
collision bağının ilk düğümü olan bu verinin arkasına 2 byte'lık
collision pointer olarak 00 koy. */
```

```
    j=hashval;
    if (ght_buf->po[hashval]==0){
        sident=altyapi(ght_buf2,gct_buf2,tempbuf);
    }
```

```
/*eğer hash girdisi 0 dan büyük ise daha önce buraya giriş yapılmış */
```

```
    else {
        j=hashval;
        follow=0;
        /* follow traverse esnasında bir kayıt geriden
        gelen pointer'dır */
        l=ght_buf->po[j] ;/* Bu pointer'ı al */
        sident=l;
        bulunam=karsilas(&gct_buf->ct[l+4],tempbuf);
        /*
        coll tablosundaki ilk değer ile karşılaştır */
        while (bulunam && (l!=0)){
            temp[3]=gct_buf->ct[l+3]; /* integer pointer
            temp[2]=gct_buf->ct[l+2]; /* integer pointer
            temp[1]=gct_buf->ct[l+1]; /* integer pointer
            paket aç */

            temp[0]=gct_buf->ct[l];
            follow=l;
            l=acpaket(temp);

            /* pointer'ı paketlenmiş olduğu karakter dizisi
            formundan integer formuna aç ve bu pointer'ı
            takip ederek coll linkte bir sonraki kayıda
            geç */
            sident=l;
            if (l>0){
                bulunam=karsilas(&gct_buf->ct[l+4],tempbuf) ;
            }
        }
        if (l==0&&bulunam){ /* collision bucket'in sonuna
            erişti ama bulamadı*/
            sident=altyapi(ght_buf2,gct_buf2,tempbuf);
        }
    }
```

```
if (bulunam==0) ; /* bulundu sident
doğru değere set edildi*/
```



```
} /* end else == if ght_buf->po[hashval]>0 */
```

/* şimdi girdi ya bulundu ya da yeni olarak yerleştirildi , bu girdiye pointer olan sident elimizde ve bu noktadan itibaren sadece sident kullanılacak */

```
m=0;
transc[crossc[pos2][pos4][1]-72+1]=sident;
/* çıkış hareket biriminde sident değeri yerine koyuldu */
} /* if pos2>=32 yani giriş hareketinden direkt geçiş değil */
} /* endif hedef>=72- yani dışarı gönderilen değer */
q+=1;
} /*while inst[q]*/
/* Komutların bu hareket için icrası bitti. şimdi
bu işlemler sonunda transv ye yapılan atamalardan yararlanarak
output hareket yaratılacak ve çıkış kuyruğuna yazılacak
yine eğer filref yapısına atama yapılmışsa bu degerler fd yapısındaki
file tanıtıcıları ile belirlenen saha dosyalarının(r) tnum
yapısında belirtilen pozisyonlarına yeni değer olarak işlenecektir.
bu numaraya sahip kayıt, mevcut "same linkten" çıkartılacak (same linkteki bir
önceki ve bir sonraki kayıtların forward/backward pointerları
buna göre ayarlanacak eğer önceki ve sonraki kayıt yoksa ct yapısındaki
başa ve sona pointer'lar ayarlanacak/sıfırlanacak ve ct yapısına
yeni sident değerini içeren kayıt (yok ise) eklendikten sonra
bu kaydın "same" linkine, .r da bu pozisyonundaki kayıt bağlanacaktır. */
```

```
transyaz(tlc,adrp2,transc,semid,&p,&v,adrp->tablo[pno][3]);
transoku(tlg,adrp2,trans,semid1,&p,&v,adrp->tablo[pno][2]);
sident=trans[1];
```

```
} /* while trcv!=-1 */
transc[1]=-1;
transyaz(tlc,adrp2,transc,semid,&p,&v,adrp->tablo[pno][3]);
semop(adrp->simtan[0],&p,1);
adrp->compubit[adrp->tablo[pno][5]]=0;
semop(adrp->simtan[0],&v,1);
semop(adrp->simtan[1],&p,1);
adrp->tablo[pno][0]=0;
adrp->procact[pno]=0;
semop(adrp->simtan[1],&v,1);
semop(adrp->simtan[2],&p,1);
adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
semop(adrp->simtan[2],&v,1);
} /* sonsuz dongu sonu */
}
```

```
/* COND SANAL İŞLEMCİSİ KAYNAK KODU */
```

```
#include <derle.h>
```

```
int kars(char *t,char *ilk){
```

```
/* bu fonksiyon karşılaştırma amaçlı kullanılmaktadır */
```

```
int n;
```

```
n=0;
```

```
for (;*t!='\0';t++){
```

```
if (*ilk!=*t)
```

```
n=0;
```

```
if (*ilk>*t){
```

```
return(1);}
```

```
if (*ilk<*t){
```

```
return(2);}
```

```
*ilk++;
```

```
}
```

```
return(n);
```

```
}
```

```
main(){ /*1*/
```

```
/* değişken deklarasyonları */
```

```
struct tms timebuf;
```

```
pid_t pid;
```

```
processorid_t gelen;
```

```
idtype_t p_pid;
```

```
long sident,sidentref,ilk,pnum,ilk,iki;
```

```
struct sembuf p={0,-1,SEM_UNDO};
```

```
struct sembuf v={0,1,SEM_UNDO};
```

```
struct q_entry s_entry;
```

```
int queue_id,rlen;
```

```
ANAYAPI *adrp;
```

```
KUYRUK *adrp2;
```

```
GHTTAB *ght_buf,*ght_buf2;
```

```
GCTTAB *gct_buf,*gct_buf2,*gct_gec,*gct_gec2;
```

```
int i,k,a,dur,say,shmid,shmid1,semid,semid1,shmid2,shmid3;
```

```
long trans[100],reftrans[100];
```

```
int pno,reftransl,inptransl,yer,adrsay,cek[200];
```

```
int sart;
```

```
char tempbuf[256],tempbuf2[256];
```

```
float cc,cci;
```



```

pnum = 1;
pid = getpid();
p_pid = getppid();
k = processor_bind(P_PID, pid, pnum, &gelen);

pno = 55;

if((shmid1 = shmget((key_t)0x15, sizeof(ANAYAPI), 0777 | IPC_CREAT)) == -1)
    printf ("\n get olmadi \n");
if((adrp = (ANAYAPI *)shmat(shmid1, (void *)0, 0)) == (ANAYAPI *)(-1))
    printf ("\n attach olmadı \n");

if((shmid = shmget((key_t)0x25, sizeof(KUYRUK), 0777 | IPC_CREAT)) == -1)
    printf ("\n get olmadi \n");
if((adrp2 = (KUYRUK *)shmat(shmid, (void *)0, 0)) == (KUYRUK *)(-1))
    printf ("\n attach kkkkolmadı \n");

if((shmid = shmget((key_t)0x35, sizeof(GHTTAB), 0777 | IPC_CREAT)) == -1)
    printf ("\n get olmadi \n");
if((ght_buf = (GHTTAB *)shmat(shmid, (void *)0, 0)) == (GHTTAB *)(-1))
    printf ("\n attach olmadı \n");
if((shmid = shmget((key_t)0x45, sizeof(GCTTAB), 0777 | IPC_CREAT)) == -1)
    printf ("\n get olmadi \n");
if((gct_buf = (GCTTAB *)shmat(shmid, (void *)0, 0)) == (GCTTAB *)(-1))
    printf ("\n attach olmadı \n");

if((shmid2 = shmget((key_t)0x55, sizeof(GHTTAB), 0777 | IPC_CREAT)) == -1)
    printf ("\n get olmadi \n");
if((ght_buf2 = (GHTTAB *)shmat(shmid2, (void *)0, 0)) == (GHTTAB *)(-1))
    printf ("\n attach olmadı \n");
if((shmid3 = shmget((key_t)0x65, sizeof(GCTTAB), 0777 | IPC_CREAT)) == -1)
    printf ("\n get olmadi \n");
if((gct_buf2 = (GCTTAB *)shmat(shmid3, (void *)0, 0)) == (GCTTAB *)(-1))
    printf ("\n attach olmadı \n");
if ((queue_id = msgget(QKEY, IPC_CREAT | QPERM)) == -1)
    perror("msg get failed");
/* çıkış kuyruğu */
while(1) { /* sonsuz döngü */
if((rlen = msgrcv(queue_id, &s_entry, 5, (long)pno, MSG_NOERROR)) == -1)
    perror("msgrcv failed");
while(adrp->procact[pno] != 1) a = a + a;

yer = adrp->tablo[pno][5];

reftransl = adrp->cndrefitl[yer];
inptransl = adrp->cndinptl[yer];

transoku(reftransl, adrp2, reftrans, semid1, &p, &v, adrp->cndrefqid[yer]);

```

```

sidentm=reftrans[1];
sidentref=reftrans[adrp->cndrefl[yer]];
transoku(reftransl,adrp2,reftrans,semid1,&p,&v,adrp->cndrefqid[yer]);
/* dummy okuma, kuyruk sonu belirteci -1'i temizliyor */
if (sidentm== -1) { printf ("\n referans data bulunamadi \n");
}else {
    if (sidentref<0) { gct_gec=gct_buf2; sidentref=sidentref*(-1); }
    else { gct_gec=gct_buf;}
    bas=sidentref+4;
    if ( adrp->cndreft[yer]==1){
        cc=acpaketd(gct_gec->ct,bas);
    }else{
        i=0;
        while (gct_gec->ct[bas+i]!='\0'){
            tempbuf[i]=gct_gec->ct[bas+i];
            i++;
        }
        tempbuf[i]='\0';
    }
    transoku(inptransl,adrp2,trans,semid1,&p,&v,adrp->cndinpqid[yer]);
    sidentm=trans[1];
    sident=trans[adrp->cndinpl[yer]];
    if (sident!= -1){
        if (sident<0) { gct_gec2=gct_buf2; sident=sident*(-1); }
        else { gct_gec2=gct_buf;}
    }
    sart=0;
    while (sidentm!= -1){
        if ( adrp->cndreft[yer]==1){
            bas=sident+4;
            if (adrp->cndinpt[yer]==1){
                cci=acpaketd(gct_gec2->ct,bas);
                if (adrp->cndsart[yer]=='e'){
                    if (cci==cc) sart=1;
                }
                if (adrp->cndsart[yer]=='n'){
                    if (cci!=cc) sart=1;
                }
                if (adrp->cndsart[yer]=='<'){
                    if (cci<cc) sart=1;
                }
                if (adrp->cndsart[yer]=='>'){
                    if (cci<cc) sart=1;
                }
                if (adrp->cndsart[yer]=='b'){
                    if (cci>=cc) sart=1;
                }
                if (adrp->cndsart[yer]=='k'){
                    if (cci<=cc) sart=1;
                }
            } else {

```



```

printf ("\n hata: condblok: karsilastirmada tip uyusmuyor \n");
}
}else{ /* tip sayisal deęil karakter tipi */
    if (adrp->cndinpt[yer]==0){
        i=0;
        bas=sident+4;
        while (gct_gec2->ct[bas+i]!='\0'){
            tempbuf2[i]=gct_gec2->ct[bas+i];
            i++;
        }
        tempbuf2[i]='\0';

        if (adrp->cndsart[yer]=='e'){
            if (kars(tempbuf,tempbuf2)==0) sart=1;
        }

        if (adrp->cndsart[yer]=='n'){
            if (kars(tempbuf,tempbuf2)!=0) sart=1;
        }
        if (adrp->cndsart[yer]=='<'){
            if (kars(tempbuf,tempbuf2)==1) sart=1;
        }
        if (adrp->cndsart[yer]=='>'){
            if (kars(tempbuf,tempbuf2)==2) sart=1;
        }
        if (adrp->cndsart[yer]=='b'){
            if ((kars(tempbuf,tempbuf2)==2)||
                (kars(tempbuf,tempbuf2)==0)) sart=1;
        }
        if (adrp->cndsart[yer]=='k'){
            if ((kars(tempbuf,tempbuf2)==1)||
                (kars(tempbuf,tempbuf2)==0)) sart=1;
        }
    } else {
        printf ("\n hata: condblok: karsilastirmada tip uyusmuyor \n");
    }
} /* else string */
if (sart){
    transyaz(inptransl,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);
    sart=0;
}
transoku(inptransl,adrp2,trans,semid1,&p,&v,adrp->cndinppid[yer]);
sidentm=trans[1];
sident=trans[adrp->cndinpl[yer]];
if (sident!=-1){
    if (sident<0) { gct_gec2=gct_buf2; sident=sident*(-1); }
    else { gct_gec2=gct_buf;}
}
} /* while */
trans[1]=-1;
transyaz(inptransl,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);

```

```
 } /* else referans data bulundu */
```

```
 semop(adrp->simtan[0],&p,1);
 adrp->cndubit[adrp->tablo[pno][5]]=0;
 semop(adrp->simtan[0],&v,1);
 semop(adrp->simtan[1],&p,1);
 adrp->tablo[pno][0]=0;
 adrp->procact[pno]=0;
 semop(adrp->simtan[1],&v,1);
 semop(adrp->simtan[2],&p,1);
 adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
 adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
 semop(adrp->simtan[2],&v,1);
 } /* sonsuz döngü sonu */
```

```
 } /* main */
```

```
 /* DELETE SANAL İŞLEMCİSİ KAYNAK KODU */
```

```
 #include <derle.h>
```

```
 main() { /*1*/
 RTTAB rt_buf[256];
 HTTAB ht_buf[256];
 CTTAB ct_buf[256];
 FBUF f_buf;
 RELINFO rel_buf;
 struct q_entry s_entry;
 int queue_id,rlen;
```

```
 /* değişken deklarasyonları */
```

```
 char atrname[50],relname[50];
 char htflen[256][12],rtflen[256][12],temp[2],dum[2],ctflen[256][12];
 char in_fifo[50];
 char relatnm[50];
 int delmat,ok,fp,fifop,fifop2;
 int sayac,sayac2,sayac3,i,k,l,j,r,sident,gec,gec2,gec3;
 int ctp[256],rtp[256],htp[256];
 int cevap,gecici,gecici1,gecici2,gecici3;
 long curpos2,curpos,hashval;
 long pos,son,lgec2,lgec,lgec3;
 int refers,maxeqc,mastfld,mastfp,eq[256],eqval[256];
```

```
 struct sembuf p={0,-1,SEM_UNDO};
 struct sembuf v={0,1,SEM_UNDO};
 ANAYAPI *adrp;
 KUYRUK *adrp2;
 int shmid,shmid1,semid,semid1;
 long trans[100];
```



```

int a,pnum,pno,giratr,instrsay,yer;
pid_t pid;
processorid_t gelen;
idtype_t p_pid;

```

```

pno=49;
pnum=2;
pid=getpid();
p_pid=getppid();
k=processor_bind(P_PID,pid,pnum,&gelen);

```

```

if((shmid1=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((adrp=(ANAYAPI *)shmat(shmid1,(void *)0,0))==(ANAYAPI *)(-1))
    printf ("\n attach olmadı \n");

```

```

if((shmid=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((adrp2=(KUYRUK *)shmat(shmid,(void *)0,0))==(KUYRUK *)(-1))
    printf ("\n attach kkkkolmadı \n");

```

```

if ((queue_id= msgget(QKEY,IPC_CREAT|QPERM))==-1)
    perror("msg get failed");

```

```

/* çıkış kuyruğu */

```

```

while(1) { /* sonsuz döngü */
if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)
    perror("msgrcv failed");
while(adrp->procact[pno]!=1)a=a+a;
}

```

```

/* maxeqc maksimum eşitlik şart sayısı
mastfld tuple aramada master saha numarası
"eq" endirekt veri yapısı içinde match için kullanılan
sahaların sıra numarasını taşır
"eqval" ise match için kullanılan değerlerin
sident kartşılıklarının depolandığı veri yapısıdır */

```

```

/* mesajları al */

```

```

yer = adrp->tablo[pno][5];
giratr=adrp->delat[yer];
delmat=adrp->delmat[yer];

```

```

strcpy(relatnm,adrp->delent[yer][0].relname);
fp=open("relinfo",O_CREAT| O_RDWR );
lseek(fp,0L,SEEK_END);

```

```

son=(tell(fp)-sizeof(RELINFO))/sizeof(RELINFO);
pos=0;
lseek(fp,pos*sizeof(RELINFO),SEEK_SET);
while ((pos <= son)&&(!(karsilas(rel_buf.relname,relatnm) == 2))){
    if (read(fp,&rel_buf,sizeof(RELINFO)) == -1)
        printf (" read failed in relinfo\n");
    pos += 1;
    lseek(fp,pos*sizeof(RELINFO),SEEK_SET);
}
if (pos > son&&(!(karsilas(rel_buf.relname,relatnm) == 2))){
    printf("\n calisma zamani hatasi \n");
    exit(0);
}
close(fp);

```

```

k=0;
mastfld=-1;
for (j=0;j < delmat;j++){
    strcpy(atrname, adrp->delent[yer][j].atrname);
    for (i=0;i <= rel_buf.atrsayi;i++){
        if (karsilas(rel_buf.atrname[i],atrname) == 2){
            eqval[k]=adrp->delent[yer][j].ind;
            if (mastfld < 0){ mastfld=i; mastfp=k;}
            eq[k]=i;
            k+=1;
        } /* if */
    } /*for i */
    if (mastfld == -1){
        printf("\n Calisma zamani hatasi, database de programda \n");
        printf("\n belirtilen attribute yok \n");
        exit(-1);
    }
} /* for j */
maxeqc=k;

```

```

/* giriş ve çıkış saha dosya isimlerini yarat */
for (i=0;i <= rel_buf.atrsayi;i++){
    htfile[i][0]='\0';
    ctfile[i][0]='\0';
    rtfile[i][0]='\0';
}
for (i=0;i <= rel_buf.atrsayi;i++){
    strcat(htfile[i],rel_buf.relname);
    strcat(ctfile[i],rel_buf.relname);
    strcat(rtfile[i],rel_buf.relname);
}
for (i=0;i <= rel_buf.atrsayi;i++){
    strcat(htfile[i],rel_buf.atrname[i]);
    strcat(htfile[i],".h");
}

```



```

    strcat(ctfilen[i],rel_buf.atrname[i]);
    strcat(ctfilen[i],".c");
    strcat(rtfilen[i],rel_buf.atrname[i]);
    strcat(rtfilen[i],".r");

}
for (i=0;i<=rel_buf.atrsayi;i++){
    rtp[i]=open(rtfilen[i], O_CREAT|O_RDWR);
    htp[i]=open(htfilen[i], O_CREAT |O_RDWR);
    ctp[i]=open(ctfilen[i], O_CREAT |O_RDWR);
}

/* kuyruk yapısını aç */

transoku(giratr,adrp2,trans,semid1,&p,&v,adrp->tablo[pno][2]);
sident=trans[1];
i=mastfld;
while(sident!=-1){
    if (eqval[mastfp]<0) sident=trans[eqval[mastfp]*(-1)];
    else sident=eqval[mastfp];
    /* eğer master saha için değer hareket indeksi
    olarak verilmiş ise (>0) okunan hareketin
    içinden sident değerini al, eğer direk verilmiş
    ise (<=0) bunun - tersini alıp sident olarak
    kullan */
    paket(temp,sident);
    r=hash(temp);
    lgec=r;
    /* kuyruktan okunan sident değeri ile hash tablosuna
    erişilecek */
    lseek(htp[i],lgec*sizeof(HTTAB),SEEK_SET);

    if (read(htp[i],&ht_buf[i],sizeof(HTTAB))===-1)
        printf("\n hash tablosu okumada hata \n");

    if (ht_buf[i].ht2!=0){ /*3 eğer collision bucket tesis edilmiş ise
        yani aynı zamanda bu girdi boş ise */
        lgec=ht_buf[i].ht2 ;
        lseek(ctp[i], lgec*sizeof(CTTAB), SEEK_SET);
        /* collision tablosuna geç */
        if (read(ctp[i], &ct_buf[i], sizeof(CTTAB)) == -1)
            printf("problem CT\n");
        while (ct_buf[i].ct2!=0&&ct_buf[i].ct1!=sident) { /*4 collision
            tablosunda bulununcaya kadar veya collision tablosunun
            sonuna varılıncaya kadar ilerle */
            lgec=ct_buf[i].ct2;
            lseek(ctp[i],lgec*sizeof(CTTAB),SEEK_SET);
            if (read(ctp[i], &ct_buf[i], sizeof(CTTAB)) == -1)
                printf("problem CT\n");
        } /*4 collision bucket içinde bulana kadar veya sonuna

```

ulaşıp bulamayana kadar ilerledi */

```

if (ct_buf[i].ct1 == sident){ /*4bucket içinde buldu */
    curpos = tell(ctp[i]);
    curpos = (curpos - sizeof(CTTAB))/sizeof(CTTAB);
    lgec = ct_buf[i].ct3;
    /* giriş sahasının collision tablosundan
    rt deki same linke atlama yapıldı */
    while(lgec != 0){ /* 5*/
        lseek(rtp[i], lgec * sizeof(RTTAB), SEEK_SET);
        /* rt tablosunda same linkte en baştaki düğüme ulaş */
        if (read(rtp[i], &rt_buf[i], sizeof(RTTAB)) == -1)
            printf("problem RT\n");
        ayris(rt_buf[i].rt, &gecici, &gecici2, &gecici3);
        /* ayrıştır */
        ok = 1;
        for (j = 0; j < maxeqc; j++){
            if (eq[j] != i){
                lseek(rtp[eq[j]], lgec * sizeof(RTTAB), SEEK_SET);
                /* çıkış sahasının rt tablosundan aynı
                numaralı kaydı oku */
                if (read(rtp[eq[j]],
                    &rt_buf[eq[j]], sizeof(RTTAB)) == -1)
                    printf("problem RTcik\n");
                ayris(rt_buf[eq[j]].rt, &gec, &gec2, &gec3);
                /* bu kaydı da ayrıştır */
                lgec2 = -gec; /* long a çevir - den kurtar */
                lseek(ctp[eq[j]], lgec2 * sizeof(CTTAB), SEEK_SET);
                if (read(ctp[eq[j]],
                    &ct_buf[eq[j]], sizeof(CTTAB)) == -1)
                    printf("\n çıkış ct okumada problem \n");
                /* çıkış sahası coll tablosundan sident
                değerini oku */
                if (eqval[j] < 0) /* hareket index */
                    refers = trans[eqval[j] * (-1)];
                else
                    refers = eqval[j];
                if (ct_buf[eq[j]].ct1 != refers){
                    ok = 0;
                }
            } /* if eq[j] != i */
        } /* for j */
        if (ok){ /* eğer tam bir uyum var ise bu tuple
        iptal edilecek */
            for (k = 0; k <= rel_buf.atrsayı; k++){
                lseek(rtp[k], lgec * sizeof(RTTAB), SEEK_SET);
                /* çıkış sahasının rt tablosundan aynı
                numaralı kaydı oku */
                if (read(rtp[k],
                    &rt_buf[k], sizeof(RTTAB)) == -1)
                    printf("problem RTcik\n");
            }
        }
    }
}

```



```

ayris(rt_buf[k].rt,&gec,&gec2,&gec3);
/* bu kayıdı da ayrıştır */
gec=999; /* long a çevir - den kurtar */
rt_buf[k].rt=birles(gec,gec2,gec3);
lseek(rtp[k],lgec*sizeof(RTTAB),SEEK_SET);
/* çıkış sahasının rt tablosundan aynı
numaralı kaydı oku */
if (write(rtp[k],
&rt_buf[k], sizeof(RTTAB)) == -1)
printf("problem RT\n");
}/* for k */
}/* if ok */
/* same linkte bir ileri hareket et */
lgec=gecici2; /* forward pointer atamasi */
} /*5 while lgec!=0 */
} /*4 if ct_buf[i].ct1 == sident */
} /*3 if ht_buf[i].ht2!=0 */

transyaz(giratr,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);
transoku(giratr,adrp2,trans,semid1,&p,&v,adrp->tablo[pno][2]);
sident=trans[1];
} /* while sident!=-1 */
trans[1]=-1;
transyaz(giratr,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);

adrp->procact[pno]=0;
adrp->tablo[pno][0]=0;
for (i=0;i<=rel_buf.atrsayi;i++){
close(rtp[i]);
close(htp[i]);
close(ctp[i]);
}
semop(adrp->simtan[0],&p,1);
adrp->delubit[adrp->tablo[pno][5]]=0;
semop(adrp->simtan[0],&v,1);
semop(adrp->simtan[1],&p,1);
adrp->tablo[pno][0]=0;
adrp->procact[pno]=0;
semop(adrp->simtan[1],&v,1);
semop(adrp->simtan[2],&p,1);
adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
semop(adrp->simtan[2],&v,1);
} /* sonsuz dongu sonu */

}/* main */

```

/*FILTER SANAL İŞLEMCİSİ KAYNAK PROGRAMI */

```
#include <derle.h>
```

```
main(){ /*1*/
```

```
/* deęişken bildirimleri */
```

```
struct q_entry s_entry;
```

```
int queue_id,rlen;
```

```
int po[5000];
```

```
char ct[20000];
```

```
int ind[10];
```

```
char isim[50],cev[2];
```

```
char temp[2],dum[2];
```

```
char *charptr;
```

```
char tempbuf[256];
```

```
int tempint,fp,ft,a;
```

```
int sayac,sayac2,k,l,j,r,dec,sign,bulunam, follow;
```

```
int gtp,gcp,ctp,rtp,htp,gecici,gecici1,gecici2,gecici3,ident1,ident2,ident3;
```

```
int ident4,ident5;
```

```
long hashval,yenirno,yenictno,lgec,curpos;
```

```
char *tempptr1="\0",*tempptr2="\0";
```

```
double tempflt;
```

```
pid_t pid;
```

```
processorid_t gelen;
```

```
long sident,pnum,ilk,iki;
```

```
struct sembuf p={0,-1,SEM_UNDO};
```

```
struct sembuf v={0,1,SEM_UNDO} ;
```

```
ANAYAPI *adrp;
```

```
KUYRUK *adrp2;
```

```
int i,shmid,shmid1,semid,semid1;
```

```
long trans[100];
```

```
int pno,transl,yer,fltsay;
```

```
struct tms timebuf;
```

```
idtype_t p_pid;
```

```
pnum=2;
```

```
pid=getpid();
```

```
p_pid=getppid();
```

```
k=processor_bind(P_PID,pid,pnum,&gelen);
```

```
pno=40;
```



```
k=1;
```

```
if((shmid1=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((adrp=(ANAYAPI *)shmat(shmid1,(void *)0,0))==(ANAYAPI *)(-1))
    printf ("\n attach olmadı \n");
```

```
if((shmid=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((adrp2=(KUYRUK *)shmat(shmid,(void *)0,0))==(KUYRUK *)(-1))
    printf ("\n attach kkkkolmadı \n");
if ((queue_id= msgget(QKEY,IPC_CREAT|QPERM))==-1)
    perror("msg get failed");
```

```
/* çıkış kuyruğu */
```

```
while(1) { /* sonsuz dongu */
    for (i=0;i<=4999;i++) po[i]=0;
    for (i=0;i<=19999;i++)ct[i]=0;
    if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)
        perror("msgrcv failed");
    while(adrp->procact[pno]!=1)a=a*a;
```

```
    transl=adrp->fltxl[yer];
    fltsay=adrp->fltsay[yer];
    for (i=0;i<fltsay;i++){
        ind[i]=adrp->fltxix[yer][i+1];
    }
    transoku(transl,adrp2,trans,semid1,&p,&v,adrp->tablo[pno][2]);
    sident=trans[1];
```

```
while (sident!=-1){
    *tempptr2='\0';
    for (i=0;i<fltsay;i++){
        paket(tempbuf,trans[ind[i]]);
        strcat(tempptr2,tempbuf);
    }
}
```

```
    hashval=hash(tempptr2);
```

```
/* eğer daha önce hiç girilmemiş ise hash tablosuna coll tablosunda
bu yeni verinin yerleşeceği yeri işle ve gidip coll tablosuna bu
değeri yerleştirdikten sonra bu hash girdisinin aynı zamanda
collision bağının ilk nodu olan bu verinin arkasına 2 byte'lik
collision pointer olarak 00 koy. */
```

```
    j=hashval;
```

```
    if (po[hashval]==0){
```

```
        transyaz(transl,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);
```

```
        /* eğer global hash tablosunun bu girdisi boş ise
        global collision tablosuna girilecek yeni değerin
        pozisyonunu belirleyen pointer (k) değerini
```

```

buraya yerleştir */

po[j]=k;
sident=k;

/* sident çok önemli indeks sakla */
l=0;
ct[k++] = 0;
ct[k++] = 0;
ct[k++] = 0;
ct[k++] = 0;
while (tempbuf[l]!='\0'){
    ct[k++] = tempbuf[l++];
}
ct[k++] = '\0';
/* girilen değer gct'a yüklendi ve sonlandırıldı */
}

/*eğer hash entry 0 dan büyük ise daha önce buraya giriş yapılmış */
else {
    j=hashval;
    follow=0;
    /* follow traverse esnasında bir kayıt geriden
    gelen pointer'dır */
    l=po[j] ;/* Bu pointer'ı al */
    sident=l;
    bulunam = karsilas(&ct[l+4],tempbuf);
    /*
    coll tablosundaki ilk değer ile karşılaştır */
    while (bulunam && (l!=0)){
        temp[3]=ct[l+3];
        temp[2]=ct[l+2];
        temp[1]=ct[l+1]; /* integer pointer
        paket ac */
        temp[0]=ct[l];
        follow=l;
        l=acpaket(temp);

        /* pointer'ı paketlenmiş olduğu karakter dizisi
        formundan integer formuna aç ve bu pointer'ı
        takip ederek coll linkte bir sonraki kayıda
        geç */
        sident=l;
        if (l>0){
            bulunam = karsilas(&ct[l+4],tempbuf) ;
        }
    }

    if (l==0&&bulunam){ /* collision bucket'in sonuna
    erişti ama bulamadı*/

```



```

j=0;

transyaz(transl,adrp2,
        trans,semid,&p,&v,adrp->tablo[pno][3]);
        sident=k;
        ct[k++]=0;
        ct[k++]=0;
        ct[k++]=0;
        ct[k++]=0;
        /* sident çok önemli indeks , sakla */
        while (tempbuf[j]!='\0'){
            ct[k++]=tempbuf[j++];
        }
        ct[k++]='\0';
        /* yeni girdi olarak yerleştirildi */
        paket(temp,sident);
        if (follow>0){
            ct[follow]=temp[0];
            ct[follow+1]=temp[1];
            ct[follow+2]=temp[2];
            ct[follow+3]=temp[3];
            /* bir kayıt geriden gelen pointer'ı
            kullanarak bir önceki kayıda eriş
            ve bu kaydın collision forward
            linkini yeni girdiyi gösterecek
            şekilde güncle */
        }
    }
    if (bulunam==0){
    }

    } /* end else == if po[hashval]>0 */
/* şimdi girdi ya bulundu ya da yeni
olarak yerleştirildi , bu girdiye pointer olan sident elimizde
ve bu noktadan itibaren sadece sident kullanılacak */

transoku(transl,adrp2,trans,semid1,&p,&v,adrp->tablo[pno][2]);
sident=trans[1];
} /* while sident!=-1 */
trans[1]=-1;
transyaz(transl,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);

semop(adrp->simtan[0],&p,1);
adrp->fltubit[adrp->tablo[pno][5]]=0;
semop(adrp->simtan[0],&v,1);
semop(adrp->simtan[1],&p,1);
adrp->tablo[pno][0]=0;
adrp->proctact[pno]=0;
semop(adrp->simtan[1],&v,1);
semop(adrp->simtan[2],&p,1);

```

```

adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
semop(adrp->simtan[2],&v,1);
} /* sonsuz döngü sonu */
}/* main */

/* GENERATE SANAL İŞLEMCİSİ KAYNAK KODU */

#include <derle.h>

main(){
CTTAB ct_buf;

struct sembuf p={0,-1,SEM_UNDO};
struct sembuf v={0,1,SEM_UNDO} ;
struct q_entry s_entry;
int queue_id,rlen;
char out_fifo[50],relname[50],atname[50],ctfilen[50],out_fk[50];
char dum[2];
int i,j,k,sayac,sayac2,ctp,fifop,a;
long lgec,son,pnum,ilk,iki;
pid_t pid;
processorid_t gelen;
idtype_t p_pid;
struct tms timebuf;
ANAYAPI *adrp;
KUYRUK *adrp2;
int semid,shmid;
long trans[100];
int pno;

pnum=2;
pno=1;
pid=getpid();
p_pid=getppid();
k=processor_bind(P_PID,pid,pnum,&gelen);
/* shared memory allocate et */
if((shmid=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
printf ("\n get olmadı \n");
if((adrp=(ANAYAPI *)shmat(shmid,(void *)0,0))==(ANAYAPI *)(-1))
printf ("\n attach olmadı \n");
if((shmid=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
printf ("\n get olmadı \n");

if((adrp2=(KUYRUK *)shmat(shmid,(void *)0,0))==(KUYRUK *)(-1))
printf ("\n attach olmadı \n");
if ((queue_id= msgget(QKEY,IPC_CREAT|QPERM))==-1)
perror("msg get failed");

while(1){
if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)

```



```

    perror("msgrcv failed");
while(adrp->procact[pno]!=1)a=a+a;

ctfilen[0]='\0';
strcat(ctfilen,adrp->genistb[adrp->tablo[pno][5]][0]);
strcat(ctfilen,adrp->genistb[adrp->tablo[pno][5]][1]);
strcat(ctfilen,".c");
ctp=open(ctfilen, O_CREAT | O_RDWR);
lseek(ctp,0L,SEEK_END);

son=(tell(ctp)-sizeof(CTTAB))/sizeof(CTTAB); /* dosyanın kayıt sayısını bul */
lgec=sizeof(CTTAB);
lseek(ctp,lgec,SEEK_SET);
son-=1; /* ilk kayıt dummy, bunu atladık */
for (sayac=0;sayac<=son;sayac++){
    if( read(ctp,&ct_buf, sizeof(CTTAB))== -1)
        printf ("\n ct dan okuma basarisiz \n ");
    trans[1]=ct_buf.ct1;
    transyaz(1,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);
}
trans[1]=-1;
transyaz(1,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);
close(ctp);

semop(adrp->simtan[0],&p,1);
adrp->genubit[adrp->tablo[pno][5]]=0;
semop(adrp->simtan[0],&v,1);
semop(adrp->simtan[1],&p,1);
adrp->tablo[pno][0]=0;
adrp->procact[pno]=0;
semop(adrp->simtan[1],&v,1);
semop(adrp->simtan[2],&p,1);
adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
semop(adrp->simtan[2],&v,1);
}
}

```

/* GENERATE TUPPLE SANAL İŞLEMCİSİ KAYNAK PROGRAMI */

```
#include <derle.h>
```

```
main(){ /*1*/
```

```
/* değişken bildirimleri */
```

```
RTTAB rt_buf[256],rt_bufc[256];
```

```
HTTAB ht_buf[256];
```

```
CTTAB ct_buf[256],ct_bufc[256];
```

```

FBUF f_buf;
RELINFO rel_buf;
struct q_entry s_entry;
int queue_id,rlen;
struct stat buf;
int retval;
char atrname[50],relname[50];
char htflen[256][12],rtflen[256][12],temp[2],dum[2],ctflen[256][12];
char rtflenl[256][12],ctflenl[256][12];
char in_fifo[50];
char relatnm[50];
int ok,fp,fifop,fifop2;
int sayac,sayac2,sayac3,i,k,l,j,r,sident,gec,gec2,gec3,a;
int ctp[256],rtp[256],htp[256],ctpc[256],rtpc[256];
int cikamaz,cevap,gecici,gecici1,gecici2,gecici3;
long curpos2,curpos,son,hashval;
long lgec2,lgec,lgec3;
int maxeqc,mastfld,eq[256],eqc[256],eqval[256],cikatr,giratr;

pid_t pid;
processorid_t gelen;
long pnum,ilk,iki;
idtype_t p_pid;

struct sembuf p={0,-1,SEM_UNDO};
struct sembuf v={0,1,SEM_UNDO};
ANAYAPI *adrp;
KUYRUK *adrp2;
int shmid,shmid1,semid,semid1;
long trans[100];
int pno;

pno=7;
pnum=2;
pid=getpid();
p_pid=getppid();
k=processor_bind(P_PID,pid,pnum,&gelen);
cikamaz=0;

if((shmid1=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
    printf("\n get olmadi \n");
if((adrp=(ANAYAPI *)shmat(shmid1,(void *)0,0))==(ANAYAPI *)(-1))
    printf("\n attach olmadi \n");

if((shmid=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
    printf("\n get olmadi \n");
if((adrp2=(KUYRUK *)shmat(shmid,(void *)0,0))==(KUYRUK *)(-1))
    printf("\n attach kkkolmad \n");
if ((queue_id= msgget(QKEY,IPC_CREAT|QPERM))==-1)

```



```

perror("msg get failed");

/* çıkış kuyruğu */

/* mesajları al */
while(1) { /* sonsuz döngü */
if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)
    perror("msgrcv failed");
while(adrp->procact[pno]!=1)a=a+a;

cikatr=>gtcas[adrp->tablo[pno][5]];

for (i=0;i<cikatr;i++){
    rtfilenc[i][0]='\0';
    ctfilenc[i][0]='\0';
    strcat(rtfilenc[i],adrp->gtistb[adrp->tablo[pno][5]][0]);
    strcat(rtfilenc[i],adrp->gtistb[adrp->tablo[pno][5]][i+1]);
    strcat(ctfilenc[i],rtfilenc[i]);
    strcat(rtfilenc[i],".r");
    strcat(ctfilenc[i],".c");
    rtpc[i]=open(rtfilenc[i], O_CREAT|O_RDWR);
    ctpc[i]=open(ctfilenc[i], O_CREAT |O_RDWR);
}

son=lseek(rtpc[0],0L,SEEK_END);
retval=fstat(rtpc[0],&buf);
son=buf.st_size;
son=(son-sizeof(RTTAB))/sizeof(RTTAB);

for (lgec=1;lgec<=son;lgec++){
    for (j=0;j<cikatr;j++){
        lseek(rtpc[j],lgec*sizeof(RTTAB),SEEK_SET);
        /* diğer giriş sahasının
        rt tablosundan aynı
        numaralı kaydı oku */
        if (read(rtpc[j],
            &rt_buf[j], sizeof(RTTAB)) == -1)
            printf("problem RT\n");
        ayris(rt_buf[j].rt,&gec,&gec2,&gec3);
        /* bu kaydı da ayırıştır */
        if (gec==999) {cikamaz=1;}
        else {
            lgec2=-gec; /* long a çevir - den kurtar */
            lseek(ctpc[j],lgec2*sizeof(CTTAB),SEEK_SET);
            if (read(ctpc[j],&ct_buf[j],sizeof(CTTAB))==-1)
                printf("\n ct okumada problem \n");
            /* giriş sahası coll tablosundan sident
            değerini oku */
            trans[j+1]=ct_buf[j].ct1;

```

```

        cikamaz=0;
    }
    } /* for j */
/* eğer daha önce iptal edilmiş bir tuple değil ise çıkışa
   yaz toggle'i tekrar eski konumuna al */
if (!cikamaz){
    transyaz(cikatr, adrp2, trans, semid, &p, &v, adrp->tablo[pno][3]);
    cikamaz=0;
}
} /* for lgec */
trans[1]=-1;
transyaz(cikatr, adrp2, trans, semid, &p, &v, adrp->tablo[pno][3]);

adrp->procact[pno]=0;
adrp->tablo[pno][0]=0;
for (i=0; i < cikatr; i++){
    close(rtpc[i]);
    close(ctpc[i]);
}
semop(adrp->simtan[0], &p, 1);
adrp->gtubit[adrp->tablo[pno][5]]=0;
semop(adrp->simtan[0], &v, 1);
semop(adrp->simtan[1], &p, 1);
adrp->tablo[pno][0]=0;
adrp->procact[pno]=0;
semop(adrp->simtan[1], &v, 1);
semop(adrp->simtan[2], &p, 1);
adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
semop(adrp->simtan[2], &v, 1);
} /* sonsuz döngü sonu */
} /* main */

```

/* GENERATE ELEMENT SANAL İŞLEMCİSİ KAYNAK KODU */

```
#include <derle.h>
```

```
main(){
```

```

    GHTTAB *ght_buf, *ght_buf2;
    GCTTAB *gct_buf, *gct_buf2;
    FILE *kp;
    struct q_entry s_entry;
    int queue_id, rlen;
    long lgec2, lgec3, tnum[10], sident, sident1, sident2;
    long yenirno, curpos, yenictno;
    int ii, tnumind[10], gecic, gec, gec2, gec3, gec1, gec12, gec13, blnmdi;
    double son, oper1, oper2;
    char *charptr, index[10], operat, hedef;

```



```

char tempbuf3[256],tempbuf2[256];
float tempf;
char tempbuf[256];
char temp[10];
int transl, numerik, yer, pos1, pos2, pos4, l, m, dec, sign, follow;
int hashval, bulunam, ind;
double gecici;
struct sembuf p={0,-1,SEM_UNDO};
struct sembuf v={0,1,SEM_UNDO};
char relname[50],atrnname[50],ctfiden[50];
char dum[2];
int devam,i,j,k,sayac,sayac2,ctp,fifop,a;
long lgec,pnum,ilk,iki;
pid_t pid;
idtype_t p_pid;
processorid_t gelen;
struct tms timebuf;
ANAYAPI *adrp;
KUYRUK *adrp2;
int semid,shmid,shmid1,shmid2,shmid3;
long trans[100];
int pno;
char deva[2],girsin[1];
int iii;
/* k gctable son girdisine gösterir */

pno=4;
pnum=2;
pid=getpid();
p_pid=getppid();
k=processor_bind(P_PID,pid,pnum,&gelen);
/* shared memory allocate et */
if((shmid=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
    printf("\n get olmadı \n");
if((adrp=(ANAYAPI *)shmat(shmid,(void *)0,0))==(ANAYAPI *)(-1))
    printf("\n attach olmadı \n");
if((shmid1=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
    printf("\n get olmadı \n");
if((adrp2=(KUYRUK *)shmat(shmid1,(void *)0,0))==(KUYRUK *)(-1))
    printf("\n attach olmadı \n");

/* global hash tablosu ve global carpisma tablosunu shared
memory olarak al */

if((shmid2=shmget((key_t)0x35,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
    printf("\n get olmadı \n");
if((ght_buf=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
    printf("\n attach olmadı \n");
if((shmid3=shmget((key_t)0x45,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
    printf("\n get olmadı \n");

```

```

if((gct_buf=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
    printf ("\n attach olmadı \n");

if((shmid2=shmget((key_t)0x55,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((ght_buf2=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
    printf ("\n attach olmadı \n");
if((shmid3=shmget((key_t)0x65,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((gct_buf2=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
    printf ("\n attach olmadı \n");
if ((queue_id= msgget(QKEY,IPC_CREAT|QPERM))==-1)
    perror("msg get failed");

while(1){
if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)
    perror("msgrcv failed");
while(adrp->procact[pno]!=1);
yer=adrp->tablo[pno][5];
semid=adrp->simtan[4];

transl=adrp->inputsbt[yer];
for (iii=1;iii<=transl;iii++){
    numerik=adrp->inputsbent[yer][iii];
    if (numerik) {
        tempf=adrp->inputsbf[yer][iii];
        paketd(tempbuf,(char *)&tempf);
    }
    else /* string */
        strcpy(tempbuf,adrp->inputsbst[yer][iii]);
    hashval=hash(tempbuf);
    /* eğer daha önce girilmemiş ise hash tablosuna coll tablosunda
    bu yeni verinin yerleşeceği yeri işle ve gidip coll tablosuna bu
    değeri yerleştirdikten sonra bu hash girdisinin aynı zamanda
    collision bağının ilk düğümü olan bu verinin arkasına 2 byte'lik
    collision pointer olarak 00 koy. */
    j=hashval;
    semop(semid,&p,1); /* global tablolara exclusive erişim */
    if (ght_buf->po[hashval]==0){
        sident=alty(i,yer,adrp,ght_buf2,gct_buf2,tempbuf,trans,numerik);
        trans[iii]=sident;
    }

    /*eğer hash girdisi 0 dan büyük ise daha önce buraya giriş yapılmış */
    else {
        j=hashval;
        follow=0;
        /* follow traverse esnasında bir kayıt geriden
        gelen pointer'dır */
        l=ght_buf->po[j] ;/* Bu pointer'ı al */

```



```

sident=1;
bulunam=karsilas(&gct_buf->ct[l+4],tempbuf);
/*
    coll tablosundaki ilk deęer ile karřılařtır */
while (bulunam && (l!=0)){
    for(ii=0;ii<=3;ii++){
        temp[ii]=gct_buf->ct[l+ii];
    }
    follow=1;
    l=acpaket(temp);

    /* pointer'ı paketlenmiř olduęu karakter dizisi
    formundan integer formuna a ve bu pointer'ı
    takip ederek coll linkte bir sonraki kayıda
    ge */
    sident=1;
    if (l>0){
        bulunam=karsilas(&gct_buf->ct[l+4],tempbuf) ;
    }

    if (l==0&&bulunam){ /* collision bucket'in sonuna
        eriřti ama bulamadı*/
        sident=alty(i,yer,adrp,
            ght_buf2,gct_buf2,tempbuf,trans,numeric);
        trans[iii]=sident;
    }

    if (bulunam==0) /* bulundu sident
        dogru deęere set edildi*/
        trans[iii]=sident;
    } /* end else == if ght_buf->po[hashval]>0 */
/* simdi girdi ya bulundu ya da yeni
olarak yerleřtirildi , bu girdiye pointer olan sident elimizde
ve bu noktadan itibaren sadece sident kullanılacak */
semop(semid,&v,1);
} /* end for iii */
transyaz(transl,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);
trans[1]=-1;
transyaz(transl,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);
close(ctp);

adrp->proact[pno]=0;
semop(adrp->simtan[1],&p,1);
adrp->tablo[pno][0]=0;
semop(adrp->simtan[1],&v,1);
semop(adrp->simtan[0],&p,1);
adrp->inpusbbit[adrp->tablo[pno][5]]=0;
semop(adrp->simtan[0],&v,1);

```

```

semop(adrp->simtan[2],&p,1);
adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
semop(adrp->simtan[2],&v,1);

} /* while 1 */
}

/* INPUT SANAL İŞLEMCİSİ KAYNAK PROGRAMI */

#include <derle.h>

main(){
    GHTTAB *ght_buf,*ght_buf2;
    GCTTAB *gct_buf,*gct_buf2;
    FILE *kp;
    long lgec2,lgec3,tnum[10],sident,sident1,sident2;
    long yenirno,curpos,yenictno;
    int ii,tnumind[10],gecic,gec,gec2,gec3,gec1,gec12,gec13,blnmdi;
    double son,oper1,oper2;
    char *charptr,index[10],operat,hedef;

    char tempbuf3[256][256],tempbuf2[256];
    float tempf[256];
    /* 256 ya kadar nümerik
       256 ya kadar karakter dizini sisteme girebilir. */
    char tempbuf[256];
    char temp[10];
    int transl,numerik,yer,pos1,pos2,pos4,l,m,dec,sign,follow;
    int hashval,bulunam,ind;
    double gecici;
    struct sembuf p={0,-1,SEM_UNDO};
    struct sembuf v={0,1,SEM_UNDO} ;
    struct q_entry s_entry;
    int queue_id,rlen,a;
    char relname[50],atrname[50],ctfilen[50];
    char dum[2];
    int devam,i,j,k,sayac,sayac2,ctp,fifop;
    long lgec,pnum,ilk,iki;

    pid_t pid;
    processorid_t gelen;
    idtype_t p_pid;
    struct tms timebuf;
    ANAYAPI *adrp;
    KUYRUK *adrp2;
    int semid,shmid,shmid1,shmid2,shmid3;
    long trans[100];
    int pno;
    char deva[2],girsin[1];

```



```

/* k gctable son girdisini gösterir */
devan=1;
while(devan<10)
{
    pnum=3;
    pid=getpid();
    p_pid=getppid();
    k=processor_bind(P_PID,pid,pnum,&gelen);
    pno=22;
    /* shared memory allocate et */
    if((shmid=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
        printf("\n get olmadı \n");
    if((adrp=(ANAYAPI *)shmat(shmid,(void *)0,0))==(ANAYAPI *)(-1))
        printf("\n attach olmadı \n");
    if((shmid1=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
        printf("\n get olmadı \n");
    if((adrp2=(KUYRUK *)shmat(shmid1,(void *)0,0))==(KUYRUK *)(-1))
        printf("\n attach olmadı \n");

    /* global hash tablosu ve global carpışma tablosunu shared
    memory olarak al */
    if((shmid2=shmget((key_t)0x35,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
        printf("\n get olmadı \n");
    if((ght_buf=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
        printf("\n attach olmadı \n");
    if((shmid3=shmget((key_t)0x45,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
        printf("\n get olmadı \n");
    if((gct_buf=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
        printf("\n attach olmadı \n");

    if((shmid2=shmget((key_t)0x55,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
        printf("\n get olmadı \n");
    if((ght_buf2=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
        printf("\n attach olmadı \n");
    if((shmid3=shmget((key_t)0x65,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
        printf("\n get olmadı \n");
    if((gct_buf2=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
        printf("\n attach olmadı \n");
    if((queue_id=msgget(QKEY,IPC_CREAT|QPERM))==-1)
        perror("msgget failed");
    while(1) { /* sonsuz döngü */
        if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)
            perror("msgrcv failed");
        while(adrp->procact[pno]!=1);
        yer=adrp->tablo[pno][5];
        semid=adrp->simtan[4];

        ilk=times(&timebuf);
    }
}

```

```

transl=adrp->inputtl[yer];
devam=1;
while(devam){

for (i=1;i<=adrp->inputmat[yer]+1;i++){

    printf(" %s.....:",adrp->inputfb[yer][i]);
    if (adrp->inputent[yer][3][i]==1) { /* nümerik */
        scanf(" %f",&tempf[i]); }
    else { /* string */
        scanf(" %s",tempbuf3[i]);
    }
}
printf("
");
printf("SISTEME GIRSINMI e/h ? ");
printf("
");
scanf(" %s",girsin);

if (girsin[0]=='e'){

for(i=1;i<=adrp->inputmat[yer]+1;i++){
    numerik=adrp->inputent[yer][3][i];
    if (numerik) /* nümerik */
        paketd(tempbuf,(char *)&tempf[i]);
    else /* string */
        strcpy(tempbuf,tempbuf3[i]);
    hashval=hash(tempbuf);
/* eğer daha önce hiç girilmemiş ise hash tablosuna coll tablosunda
bu yeni verinin yerleşeceği yeri işle ve gidip coll tablosuna bu
değeri yerleştirdikten sonra bu hash girdisinin aynı zamanda
collision bağının ilk nodu olan bu verinin arkasına 2 bytelik
collision pointer olarak 00 koy. */
    j=hashval;
    semop(semid,&p,1); /* global tablolara exclusive erişim */
    if (ght_buf->po[hashval]==0){
        atyapi(i,yer,adrp,ght_buf2,gct_buf2,tempbuf,trans,numerik);
    }
}

/*eğer hash entry 0 dan büyük ise daha önce buraya giriş yapılmış */
else {
    j=hashval;
    follow=0;
    /* follow traverse esnasında bir kayıt geriden
    gelen pointer'dır */
    l=ght_buf->po[j]; /* Bu pointer'ı al */
    sident=1;
    bulunam=karsilas(&gct_buf->ct[l+4],tempbuf);
    /*
        coll tablosundaki ilk değer ile karşılaştırmak */
    while (bulunam && (!=0)){
        for(ii=0;ii<=3;ii++){

```



```

temp[ii]=gct_buf->ct[l+ii];
    }
    follow=1;
    l=acpaket(temp);

    /* pointer'ı paketlenmiş olduğu karakter dizisi
    formundan integer formuna aç ve bu pointer'ı
    takip ederek coll linkte bir sonraki kayıda
    geç */
    sident=1;
    if (l>0){
        bulunam=karsilas(&gct_buf->ct[l+4],tempbuf) ;
    }
}

if (l==0&&bulunam){ /* collision bucket'in sonuna
erişti ama bulamadı*/
    altyapi(i,yer,adrp,ght_buf2,gct_buf2,tempbuf,trans,numerik);
}

if (bulunam==0) /* bulundu sident
dogru değere set edildi*/
    trans[adrp->inputent[yer][2][i]]=sident;
} /* end else == if ght_buf->po[hashval]>0 */
/* şimdi entry ya bulundu ya da yeni
olarak yerleştirildi , bu girdiye pointer olan sident elimizde
ve bu noktadan itibaren sadece sident kullanılacak */
semop(semid,&v,1);
} /* for i= inputmat[yer] */
transyaz(transl,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);
} /* sisteme girsin */
printf(" ");
printf("\n DEVAM e/h ");
scanf("%s",deva);
if (deva[0]=='e') devam=1; else devam=0;
} /* while devam */
trans[1]=-1;
transyaz(transl,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);
close(ctp);
semop(adrp->simtan[0],&p,1);
adrp->inpubit[adrp->tablo[pno][5]]=0;
semop(adrp->simtan[0],&v,1);
semop(adrp->simtan[1],&p,1);
adrp->tablo[pno][0]=0;
adrp->procact[pno]=0;
semop(adrp->simtan[1],&v,1);
semop(adrp->simtan[2],&p,1);
adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
semop(adrp->simtan[2],&v,1);

```

```

} /* Sonsuz döngü */

}/* main */

/* MERGE BLOK SANAL İŞLEMCİSİ KAYNAK PROGRAM KODU */

#include <derle.h>

main(){ /*1*/

/* degisken bildirimleri */

struct tms timebuf;

pid_t pid;
processorid_t gelen;
idtype_t p_pid;
long sident,pnum,ilk,iki;

struct sembuf p={0,-1,SEM_UNDO};
struct sembuf v={0,1,SEM_UNDO} ;
ANAYAPI *adrp;
KUYRUK *adrp2;
struct q_entry s_entry;
int queue_id,rlen;
int i,k,a,dur,say,shmid,shmid1,semid,semid1;
long trans[100];
int pno,transl,yer,adrsay,cek[200];

pno=64;
pnum=3;
pid=getpid();
p_pid=getppid();
k=processor_bind(P_PID,pid,pnum,&gelen);

if((shmid1=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((adrp=(ANAYAPI *)shmat(shmid1,(void *)0,0))==(ANAYAPI *)(-1))
    printf ("\n attach olmadı \n");

if((shmid=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((adrp2=(KUYRUK *)shmat(shmid,(void *)0,0))==(KUYRUK *)(-1))
    printf ("\n attach kkkkolmadı \n");
if ((queue_id= msgget(QKEY,IPC_CREAT|QPERM))==-1)
    perror("msg get failed");

/* çıkış kuyruğu */
while(1) { /* sonsuz döngü */

```



```

if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)
    perror("msgrcv failed");
while(adrp->procact[pno]!=1)a=a+a;

yer=adrp->tablo[pno][5];
transl=adrp->mergetl[yer];
adrsay=adrp->mergeadrs[yer];

dur=0;
say=0;
while (!dur){
    i=0;
    while (i < adrsay){
        if (adrp->mergeqids[yer][i] > 0) {
            transoku(transl,adrp2,trans,semid1,&p,&v,adrp->mergeqids[yer][i]);
            if (trans[1]!=-1)
                transyaz(transl,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);
        }
        sident=trans[1];
        if (sident == -1) {
            adrp->mergeqids[yer][i] = -1;
            say ++;
        }
        i ++;
    }
    if (say == adrsay) dur = 1;
} /* while !dur */

trans[1] = -1;
transyaz(transl,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);

semop(adrp->simtan[0],&p,1);
adrp->applyubit[adrp->tablo[pno][5]] = 0;
semop(adrp->simtan[0],&v,1);
semop(adrp->simtan[1],&p,1);
adrp->tablo[pno][0] = 0;
adrp->procact[pno] = 0;
semop(adrp->simtan[1],&v,1);
semop(adrp->simtan[2],&p,1);
adrp->serbkuy[adrp->tablo[pno][2]][1] = 0;
adrp->serbkuy[adrp->tablo[pno][3]][2] = 0;
semop(adrp->simtan[2],&v,1);
} /* sonsuz döngü sonu */

}/* main */

/* REPORT BLOK SANAL İŞLEMCİSİ KAYNAK PROGRAM KODU */

```

```

#include <derle.h>

main(){

    GHTTAB *ght_buf,*ght_buf2,*ght_gec;
    GCTTAB *gct_buf,*gct_buf2,*gct_gec;
    FILE *kp;
    struct q_entry s_entry;
    int queue_id,rlen;
    long lgec2,lgec3,tnum[10],sident,sident1,sident2;
    long yenirno,curpos,yenictno;
    int tnumind[10],gecic,gec,gec2,gec3,gec1,gec12,gec13,blnmdi;
    double son,oper1,oper2;
    char *charptr,index[10],operat,hedef;
    float cc;
    char tempbuf3[256][256],tempbuf2[256];
    float tempf[256];
    /* 256 ya kadar numerik
       256 ya kadar karakter dizini sisteme girebilir. */
    char tempbuf[256];
    char temp[10];
    int transl,numerik,yer,pos1,pos2,pos4,l,m,dec,sign,follow;
    int hashval,bulunam,ind;
    double gecici;
    struct sembuf p={0,-1,SEM_UNDO};
    struct sembuf v={0,1,SEM_UNDO};
    char relname[50],atrname[50],ctfilen[50];
    char dum[2];
    int devam,i,j,k,sayac,sayac2,ctp,fifop,a;
    long lgec,pnum,ilk,iki;
    pid_t pid;
    processorid_t gelen;
    idtype_t p_pid;
    struct tms timebuf;
    ANAYAPI *adrp;
    KUYRUK *adrp2;
    int semid,shmids,shmids1,shmids2,shmids3;
    long trans[100];
    int pno,ii;
    char deva[2],girsin[1];
    /* k gctable son girdisini gösterir */

    pnum=3;
    pno=25;
    pid=getpid();
    p_pid=getppid();
    k=processor_bind(P_PID,pid,pnum,&gelen);
    printf("\n p=%d \n",pno);
    /* shared memory allocate et */

```



```

if((shmid=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((adrp=(ANAYAPI *)shmat(shmid,(void *)0,0))==(ANAYAPI *)(-1))
    printf ("\n attach olmadı \n");
if((shmid1=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");

if((adrp2=(KUYRUK *)shmat(shmid1,(void *)0,0))==(KUYRUK *)(-1))
    printf ("\n attach olmadı \n");

/* Global hash tablosu ve global çarpışma tablosunu shared
memory olarak al */

if((shmid2=shmget((key_t)0x35,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((ght_buf=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
    printf ("\n attach olmadı \n");
if((shmid3=shmget((key_t)0x45,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((gct_buf=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
    printf ("\n attach olmadı \n");

if((shmid2=shmget((key_t)0x55,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((ght_buf2=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
    printf ("\n attach olmadı \n");
if((shmid3=shmget((key_t)0x65,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((gct_buf2=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
    printf ("\n attach olmadı \n");
if((queue_id=msgget(QKEY,IPC_CREAT|QPERM))==-1)
    perror("msgget failed");

while(1) { /* sonsuz döngü */
if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)
    perror("msgrcv failed");

while(adrp->procact[pno]!=1)a=a+a;
ilk=times(&timebuf);

kp=fopen("cik4","w");
semid=adrp->simtan[4];
yer=adrp->tablo[pno][5];

transl=adrp->outputtl[yer];

transoku(transl,adrp2,trans,semid,&p,&v,adrp->tablo[pno][2]);
devam=1;
while(devam&&trans[1]!=-1){

```

```

for (i=1;i<=adrp->outmat[yer]+1;i++){
    printf("\n %s .....: ",adrp->outputfb[yer][i]);
    sident=trans[adrp->outputent[yer][2][i]];
    if (sident>0){
        sident=trans[adrp->outputent[yer][2][i]]+4;
        gct_gec=gct_buf;
    }
    else {
        sident=(trans[adrp->outputent[yer][2][i]]*(-1)+4;
        gct_gec=gct_buf2;
    }
    ii=sident;
    if (adrp->outputent[yer][3][i]==1) { /* nümerik */
        cc=acpaketd(gct_gec->ct,ii);
        printf("%f\n",cc);
    }
    else { /* string */
        printf("%s\n",&gct_gec->ct[sident]);
    }
}
transoku(transl,adrp2,trans,semid,&p,&v,adrp->tablo[pno][2]);
} /* while devam */
close(ctp);
iki=times(&timebuf);
printf("\n son an = %d \n",iki);
semop(adrp->simtan[0],&p,1);
adrp->outpubit[adrp->tablo[pno][5]]=0;
semop(adrp->simtan[0],&v,1);
semop(adrp->simtan[1],&p,1);
adrp->tablo[pno][0]=0;
adrp->procact[pno]=0;
semop(adrp->simtan[1],&v,1);
semop(adrp->simtan[2],&p,1);
adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
semop(adrp->simtan[2],&v,1);
fclose(kp);
} /* sonsuz döngü sonu */
}

```

/* NOTMEMBER SANAL İŞLEMCİSİ KAYNAK PROGRAM KODU */

```
#include <derle.h>
```

```

main(){ /*1*/
RTTAB rt_buf[256],rt_bufc[256];
HTTAB ht_buf[256];
CTTAB ct_buf[256],ct_bufc[256];
FBUF f_buf;

```



```

RELINFO rel_buf;

/* deęişken bildirimleri */

char atrname[50],relname[50];
char htfile[256][12],rtfile[256][12],temp[2],dum[2],ctfile[256][12];
char rtfilec[256][12],ctfilec[256][12];
char in_fifo[50];
char relatnm[50];
int ok,fp,fifop,fifop2;
int yer,ii,sayac,sayac2,sayac3,i,k,l,j,r,sident,gec,gec2,gec3;
int ctp[256],rtp[256],htp[256],ctpc[256],rtpc[256];
int cevap,gecici,gecici1,gecici2,gecici3;
int memtxix[10][2];
long curpos2,curpos,son,hashval;
long lgec2,lgec,lgec3;
int maxeqc,mastfld,eq[256],eqc[256],eqval[256],cikatr,giratr;
int translc,translg;
struct tms timebuf;
struct q_entry s_entry;
int queue_id,rln;

pid_t pid;
processorid_t gelen;
idtype_t p_pid;
long pnum,ilk,iki;
int ka;

struct sembuf p={0,-1,SEM_UNDO};
struct sembuf v={0,1,SEM_UNDO};
ANAYAPI *adrp;
KUYRUK *adrp2;
int shmid,shmid1,semid,semid1;
long trans[100],transv[100],transc[100];
int pno;

pno=58;
pnum=1;
pid=getpid();
p_pid=getppid();
k=processor_bind(P_PID,pid,pnum,&gelen);

if((shmid1=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
    printf("\n get olmadı \n");
if((adrp=(ANAYAPI *)shmat(shmid1,(void *)0,0))==(ANAYAPI *)(-1))
    printf("\n attach olmadı \n");

if((shmid=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
    printf("\n get olmadı \n");
if((adrp2=(KUYRUK *)shmat(shmid,(void *)0,0))==(KUYRUK *)(-1))
    printf("\n attach kkkkolmadı \n");

```

```

if((queue_id=msgget(QKEY,IPC_CREAT|QPERM))==-1)
    perror("msgget failed");

/* çıkış kuyruğu */

/* maxeqc maksimum eşitlik şart sayısı
mastfld tuple aramada master saha numarası
eq endirekt veri yapısı içinde match için kullanılan
sahaların sıra numarasını taşır
eqval ise match için kullanılan değerlerin
sident karşılıklarının depolandığı veri yapısıdır */

/* mesajları al */
while(1){
if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)
    perror("msgrcv failed");
while(adrp->tablo[pno][0]!=1);

ilk=times(&timebuf);

giratr=adrp->memgas[adrp->tablo[pno][5]];
translg=adrp->memtlg[adrp->tablo[pno][5]];
yer=adrp->tablo[pno][5];

ii=1;
while (adrp->memtxix[yer][ii][1]!=0){
    memtxix[ii][1]=adrp->memtxix[yer][ii][1];
    memtxix[ii][0]=adrp->memtxix[yer][ii][0];
    ii+=1;
}
for (i=0;i<giratr;i++){
    rtflen[i][0]='\0';
    htflen[i][0]='\0';
    ctflen[i][0]='\0';
    strcat(rtflen[i],adrp->memrel[adrp->tablo[pno][5]]);
    strcat(rtflen[i],adrp->memga[adrp->tablo[pno][5]][i]);
    strcat(htflen[i],rtflen[i]);
    strcat(ctflen[i],rtflen[i]);
    strcat(rtflen[i],".r");
    strcat(htflen[i],".h");
    strcat(ctflen[i],".c");
    rtp[i]=open(rtflen[i], O_CREAT|O_RDWR);
    htp[i]=open(htflen[i], O_CREAT|O_RDWR);
    ctp[i]=open(ctflen[i], O_CREAT|O_RDWR);
}

/* buraya kuyruktan okuma bölümü girecek */
transoku(translg,adrp2,transv,semid1,&p,&v,adrp->tablo[pno][2]);
for (ii=0;ii<=translg;ii++){
    if (transv[ii]<0&&transv[ii]!=-1) trans[ii]=transv[ii]*(-1);

```



```

    else trans[ii]=transv[ii];
}
ii=1;
while (memtxix[ii][1]!=0){
    transc[memtxix[ii][1]]=trans[memtxix[ii][0]];
    ii+=1;
}
/* mem blokta çıkgrupta iki noktadan sonra yer alan bypass
saha ve değişkenlerinin çıkış hareketindeki yerlerine taşınması */

sident=trans[1];

/* mem operasyonunda kullanılacak en az bir saha
trans[1] da yer alacak */

i=0;
while(sident!=-1){
    paket(temp,sident);
    r=hash(temp);
    lgec=r;
    /* kuyruktan okunan sident değeri ile hash tablosuna
erişilecek */
    lseek(htp[i],lgec*sizeof(HTTAB),SEEK_SET);

    if (read(htp[i],&ht_buf[i],sizeof(HTTAB))== -1)
        printf("\n hash tablosu okumada hata \n");

    if (ht_buf[i].ht2!=0){ /*3 eğer collision bucket tesis edilmiş ise
yani aynı zamanda bu entry boş ise */
        lgec=ht_buf[i].ht2 ;
        lseek(ctp[i], lgec*sizeof(CTTAB), SEEK_SET);
        /* collision tablosuna geç */
        if (read(ctp[i], &ct_buf[i], sizeof(CTTAB)) == -1)
            printf("problem CT\n");
        while (ct_buf[i].ct2!=0&&ct_buf[i].ct1!=sident) { /*4 collision
tablosunda bulununcaya kadar veya collision tablosunun
sonuna varılincaya kadar ilerle */
            lgec=ct_buf[i].ct2;
            lseek(ctp[i],lgec*sizeof(CTTAB),SEEK_SET);
            if (read(ctp[i], &ct_buf[i], sizeof(CTTAB)) == -1)
                printf("problem CT\n");
        } /*4 collision bucket içinde bulana kadar veya sonuna
ulaşıp bulamayana kadar ilerledi */

        if (ct_buf[i].ct1==sident){ /*4 bucket içinde buldu */
            curpos=tell(ctp[i]);
            curpos=(curpos-sizeof(CTTAB))/sizeof(CTTAB);
            lgec=ct_buf[i].ct3;
            /* giriş sahasının collision tablosundan
rt deki "same linke" atlama yapıldı */
            ok=1;

```

```

while(lgec!=0){/* 5*/
    lseek(rtp[i],lgec*sizeof(RTTAB),SEEK_SET);
    /* rt tablosunda same linkte en baştaki düğüme
    ulaş */
    if (read(rtp[i], &rt_buf[i], sizeof(RTTAB)) == -1)
        printf("problem RT\n");
    ayris(rt_buf[i].rt,&gecici,&gecici2,&gecici3);
    /* ayırıştır */
    /* ok=1; */
    for (j=1;j<giratr;j++){
        lseek(rtp[j],lgec*sizeof(RTTAB),SEEK_SET);
        /* diğer giriş sahasının
        rt tablosundan aynı
        numaralı kaydı oku */
        if (read(rtp[j],
            &rt_buf[j], sizeof(RTTAB)) == -1)
            printf("problem RT\n");
        ayris(rt_buf[j].rt,&gec,&gec2,&gec3);
        /* bu kaydı da ayırıştır */
        lgec2=-gec; /* long a çevir - den kurtar */
        lseek(ctp[j],lgec2*sizeof(CTTAB),SEEK_SET);
        if (read(ctp[j],&ct_buf[j],sizeof(CTTAB)) == -1)
            printf("\n ct okumada problem \n");
        /* giriş sahası coll tablosunda sident
        değerini oku */
        if(ct_buf[j].ct1!=trans[j+1]){
            ok=0; /* uyum bozuldu */
        }
    } /* for j */
    /* same linkte bir ileri hareket et */
    lgec=gecici2; /* forward pointer ataması */
} /*5 while lgec!=0 */
if (ok==0){ /* eğer bir uyum yok ise bu tuple
olduğu gibi çıkışa gidecek */
    transyaz(translg,adrp2,
        trans,semid,&p,&v,adrp->tablo[pno][3]);
    }/* if !ok */
    } /*4 if ct_buf[i].ct1==sident */
} /*3 if ht_buf[i].ht2!=0 */

transoku(translg,adrp2,transv,semid1,&p,&v,adrp->tablo[pno][2]);
for (ii=0;ii<=translg;ii++){
    if (transv[ii]<0&&transv[ii]!=-1) trans[ii]=transv[ii]*(-1);
    else trans[ii]=transv[ii];
}
ii=1;
while (memtxix[ii][1]!=0){
    transc[memtxix[ii][1]]=trans[memtxix[ii][0]];
    ii+=1;
}
/* mem blokta çıkgrupta iki noktadan sonra yer alan bypas

```



```
saha ve deęişkenlerinin çıkış hareketindeki yerlerine taşınması */
sident=trans[1];
} /* while sident!=-1 */
trans[1]=-1;
transyaz(translg,adrp2,trans ,semid,&p,&v,adrp->tablo[pno][3]);
```

```
iki=times(&timebuf);
for (i=0;i < giratrs;i++){
    close(rtp[i]);
    close(htp[i]);
    close(ctp[i]);
}

semop(adrp->simtan[0],&p,1);
adrp->applyubit[adrp->tablo[pno][5]]=0;
semop(adrp->simtan[0],&v,1);
semop(adrp->simtan[1],&p,1);
adrp->tablo[pno][0]=0;
adrp->proctact[pno]=0;
semop(adrp->simtan[1],&v,1);
semop(adrp->simtan[2],&p,1);
adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
semop(adrp->simtan[2],&v,1);
} /* while sonsuz döngü */

} /* main */
```

```
/* SINK BLOK SANAL İŞLEMCİSİ KAYNAK PROGRAM KODU */
```

```
#include <derle.h>

main(){
    sident=trans[1];
    /* deęişken bildirimleri */
    while (sident!=-1){
        transyaz(translg,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);

        struct tms timebuf;
        struct q_entry s_entry;
        int queue_id,rlen,a,k;

        pid_t pid;
        processorid_t gelen;
        idtype_t p_pid;
        long sident,pnum,ilk,iki;

        struct sembuf p={0,-1,SEM_UNDO};
        struct sembuf v={0,1,SEM_UNDO};
        ANAYAPI *adrp;
        KUYRUK *adrp2;
```

```

int i,shmid,shmid1,semid,semid1;
long trans[100];
int pno,transl,yer,adrsay;

pno=37;
pnum=0;
pid=getpid();
p_pid=getppid();
k=processor_bind(P_PID,pid,pnum,&gelen);

if((shmid1=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((adrp=(ANAYAPI *)shmat(shmid1,(void *)0,0))==(ANAYAPI *)(-1))
    printf ("\n attach olmadı \n");

if((shmid=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((adrp2=(KUYRUK *)shmat(shmid,(void *)0,0))==(KUYRUK *)(-1))
    printf ("\n attach kkkkolmadı \n");
if((queue_id=msgget(QKEY,IPC_CREAT|QPERM))==-1)
    perror("msgget failed");

while(1){
    if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)
        perror("msgrcv failed");

    while(adrp->proctact[pno]!=1)a=a+a;

    ilk=times(&timebuf);
    yer=adrp->tablo[pno][5];
    /* buraya kuyruktan okuma bölümü girecek */
    transl=adrp->sinktl[yer];
    transoku(transl,adrp2,trans,semid1,&p,&v,adrp->tablo[pno][2]);
    sident=trans[1];

    while (sident!=-1){
        transoku(transl,adrp2,trans,semid1,&p,&v,adrp->tablo[pno][2]);
        sident=trans[1];
    } /* while sident!=-1 */
    iki=times(&timebuf);

    semop(adrp->simtan[0],&p,1);
    adrp->applyubit[adrp->tablo[pno][5]]=0;
    semop(adrp->simtan[0],&v,1);
    semop(adrp->simtan[1],&p,1);
    adrp->tablo[pno][0]=0;
    adrp->proctact[pno]=0;
    semop(adrp->simtan[1],&v,1);
    semop(adrp->simtan[2],&p,1);
    adrp->serbkuy[adrp->tablo[pno][2]][1]=0;

```



```

adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
semop(adrp->simtan[2],&cv,1);
} /* Sonsuz döngü */

/* main */

```

```

/* SORT BLOK SANAL İŞLEMCİSİ KAYNAK PROGRAM KODU */

```

```

#include <derle.h>

```

```

main(){ /*1*/

```

```

/* değişken bildirimleri */

```

```

struct tms timebuf;

```

```

pid_t pid;

```

```

processorid_t gelen;

```

```

idtype_t p_pid;

```

```

long sident,pnum,ilk,iki;

```

```

struct sembuf pi={0,-1,SEM_UNDO};

```

```

struct sembuf vi={0,1,SEM_UNDO} ;

```

```

ANAYAPI *adrp;

```

```

KUYRUK *adrp2;

```

```

struct q_entry s_entry;

```

```

int queue_id,rlen;

```

```

int i,a,shmid,shmid1,shmid3,shmid2,semid,semid1;

```

```

long trans[100];

```

```

int pno,transl,yer,adrsay;

```

```

float cc;

```

```

char ct[30000];

```

```

GHTTAB *ght_buf,*ght_buf2,*ght_gec;

```

```

GCTTAB *gct_buf,*gct_buf2,*gct_gec;

```

```

char isim[50],cev[2];

```

```

char temp[4],dum[2];

```

```

char temp2[40];

```

```

char htflen[12],rtflen[12],ctflen[12];

```

```

char *charptr;

```

```

char tempbuf[256];

```

```

float cca[256];

```

```

int ccp[256];

```

```

int tempint,fp,ft,sonuc,demir,head,ident[10];

```

```

int numerik,sayac,sayac2,k,l,j,p,r,s,dec,sign,bulunam,follow;

```

```

int ii,mm,gtp,gcp,ctp,rtp,htp,gecici,gecici1,gecici2,gecici3,ident2,ident3;

```

```

int t,enbasta,bas,ident4,ident5,numapd;

```

```

long hashval,yenirno,yenictno,lgec,curpos;
char *tempPtr2="\0";
double tempflt;
int sonuca[256];

```

```

pno=28;
pnum=3;
pid=getpid();
p_pid=getppid();
mm=processor_bind(P_PID,pid,pnum,&gelen);

```

```

if((shmid1=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
    printf("\n get olmadi \n");
if((adrp=(ANAYAPI *)shmat(shmid1,(void *)0,0))==(ANAYAPI *)(-1))
    printf("\n attach olmadı \n");

```

```

if((shmid=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
    printf("\n get olmadi \n");
if((adrp2=(KUYRUK *)shmat(shmid,(void *)0,0))==(KUYRUK *)(-1))
    printf("\n attach kkkkolmadı \n");

```

```

if((shmid=shmget((key_t)0x35,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
    printf("\n get olmadi \n");
if((ght_buf=(GHTTAB *)shmat(shmid,(void *)0,0))==(GHTTAB *)(-1))
    printf("\n attach olmadı \n");
if((shmid=shmget((key_t)0x45,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
    printf("\n get olmadi \n");
if((gct_buf=(GCTTAB *)shmat(shmid,(void *)0,0))==(GCTTAB *)(-1))
    printf("\n attach olmadı \n");

```

```

if((shmid2=shmget((key_t)0x55,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
    printf("\n get olmadi \n");
if((ght_buf2=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
    printf("\n attach olmadı \n");
if((shmid3=shmget((key_t)0x65,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
    printf("\n get olmadi \n");
if((gct_buf2=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
    printf("\n attach olmadı \n");
if((queue_id=msgget(QKEY,IPC_CREAT|QPERM))==-1)
    perror("msg get failed");

```

```

/* çıkış kuyruğu */

```

```

while(1) { /* sonsuz döngü */

```

```

k=1;

```

```

for (i=0;i<=20000;i++){ct[i]='\0';}

```

```

if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)
    perror("msgrcv failed");

```



```

while(adrp->procact[pno]!=1)a=a+a;

yer=adrp->tablo[pno][5];
transl=adrp->sorttl[yer];
numapd=adrp->sorttsay[yer];

transoku(transl,adrp2,trans,semid1,&pi,&vi,adrp->tablo[pno][2]);
sident=trans[1];
/* buya kuyruktan okuma bolumu girecek */
numerik=1;
while (sident!=-1){
    i=0;
    for (s=0;s<=numapd;s++){
        ident[s]=trans[adrp->sorttxix[yer][s]];
        numerik= adrp->sorttyp[yer][s+1];
        if (ident[s]>0){
            bas=ident[s]+4;
            gct_gec=gct_buf;
        }else {
            bas=ident[s]*(-1)+4;
            gct_gec=gct_buf2;
        }
        if (numerik) /* nümerik */ {
            cc=acpaketd(gct_gec->ct,bas);
            cca[s]=cc;
            ccp[s]=s;
        } else { /* character */
            ccp[s]=i;
            while(gct_gec->ct[bas]!='\0'){
                tempbuf[i++] = gct_gec->ct[bas++];
            }
            tempbuf[i++] = '\0';
        }
    }
    enbasta=0;

    if (k==1){
        head=1;
        sident=k;
        p=0;
        ct[k++] = 0;
        ct[k++] = 0;
        ct[k++] = 0;
        ct[k++] = 0;
        for (s=0;s<=numapd;s++){
            paket(temp,ident[s]);
            for(p=0;p<=3;p++){
                ct[k++] = temp[p];
            }
            p=0;
        }
        ct[k++] = '\0';
    }
}

```

```

}
```

```

/*eğer k 1 den büyük ise daha önce buraya giriş yapılmış */
```

```

else {
```

```

    follow=0;
```

```

    /* follow traverse esnasında bir kayıt geriden
```

```

    gelen pointer'dır */
```

```

    l=head ;/* Bu pointer'ı al */
```

```

    sonuc=0;
```

```

    tempptr2=tempbuf;
```

```

    for (s=0;s <= numapd*4;s += 4){
```

```

        temp[0]=ct[l+s+4];
```

```

        temp[1]=ct[l+s+5];
```

```

    temp[2]=ct[l+s+6];
```

```

    temp[3]=ct[l+s+7];
```

```

        ident2=acpaket(temp);
```

```

    if (ident2 < 0) {
```

```

        ident2=ident2*(-1);
```

```

        gct_gec=gct_buf2;
```

```

    }else {
```

```

        gct_gec=gct_buf;
```

```

    }
```

```

    if( adrp->sorttyp[yer][s/4]==1) {
```

```

        cc=acpaketd(gct_gec->ct,ident2+4);
```

```

        if (cc > cca[ccp[s/4]]) sonuc=2;
```

```

        if (cc < cca[ccp[s/4]]) sonuc=1;
```

```

        if (cc == cca[ccp[s/4]]) sonuc=0;
```

```

    }else{
```

```

        sonuc=kars(&gct_gec->ct[ident2+4],
```

```

        &tempptr2[ccp[s/4]]);
```

```

    }
```

```

        sonuca[s/4]=sonuc;
```

```

    }
```

```

    s=0;
```

```

    sonuc=sonuca[s];
```

```

    while(sonuca[s]==0&& s!=numapd){
```

```

        s++;
```

```

        sonuc=sonuca[s];
```

```

    }
```

```

    if (sonuc==2){ /* En baştaki değerden küçük veya eşit */
```

```

        enbasta=1;
```

```

        follow=head;
```

```

    }
```

```

    while (sonuc!=2 && (l!=0)){
```

```

        temp[0]=ct[l];
```

```

        temp[1]=ct[l+1]; /* integer pointer
```

```

        paket aç */
```



```
temp[2]=ct[l+2];
temp[3]=ct[l+3];
follow=1;
```

```
l=acpaket(temp);
```

```
/* pointer'ı paketlenmiş olduğu karakter dizisi
formundan integer formuna aç ve bu pointer'ı
takip ederek coll linkte bir sonraki kayıda
geç */
```

```
if (l>0){
    sonuc=0;
    h=tempptr2=tempptr;
    for (s=0;s<=numapd*4;s+=4){

        temp[0]=ct[l+s+4];
        temp[1]=ct[l+s+5];
        temp[2]=ct[l+s+6];
        temp[3]=ct[l+s+7];
        ident2=acpaket(temp);
        if (ident2<0) {
            ident2=ident2*(-1);
            gct_gec=gct_buf2;
        }else {
            gct_gec=gct_buf;
        }
        if (adrp->sorttyp[yer][s/4]==1) {
            cc=acpaketd(gct_gec->ct,ident2+4);
            if (cc>cca[ccp[s/4]]) sonuc=2;
            if (cc<cca[ccp[s/4]]) sonuc=1;
            if (cc==cca[ccp[s/4]]) sonuc=0;
        }else{
            sonuc=kars(&gct_gec->ct[ident2+4],&tempptr2[ccp[s/4]]);
        }

        sonuca[s/4]=sonuc;

    }
    s=0;
    sonuc=sonuca[s];
    while(sonuca[s]==0&& s!=numapd){
        s++;
        sonuc=sonuca[s];
    }

}

}
if (l==0&&sonuc!=2){ /* collision bucket'in sonuna
erişti ama daha büyük bulamadı
```

```

en sona eklenecek*/
j=0;
sident=k;
ct[k++] = 0;
ct[k++] = 0;
ct[k++] = 0;
ct[k++] = 0;
for (s=0; s<=numapd; s++){
    paket(temp,ident[s]);
    for(j=0; j<=3; j++){
        ct[k++] = temp[j];
    }
    j=0;
    ct[k++] = '\0';
    paket(temp,sident);
    if (follow>0){
        ct[follow] = temp[0];
        ct[follow+1] = temp[1];
        ct[follow+2] = temp[2];
        ct[follow+3] = temp[3];
        /* bir kayıt geriden gelen pointer'ı
        kullanarak bir önceki kayıda eriş
        ve bu kaydın collision forward
        linkini yeni girdiyi gösterecek
        şekilde update et */
    }
}

if (sonuc==2){/* daha büyük veya eşiti bulundu */
    /* bu durumda linklenecek */
    /* bulunan kaydın next linkini
    yeni kayda yönlendir. Yeni kaydın
    next linkini bulunan kaydın next
    linkine eşitle. Böylece yeni kayıt
    linked list de araya sokmuş
    oluyoruz */
    j=0;
    demir=k;
    paket(temp,l);
    ct[k++] = temp[0];
    ct[k++] = temp[1];
    ct[k++] = temp[2];
    ct[k++] = temp[3];
    /* bu kaydın next linki bulunan kaydın next
    linkine eşitlendi */
    for (s=0; s<=numapd; s++){
        paket(temp,ident[s]);
        for (j=0; j<=3; j++){
            ct[k++] = temp[j];
        }
        j=0;
    }
}

```



```

ct[k++] = '\0';
paket(temp, demir);
if (follow > 0) {
    if (enbasta) {
        head = demir;
    } else {
        ct[follow] = temp[0];
        ct[follow + 1] = temp[1];
        ct[follow + 2] = temp[2];
        ct[follow + 3] = temp[3];
    }
}

/* TRANSFER R. (ANAL. KODU) */
#include <dire.h>
}

main() { /* */

/* degışken bildirgeni */

} /* end else === k > 1 */

struct trans {
    struct trans *next;
    int sident;
    int l;
    int demir;
    int temp[4];
    int follow;
    int l;
    int s;
    int ident2;
    int trans[4];
};

transoku(transl, adrp2, trans, semid1, &pi, &vi, adrp->tablo[pno][2]);
sident = trans[1];
} /* while sident != -1 */
l = head;
while (l != 0) {
    demir = l;
    temp[0] = ct[l];
    temp[1] = ct[l + 1]; /* integer pointer
    paket aç */
    temp[2] = ct[l + 2];
    temp[3] = ct[l + 3];
    follow = l;
    l = acpaket(temp);
    for (s = 0; s <= numapd * 4; s += 4) {
        temp[0] = ct[demir + s + 4];
        temp[1] = ct[demir + s + 5];
        temp[2] = ct[demir + s + 6];
        temp[3] = ct[demir + s + 7];
        ident2 = acpaket(temp);
        trans[adrp->sorttxix[yer][s/4]] = ident2;
    }
    transyaz(transl, adrp2, trans, semid, &pi, &vi, adrp->tablo[pno][3]);
}
trans[1] = -1;
transyaz(transl, adrp2, trans, semid, &pi, &vi, adrp->tablo[pno][3]);

semop(adrp->simtan[0], &pi, 1);
adrp->sortubit[adrp->tablo[pno][5]] = 0;
semop(adrp->simtan[0], &vi, 1);
semop(adrp->simtan[1], &pi, 1);
adrp->tablo[pno][0] = 0;
adrp->procact[pno] = 0;

```

```

semop(adrp->simtan[1],&vi,1);
semop(adrp->simtan[2],&pi,1);
adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
semop(adrp->simtan[2],&vi,1);
} /* sonsuz döngü sonu */

}/* main */

/* TRANSFER BLOĞU SANAL İŞLEMCİSİ KAYNAK PROGRAM KODU */

#include <derle.h>

main(){ /*1*/

/* değişken bildirimleri */

struct tms timebuf;
struct q_entry s_entry;
int queue_id,rlen;

pid_t pid;
processorid_t gelen;
idtype_t p_pid;
long sident,pnum,ilk,iki;

struct sembuf p={0,-1,SEM_UNDO};
struct sembuf v={0,1,SEM_UNDO} ;
ANAYAPI *adrp;
KUYRUK *adrp2;
int i,k,a,shmid,shmid1,semid,semid1;
long transl[100];
int pno,transl,yer,adrsay;

pnum=3;
pno=34;
pid=getpid();
p_pid=getppid();
k=processor_bind(P_PID,pid,pnum,&gelen);

if((shmid1=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
    printf ("\n get olmadı \n");
if((adrp=(ANAYAPI *)shmat(shmid1,(void *)0,0))==(ANAYAPI *)(-1))
    printf ("\n attach olmadı \n");

if((shmid=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)

```



```

    printf ("\n get olmadı \n");
if((adrp2=(KUYRUK *)shmat(shmid,(void *)0,0))==(KUYRUK *)(-1))
    printf ("\n attach kkkkolmadı \n");
if ((queue_id= msgget(QKEY,IPC_CREAT|QPERM))==-1)
    perror("msg get failed");

/* çıkış kuyruğu */
while(1) { /* sonsuz döngü */
if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)
    perror("msgrcv failed");
while(adrp->proctact[pno]!=1)a=a+a;

yer=adrp->tablo[pno][5];

transl=adrp->splttl[yer];
adrsay=adrp->spltadrs[yer];
transoku(transl,adrp2,trans,semid1,&p,&v,adrp->tablo[pno][2]);
sident=trans[1];

while (sident!=-1){
    i=0;
    while (i < adrsay){
        transyaz(transl,adrp2,trans,semid,&p,&v,adrp->spltqids[yer][i]);
        i++;
    }
    transoku(transl,adrp2,trans,semid1,&p,&v,adrp->tablo[pno][2]);
    sident=trans[1];
} /* while sident!=-1 */
trans[1]=-1;
i=0;
while (i < adrsay){
    transyaz(transl,adrp2,trans,semid,&p,&v,adrp->spltqids[yer][i]);
    i++;
}

semop(adrp->simtan[0],&p,1);
adrp->splutubit[adrp->tablo[pno][5]]=0;
semop(adrp->simtan[0],&v,1);
semop(adrp->simtan[1],&p,1);
adrp->tablo[pno][0]=0;
adrp->proctact[pno]=0;
semop(adrp->simtan[1],&v,1);
semop(adrp->simtan[2],&p,1);
adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
semop(adrp->simtan[2],&v,1);
} /* sonsuz döngü sonu */

}/* main */

```

```
/* TEST BLOĞU SANAL İŞLEMCİSİ KAYNAK PROGRAM KODU */
```

```
#include <derle.h>
```

```
main(){
GHTTAB *ght_buf,*ght_buf2;
GCTTAB *gct_buf,*gct_buf2,*gct_gec;
CTTAB ct_buf[10];
RTTAB rt_buf[10],rt_buf2;
HTTAB ht_buf[10];
FILE *kp;
struct q_entry s_entry;
int queue_id,rlen;
char dumy,a,mesaj1[2],mesaj2[2],mesaj3[1],mesaj4[2],temp1[10],temp2[10];
char temp[100],dum[2],mesaj[2],mesajh[2],rname[20],atname[20];
char rtflen[20],htflen[20],ctflen[20];
char *r=".r",*h=".h",*c=".c";
int i,translg,translc,inst[100],bas,q,trcv,trov,r1,r2,fdsay;
int fd[10][3],htp,ctp,rtp,ltsay,sayac,deleteon;
long lt[10];
float cc,temps[30],filref[10],transv[32];
long lgec,lgec2,lgec3,tnum[10],sident,sident1,sident2;
long yenirno,curpos,yenictno;
int tnumind[10],gecic,gec,gec2,gec3,gec1,gec12,gec13,blnmdi;
int boolv1,boolv2,tempsb[100];
float oper1,oper2;
double son;
char crossc[100][100][1];
char tempbuf[100],tempbuf2[100];
char *charptr,index[10],operat,hedef;
int pos1,pos2,pos4,j,k,l,m,dec,sign,follow;
int hashval,bulunam,ind;
double gecici;
struct sembuf p={0,-1,SEM_UNDO};
struct sembuf v={0,1,SEM_UNDO};
ANAYAPI *adrp;
KUYRUK *adrp2;
int shmid,shmid1,shmid2,shmid3,semid,semid1;
long trans[100];
int pno,pnum,giratr,cikatr;
int instrsay,yer;

pid_t pid;
processorid_t gelen;
idtype_t p_pid;
```

```
deleteon=0;
```



```

pno=10;
pnum=2;
pid=getpid();
p_pid=getppid();
k=processor_bind(P_PID,pid,pnum,&gelen);

if((shmid1=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((adrp=(ANAYAPI *)shmat(shmid1,(void *)0,0))==(ANAYAPI *)(-1))
    printf ("\n attach olmadı \n");

if((shmid=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((adrp2=(KUYRUK *)shmat(shmid,(void *)0,0))==(KUYRUK *)(-1))
    printf ("\n attach kkkkolmadı \n");

/* global hash tablosu ve global carpisma tablosunu shared
memory olarak al */

if((shmid2=shmget((key_t)0x35,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((ght_buf=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
    printf ("\n attach olmadı \n");
if((shmid3=shmget((key_t)0x45,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((gct_buf=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
    printf ("\n attach olmadı \n");

if((shmid2=shmget((key_t)0x55,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((ght_buf2=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
    printf ("\n attach olmadı \n");
if((shmid3=shmget((key_t)0x65,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
    printf ("\n get olmadi \n");
if((gct_buf2=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
    printf ("\n attach olmadı \n");

if ((queue_id= msgget(QKEY,IPC_CREAT|QPERM))==-1)
    perror("msg get failed");

/* çıkış kuyruğu */
while(1) { /* sonsuz döngü */
if((rlen=msgrcv(queue_id,&s_entry,5,(long)pno,MSG_NOERROR))==-1)
    perror("msgrcv failed");
while(adrp->procact[pno]!=1)a=a+a;

yer=adrp->tablo[pno][5];

```

```

giratrs=adrp->testgas[yer];
cikatrs=adrp->testcas[yer];
instrsay=adrp->testksay[yer];
fdsay=giratrs; /* file descriptor tablosu pointer'ını sıfırla */
translg=adrp->testtlg[yer];
translc=adrp->testtlc[yer];

for (i=0;i<giratrs;i++){
    tnumind[i]=adrp->testuse[yer][i].ind;
    strcpy(rname,adrp->testuse[yer][i].relname);
    strcpy(atname,adrp->testuse[yer][i].atname);
    strcat(rname,atname);
    strcpy(rtfile,rname);
    strcpy(htfile,rname);
    strcpy(ctfile,rname);
    strcat(rtfile,r);
    strcat(htfile,h);
    strcat(ctfile,c);

    fd[i][0]=open(rtfile, O_CREAT|O_RDWR);
    fd[i][1]=open(htfile, O_CREAT|O_RDWR);
    fd[i][2]=open(ctfile, O_CREAT|O_RDWR);
}

for (i=0;i<giratrs;i++){
    lt[i]=adrp->testlt[yer][i];
}

for (i=0;i<instrsay;i++){
    crossc[adrp->testinst[yer][i][1]]
        [adrp->testinst[yer][i][2]] [0]=adrp->testop[yer][i];

    crossc[adrp->testinst[yer][i][1]]
        [adrp->testinst[yer][i][2]] [1]=adrp->testinst[yer][i][0];
}

for (i=0;i<instrsay;i++){
    inst[i]=adrp->testicrk[yer][i];
}

transoku(translg,adrp2,trans,semid,&p,&v,adrp->tablo[pno][2]);
sident=trans[1];

while (sident!=-1){
    /* yerleştirme yaptı sıfırlama yapmadı */
    /* bu noktada eger locate'ler varsa yapılacak */
    if (giratrs>0){
        sayac=0;
        while (sayac<giratrs){
            if (lt[sayac]!=0){
                if (lt[sayac]>0){ /* sident değeri direkt */
                    sident=lt[sayac];

```



```

} else{
    sident=trans[lt[sayac]*(-1)];
} /* eğer - ise hareket indeksi ile geliyor */
paket(temp,sident); /* sident paketlenerek
                        hash parametresi olarak kullanılmalı */
htp=fd[sayac][1];
ctp=fd[sayac][2];
hashval=hash(temp);
lseek(htp,hashval*sizeof(HTTAB),SEEK_SET);
/* ht fileda git bul */
if (read(htp, &ht_buf[sayac], sizeof(HTTAB)) == -1)
    printf("problem reading HT\n");

if (ht_buf[sayac].ht2!=0){
    /*eğer collision bucket tesis edilmiş ise
       yani aynı zamanda bu entry boş ise */
    lgec=ht_buf[sayac].ht2 ;
    lseek(ctp, lgec*sizeof(CTTAB), SEEK_SET);
    /* collision tablosuna geç */
    if (read(ctp, &ct_buf[sayac], sizeof(CTTAB)) == -1)
        printf("problem C\n");
    while (ct_buf[sayac].ct2!=0&&ct_buf[sayac].ct1!=sident) {
        lgec=ct_buf[sayac].ct2;
        lseek(ctp,lgec*sizeof(CTTAB),SEEK_SET);
        if (read(ctp, &ct_buf[sayac], sizeof(CTTAB)) == -1)
            printf("problem CT\n");
    } /* collision bucket içinde bulana kadar veya sonuna
       ulaşip bulamayana kadar ilerledi */
    if (ct_buf[sayac].ct1==sident){ /* bucket içinde buldu */
        tnum[sayac]=ct_buf[sayac].ct3;
    }else tnum[sayac]=0;
} /* if ht_buf[sayac].ht2!=0 */
else tnum[sayac]=0;
} /* if lt[sayac]!=0 */
sayac += 1;
} /* while sayac <= giratrs */

} /* if giratrs > 0 */

/* komutlari sırasıyla al */
q=0;
while(inst[q]!=0){
    gecici=inst[q]/100;
    pos2=gecici;
    pos4=inst[q]-pos2*100;
    /* hareket değişkenlerinden ilk mesajda belirtilenleri
    al */

    if (pos2 < 92&&pos4 < 98){ /* aşağıdaki blokun boşuna icra etmemesi için */
        if(pos2 < 32&&pos2 > 0){
            sident1=trans[pos2];

```

```

if (sident1 > 0){
    bas=sident1+4;
    oper1=acpaketd(gct_buf->ct,bas);
}else {
    bas=sident1*(-1)+4;
    oper1=acpaketd(gct_buf2->ct,bas);
}

}

if (pos4 < 32 && pos4 > 0){
    sident2=trans[pos4];
    if (sident2 > 0){
        bas=sident2+4;
        oper2=acpaketd(gct_buf->ct,bas);
    }else {
        bas=sident2*(-1)+4;
        oper2=acpaketd(gct_buf2->ct,bas);
    }
}

if (pos2 > = 32 && pos2 < 62){
    oper1=temps[pos2-32];
}
if (pos4 > = 32 && pos4 < 62){
    oper2=temps[pos4-32];
}
if (pos2 > = 62 && pos2 < 82)
    boolv1=tempsb[pos2-62];
if (pos4 > = 62 && pos4 < 82)
    boolv2=tempsb[pos4-62];
if (pos2 > = 82 && pos2 < 92){
    sayac=pos2-82;
    if (tnumind[sayac] != 0){
        lseek(fd[sayac][0],
            tnum[tnumind[sayac]-1]*sizeof(RTTAB),SEEK_SET);
        /* çıkış attribute unun rt tablosundan aynı
        numaralı kayıdı oku */
        if (read(fd[sayac][0], &rt_buf[sayac], sizeof(RTTAB)) == -1)
            printf("problem RTcik\n");
        ayris(rt_buf[sayac].rt,&gec,&gec2,&gec3);
        /* bu kayıdı ayırıştır */
        lgec2=-gec; /* long a çevir - den kurtar */
        lseek(fd[sayac][2],lgec2*sizeof(CTTAB),SEEK_SET);
        if (read(fd[sayac][2],&ct_buf[sayac],sizeof(CTTAB)) == -1)
            printf("\n çıkış ct okumada problem \n");
        /* çıkış sahası çarpışma tablosundan sident
        değerini oku */
        sident1=ct_buf[sayac].ct1;
    } else { /* tnumind[sayac] == 0 */
        if (lt[sayac] > 0){ /* sident değeri direkt */
            sident1=lt[sayac];

```



```

    } else{
        sident1=trans[l[sayac]*(-1)];
    }

    /* else */
    if (sident1 > 0){
        bas=sident1 + 4;
        oper1=acpaketd(gct_buf->ct,bas);
    }else {
        bas=sident1*(-1)+4;
        oper1=acpaketd(gct_buf2->ct,bas);
    }
}

if (pos4 >= 82 && pos4 < 92){
    sayac=pos4-82;
    if (tnumind[sayac] != 0){
        lseek(fd[sayac][0],
            tnum[tnumind[sayac]-1]*sizeof(RTTAB),SEEK_SET);
        /* çıkış sahasının rt tablosundan aynı
        numaralı kaydı oku */
        if (read(fd[sayac][0], &rt_buf[sayac], sizeof(RTTAB)) == -1)
            printf("problem RTcik\n");
        ayris(rt_buf[sayac].rt,&gec,&gec2,&gec3);
        scanf("%c",&dumy);
        /* bu kaydı ayrıştır */
        lgec2=-gec; /* long a çevir - den kurtar */
        lseek(fd[sayac][2],lgec2*sizeof(CTTAB),SEEK_SET);
        if (read(fd[sayac][2],&ct_buf[sayac],sizeof(CTTAB)) == -1)
            printf("\n çıkış ct okumada problem \n");
        /* çıkış sahası çarpışma tablosundan sident
        değerini oku */
        sident2=ct_buf[sayac].ct1;
    } else { /* tnumind[sayac] == 0 */
        if (lt[sayac] > 0){ /* sident değeri direkt */
            sident2=lt[sayac];
        } else{
            sident2=trans[l[sayac]*(-1)];
        }
    }

    /* else */
    if (sident2 > 0){
        bas=sident2 + 4;
        oper2=acpaketd(gct_buf->ct,bas);
    }else {
        bas=sident2*(-1)+4;
        oper2=acpaketd(gct_buf2->ct,bas);
    }
}

} /* if pos2 < 92 && pos4 < 98 */

```

```

operat = crossc[pos2][pos4][0];
hedef = crossc[pos2][pos4][1];
if (operat == '*') {
    if (hedef < 32)
        transv[hedef] = oper1 * oper2;
    if (hedef >= 32 && hedef < 62) {
        temps[hedef-32] = oper1 * oper2;
    }
}
if (operat == '+') {
    if (hedef < 32)
        transv[hedef] = oper1 + oper2;
    if (hedef >= 32 && hedef < 62)
        temps[hedef-32] = oper1 + oper2;
}
if (operat == '-') {
    if (hedef < 32)
        transv[hedef] = oper1 - oper2;
    if (hedef >= 32 && hedef < 62)
        temps[hedef-32] = oper1 - oper2;
}
if (operat == '/') {
    if (hedef < 32)
        transv[hedef] = oper1 / oper2;
    if (hedef >= 32 && hedef < 62)
        temps[hedef-32] = oper1 / oper2;
}
if (operat == '=') {
    if (hedef < 32)
        transv[hedef] = oper1;
    if (hedef >= 32 && hedef < 62)
        temps[hedef-32] = oper1;
    if (hedef >= 62 && hedef < 82) {
        if (pos2 >= 92) {
            if (trans[pos4] < 0) {
                gct_gec = gct_buf2;
                sident1 = trans[pos4] * (-1);
            }
            else {
                gct_gec = gct_buf;
                sident1 = trans[pos4];
            }
        }
        tempsb[hedef-62] = strcmp(adrp->teststr[yer][pos2-92],
                                &gct_gec->ct[sident1+4]);
    }
    else {
        tempsb[hedef-62] = boolv1;
    }
}
}
if (operat == '$') {
    if (hedef >= 32 && hedef < 62)

```



```

        tempsb[hedef-32]=adrp->testsbt[yer][pos2-32];
    }
    if (operat=='>') {
        tempsb[crossc[pos2][pos4][1]-62]=oper2>oper1;
    }
    if (operat=='<')
        tempsb[crossc[pos2][pos4][1]-62]=oper2<oper1;
    if (operat=='b')
        tempsb[crossc[pos2][pos4][1]-62]=oper2>=oper1;
    if (operat=='k')
        tempsb[crossc[pos2][pos4][1]-62]=oper2<=oper1;
    if (operat=='n') {
        if (pos2>=92){
            if (trans[pos4]<0 ){
                gct_gec=gct_buf2;
                sident1=trans[pos4]*(-1);
            }
            else{
                gct_gec=gct_buf;
                sident1=trans[pos4];
            }
            if(strcmp(adrp->teststr[yer][pos2-92],
                &gct_gec->ct[sident1+4])!=0)
                tempsb[hedef-62]=1;
            else
                tempsb[hedef-62]=0;
        }
        else{
            tempsb[crossc[pos2][pos4][1]-62]=oper1!=oper2;
        }
    }
    if (operat=='e')
        if (pos2>=92){
            if (trans[pos4]<0 ){
                gct_gec=gct_buf2;
                sident1=trans[pos4]*(-1);
            }
            else{
                gct_gec=gct_buf;
                sident1=trans[pos4];
            }
            if(strcmp(adrp->teststr[yer][pos2-92],
                &gct_gec->ct[sident1+4])==0)
                tempsb[hedef-62]=1;
            else
                tempsb[hedef-62]=0;
        }
        else{
            tempsb[crossc[pos2][pos4][1]-62]=oper1==oper2;
        }
    if (operat=='&')
        tempsb[crossc[pos2][pos4][1]-62]=boolv1&&boolv2;
    if (operat=='|')

```

```

    tempsb[crossc[pos2][pos4][1]-62]=boolv1 || boolv2;
    if (operat == '!')
        tempsb[crossc[pos2][pos4][1]-62]=!boolv1;

    q += 1;
} /*while inst[q]*/

/* Komutların bu hareket için icrası bitti. şimdi
bu işlemler sonunda transv ye yapılan atamalardan yararlanarak
çıkış hareketi yaratılacak ve çıkış kuyruğuna yazılacak
yine eğer filref yapısına atama yapılmışsa bu değerler fd yapısındaki
file descriptionlar ile belirlenen saha dosyalarının(r) tnum
yapısında belirtilen pozisyonlarına yeni değer olarak işlenecektir.
bu numaraya sahip kayıt, mevcut same linkten çıkartılacak (same linkteki bir
önceki ve bir sonraki kayıtların forward/backward pointer'ları
buna göre ayarlanacak, eğer önceki ve sonraki kayıt yoksa, ct yapısındaki
başa ve sona pointer'lar ayarlanacak/sıfırlanacak ve ct yapısına
yeni sident değerini içeren kayıt (yok ise) eklendikten sonra
bu kaydın same linkine, .r da bu pozisyonadaki kayıt linklenecektir. */
char temp[4],dum[2];
char *chp;
if (tempsb[0] == 0) {
    if (adrp->tablo[pno][3] != 0){
        transyaz(translc,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);
    }
} else {
    if (adrp->tablo[pno][1] != 0){
        /* eğer bu kuyruk no varsa ikinci etiket de var */
        transyaz(translc,adrp2,trans,semid,&p,&v,adrp->tablo[pno][1]);
    }
}
transoku(translg,adrp2,trans,semid1,&p,&v,adrp->tablo[pno][2]);
sident=trans[1];

} /* while trcv!=-1 */
trans[1]=-1;
if (adrp->tablo[pno][3] != 0)
    transyaz(translc,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);
/* bu kuyruğu sonlandır */
if (adrp->tablo[pno][1] != 0)
    transyaz(translc,adrp2,trans,semid,&p,&v,adrp->tablo[pno][1]);
/* eğer ikinci etiket varsa bu kuyruğu da sonlandır */
semop(adrp->simtan[0],&p,1);
adrp->testubit[adrp->tablo[pno][5]] = 0;
semop(adrp->simtan[0],&v,1);
semop(adrp->simtan[1],&p,1);
adrp->tablo[pno][0] = 0;
adrp->procact[pno] = 0;
semop(adrp->simtan[1],&v,1);

```



```

semop(adrp->simtan[2],&p,1);
adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
semop(adrp->simtan[2],&v,1);
} /* sonsuz döngü sonu */
} /* main */

```

```
/* TXCNT BLOĞU SANAL İŞLEMCİSİ KAYNAK PROGRAMI */
```

```
#include <derle.h>
```

```
main(){ /*1*/
```

```
/* değişken bildirimleri*/
```

```
float txsayisi=0 ; /* geçen hareket sayısı */
```

```
char isim[50],cev[2];
```

```
char temp[4],dum[2];
```

```
char *charptr;
```

```
char tempbuf[256],gec[256];
```

```
int tempint,ft;
```

```
int sayac,sayac2,cntsay,k,l,j,r,s,demir,dec,sign,bulunam,follow;
```

```
int gtp,gcp,ctp,rtp,htp,gecici,gecici1,gecici2,gecici3,ident1,ident2,ident3;
```

```
int ident4,ident5;
```

```
long hashval,yenirno,yenictno,lgec,curpos;
```

```
char *tempptr1="\0",*tempptr2="\0";
```

```
double tempflt;
```

```
float cc;
```

```
pid_t pid;
```

```
processorid_t gelen;
```

```
idtype_t p_pid;
```

```
long sident,sident2, pnum,ilk,iki;
```

```
struct sembuf p={0,-1,SEM_UNDO};
```

```
struct sembuf v={0,1,SEM_UNDO} ;
```

```
ANAYAPI *adrp;
```

```
KUYRUK *adrp2;
```

```
GCTTAB *gct_buf2;
```

```
GHTTAB *ght_buf2;
```

```
struct q_entry s_entry;
```

```
int queue_id,rlen;
```

```
int i,a,shmid,shmid1,shmid2,shmid3,semid,semid1;
```

```
long trans[100];
```

```
int pno,transl,yer,fltsay;
```

```
struct tms timebuf;
```

```
FILE *fp;
```

```

char fname[10];

pno = 52;
pnum = 2;
pid = getpid();
p_pid = getppid();
k = processor_bind(P_PID, pid, pnum, &gelen);

if((shmid1 = shmget((key_t)0x15, sizeof(ANAYAPI), 0777 | IPC_CREAT)) == -1)
    printf ("\n get olmadi \n");
if((adrp = (ANAYAPI *)shmat(shmid1, (void *)0, 0)) == (ANAYAPI *)(-1))
    printf ("\n attach olmadı \n");

if((shmid = shmget((key_t)0x25, sizeof(KUYRUK), 0777 | IPC_CREAT)) == -1)
    printf ("\n get olmadi \n");
if((adrp2 = (KUYRUK *)shmat(shmid, (void *)0, 0)) == (KUYRUK *)(-1))
    printf ("\n attach kkkkolmadı \n");
if((shmid2 = shmget((key_t)0x55, sizeof(GHTTAB), 0777 | IPC_CREAT)) == -1)
    printf ("\n get olmadi \n");
if((ght_buf2 = (GHTTAB *)shmat(shmid2, (void *)0, 0)) == (GHTTAB *)(-1))
    printf ("\n attach olmadı \n");
if((shmid3 = shmget((key_t)0x65, sizeof(GCTTAB), 0777 | IPC_CREAT)) == -1)
    printf ("\n get olmadi \n");
if((gct_buf2 = (GCTTAB *)shmat(shmid3, (void *)0, 0)) == (GCTTAB *)(-1))
    printf ("\n attach olmadı \n");
if ((queue_id = msgget(QKEY, IPC_CREAT | QPERM)) == -1)
    perror("msg get failed");

/* ANAYAPI SİSTEM YERİ YAPILARI */
while(1) { /* sonsuz döngü */
if((rlen = msgrcv(queue_id, &s_entry, 5, (long)pno, MSG_NOERROR)) == -1)
    perror("msgrcv failed");
printf("\n mesaj alındı %ld \n", s_entry.mtype);
txsayisi = 0;
while(adrp->procact[pno] != 1) a = a + a;

yer = adrp->tablo[pno][5];

transl = adrp->txstl[yer];
transoku(transl, adrp2, trans, semid1, &p, &v, adrp->tablo[pno][2]);
sident = transl[1];

while (sident != -1){
    txsayisi += 1;
    transoku(transl, adrp2, trans, semid1, &p, &v, adrp->tablo[pno][2]);
    sident = transl[1];
} /* while sident != -1 */
paketd(gec, (char *)&txsayisi);

```



```

sident2=altyapi(ght_buf2,gct_buf2,gec);
trans[1]=sident2;
transyaz(1,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);
trans[1]=-1;
transyaz(1,adrp2,trans,semid,&p,&v,adrp->tablo[pno][3]);

```

```

printf("\n bu islem %ld tick surdu \n",iki-ilk); */
semop(adrp->simtan[0],&p,1);
adrp->txsubit[adrp->tablo[pno][5]]=0;
semop(adrp->simtan[0],&v,1);
semop(adrp->simtan[1],&p,1);
adrp->tablo[pno][0]=0;
adrp->procact[pno]=0;
semop(adrp->simtan[1],&v,1);
semop(adrp->simtan[2],&p,1);
adrp->serbkuy[adrp->tablo[pno][2]][1]=0;
adrp->serbkuy[adrp->tablo[pno][3]][2]=0;
semop(adrp->simtan[2],&v,1);
} /* sonsuz döngü sonu */
}/* main */

```

```

/* AŞAĞIDA ÇIKTISI VERİLEN
ANAYAPI.H          SİSTEM VERİ YAPILARININ TANIMLARINI
DERLE.H            SİSTEM FONKSİYON KÜTÜPHANESİNİ
PAKET.H            PAKETLEME VE PAKET AÇMA
                   FONKSİYONLARINI İÇERİR.
*/

```

```

/* ANAYAPI.H    SİSTEM VERİ YAPILARI */

```

```

#define MAXBLOK 500
typedef struct {
    char relname[24];
    char atrname[24];
    int ind;
} COMPENT;
typedef struct {
    int simtan[4]; /*          simafor tanıtıcı
                        tablo allocate bit koruma   ===0
                        serbkuy used bit koruma    ===1
                        mesaj blokları koruma      ===2
                        GHT,GCT yapısı k koruma    ===3
                    */
    int tablo [100][8]; /* süreç bilgi tablosu */
    int procact[100]; /* süreç aktivasyon tablosu */
    /* Kuyruk tablosu */
    int serbkuy[200][4];/* 0. indis = kuyruk numarası
                        1. indis = giriş used bit
                        2. indis = çıkış used bit
                        3. indis = ref used bit

```

```

*/
/* Generate attribute blok mesaj bloklari */
int genubit[3]; /* sırasıyla 0,1,2 numaralı
                mesaj blokları used bit'leri */

char genistb[3][10][24];
long genind[3]; /* isim tablosunun ilk elemanı ilişki ismi
                geriye kalanı saha ismidir. */

int gencas[3];
/* Cartesp mesaj bloklari */
int cpubit[3]; /* sırasıyla 0,1,2 numaralı
                mesaj blokları used bit'leri */

char cpistb[3][10][24];
int cptlg[3];
long cpind[3]; /* isim tablosunun ilk elemanı rel ismi
                geriye kalanı attribute ismidir. */
int cpcas[3];

/* Generate tuple blok mesaj bloklari */
int gtubit[3]; /* sırasıyla 0,1,2 numaralı
                mesaj blokları used bit'leri */

char gtistb[3][10][24];
long gtind[3]; /* isim tablosunun ilk elemanı rel ismi
                geriye kalanı attribute ismidir. */
int gtcas[3];
/* Generate unq blok mesaj bloklari */
int gnubit[3]; /* sırasıyla 0,1,2 numaralı
                mesaj blokları used bitleri */

char gnistb[3][10][24];
long gnind[3]; /* isim tablosunun ilk elemanı ilişki ismi
                geriye kalanı saha ismidir. */

int gncas[3];
/* Test blok mesaj bloklari */
long testlt[3][10]; /* locate veri yapısı */
int testubit[3];
COMPENT testuse[3][10]; /* USE blok */
int testinst[3][100][3]; /* komut operandları */
char testop[3][100]; /* komut operatörleri */
int testicrk[3][100]; /* komut dizisi */
char teststr[3][5][256]; /* karakter dizisi tipindeki parametreler */
int testgas[3]; /* giriş değişken sayısı */
int testcas[3]; /* çıkış değişken sayısı */
int testksay[3]; /* test komut sayısı */
int testtlg[3]; /* giriş hareket uzunluğu */
int testtlc[3]; /* çıkış hareket uzunluğu */
float testsbt[3][30]; /* floating point tipindeki parametreler */

```



```

/* Apply blok mesaj bloklari */
int applyubit[3]; /* used biti */
char applyrel[3][24]; /* ilişki ismi */
char applyga[3][10][24]; /* apply giriş saha isimleri */
char applyca[3][10][24]; /* apply çıkış saha isimleri */
int applygas[3]; /* giriş ve çıkış saha sayıları */
int applycas[3];
char applyst[3][24]; /* eğer kullanılıyorsa standart fonksiyon
ismi */
int applytxix[3][10][2]; /* direk bypass edilen var atr map yapısı */
int applytlg[3];
int applytlc[3]; /* giriş ve çıkış hareket uzunlukları */
/* Mem blok mesaj bloklari */
int memubit[3]; /* used biti */
char memrel[3][24]; /* ilişki ismi */
char memga[3][10][24]; /* mem giriş saha isimleri */
int memgas[3]; /* giriş saha sayıları */
int memtlg[3]; /* giriş hareket uzunlukları */
int memtxix[3][10][2]; /* direk bypass edilen var atr map yapısı */
/*Compute fix blok mesaj bloklari */
int compufbit[3];
COMPENT compufse[3][10];
long compflt[3][10]; /* Locate veri yapısı */
int compfinst[3][100][3]; /* Komut yapısı */
char compfop[3][100]; /* operatörler */
int compficrk[3][100]; /* komut dizisi */
int compfgas[3]; /* giriş değişken sayısı */
int compfcas[3]; /* çıkış değişken sayısı */
int compfksay[3]; /* komut sayısı */
int compftlg[3]; /* giriş hareket uzunluğu */
int compftlc[3]; /* çıkış hareket uzunluğu */
float compfsbt[3][30]; /* floating point tipindeki parametreler */
/*Compute blok mesaj bloklari */
int compubit[3]; /* used bit */
COMPENT compuse[3][10]; /* USE blok */
long complt[3][10]; /* Locate block */
int compinst[3][100][3]; /* komut tanımları */
char compop[3][100]; /* komut opcode'ları */
int compicrk[3][100]; /* komut dizisi */
int compgas[3]; /* giriş değişken sayısı */
int compcas[3]; /* çıkış değişken sayısı */
int compksay[3]; /* komut sayısı */
int comptlg[3]; /* giriş hareket uzunluğu */
int comptlc[3]; /* çıkış hareket uzunluğu */
float compsb[3][30]; /* floating point tipindeki parametreler */
/* Input blok mesaj bloklari */
int inpubit[3]; /* used bit */
int inputent[3][4][100]; /* 0 satır
1 sutun
2 hareket'deki pozisyon
3 tip

```

```

    */
char inputfb[3][100][10]; /* saha başlıkları */
int inputmat[3]; /* maksimum değişken sayısı sayısı */
int inputtl[3]; /* hareket uzunluğu */
/* Generate Element blok mesaj blokları */
int inpusbbit[3]; /* used bit */
int inputsbent[3][10]; /* =1 numerik =0 string */
/* hareket uzunluğu en çok 10 olabilir */
float inputsbf[3][10]; /*sayısal sabit (eger varsa) */
char inputsbst[3][10][256]; /* karakter dizin sabit (eger varsa) */
int inputsbtl[3]; /* hareket uzunluğu */
/* Report Blok mesaj blokları */
int outpubit[3]; /* output used bit */
int outputent[3][4][100]; /* output girdisi */
char outputfb[3][100][10]; /* output saha başlıkları */
int outmat[3]; /* report max değişken sayısı */
int outputtl[3]; /* hareket uzunluğu */
/* Split Blok mesaj blokları */
int splitubit[3]; /* used bitler */
int spltqids[3][10]; /*çıkış kuyruk tanıtıcıları */
int splttrsl[3]; /* hareket uzunluğu */
int spltadrsl[3]; /* dallanma sayısı */
int splttl[3]; /* hareket uzunluğu giriş ve çıkış aynı */
/* Merge Blok mesaj blokları */
int mergeubit[3]; /* used bitler */
int mergeqids[3][10]; /*çıkış kuyruk tanıtıcıları */
int mergeadrsl[3]; /* dallanma sayısı */
int mergetl[3]; /* hareket uzunluğu giriş ve çıkış aynı */
/* Cndblok Blok mesaj blokları */
int cndubit[3]; /* used bitler */
int cndreftl[3]; /* cnd referans kuyruk hareket uzunluğu */
int cndrefsl[3]; /* cnd referans verinin hareketdeki mevkii */
int cndrefst[3]; /* referans veri tipi */
int cndrefqid[3]; /* referans kuyruk tanıtıcısı */
int cndinptl[3]; /* standart input hareket uzunluğu */
int cndinpsl[3]; /* karşılaştırılacak verinin giriş tx'daki mevkii */
int cndinpt[3]; /* giriş veri tipi */
int cndinpqid[3]; /* standart giriş kuyruğu tanıtıcısı */
char cndsart[3]; /* karşılaştırma şart indikatörü */
/* Sort Blok mesaj blokları */
int sortubit[3]; /* mesaj blok used bit */
int sorttl[3]; /* sort hareket boyu */
int sorttsay[3]; /* sort tx index sayısı */
int sorttxix[3][10]; /* sort tx indeksleri veri yapısı */
int sorttyp[3][10]; /* Sort tipi */
/* Count obje mesaj blokları */
int cntubit[3]; /* count blok kullanım biti */
int cnttxix[3][10]; /* hareketde hangi sahaların
kombine anahtarda yer alacağı
ve sıraları */

```



```

int cnttl[3]; /* bu bloğa giren hareket uzunluğu */
int cntsay[3]; /* cnttx de kac adet indeks verildiği */
int cntloc[3]; /* sayaç bilgisinin hareketde
                depolanacağı indeks */
/* Txsay Blok mesaj bloklari */
int txsubit[3]; /* used biti */
int txstl[3]; /* bu bloğa giren hareket uzunluğu */
/* Apply Cumulative Blok mesaj bloklari */
int cumubit[3]; /* used biti */
int cumtxix[3][10]; /* hareketde hangi sahaların
                    kombine anahtarda yer alacağı
                    ve sıraları */
int cumtl[3]; /* bu bloğa giren hareket uzunluğu */
int cumtsay[3]; /* cumtxix de kac adet indeks verildiği */
int cumtloc[3]; /* sayaç bilgisinin hareketde
                depolanacağı indeks */
int cumfld[3]; /* Birikimsel fonksiyonun hangi sahaya uygulanacağı */
char cumop[3]; /* Birikimsel fonksiyon operatörü */
/* Transfer Blok mesaj bloklari */
int trsfubit[3]; /* used bit */
int trsfrsl[3]; /* hareket uzunluğu */
/* Herhangi bir özel mesaj almasına gerek yok */
/* Sink Blok mesaj bloklari */
int sinkubit[3]; /* used bit */
int sinktl[3]; /* sink hareket boyu */
/* Filter Blok mesaj bloklari */
int fltubit[3]; /* used biti */
int flttxl[3]; /* bloğa giren hareket uzunluğu */
int fltsay[3]; /* filitre edilecek tx indeks sayısı */
int fltxix[3][10]; /* hangi nolu tx girdileri
                  kombine olarak filitre
                  edilecek max 10 */
/* Append Blok mesajlari */
int updtubit[3];
int updttl[3]; /* update hareket uzunluğu */
COMPENT updtent[3][10]; /* update girdisi pozitif ise
                        hareket indeksi
                        negatif ise sident'in kendisi*/
char updtreln[3][24]; /* ilişki ismi */
int updtmax[3]; /* maksimum saha sayısı
                updtent de bulunacak endirek tamsayı
                pointer maksimum sayısı */
/* Delete Blok mesajlari */
int delubit[3]; /* delete used bit */
int delttl[3]; /* delete hareket uzunluğu */
int delmat[3]; /* maksimum kriter sayısı */
int delmast[3][10]; /* hangi saha master saha olarak kullanılacak*/
COMPENT delent[3][10]; /* delete girdisi */
} ANAYAPI;

```

```
/* DERLE.H DOSYASI (SİSTEM FONKSİYON KÜTÜPHANESİ) */
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/times.h>
#include <unistd.h>
#include <sys/msg.h>
#include <sys/procset.h>
#include <sys/processor.h>
#include </home/suha/final/anayapi.h>
#include </home/suha/final/kuyruk.h>
#include </home/suha/final/paket.h>
```

```
#define MAXATR 100;
#define QKEY (key_t)0105
#define QPERM 0660
#define MAXOBN 5
```

```
#define hashsize 5000
```

```
struct q_entry{
    long mtype;
    char mtext[MAXOBN + 1];
};
```

```
typedef struct{
    char relname[50];
    char atrname[256][50];
    int atrtype[256];
    int atrsize[256];
    int atrsayi;
    int rtsize;
} RELINFO;
```

```
typedef struct {
    long rt;
} RTTAB;
```

```
typedef struct {
    int ht2;
} HTTAB;
```

```
typedef struct {
    long ct1;
    int ct2;
    int ct3;
    int ct4;
} CTTAB;
```



```

typedef struct {
    int po[5000];
} GHTTAB;

typedef struct {
    int k;
    char ct[15000];
} GCTTAB;

typedef struct {
    long k;
} FBUF;

/* Kuyruktan bir hareket okur */

transoku(transl,qbuf,trans,semid,p,v,kuyno)
int transl,semid;
KUYRUK *qbuf;
long *trans;
struct sembuf *p;
struct sembuf *v;
int kuyno;

{
    int i,read_p,write_p, readm_p,writem_p,beg;
    read_p=qbuf->q[kuyno][QLENG];
    write_p=qbuf->q[kuyno][QLENG+1];
    readm_p=read_p%QLENG;
    writem_p=write_p%QLENG;
    /* yazma pointer'ı okuma pointer'ından transl kadar büyük
       oluncaya kadar bekle */
    while((write_p-read_p)<=transl-1){
        read_p=qbuf->q[kuyno][QLENG];
        write_p=qbuf->q[kuyno][QLENG+1];
        readm_p=read_p%QLENG;
    }
    /* buffer'ın sonuna gelinip circular buffer nedeniyle
       başa döndüyse q0 ve q1'i atla
    */
    if (readm_p==0) beg=0;
    else beg=readm_p;
    for (i=1;i<=transl;i++){
        trans[i]=qbuf->q[kuyno][beg];
        beg++;
        if (beg==QLENG-1){
            beg=0;
            qbuf->q[kuyno][QLENG]+ = 1;
        }else {
            qbuf->q[kuyno][QLENG]+ +;
        }
    }
}

```

```

}
}
}

/* Kuyruğa bir hareket yazar */
transyaz(transl,qbuf,trans,semid,p,v,kuyno)
int transl,semid; /* hareket uzunluğu ve semafor tanıtıcısı */
KUYRUK *qbuf; /* gerçek kuyruk */
long *trans; /* hareket veri yapısı */
struct sembuf *p; /* semafor bufferları fonksiyona sadece pointer geçer*/
struct sembuf *v;
int kuyno;
{
    int i,read_p,write_p, readm_p,writem_p,beg;
    /* tamsayı pointer'lar
        read_p kuyrukta bir sonra yapılacak yazma işleminin
        başlayacağı yer
        write_p kuyrukta bir sonra yapılacak yazma işleminin
        başlayacağı yer
        readm_p mod read_p
        writem_p mod write_p circular buffer kullanımı için
        beg house keeping için kullanılan tamsayı pointer
    */
    read_p=qbuf->q[kuyno][QLENG];
    write_p=qbuf->q[kuyno][QLENG+1];
    readm_p=read_p%QLENG;
    writem_p=write_p%QLENG;
    /* shared memory deki son durumu al ve mod operasyonu
        ile kuyruk üzerindeki indisleri elde et */

    /* okuma ve yazma mod pointer'ları birbirine eşit olduğunda
        okuma ve yazma pointer'ları da birbirine eşit değilse
        yazma mod pointer'ı okuma yazma pointer'ından büyük
        olduğunda bunların arasındaki fark hareket uzunluğundan
        büyük değilse okuma modulo pointer'ı yazma modulo pointer'ından
        büyük hale geldiyse yani yazma modulo pointer'ı bir tur ileri geçip
        geriden gelip tekrar yetiştiyse ve bu durumda ters yönden
        aradaki fark, hareket uzunluğundan büyük değilse
        BU SARTLAR SAGLANANA KADAR BEKLE.
    */
    while (!((((readm_p==writem_p)&&(read_p==write_p))||
        (writem_p>readm_p)&&((writem_p-transl)>=readm_p))||
        ((readm_p>=writem_p)&&((readm_p+QLENG-writem_p<=transl))))))
    {
        read_p=qbuf->q[kuyno][QLENG];
        write_p=qbuf->q[kuyno][QLENG+1];
        readm_p=read_p%QLENG;

```



```

    writem_p=write_p%QLENG;
}
beg=writem_p;
for (i=1;i<=transl;i++){

    qbuf->q[kuyno][beg]=trans[i];
    if (beg==QLENG-1){
        beg=0;
        qbuf->q[kuyno][QLENG+1]+=1;
    }else{
        beg++;
        qbuf->q[kuyno][QLENG+1]++;
    }
    /* yukarida exclusive erişim içinde pointer'lar
    hareket ettirilerek bellekteki hareket
    shared memory'deki kuyruk yapısına kopya ediliyor
    */
}
}

```

```

/* hash fonksiyonu */

```

```

unsigned hash(s)
char *s;
{ unsigned hashval,temp;
  for (hashval=0;*s!='\0';s++)
    hashval=*s+31*hashval;
  temp=hashval % hashsize;
  return hashval % hashsize;
}

```

```

/* üç ayrı tam sayıyı tek bir uzun tam sayı (long integer) halinde
birleştirir */

```

```

long birles(i,j,k)
/* Ayrıs fonksiyonunun tersini yapar */

```

```

int i,j,k;
{
  long l,o;
  l=i*1000000;
  o=j;
  if (l<0){
    o=-o;
    k=-k;
  }
  l=l+o*1000;
  l=l+k;
}

```

```

return(l);
}
/* uzun tam sayıya (long integer) paketlenmiş üç ayrı tam sayıyı
ayrıştırır */

```

```

ayris(l,i,j,k)

```

```

long l;

```

```

int *i,*j,*k;

```

```

{
    *i=1/1000000;
    *j=(1/1000)-(*i*1000);
    *k=1-(*i*1000+*j)*1000;
    *j=abs(*j);
    *k=abs(*k);
}

```

```

int karsilas(char *t,char *ilk){

```

```

/* bu fonksiyon karşılaştırma amaçlı kullanılmaktadır */

```

```

int n;
n=0;
for (;*ilk!='\0';ilk++){
    if (*ilk!=*t)n=1;
    t++;
}
return(n);
}

```

```

int kars(char *t,char *ilk){

```

```

/* bu fonksiyon karşılaştırma amaçlı kullanılmaktadır */

```

```

int n;
n=0;
for (;*t!='\0';t++){
    h=ilk;
    if (*ilk!=*t)
        n=0;
    if (*ilk>*t){
        return(1);}
    if (*ilk<*t){
        return(2);}
    *ilk++;
}
return(n);
}

```

```

/* global tablolarda bulunmayan her tip veriyi global tablolara
yerleştirerek yeni sident değeri elde eder */

```

```

long alty(int i, int yer, ANAYAPI *adrp,
GHTTAB *ght_buf,GCTTAB *gct_buf,
char *tempbuf,long *trans,int numerik) {

```




```

int k,ii,bulunam,l,j, follow;
char temp[10];
unsigned hashval;
long sident;

```

```

j=hashval=hash(tempbuf);
if (ght_buf->po[hashval]==0){
    /* eger global hash tablosunun bu girdisi boş ise
    global collision tablosuna girilecek yeni deęerin
    pozisyonunu belirleyen pointer (k) deęerini
    buraya yerleřtir */
    k=gct_buf->k; /* k = shared memory deki
    ct[0] ONEMLI */
    ght_buf->po[j]=k;
    sident=k;
    /* sident çok önemli indeks sakla */
    l=0;
    for (ii=0;ii<=3;ii++){
        gct_buf->ct[k++]=0;

        gct_buf->k+=4;
        if (numerik){
            for (ii=0;ii<=7;ii++){
                gct_buf->ct[gct_buf->k++]=tempbuf[l++];
                gct_buf->ct[gct_buf->k++]='\0';
            }
        }
        while (tempbuf[l]!='\0'){
            gct_buf->ct[gct_buf->k++]=tempbuf[l++];
            gct_buf->ct[gct_buf->k++]='\0';
        }
        return(sident*(-1));
        /* girilen deęer gct'a yüklendi ve sonlandırıldı */
    }

    /*eđer hash entry 0'dan büyük ise daha önce buraya giriş yapılmıř */
    else {
        j=hashval;
        follow=0;
        /* follow traverse esnasında bir kayıt geriden
        gelen pointer'dır */

        l=ght_buf->po[j] ;/* Bu pointer i al */
        sident=l;
        bulunam=karsilas(&gct_buf->ct[l+4],tempbuf);
        /*
        coll tablosundaki ilk deęer ile karřılařtır */
        while (bulunam && (l!=0)){
            for(ii=0;ii<=3;ii++){

```

```

temp[ii]=gct_buf->ct[l+ii];
}
follow=1;
l=acpaket(temp);

/* pointer'ı paketlenmiş olduğu karakter dizisi
formundan integer formuna aç ve bu pointer'ı
takip ederek coll linkte bir sonraki kayıda
geç */
sident=1;
if (l>0){
bulunam=karsilas(&gct_buf->ct[l+4],tempbuf) ;
}

if (l==0&&bulunam){ /* collision bucket'in sonuna
erişti ama bulamadı*/
j=0;

sident=k=gct_buf->k;
for(ii=0;ii<=3;ii++)
gct_buf->ct[gct_buf->k++]=0;
/* sident çok önemli indeks, sakla */
if (numerik){
for (ii=0;ii<=7;ii++)
gct_buf->ct[gct_buf->k++]=tempbuf[l++];
}else{
while (tempbuf[l]!='\0'){
gct_buf->ct[gct_buf->k++]=tempbuf[l++];
}
gct_buf->ct[gct_buf->k]='\0';
}
/* yeni girdi olarak yerleştirildi */
paket(temp,sident);
if (follow>0){
for(ii=0;ii<=3;ii++)
gct_buf->ct[follow+ii]=temp[ii];
/* bir kayıt geriden gelen pointer'ı
kullanarak bir önceki kayıda eriş
ve bu kaydın collision forward
linkini yeni girdiyi gösterecek
şekilde güncle */
}
return(sident*(-1));
}
if (bulunam==0) /* bulundu sident
dogru değere set edildi*/
return(sident*(-1));
} /* end else === if ght_buf->po[hashval]>0 */

} /* function alty */

```


/* global tablolarda bulunmayan sayısal veriyi global tablolara yerleştirerek yeni sident değeri elde eder */

```
long altyapi(GHTTAB *ght_buf, GCTTAB *gct_buf, char *tempbuf) {
```

```
int k, ii, bulunam, l, j, follow;
```

```
char temp[10];
```

```
unsigned hashval;
```

```
long sident;
```

```
    j=hashval=hash(tempbuf);
```

```
    if (ght_buf->po[hashval]==0){
```

```
        /* eğer global hash tablosunun bu girdisi boş ise
        global collision tablosuna girilecek yeni değerin
        pozisyonunu belirleyen pointer (k) değerini
        buraya yerleştir */
```

```
        k=gct_buf->k; /* k = shared memory deki
```

```
        sident=k=gct_buf->k;
        for (ct[0] ONEMLI */
```

```
        ght_buf->po[j]=k;
```

```
        sident=k; /* sident çok önemli indeks, sakla */
```

```
        /* sident çok önemli indeks sakla */
```

```
        l=0;
```

```
        for (ii=0; ii<=3; ii++){
```

```
            gct_buf->ct[k++] = 0;
```

```
        gct_buf->k+=4;
```

```
        for (ii=0; ii<=7; ii++){
```

```
            gct_buf->ct[gct_buf->k++] = tempbuf[l++];
```

```
        gct_buf->ct[gct_buf->k++] = '\0';
```

```
        return sident*(-1);
```

```
        /* girilen değer gct'a yüklendi ve sonlandırıldı */
```

```
    }
```

```
/*eğer hash entry 0 dan büyük ise daha önce buraya giriş yapılmış */
```

```
else {
```

```
    j=hashval;
```

```
    follow=0;
```

```
    /* follow traverse esnasında bir kayıt geriden
```

```
    gelen pointer'dır */
```

```
    l=ght_buf->po[j]; /* Bu pointer'ı al */
```

```
    sident=l;
```

```
    bulunam = karsilas(&gct_buf->ct[l+4], tempbuf);
```

```
    /* coll tablosundaki ilk değer ile karşılaştır */
```

```
    while (bulunam && (l!=0)){
```

```
        for(ii=0; ii<=3; ii++){
```

```
            temp[ii]=gct_buf->ct[l+ii];
```

```
        }
```



```

        follow=1;
        l=acpaket(temp);

        /* pointer'ı paketlenmiş olduğu karakter dizisi
        formundan tamsayı formuna aç ve bu pointer'ı
        takip ederek coll linkte bir sonraki kayıda
        geç */
        sident=1;
        if (l>0){
            bulunam = karsilas(&gct_buf->ct[l+4],tempbuf) ;
        }

    }

    if (l==0&&bulunam){ /* collision bucket'in sonuna
                        erişti ama bulamadı*/
        j=0;

        sident=k=gct_buf->k;
        for(ii=0;ii<=3;ii++)
            gct_buf->ct[gct_buf->k++]=0;
        /* sident çok önemli indeks, sakla */
        for (ii=0;ii<=7;ii++)
            gct_buf->ct[gct_buf->k++]=tempbuf[l++];
        gct_buf->ct[gct_buf->k++]='\0';
        /* yeni girdi olarak yerleştirildi */
        paket(temp,sident);
        if (follow>0){
            for(ii=0;ii<=3;ii++)
                gct_buf->ct[follow+ii]=temp[ii];
            /* bir kayıt geriden gelen pointer'ı
            kullanarak bir önceki kayıda eriş
            ve bu kaydın collision forward
            linkini yeni girdiyi gösterecek
            şekilde güncle */
        }
        return sident*(-1);
    }
    if (bulunam==0) /* bulundu sident
                    doğru değere set edildi*/
        return sident*(-1);
    } /* end else == if ght_buf->po[hashval]>0 */
} /* alt yapı */

/* Diske yazılmış hareketleri okur */
transokd(transl,trans,fp)
int transl;
long *trans;
FILE *fp;
{

```



```

int i;
for(i=1;i<=transl;i++){
    fscanf(fp, "%ld",&trans[i]);
}
}

```

```

/* Diske hareket yazar */
transyazd(transl,trans,fp)
int transl;
long *trans;
FILE *fp;

```

```

{
int i;
for(i=1;i<=transl;i++){
    fprintf(fp, "%ld\n",trans[i]);
}
}

```

```

/* PAKET.H */

```

```

/* bu fonksiyon integer icinde bulunan 4 byte'ı karakter dizinin
dört girdisine paketler */

```

```

paket(char *temp, int r){
    *temp=r;
    r=r>>8;
    *(temp+1)=r;
    r=r>>8;
    *(temp+2)=r;
    r=r>>8;
    *(temp+3)=r;
}

```

```

/* paketlemenin tersi */
int acpaket(char *temp){

```

```

int j,r,k,m,l ;
j=k=m=r=0;
j=j|*(temp+3);
j=j&0x00FF;
j=j<<24;
k=k|*(temp+2);
k=k&0x00FF;
k=k<<16;
m=m|*(temp+1);
m=m&0x00FF;

```

```

    m=m<<8;
    r=0;
    r=r|*temp;
    if (*temp<0){
        r=r&0x00FF;
    }
    l=j|r|m|k;
    return(l);
}

```

/* Kayar noktalı sayıyı 7 elemanlı bir karakter dizin içine paketler. Global Çarpışma tablosu karakter tipinde bir tablo olup, gerek karakter dizin tipindeki gerek se sayısal veriler bu tablo içinde karakter dizinleri şeklinde depolanırlar */

```

paketd(char *temp,char *r){
int i;
for (i=0;i<=7;i++){
    *temp=(*r)+1;
    temp++;
    r++;
}
*temp='\0';
}

```

/* paketd fonksiyonunun tersi */

```

float acpaketd(char *temp,int bas){
char temp2[20];
int i;
for (i=0;i<=7;i++) *(temp2+i)=(*temp+bas+i)-1 ;
return(*(float *)temp2);
}

```

/* kullanım örneği

```

c=678.93;
paketd(gec,(char *)&c);
for(a=0;a<=7;a++) çıkış[a+5]=gec[a];
cc=acpaketd(çıkış,5);
printf("-----%f",cc);
*/

```

/* AŞAĞIDAKİ ANA.C ADLI PROGRAM SİSTEM AÇILIŞINDA PROGRAMLAMA VE PROGRAM

ÇALIŞTIRMA ORTAMININ DEVREYE GİRMESİNİ SAĞLAR. */

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <fcntl.h>
#include <stdio.h>
#include </home/suha/final/anayapi.h>
#include </home/suha/final/kuyruk.h>

typedef struct {
    int po[5000];
} GHTTAB;

typedef struct {
    int k;
    char ct[15000];
} GCTTAB;

main(){
    struct sembuf p={0,-1,SEM_UNDO};
    struct sembuf v={0,1,SEM_UNDO};
    FILE *fp;
    ANAYAPI *adrp;
    KUYRUK *adrp2,l;
    GHTTAB *ght_buf,*ght_buf2;
    GCTTAB *gct_buf,*gct_buf2;
    FILE *kp;
    int k,kk,gtp,gcp,shmid3,semid,shmid,semid1,semid2,semid3,shmid1,shmid2,i,j;
    int shmid4,shmid5;
    long trans[100];
    /* shared memory allocate et */
    if((shmid=shmget((key_t)0x15,sizeof(ANAYAPI),0777|IPC_CREAT))==-1)
        printf("\n get olmadı \n");
    fp=fopen("transfer","w");
    fprintf(fp,"%d\n",shmid);
    /* ortak bellek tanıtıcısını diğer süreçlere disk dosyası üzerinden ilet */
    /* bu ortak belleğe attach ol */
    if((adrp=(ANAYAPI *)shmat(shmid,(void *)0,0))==(ANAYAPI *)(-1))
        printf("\n attach olmadı^2 \n");

    if((shmid1=shmget((key_t)0x25,sizeof(KUYRUK),0777|IPC_CREAT))==-1)
        printf("\n get olmadı \n");
    fprintf(fp,"%d\n",shmid1);
    /* ortak bellek tanıtıcısını
    diğer süreçlere disk dosyası üzerinden ilet */
    /* bu ortak belleğe attach ol */
    if((adrp2=(KUYRUK *)shmat(shmid1,(void *)0,0))==(KUYRUK *)(-1))
        printf("\n attach olmadı^2 \n");
```

```

if((shmid2=shmget((key_t)0x35,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
    printf("\n get olmadı \n");
fprintf(fp,"%d\n",shmid2);
/* ortak bellek tanıtıcısını
   diğer süreçlere disk dosyası üzerinden ilet */
/* bu ortak belleğe attach ol */
if((glt_buf=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
    printf("\n attach olmadı \n");

if((shmid3=shmget((key_t)0x45,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
    printf("\n get olmadı \n");
printf("shmid3 %d\n",shmid3);
fprintf(fp,"%d\n",shmid3);
/* ortak bellek tanıtıcısını
   diğer süreçlere disk dosyası üzerinden ilet */
/* bu ortak belleğe attach ol */
if((gct_buf=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
    printf("\n attach olmadı \n");

if((shmid4=shmget((key_t)0x55,sizeof(GHTTAB),0777|IPC_CREAT))==-1)
    printf("\n get olmadı \n");
fprintf(fp,"%d\n",shmid4);
/* ortak bellek tanıtıcısını
   diğer process e disk dosyası üzerinden ilet */
/* bu ortak belleğe attach ol */
if((glt_buf2=(GHTTAB *)shmat(shmid4,(void *)0,0))==(GHTTAB *)(-1))
    printf("\n attach olmadı \n");

if((shmid5=shmget((key_t)0x65,sizeof(GCTTAB),0777|IPC_CREAT))==-1)
    printf("\n get olmadı \n");
fprintf(fp,"%d\n",shmid5);
if((gct_buf2=(GCTTAB *)shmat(shmid5,(void *)0,0))==(GCTTAB *)(-1))
    printf("\n attach olmadı \n");
/* k gctable son girdisine point eder */
k=0;
kk=2; /* musvette yapıda gct 2 den itibaren başlar çünkü
        bu yapıya dönük sidentler - ile markalanır
        ve -1 in hareket dizini sonlandırma
        fonksiyonu nedeniyle birinci girdi kullanılamaz */

kp=fopen("k","r");
fscanf(kp,"%d\n",&k);
fscanf(kp,"%d\n",&kk);
printf("\nk=%d \n",k);
/* global veri yapılarını oku */

gtp=open("ghtable",O_CREAT|O_RDWR);
if (read(gtp,glt_buf,sizeof(GHTTAB))==-1);

```



```

close(gtp);
gcp=open("gctable",O_CREAT| O_RDWR);
if (read(gcp,gct_buf,sizeof(GCTTAB))==-1);
close(gcp);
gct_buf->k=k; /* sistem kapatma yönteminde
               bu değer ve shared memory'deki GHTTAB
               ve GCTTAB yapıları diske geriye yazılmalı */

printf("\n gct_buf->k=%d \n",gct_buf->k);
gct_buf2->k=kk;
for (i=0;i<=5000;i++) ght_buf2->po[i]=0;

if ((semid=semget((key_t)0x60,1,0600|IPC_CREAT))<0)
    printf("\n simafor set yaratilamadi \n");
if (semctl(semid,0,SETVAL,1)<0)
    printf("\n simafor baslangic deg almadı \n");
fprintf(fp,"%d\n",semid);

if ((semid1=semget((key_t)0x70,1,0600|IPC_CREAT))<0)
    printf("\n simafor set yaratilamadi \n");
if (semctl(semid1,0,SETVAL,1)<0)
    printf("\n simafor baslangic deg almadı \n");
fprintf(fp,"%d\n",semid1);

if ((semid2=semget((key_t)0x80,1,0600|IPC_CREAT))<0)
    printf("\n simafor set yaratilamadi \n");
if (semctl(semid2,0,SETVAL,1)<0)
    printf("\n simafor baslangic deg almadı \n");
fprintf(fp,"%d\n",semid2);

if ((semid3=semget((key_t)0x90,1,0600|IPC_CREAT))<0)
    printf("\n simafor set yaratilamadi \n");
if (semctl(semid3,0,SETVAL,1)<0)
    printf("\n simafor baslangic deg almadı \n");
fprintf(fp,"%d\n",semid3);

fclose(fp);

k=gct_buf->k;
kk=gct_buf2->k;
kp=fopen("k","w");
fprintf(kp,"%d\n",k);
fprintf(kp,"%d\n",kk);

/* global veri yapılarını oku */

gtp=open("ghtable",O_CREAT| O_RDWR);
if (write(gtp,ght_buf,sizeof(GHTTAB))==-1);
close(gtp);
gcp=open("gctable",O_CREAT| O_RDWR);
if (write(gcp,gct_buf,sizeof(GCTTAB))==-1);

```

```
close(gcp);
}
```

/* AŞAĞIDAKİ C PROGRAM PARÇALARI HER ÜRETİLEN HEDEF C PROGRAMININ BAŞINA STANDART OLARAK EKLENEN KOD PARÇALARIDIR */

/* CIKBAS.C */

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/msg.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/times.h>
#include </home/suha/final/anayapi.h>
#include </home/suha/final/kuyruk.h>
#include </home/suha/final/paket.h>
#include <sys/procset.h>
#include <sys/processor.h>
```

```
#define QKEY (key_t)0105
#define QPERM 0660
#define MAXOBN 5
struct q_entry{
    long mtype;
    char mtext[MAXOBN+1];
};
typedef struct{
    int po[5000];
} GHTTAB;
typedef struct{
    int k;
    char ct[5000];
} GCTTAB;
#define hashsize 5000
```

```
int tabloman(semid,bas,son,adrp,p,v)
struct sembuf *p;
struct sembuf *v;
int bas,son;
ANAYAPI *adrp;
int semid;
{
    int cik,i,pno;
    cik=1;
    while(cik){
        for (i=bas;i<=son&& cik!=0;i++){
```



```

    if (adrp->tablo[i][0] == 0) {
        semop(semid,p,1);
        adrp->tablo[i][0] = 1;
        semop(semid,v,1);
        pno=i;
        cik=0;
    }
}
}
return pno;
}

int kuybul(semid1,adrp,adrp2,p,v)
struct sembuf *p;
struct sembuf *v;
ANAYAPI *adrp;
KUYYRUK *adrp2;
int semid1;
{
    if (adrp->tablo[pno][0] == 0) {
        semop(semid1,p,1);
        adrp->tablo[pno][0] = 1;
        semop(semid1,v,1);
        yer=i;
        adrp2->q[i][100] = 2;
        adrp2->q[i][101] = 2;
        cik=0;
    }
}
}
return yer;
}

unsigned hash(char *s)
{
    unsigned hashval,temp;
    for (hashval=0;*s!='\0';s++)
        hashval = *s + 31*hashval;
    return hashval % hashsize;
}

int karsilas(char *t,char *ilk){
/* bu fonksiyon karşılaştırma amaçlı kullanılmaktadır */
    int n;
    n=0;
    for (;*ilk!='\0';ilk++){

```

```

    if (*ilk! = *t)n = 1;
    t++;
}
return(n);
}

```

/* Bu fonksiyon computes/comfixs/test deki halinden farklı olarak veri ght ve gct de bulunmadığı takdirde run time hatası verir */

```
long altyapi(GHTTAB *ght_buf, GCTTAB *gct_buf, char *tempbuf) {
```

```

int k, ii, bulunam, l, j, follow;
char temp[10];
unsigned hashval;
long sident;

```

```

    j = hashval = hash(tempbuf);
    if (ght_buf->po[hashval] == 0){
        printf("\n bu değer database'de yok, çalışma zamanı hatası \n");
        sident = 0;
        return sident;
    }

```

/*eğer hash entry 0 dan büyük ise daha önce buraya giriş yapılmış */

```

    else {
        j = hashval;
        follow = 0;
        /* follow traverse esnasında bir kayıt geriden
        gelen pointer'dır */
        l = ght_buf->po[j]; /* Bu pointer'ı al */
        sident = l;
        bulunam = karsilas(&gct_buf->ct[l+4], tempbuf);
        /*
        coll tablosundaki ilk değer ile karşılaştır */
        while (bulunam && (!l == 0)){
            for(ii = 0; ii <= 3; ii++){
                temp[ii] = gct_buf->ct[l+ii];
            }
            follow = l;
            l = acpaket(temp);

            /* pointer'ı paketlenmiş olduğu karakter dizisi
            formundan tamsayı formuna aç ve bu pointer'ı
            takip ederek coll linkte bir sonraki kayıda
            geç */
            sident = l;
            if (l > 0){
                bulunam = karsilas(&gct_buf->ct[l+4], tempbuf);
            }
        }
    }
}

```



```

    }
    if (l==0&&bulunam){ /* collision bucket'in sonuna
                        erişti ama bulamadı*/
        sident=0;
        printf("\n bu değer database'de yok, calisma zamanı hatası \n");
        return sident;
    }
    if (bulunam==0) /* bulundu sident
                    doğru değere set edildi*/
        return sident;
} /* end else == if ght_buf->po[hashval]>0 */

} /* function altiyapi */

```

```

/* shared memory allocate et */
/* times&timebuf;
printf("Ana surec baslangic %d \n",ilk);
/* bu orak belige atash ol */

main(){
    struct sembuf p={0,-1,SEM_UNDO};
    struct sembuf v={0,1,SEM_UNDO} ;
    struct tms timebuf;
    ANAYAPI *adrp;
    KUYRUK *adrp2;
    GHTTAB *ght_buf;
    GCTTAB *gct_buf;
    FILE *pp;
    struct q_entry s_entry;
    int queue_id,rlen; /* mesaj kuyruk tanitıcısı */
    int bas,s,m,son,yer,pno,cik,serbmax=200,blokmax=3;
    int k,kk,gtp,gcp,shmid3,semid,shmid,semid1,semid2,semid3,shmid1,shmid2,i,j;
    int shmid4,shmid5;
    int blokmap[MAXBLOK][20],kuymap[MAXBLOK][20],girkuy[MAXBLOK];

```

/* blokmap blok numaralari ile ilişkili , kuymap kuyruk noları ile ilişkili
girkuy sadece MERGE blogu icin gecerli olan kuymap'in standart yapisi
dışında bu özel durum için birden fazla giriş kuyruğunun merge edilmek
uzere MERGE bloguna "margeinc" içinde pass edilecek sekilde giriş
kuyruk no'larının yerleştirildiği geçici veri yapısıdır */

```

    int bloksay,yeri,bul,progbloks;
    int ilk,iki;
    long sident;
    char tempbuf[100];
    float cc;

```

```

pid_t pid;
processorid_t gelen;

```

```

idtype_t p_pid;
long pnum;
int ka;

pnum=0;
pid=getpid();
p_pid=getppid();
ka=processor_bind(P_PID,pid,pnum,&gelen);

```

```

/* CIKBAS1.C */

```

```

/* shared memory allocate et */
ilk=times(&timebuf);
printf("\n Ana surec baslangic %d \n",ilk);
/* bu ortak belleğe attach ol */

if((adrp=(ANAYAPI *)shmat(shmid,(void *)0,0))==(ANAYAPI *)(-1))
printf ("\n attach olmadı \n");

/* bu ortak belleğe attach ol */
if((adrp2=(KUYRUK *)shmat(shmid1,(void *)0,0))==(KUYRUK *)(-1))
printf ("\n attach olmadı \n");

if((ght_buf=(GHTTAB *)shmat(shmid2,(void *)0,0))==(GHTTAB *)(-1))
printf ("\n attach olmadı \n");
if((gct_buf=(GCTTAB *)shmat(shmid3,(void *)0,0))==(GCTTAB *)(-1))
printf ("\n attach olmadı \n");

/* ght ve gct ye erisilerek zaten orada bulunması gereken sident deęerleri
bulunup compute/comfix/test bloklarındaki locate'de eřitlik sartı iinde
yer aldıkları noktaya yerlestirilecek bu nedenle bu segmentlere de
attach olunuyor */
if ((queue_id= msgget(QKEY,IPC_CREAT|QPERM))===-1)
perror("msg get olmadı");
/* srelere uyarı mesajı gnderme yapısını oluřtur */

bloksay=1;

/* CIKBAS2.C */

for (i=1;i<=progbloks;i++){

if (blokmap[i][2]<0){ /* merge veya cond bloęu ise */
yer=kuybul(semid1,adrp,adrp2,&p,&v);
kuymap[i][1]=yer;
for (k=2;k<=20&&blokmap[i][k]!=0;k++){

```



```

bul=0;
for (j=1;j<=20&&bul!=1;j++){
    if(blokmap[blokmap[i][k]*(-1)][j]==i){
        bul=1;
        girkuy[m]=kuymap[blokmap[i][k]*(-1)][j];
        blokmap[blokmap[i][k]*(-1)][j]=800000000; /* körelt */
        /* aynı transfer bloğundan aynı merge
        bloğuna kazai transferleri iki
        ayrı paralel kuyruktan yaparak
        hata vermesi önleniyor */
        m++;
        yeri=j;
    } /* if */
} /* for */
if (bul==0){ /* bu bloğa input sağlayan bloğun blokmap
girdisi içinde bu bloğu işaret eden
bir çıkış girdisi bulunamadı blok map
istikrarsız */
    printf("\n çalışma zamanı hatası blok map \n");
    exit(-1);
} /* kuymap de karşı gelen noktadaki kuyruk numarasını al
ve bu bloğun kuymap'deki giriş kuyruğu hanesine işle */
} /* for k= blokmap satır içinde ilerleme
bir blok birden fazla çıkış veriyorsa bu sayac 1 den
ileri gidebilir. */
}else{ /*merge bloğu değil ise */
    for (k=1;k<=20&&blokmap[i][k]!=0;k++){
        yer=kuybul(semid1,adrp,adrp2,&p,&v);
        if(i==1 || blokmap[i][0]<=0) kuymap[i][1]=yer;
        else{ /* ilk blok değil ise buna input sağlayan bloğu bulup
        bunun input alacağı kuyruğun blokmapdeki kuymap'e
        karşı gelen noktasını bul */
            bul=0;
            for (j=1;j<=20&&bul!=1;j++){
                if(blokmap[blokmap[i][0]][j]==i){
                    bul=1;
                    yeri=j;
                }
            }
        }
        if (bul==0){ /* bu bloğa input sağlayan bloğun blokmap
        girdisi içinde bu bloğu işaret eden
        bir çıkış girdisi bulunamadı blok map
        istikrarsız */
            printf("\n çalışma zamanı hatası blok map \n");
            exit(1);
        } /* kuymap'de karşı gelen noktadaki kuyruk numarasını al
        ve bu bloğun kuymap'deki giriş kuyruğu hanesine işle */
        kuymap[i][0]=kuymap[blokmap[i][0]][yeri];
        kuymap[i][k]=yer; /* bu bloğun yeni elde edilmiş olan
        çıkış kuyruk no/nolarını yerleştir */
    } /* else */
}

```

```

} /* for k= blokmap satir icinde ilerleme
    bir blok birden fazla çıkış veriyorsa bu sayaç 1 den
    ileri gidebilir. */
} /* else merge bloğu değil */
} /* for i= blok sayacı */
/* bu noktada bu programın ihtiyacı olan tüm kuyruklar allocate edildi

    ve kuymap içinde yer aldı */

bloksay=1;

/* AŞAĞIDAKİ C PROGRAM PARÇALARI HEDEF C PROGRAMINA HER BLOK TİPİNE GÖRE
HER BLOK İÇİN AYRI AYRI YERLEŞTİRİLEN STANDART KOD PARÇALARIDIR */

/* APPEND BLOĞU STANDART C KOD PARÇASI */

bas=46;
son=48; /* süreç tablosunda bu cins sürece ayrılan yerin
    başı ve sonu */

pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][2]=kuymap[bloksay][0];
adrp->tablo[pno][3]=kuymap[bloksay][1];
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
    /* eğer bu entry 0 değilse bu test bloğudur dolayısıyla
    process tablosundaki 1 numaralı girdi alternatif
    kuyruğu içermelidir */
/*mesaj bloğu bul işle */
cik=1;
while(cik){
    for (i=0;i<=blokmax&&cik!=0;i++){
        if (adrp->updtubit[i]==0){
            semop(semid2,&p,1);
            adrp->updtubit[i]=1;
            semop(semid2,&v,1);

            yer=i;
            adrp->tablo[pno][5]=yer;
            cik=0;
        }
    }
}

bloksay+=1;

/* APPLY BLOĞU STANDART C KOD PARÇASI */

bas=13;
son=15; /* süreç tablosunda bu cins sürece ayrılan yerin
    başı ve sonu */

```



```

pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][2]=kuymap[bloksay][0];
adrp->tablo[pno][3]=kuymap[bloksay][1];
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
    /* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
       süreç tablosundaki 1 numaralı girdi alternatif
       kuyruğu içermelidir */
/*mesaj bloğu bul işle */
cik=1;
while(cik){
    for (i=0;i<=blokmax&&cik!=0;i++){
        if (adrp->applyubit[i]==0){
            semop(semid2,&p,1);
            adrp->applyubit[i]=1;
            semop(semid2,&v,1);
            yer=i;
            adrp->tablo[pno][5]=yer;
            cik=0;
        }
    }
}
bloksay+=1;

```

/* APPLY CUMULATIVE BLOĞU STANDART C KOD PARÇASI */

```

bas=61;
son=63; /* süreç tablosunda bu cins sürece ayrılan yerin
        başı ve sonu */

```

```

pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][2]=kuymap[bloksay][0];
adrp->tablo[pno][3]=kuymap[bloksay][1];
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
    /* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
       süreç tablosundaki 1 numaralı girdi alternatif
       kuyruğu içermelidir */
/*mesaj bloğu bul işle */
cik=1;
while(cik){
    for (i=0;i<=blokmax&&cik!=0;i++){
        if (adrp->cumubit[i]==0){
            semop(semid2,&p,1);
            adrp->cumubit[i]=1;
            semop(semid2,&v,1);

```

```

        yer=i;
        adrp->tablo[pno][5]=yer;
        cik=0;
    }
}
}

```

```
bloksay += 1;
```

```
/* CARTESIAN PRODUCT BLOĞU STANDART C KOD PARÇASI */
```

```

bas=43;
son=45; /* süreç tablosunda bu cins sürece ayrılan yerin
        başı ve sonu */

```

```

pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][2]=kuymap[bloksay][0];
adrp->tablo[pno][3]=kuymap[bloksay][1];
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
    /* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
        süreç tablosundaki 1 numaralı girdi alternatif
        kuyruğu içermelidir */

```

```
/*mesaj bloğu bul işle */
```

```

cik=1;
while(cik){
    for (i=0;i <=blokmax&& cik!=0;i++){
        printf("\n mesaj blok \n");
        if (adrp->cpubit[i]==0){
            semop(semid2,&p,1);
            adrp->cpubit[i]=1;
            semop(semid2,&v,1);
            yer=i;
            adrp->tablo[pno][5]=yer;
            cik=0;
        }
    }
}
bloksay += 1;

```

```
/* COUNTRY BLOĞU STANDART C KOD PARÇASI */
```

```

bas=31;
son=33; /* süreç tablosunda bu cins sürece ayrılan yerin
        başı ve sonu */

```

```

pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][2]=kuymap[bloksay][0];
adrp->tablo[pno][3]=kuymap[bloksay][1];
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
    /* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
        süreç tablosundaki 1 numaralı girdi alternatif

```



```

/* Kuyruğu içermelidir */
/* mesaj bloğu bul işle */
cik=1;
while(cik){
  for (i=0; i <= blokmax && cik!=0; i++){
    if (adrp->cntubit[i]==0){
      semop(semid2,&p,1);
      adrp->cntubit[i]=1;
      semop(semid2,&v,1);
      yer=i;
      adrp->tablo[pno][5]=yer;
      cik=0;
    }
  }
}

bloksay += 1;

/* COMPUTE FIX BLOĞU STANDART C KOD PARÇASI */

bas=19;
son=20; /* süreç tablosunda bu cins sürece ayrılan yerin
baş ve sonu */

pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][2]=kuymap[bloksay][0];
adrp->tablo[pno][3]=kuymap[bloksay][1];
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
/* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
süreç tablosundaki 1 numaralı girdi alternatif
kuyruğu içermelidir */
/* mesaj bloğu bul işle */
cik=1;
while(cik){
  for (i=0; i <= blokmax && cik!=0; i++){
    if (adrp->compufbit[i]==0){
      semop(semid2,&p,1);
      adrp->compufbit[i]=1;
      semop(semid2,&v,1);
      yer=i;
      adrp->tablo[pno][5]=yer;
      cik=0;
    }
  }
}

bloksay += 1;

```

```
/* COMPUTE BLOĞU STANDART C KOD PARÇASI */
```

```
bas=16;
son=18; /* süreç tablosunda bu cins sürece ayrılan yerin
        başı ve sonu */

pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][2]=kuymap[bloksay][0];
adrp->tablo[pno][3]=kuymap[bloksay][1];
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
/* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
   süreç tablosundaki 1 numaralı girdi alternatif
   kuyruğu içermelidir */
/*mesaj bloğu bul işle */
cik=1;
while(cik){
  for (i=0;i<=blokmax&&cik!=0;i++){
    if (adrp->compubit[i]==0){
      semop(semid2,&p,1);
      adrp->compubit[i]=1;
      semop(semid2,&v,1);

      yer=i;
      adrp->tablo[pno][5]=yer;
      cik=0;
    }
  } /* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
     süreç tablosundaki 1 numaralı girdi alternatif
     kuyruğu içermelidir */
  bloksay+=1;
  cik=1;
  while(cik){
    for (i=0;i<=blokmax&&cik!=0;i++){
      if (adrp->compubit[i]==0)
```

```
/* COND BLOĞU STANDART C KOD PARÇASI */
```

```
bas=55;
son=57; /* süreç tablosunda bu cins sürece ayrılan yerin
        başı ve sonu */
```

```
pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][3]=kuymap[bloksay][1];
cik=1;
while(cik){
  for (i=0;i<=blokmax&&cik!=0;i++){
    if (adrp->cndubit[i]==0){
      semop(semid2,&p,1);
      adrp->cndubit[i]=1;
      semop(semid2,&v,1);

      yer=i;
```



```

adrp->tablo[pno][5]=yer;
cik=0;
}
}
}

adrp->cndrefqid[yer]=girkuy[0];
adrp->cndinpqid[yer]=girkuy[1];
if (girkuy[0]==0 || girkuy[1]==0){
    printf("\n COND kuyruk dagitiminda hata (run time) devam edilemez \n");
    exit(-1);
}

bloksay += 1;

```

/* DELETE BLOĞU STANDART C KOD PARÇASI */

```

bas=49;
son=51; /* process tablosunda bu cins porcess e ayrılan yerin
        basi ve sonu */

pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][2]=kuymap[bloksay][0];
adrp->tablo[pno][3]=kuymap[bloksay][1];
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
/* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
   süreç tablosundaki 1 numaralı girdi alternatif
   kuyruğu içermelidir */

/*mesaj bloğu bul işle */
cik=1;
while(cik){
    for (i=0;i <= blokmax&& cik!=0;i++){
        if (adrp->delubit[i]==0){
            semop(semid2,&p,1);
            adrp->delubit[i]=1;
            semop(semid2,&v,1);
            yer=i;
            adrp->tablo[pno][5]=yer;
            cik=0;
        }
    }
}

bloksay += 1;

```

/* FILTER BLOĞU STANDART C KOD PARÇASI */

```

bas=40;
son=42; /* süreç tablosunda bu cins sürece ayrılan yerin
        başı ve sonu */

pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][2]=kuymap[bloksay][0];
adrp->tablo[pno][3]=kuymap[bloksay][1];
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
/* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
   süreç tablosundaki 1 numaralı girdi alternatif
   kuyruğu içermelidir */
/*mesaj bloğu bul işle */
cik=1;
while(cik){
    for (i=0;i<=blokmax&&cik!=0;i++){
        if (adrp->fltubit[i]==0){
            semop(semid2,&p,1);
            adrp->fltubit[i]=1;
            semop(semid2,&v,1);
            yer=i;
            adrp->tablo[pno][5]=yer;
            cik=0;
        }
    }
}
bloksay+=1;

/* GENERATE BLOĞU STANDART C KOD PARÇASI */

bas=1;
son=3; /* süreç tablosunda bu cins sürece ayrılan yerin
        başı ve sonu */

pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][2]=kuymap[bloksay][0];
adrp->tablo[pno][3]=kuymap[bloksay][1];
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
/* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
   süreç tablosundaki 1 numaralı girdi alternatif
   kuyruğu içermelidir */
/*mesaj bloğu bul işle */
cik=1;
while(cik){
    for (i=0;i<=blokmax&&cik!=0;i++){
        printf("\n mesaj blok \n");
        if (adrp->genubit[i]==0){
            semop(semid2,&p,1);
            adrp->genubit[i]=1;

```



```

semop(semid2,&v,1);
adrp->tablo[pno][5]=yer;
cik=0;
}
}
}
bloksay += 1;

/* GENERATE TUPLE BLOĞU STANDART C KOD PARÇASI */

bas=7;
son=9; /* süreç tablosunda bu cins sürece ayrılan yerin
        başı ve sonu */

pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][2]=kuymap[bloksay][0];
adrp->tablo[pno][3]=kuymap[bloksay][1];
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
/* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
   süreç tablosundaki 1 numaralı girdi alternatif
   kuyruğu içermelidir */
/*mesaj bloğu bul işle */
cik=1;
while(cik){
    for (i=0;i<=blokmax&& cik!=0;i++){
        printf("\n mesaj blok \n");
        if (adrp->gtubit[i]==0){
            semop(semid2,&p,1);
            adrp->gtubit[i]=1;
            semop(semid2,&v,1);
            yer=i;
            adrp->tablo[pno][5]=yer;
            cik=0;
        }
    }
}
bloksay += 1;

/* GENERATE ELEMENT BLOĞU STANDART C KOD PARÇASI */

bas=4;
son=6; /* süreç tablosunda bu cins sürece ayrılan yerin
        başı ve sonu */

```

```

pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][2]=kuymap[bloksay][0];
adrp->tablo[pno][3]=kuymap[bloksay][1];
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
/* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
   süreç tablosundaki 1 numaralı girdi alternatif
   kuyruğu içermelidir */
/*mesaj bloğu bul işle */
cik=1;
while(cik){
for (i=0;i<=blokmax&&cik!=0;i++){
if (adrp->inpusbbit[i]==0){
semop(semid2,&p,1);
adrp->adrp->inpusbbit[i]=1;
semop(semid2,&v,1);
while(cik){
yer=i;
adrp->tablo[pno][5]=yer;
cik=0;
}
adrp->mergoubit[i]=1;
semop(semid2,&v,1);
}
}
}
bloksay+=1;

```

/* INPUT BLOĞU STANDART C KOD PARÇASI */

```

bas=22;
son=24; /* süreç tablosunda bu cins sürece ayrılan yerin
        başı ve sonu */
pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][2]=kuymap[bloksay][0];
adrp->tablo[pno][3]=kuymap[bloksay][1];
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
/* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
   süreç tablosundaki 1 numaralı girdi alternatif
   kuyruğu içermelidir */
/*mesaj bloğu bul işle */
cik=1;
while(cik){
for (i=0;i<=blokmax&&cik!=0;i++){
if (adrp->inpubit[i]==0){
semop(semid2,&p,1);
adrp->adrp->inpubit[i]=1;
semop(semid2,&v,1);
yer=i;
adrp->tablo[pno][5]=yer;
cik=0;
}
}
}

```



```

if ( }
}
}

bloksay += 1;

/* MERGE BLOĞU STANDART C KOD PARÇASI */

bas=64;
son=66; /* süreç tablosunda bu cins sürece ayrılan yerin
        başı ve sonu */

pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][3]=kuymap[bloksay][1];
cik=1;
while(cik){
    for (i=0;i<=blokmax&&cik!=0;i++){
        if (adrp->mergeubit[i]==0){
            semop(semid2,&p,1);
            adrp->mergeubit[i]=1;
            semop(semid2,&v,1);

            yer=i;
            adrp->tablo[pno][5]=yer;
            cik=0;
        }
    }
}
k=0;
while (k<m){
    adrp->mergeqids[yer][k]=girkuy[k];
    k++;
}
if (k==0){
    printf("\n MERGE kuyruk dagitiminda hata (run time) devam edilemez \n");
    exit(-1);
}
adrp->mergeadrs[yer]=k+1;

bloksay += 1;

/* NOTMEMBER BLOĞU STANDART C KOD PARÇASI */

bas=58;
son=60; /* process tablosunda bu cins porcess e ayrılan yerin
        başı ve sonu */

pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][2]=kuymap[bloksay][0];
adrp->tablo[pno][3]=kuymap[bloksay][1];

```

```
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
```

```
/* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
   süreç tablosundaki 1 numaralı girdi alternatif
   kuyruğu içermelidir */
```

```
/*mesaj bloğu bul işle */
```

```
cik=1;
```

```
while(cik){
```

```
for (i=0;i <= blokmax&& cik!=0;i++){
```

```
if (adrp->memubit[i]==0){
```

```
semop(semid2,&p,1);
```

```
adrp->memubit[i]=1;
```

```
semop(semid2,&v,1);
```

```
adrp->tablo[pno][5]=yer;
```

```
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
```

```
/* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
```

```
   süreç tablosundaki 1 numaralı girdi alternatif
```

```
   kuyruğu içermelidir */
```

```
/*mesaj bloğu bul işle */
```

```
cik=1;
```

```
while(cik){
```

```
bloksay += 1;
```

```
if (adrp->memubit[i]==0){
```

```
semop(semid2,&p,1);
```

```
adrp->memubit[i]=1;
```

```
/* REPORT BLOĞU STANDART C KOD PARÇASI */
```

```
semop(semid2,&v,1);
```

```
bas=25;
```

```
son=27; /* süreç tablosunda bu cins sürece ayrılan yerin
```

```
   başı ve sonu */
```

```
pno=tabloman(semid,bas,son,adrp,&p,&v);
```

```
adrp->tablo[pno][2]=kuymap[bloksay][0];
```

```
adrp->tablo[pno][3]=kuymap[bloksay][1];
```

```
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
```

```
/* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
```

```
   süreç tablosundaki 1 numaralı girdi alternatif
```

```
   kuyruğu içermelidir */
```

```
/*mesaj bloğu bul işle */
```

```
cik=1;
```

```
while(cik){
```

```
for (i=0;i <= blokmax&& cik!=0;i++){
```

```
if (adrp->outpubit[i]==0){
```

```
semop(semid2,&p,1);
```

```
adrp->outpubit[i]=1;
```

```
semop(semid2,&v,1);
```

```
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
```

```
/* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
```

```
   süreç tablosundaki 1 numaralı girdi alternatif
```

```
   kuyruğu içermelidir */
```

```
/*mesaj bloğu bul işle */
```

```
cik=1;
```

```
while(cik){
```



```

}
bloksay += 1;

/* SINK BLOĞU STANDART C KOD PARÇASI */

bas=37;
son=39; /* süreç tablosunda bu cins sürece ayrılan yerin
        başı ve sonu */

pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][2]=kuymap[bloksay][0];
adrp->tablo[pno][3]=kuymap[bloksay][1];
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
/* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
   süreç tablosundaki 1 numaralı girdi alternatif
   kuyruğu içermelidir */
/*mesaj bloğu bul işle */
cik=1;
while(cik){
    for (i=0;i<=blokmax&&cik!=0;i++){
        if (adrp->sinkubit[i]==0){
            semop(semid2,&p,1);
            adrp->sinkubit[i]=1;
            semop(semid2,&v,1);
            yer=i;
            adrp->tablo[pno][5]=yer;
            cik=0;
        }
    }
}
bloksay += 1;

```

/* SORT BLOĞU STANDART C KOD PARÇASI */

```

bas=28;
son=30; /* süreç tablosunda bu cins sürece ayrılan yerin
        başı ve sonu */

pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][2]=kuymap[bloksay][0];
adrp->tablo[pno][3]=kuymap[bloksay][1];
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
/* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
   süreç tablosundaki 1 numaralı girdi alternatif
   kuyruğu içermelidir */
/*mesaj bloğu bul işle */
cik=1;

```

```

while(cik){
    for (i=0;i<=blokmax&&cik!=0;i++){
        if (adrp->sortubit[i]==0){
            semop(semid2,&p,1);
            adrp->sortubit[i]=1;
            semop(semid2,&v,1);

            yer=i;
            adrp->tablo[pno][5]=yer;
            cik=0;
        }
    }
}

bloksay += 1;

/* TEST BLOĞU STANDART C KOD PARÇASI */

bas=10;
son=12; /* süreç tablosunda bu cins sürece ayrılan yerin
        başı ve sonu */

pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][2]=kuymap[bloksay][0];
adrp->tablo[pno][3]=kuymap[bloksay][1];
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
/* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
   süreç tablosundaki 1 numaralı girdi alternatif
   kuyruğu içermelidir */
/*mesaj bloğu bul işle */
cik=1;
while(cik){
    for (i=0;i<=blokmax&&cik!=0;i++){
        if (adrp->testubit[i]==0){
            semop(semid2,&p,1);
            adrp->testubit[i]=1;
            semop(semid2,&v,1);

            yer=i;
            adrp->tablo[pno][5]=yer;
            cik=0;
        }
    }
}

bloksay += 1;

/* TRANSFER BLOĞU STANDART C KOD PARÇASI */

bas=34;
son=36; /* süreç tablosunda bu cins sürece ayrılan yerin

```


başı ve sonu */

```
pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][2]=kuymap[bloksay][0];
cik=1;
while(cik){
    for (i=0;i<=blokmax&&cik!=0;i++){
        if (adrp->splitubit[i]==0){
            semop(semid2,&p,1);
            adrp->splitubit[i]=1;
            semop(semid2,&v,1);
            yer=i;
            adrp->tablo[pno][5]=yer;
            cik=0;
        }
    }
}
k=1;
while (kuymap[bloksay][k]!=0){
    adrp->spltqids[yer][k-1]=kuymap[bloksay][k];
    k++;
}
if (k==1){
    printf("\n Transfer kuyruk dağıtımında hata (run time) devam edilemez \n");
    exit(-1);
}
adrp->spltadrs[yer]=k-1;

bloksay+=1;
```

/* TXCNT BLOĞU STANDART C KOD PARÇASI */

```
bas=52;
son=54; /* süreç tablosunda bu cins sürece ayrılan yerin
        başı ve sonu */
```

```
pno=tabloman(semid,bas,son,adrp,&p,&v);
adrp->tablo[pno][2]=kuymap[bloksay][0];
adrp->tablo[pno][3]=kuymap[bloksay][1];
if (kuymap[bloksay][2]!=0) adrp->tablo[pno][1]=kuymap[bloksay][2];
    /* eğer bu girdi 0 değilse bu test bloğudur dolayısıyla
       süreç tablosundaki 1 numaralı girdi alternatif
       kuyruğu içermelidir */
/*mesaj blogu bul isle */
cik=1;
while(cik){
    for (i=0;i<=blokmax&&cik!=0;i++){
        if (adrp->txsubit[i]==0){
            semop(semid2,&p,1);
            adrp->txsubit[i]=1;
```


EK-5 LUGATÇE

Bu bölümde tez içinde yer alan Türkçe Bilgisayar Terimlerinin, nisbeten az yerleşmiş olanlarının İngilizce karşılıkları verilmektedir.

Amaç	Goal
Anlambilim	Semantic
Biçimbirimsel	Lexical
Bildirim	Declaration, to Declare.
Boş veri	Null
Çarpışma Tablosu	Collision Table
Dağıtık	Distributed
Dağıtılmış	Distributed
Dağıtım Kodlama Tablosu	Hash Table
Dağıtım Kodlaması	Hash Coding, Hashing
Değer Geçirme	Call by Value
Değişken Geçerlilik Bölgesi	Scope
Eş Zamanlama	Synchronization
Eşitlik Şartı ile Katışım	Natural Join
İcra zinciri	Execution Sequence
İcra etme	to Execute
İcra	Execution
İlişki Kaydı	Tuple
İlişki Sahası	Attribute
İşaretçi	Pointer
İsimlendirme Ortamı	Context
İşleme Giren	Operand
Kapsam	Scope
Katışım	Join
Kayıt	Tuple
Kesişim	Intersection
Mantıksal	Logical
Miras	Inheritance
Müsvette	Scratch
Nesne	Object
Nesne yönelimli	Object Oriented
Nesne Temelli	Object Based
Referans Geçirme	Call by Reference
Saha	Attribute
Simge	Token
Sınıf	Class
Sözdizimsel	Syntactic
Tanım Kümesi	Domain
Uç uca ekleme	Concatenation
Yan Etki	Side Effect
Yığıt	Stack
Yol	Bus
Zamanuyumlu	Synchronous
Zamanuyumsuz	Asynchronous

ÖZGEÇMİŞ

ADI, SOYADI	Süha FUTACI
DOĞUM YERİ VE YILI	Bursa 1956
ORTA ÖĞRENİM	Bursa Namık Sözeri Koleji Bursa.(1970) Deniz Lisesi Heybeliada-İstanbul.(1973)
YÜKSEK ÖĞRENİM	Deniz Harp Okulu Heybeliada-İstanbul.(1976)
ÖĞRENİM SONRASI	- Deniz Kuvvetleri Komutanlığı Derince Elektrik-Elektronik Okulu.(1977) - Deniz Kuvvetleri Komutanlığı Gölcük Tersanesi Komutanlığı Bilgisayar ve Sayısal Devreler Kursu(1978) -Deniz Kuvvetleri Komutanlığı Yabancı Dil Okulu, İngilizce Kursu(1979) -Öğretim Tekniği Kursu. Deniz Harp Okulu-1984
LİSANS ÜSTÜ	Naval Postgraduate School Monterey-California-USA. (MS in Computer Science) (1982)

İŞ DENEYİMLERİ

- TCG GAYRET Muhribi Silah Elektronik Subayı
- Dz.K.K. Gölcük Tersanesi K.lığı Araştırma Geliştirme Baş Mühendisliği/SAPAN Atış Kontrol Sistemi Projesi.
- TCG ANITTEPE Muhribi Savaş Harekat Merkezi Subayı.
- Deniz Harp Okulu, Fen Bilimleri Bölüm Başkanlığı Programla Dersi Öğretim üyesi.
- Deniz Harp Okulu, Obim Sistem İşletme Mühendisi, ek görevle öğretim üyesi.(Programlama dersi)
- Deniz Harp Okulu, Obim Müdürü, ek görevli öğretim üyesi. (Programlama, İleri Programlama ve Programlama Dilleri dersleri.)
- Setra A.Ş. Genel Müdür Yardımcısı. (Teknik)
- Yıldız Teknik Üniversitesi Bilgisayar Bilimleri Mühendisliği Bölümü Saat Ücretli Öğretim Görevlisi. (File Organization ve Data Base dersleri)
- Setra A.Ş. Genel Müdürü.

BİLDİRİLER

- "Bilgisayar Destekli Eğitimde Benzetim Uygulamaları", Eğitim Teknolojisi ve Bilgisayar Destekli Eğitim I. Sempozyumu, Ankara Üniversitesi, Ankara, Bildiriler Kitabı, 1991.
- "Dağıtım Bilgisayar Sistemlerinde Yönelik Programlama Dili", Bilgisayar Bilimleri Kitabı, 1991.

YAYIN	- "Relational Languages and the Worst Case Asymptotical Time Complexity Analysis of the Related Algorithms.", MS Thesis, Naval Postgraduate School- Monterey California-USA. 1982. - "Deniz Harp Okulu Programlama Dersi Fortran IV Çabuk Referans Kitabı", Süha FUTACI, Metin YAVAŞ. Deniz Harp Okulu Basımevi 1984. - "Mikrobilgisayarlar", Deniz Kuvvetleri Dergisi, Ocak 1984, Cilt-100, Sayı-524. - "Kompüterize Silah Sistemlerinde Yazılımın Önemi", Deniz Kuvvetleri Dergisi, Temmuz 1984, Cilt 100, Sayı-526. - "Bilgisayarla Benzetim ve Kullanılan Programlama Dilleri", Sanyai Mühendisliği, Ocak 1985, Cilt 4 0 Sayı 13. - "Bilgisayar Temelleri", (Ders Kitabı). Deniz Harp Okulu Basımevi.1989. - "Bilgisayar Yazılımı", (Ders Kitabı). Deniz Harp Okulu Basımevi.1989.
BİLDİRİLER	- "Bilgisayar Destekli Eğitimde Benzetim Uygulamaları", Eğitim Teknolojisi ve Bilgisayar Destekli Eğitim 1.Sempozyumu, Anadolu Üniversitesi-Eskişehir. Bildiriler Kitabı. 1991. - "Dağıtılmış Bilgisayar Sistemlerine Yönelik Programlama Dilleri", Bilişim'93 Bildiri Kitabı. 1993.

TAKDİRNAMELER

- Naval Postgraduate School
Monterey CA. Academic Dean Letter
NC4(01)tyc, 27 Feb. 1981. Inclusion
of the Dean's List- Outstanding
Academic Performance.

- Naval Postgraduate School-
Monterey CA. Academic Dean Letter
NC4(01), 11 May 1981. Inclusion of
the Dean's List- Outstanding
Academic Performance.

- Naval Postgraduate School-
Monterey CA. Academic Dean Letter
NC4(0301)tyc, 6 Aug. 1981.
Inclusion of the Dean's List
Outstanding Academic Performance.

- Naval Postgraduate School-
Monterey CA. Academic Dean Letter
NC4(0301), 20 Nov. 1981. Inclusion
of the Dean's List-Outstanding
Academic Performance.

- Naval Postgraduate School-
Monterey CA. Academic Dean Letter
NC4(0301)tyc, 2 Feb. 1982.
Inclusion of the Dean's List
Outstanding Academic Performance.

- Naval Postgraduate School-
Monterey CA. Academic Dean Letter
NC4(0301)tyc, 6 May. 1982.
Inclusion of the Dean's List
Outstanding Academic Performance.

