

39198

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

ÇOK İŞLEMCİLİ SİSTEMLER İÇİN

BİR YOK AKIS DİLİ

YÜKSEK LİSANS TEZİ

Müh. A. Olcay AKGÜN

Tezin Enstitüye Verildiği Tarih: 25 Ocak 1993

Tezin Savunulduğu Tarih : 8 Temmuz 1993

Tez Danışmanı

: Prof.Dr. Emre HARMANCI

Diğer Jüri Üyeleri

: Prof.Dr. Nadir YÜCEL

Doç.Dr. Bülent ÖRENCİK

TEMMUZ 1993

ÖNSÖZ

Çalışmamın tamamlanması için beni teşvik eden, her aşamada düşünceleri ile destek olan danışman hocam sayın Prof.Dr.A.Emre Harmancı'ya içten teşekkürlerimi sunarım.

Çalışmamın gerçekleşmesinde değerli önerileri, düşünceleri ve sabrı ile beni destekleyen hocam ve eşim Y.Doç.Dr.B.Tevfik Akgün'e sevgi ve saygıyla teşekkür ederim.

Çalışma sırasında ilgisini esirgemeyen hocam Doç.Dr. Nadia Erdogan'a teşekkür ederim. Ve yine, çalışmalarımın en yoğun günlerinde gösterdikleri ilgi ve katkıları nedeniyle İ.T.Ü. Bilgi İşlem Merkezi'ndeki çalışma arkadaşlarıma teşekkür etmek isterim.

22 Ocak 1993
İstanbul

Ayşe Olcay Akgün

İÇİNDEKİLER

ÖZET	VII
SUMMARY	VIII
BÖLÜM 1. GİRİŞ	1
1.1 Çalışmanın Amacı	1
1.2 Çalışmanın Kapsamı	2
BÖLÜM 2. FADOS DAĞINIK GERÇEK ZAMANLI İŞLETİM SİSTEMİ	6
2.1 İşletim Sisteminin Özellikleri	7
2.2 İşletim Sisteminin Gerçeklenmesi	9
2.3 Uygulama	10
2.4 MPWFL Diline Yönelik Belirlenen İşlevler ..	13
BÖLÜM 3. MPWFL DİLİ	15
3.1 MPWFL Dilinin Yetenekleri	15
3.2 MPWFL Dilinin Özellikleri	17
3.2.1 Program Yapısı	18
3.2.1.1 Programın Genel Yapısı	18
3.2.1.2 Değişken Tanımlama	19
3.2.1.3 Altprogramlar	20
3.2.1.4 Veri Tipleri	21
3.2.1.4.1 Mantıksal Veri Tipi	21
3.2.1.4.2 Tamsayı Veri Tipi	21
3.2.1.4.3 Katar Veri Tipi	21
3.2.1.4.4 Veri Tamponu Değişken Tipi	22
3.2.1.4.5 Görev Değişken Tipi	22
3.2.1.4.6 İş Sahibi Değişkeni Tipi	23
3.2.1.4.7 Grup Değişkeni Tipi	23
3.2.2 Komut Kümesi	25
3.2.2.1 Atama Komutları	25
3.2.2.1.1 Mantıksal Değişkenlere Atama ..	25
3.2.2.1.2 Tamsayı Değişkenlere Atama	26
3.2.2.1.3 Katar Değişkenlerine Atama	27
3.2.2.1.4 Görev Değişkenlerine Atama	27
3.2.2.1.5 Veri Tamponu Değişkenlerine	
Atama	28
3.2.2.2 Grup Komutları	28
3.2.2.2.1 GROUPNAME	28
3.2.2.2.2 GROUPMAKE	29
3.2.2.3 Görev Komutları	30
3.2.2.3.1 TASKINIT	31

3.2.2.3.2 TASKCOPY	31
3.2.2.4 Görüntüleme Komutları	31
3.2.2.4.1 DISPLAY	31
3.2.2.4.2 PLINE	32
3.2.2.4.3 PCONTROL	32
3.2.2.5 Akış kontrol Komutları	33
3.2.2.5.1 IF	33
3.2.2.5.2 WHILE	34
3.2.2.5.3 REPEAT	34
3.2.2.5.4 SELECT	35
3.2.2.6 İş Sahibi Komutları	36
3.2.2.6.1 RESETOWNER	36
3.2.2.6.2 ADDOWNER	36
3.2.2.6.3 SUBOWNER	37
3.2.2.6.4 MOVEOWNER	38
3.2.2.7 Fonksiyonlar	39
3.2.2.7.1 Katar Fonksiyonları	39
3.2.2.7.1.1 ACCEPT	39
3.2.2.7.1.2 CONCAT	39
3.2.2.7.1.3 ASCII	40
3.2.2.7.1.4 TAKE	40
3.2.2.7.1.5 DROP	41
3.2.2.7.2 Tamsayı Fonksiyonları	41
3.2.2.7.2.1 DECIMAL	41
3.2.2.7.2.2 LENGTH	42
3.2.2.7.2.3 ISFILE	42
3.2.2.7.3 Veri Tamponu Fonksiyonları	43
3.2.2.7.3.1 ALLOC	43
3.2.2.7.3.2 REVERSE	43
3.2.2.7.3.3 PACK	44
3.2.2.7.3.4 UNPACK	45
3.2.2.7.4 Mantıksal Fonksiyonlar	46
3.2.2.7.4.1 LOAD	46
3.2.2.7.4.2 SAVE	46
3.2.2.7.4.3 GETTASK	47
3.2.2.7.4.4 SENDTASK	48
3.2.2.7.4.5 RUNTASK	49
3.2.2.7.4.6 ABORTTASK	49
3.2.2.7.4.7 KILLTASK	50
3.2.2.7.4.8 GETGROUP	50
3.2.2.7.4.9 SENDGROUP	51
3.2.2.7.5 Basit Komutlar	52
3.2.2.7.5.1 NOP	52
3.2.2.7.5.2 RETURN	52
3.2.2.7.5.3 BREAK	52
3.3 Ana Bilgisayar ve Çok İşlemcili Sistem	
Arasındaki İletişim	52
3.3.1 İletişimin Yapısı	52
3.3.2 İletişim Protokolları	54
3.3.2.1 SENDTASK İletişim Formatı	54
3.3.2.2 RUNTASK İletişim Formatı	55
3.3.2.3 ABORTTASK İletişim Formatı	56
3.3.2.4 KILLTASK İletişim Formatı	56
3.3.2.5 GETTASK İletişim Formatı	57

3.3.2.6 SENDGROUP İletişim Formatı	57
3.3.2.7 GETGROUP İletişim Formatı	58
 BÖLÜM 4. DERLEYİCİ TASARIMI	59
4.1 Tarayıcı	60
4.2 Ayırıştırıcı	61
4.3 Arakod Üretilmesi	62
4.4 Kod Üretimi	63
4.5 Kod Optimizasyonu	63
4.6 LR Ayırıştırıcılar	63
4.7 LR Ayırıştırıcı Programı	64
4.8 LR Ayırıştırıcı Uygulaması	67
4.9 YACC Destek Programı	69
4.9.1 Tanımlama Bölümü	70
4.9.2 Dönüştürme Kuralları Bölümü	71
4.9.3 Altprogramlar Bölümü	72
4.9.4 Örnek YACC Programı	72
 BÖLÜM 5. MPWFL DİLİ GRAMERİ	74
5.1 Program Gövdessi	74
5.2 Komutlar	75
5.2.1 Özel İşlev Komutları	76
5.2.2 Akış Kontrol Komutları	76
5.2.3 Atama Komutları	77
5.2.3.1 Mantıksal atama komutları	77
5.2.3.2 Tamsayı Atama Komutları	78
5.2.3.3 Katar Atama Komutları	79
5.2.3.4 Veri Tamponu Atama Komutları	80
5.2.3.5 Görev Atama Komutları	80
5.3 Gramer	81
 BÖLÜM 6. ÇEVİRİCİNİN GERÇEKLENMESİ	85
6.1 Ayırıştırma Tablosu	85
6.2 Gerçekleme Aşamaları	87
6.3 Çevirici Programının Genel Yapısı	88
6.3.1 Ana Program İşlem Adımları	89
6.3.2 Altprogramlar	90
6.3.2.1 INITFILE	90
6.3.2.2 ISFUNC	90
6.3.2.3 ISDELIM	90
6.3.2.4 IS_IN	91
6.3.2.5 NEXTTOKEN	91
6.3.2.6 PRINT_TOKEN	92
6.3.2.7 PUSH	92
6.3.2.8 POP	92
6.3.2.9 GOTOSTATE	92
6.3.2.10 SHIFT	93
6.3.2.11 REDUCE	93
6.3.2.12 MAKE_ID	93
6.3.2.13 PARSE	93

6.3.3 Veri Yapıları ve Tablolar	94
6.3.3.1 Veri Yapıları	94
6.3.3.2 Tablolar	95
 BÖLÜM 7. UYGULAMA PROGRAMLARI	97
7.1 Birinci Uygulama Programı	98
7.1.1 Birinci Uygulama Programının MPWFL Dili Dökümü	99
7.1.2 Birinci Uygulama Programının C Dili Dökümü	102
7.2 İkinci Uygulama Programı	108
7.2.1 İkinci Uygulama Programının MPWFL Dili Dökümü	109
7.2.2 İkinci Uygulama Programının C Dili Dökümü	112
7.2.3 İkinci Uygulama Programı Ekinin C Dili Dökümü	117
7.2.4 C Dili Ana Programının Dökümü	119
 SONUÇLAR VE ÖNERİLER	121
KAYNAKLAR	123
EKLER	124
ÖZGEÇMİŞ	198

ÖZET

Bu çalışmada, kişisel bilgisayar veya iş istasyonu türünden bir ana bilgisayar ve ona bağlantılı çok işlemcili sistemden oluşan yapıya yönelik bir yük akış dili tasarlanmıştır. Tasarımının başlangıç evresinde, hedef alınan yapıya sahip sistemler incelenmiştir. Ana bilgisayar ve çok işlemcili sistem arasındaki mesajlaşmanın türleri ve gerçekleşme şekli ortaya çıkarılmıştır. Bu evrenin sonucunda, MPWFL olarak adlandırılan dilin özel amaçlı işlevler ve hizmetler kümesi belirlenmiştir.

Yüksek seviyeli dillerde görülen; temel veri yapılarından tamsayılar, karakter katarları, mantıksal ifadeler ve program akış kontrolü komut yapıları (if-then-else, while, repeat, select) MPWFL dilinde kapsamaktadır. Ana bilgisayar ve çok işlemcili sistem arasında tanımlanan işlevleri ve hizmetleri yerine getirmede kullanılan; görev, veri tamponu, iş sahibi, grup değişkenleri olarak adlandırılan özel işlevli veri yapıları ve komutlar türetilmiştir.

Belirlenen veri yapıları, işlevler ve hizmetler kümesi taban alınarak, MPWFL dilinin grameri oluşturulmuştur. Gramerin tasarımında LR ayrıştırma kuralları uygulanmıştır. UNIX işletim sistemli bilgisayar üzerinde oluşturulan YACC destek yazılımına giriş olarak gramerin metni verilmiştir. YACC programının çıkış olarak ürettiği Ayrıştırma Tablosu kullanılarak bir Çevirici programı hazırlanmıştır. Çevirici program giriş olarak MPWFL dilinde yazılmış kaynak programı almakta ve amaç program olarak C dili bir metin üretmektedir. Özel işlevler için hazırladığımız destek programları ile C dili metin birlikte derlenerek yürütülebilir bir kod elde edilmektedir.

Yürütülebilir program kodunun tanımlanan işlevleri yerine getirdiğini sınamak için bir deney düzeni oluşturulmuştur. Deney düzeninde, seri haberleşme hattı ile bağlantılanmış iki adet kişisel bilgisayar yer almaktadır. Uygulama programların yürütülmesi sonucunda tüm yazılımların kendinden beklenen işlevleri gerçeklediği gözlenmiştir.

A WORK FLOW LANGUAGE FOR MULTIPROCESSOR SYSTEMS

SUMMARY

The aim of this study has been to develop a "workflow language" for a multiprocessor system in combination with a host (personal computer). This language has been called as MPWFL (Multiple Processor Work Flow Language). MPWFL is easily portable to different host systems and can be used on different microprocessor based multiprocessor systems.

The application programs written in MPWFL are translated in "C" language. The program which performs the translation procedure and other utilities have been written in "C". Consequently MPWFL could be portable to different host systems. As a result, an application program written in MPWFL can be used on different hosts or multiprocessor systems without any modification by compiling on the new computer or by modifying the utility programs.

Program development, program compilation and user interface utilities are implemented on the host system. Using a separate machine (namely the host) to edit and compile programs and then to transfer the object code to target machine allows the latter to compute faster and not to loose time with user interface procedures. A program development system imposes different requirements on hardware as well as software (memory management, terminals, disks ..). After program development has been finished, a large part of the system remains unused. As a result it's more efficient to use another computer (the host) for program development.

The host is connected to the processors of multiprocessor system by means of exchanging messages with different lengths. The main tasks for managing and operating the multiprocessor system are:

- 1- to transfer data and program code
- 2- to monitor the multiprocessor system

The data and programs which are to be transferred from

host to the multiprocessor system are stored in the host as files. Data that have been produced by the programs on the multiprocessor system are saved in the files on the host. It is also possible to transfer data (produced within a user application program) to the multiprocessor system without saving in a file. The commands that control the multiprocessor system are sent after they have been transformed into short messages.

Access to the information of system operational characteristics has been possible through monitoring the multiprocessor's system objects. The type and facilities of the system objects vary for different multiprocessor systems. Therefore the size and contents of the messages containing the parameters for system monitoring may vary for different multiprocessor systems.

To realize the facilities listed above in an efficient and flexible way, some additional commands and data structures have been implemented in MPWFL. Owner, buffer, task and group type variables are examples of additional data structures implemented in MPWFL. Some data structures and facilities included in general purpose high level languages, such as arrays, floating point numbers, trigonometric library functions haven't been implemented in this study in order to decrease the complexity of developing MPWFL. Since this study has been concerned with the efficient implementation of a workflow language, first the workflow operations have been determined. To implement these operations efficiently, among the commands and data structures of the general purpose high level languages, we have chosen a group sufficient for a workflow language. In order to meet the conceptual requirements of a workflow language, some additional commands and data structures have been designed for MPWFL.

A program written in MPWFL has a structure given below:

program:

```
type_definition variable;  
type_definition variable, variable;  
Subprogram_definition;
```

```
SUBBODY subprogram_id  
    SUBBEGIN  
        command(s)  
    SUBEND
```

MPWFL has the following variable types:

- 1- Boolean
- 2- Integer
- 3- String
- 4- Task
- 5- Buffer
- 6- Owner
- 7- Group
- 8- Subroutine

MPWFL offers the following functionally grouped instructions:

- 1- Assignment
- 2- Group
- 3- Task
- 4- Display
- 5- Control
- 6- Owner
- 7- String

Buffer, task, owner and group variables are designed for MPWFL in order to satisfy the needs of a workflow language. Buffer variables are translated as structures in "C" language. There are three records in the buffer structure:

COUNT : the number of items in the buffer
MAX : the maximum number of items that can exist in the buffer
POINTER : the beginning address of the buffer

Task variables are translated as structures in "C" language. There are three records in the buffer structure:

NAME : the name of the task variable
FIELD : the information about task
POINTER : the beginning address of the buffer which contains the program code of task

Owner variables are defined as a character array (char[16]) in "C" language after the translation. This array represents the processors in the multiprocessor system, from 0 to 127.

It's possible to determine the format of messages, which provide the transfer of data between the host and the multiprocessor system by group variables. Group variables are defined as a linked list in "C" language

after the translation. The nodes in the linked list represent a structure. The records of this structure are:

POINTER : the beginning address of the group variable
COUNT : the number of bytes of the group variable to be sent.
TYPE : the type of the group variable
NEXT : the beginning address of the next node in the linked list (next group variable)

The statements and functions of the MPWFL are listed below:

Assignment statements are boolean assignment, integer assignment, string assignment, task assignment, and buffer assignment.

Group statements are groupname and groupmake. Groupname is used to assign a name to the group list. Groupmake is used to add a new node to the group list.

Task statements are taskinit and taskcopy. Taskinit is used to assign the initial values to a task variable. Taskcopy is used to copy the source task variable to the destination task variable.

Display statements are display, pline, and pcontrol.

Flow control statements are if-then-else, while, repeat, and select.

Owner statements are resetowner, addowner, subowner, and moveowner. Resetowner is used to assign the initial value to an owner variable. Addowner is used to add a new owner to an owner variable. Subowner is used to remove an owner from an owner variable. Moveowner is used to add new owners to an owner variable.

There are string, integer, buffer and boolean functions in MPWFL.

String functions are accept, concat, ascii, take, and drop. Accept is an I/O function. Concat function concatenates two strings. Ascii function transforms its integer parameter into a string. Take and Drop are functions for partitioning the string.

Integer functions are decimal, length, and isfile. Decimal function transforms its string parameter into

integer. Length function returns the length of a string. Isfile function returns the length of a file.

Buffer functions are alloc, reverse, pack, and unpack. Alloc function allocates the blocks of memory. Reverse function reverses the order of bytes in a buffer. Pack function packs the items in a buffer (transforms the item from ASCII to HEX). Unpack function unpacks the items in a buffer which were packed before by pack function (transforms the item from HEX to ASCII).

Boolean functions are load, save, gettask, sendtask, runtask, aborttask, killtask, getgroup, and sendgroup. Load function loads a file to a buffer. Save function saves data from a buffer into a file. Gettask function gets the data of the FIELD record of a task from a processor. Sendtask function sends the values of the NAME, FIELD and BUFFER records of a task to the processors defined with owner variable. Runtask function allows the processors defined with owner variable to run a specific task. Aborttask function provides suspending a specific task in processors defined with owner variable. Killtask function kills a task in the processors defined with owner variable. Getgroup function gets the information about a group from the processor defined with owner variable. Sendgroup function sends the information about a group to the processors defined with owner variable.

The interaction between host and multiprocessor system is summarized as follows. Host system can be either a personal computer or a workstation. It is used for program development; for monitoring the multiprocessor system; for the transfer of data and program code to or from the multiprocessor; for storing the files produced by the programs on the multiprocessor system or for storing the files providing the data to the programs on the multiprocessor system. The host is connected to a manager processor of the multiprocessor system by an asynchronous serial connection with 9600 baud rate. It's possible to modify the utilities written in "C" in order to support another connection (a LAN, for instance).

In this study, first, the instructions, commands, variables and then the grammar of the MPWFL language has been determined. Then, the grammar and the tokens have been arranged to form an input to a UNIX operating system utility "YACC". The translator program has been developed by using the parser table generated by YACC.

ÖZGEÇMİŞ

Ayşe Olcay Akgün 22 Ekim 1967'de İstanbul'da doğdu. Orta öğrenimini 1985 yılında Küçükçekmece Lisesinde Birincilikle bitirdi. Matematik Derneğinin 1985 yılında düzenlemiş olduğu İstanbul Liselerarası Matematik Yarışmasında 12.lık derecesini aldı. 1989 yılında İ.T.Ü. Elektrik-Elektronik Fakültesi Kontrol ve Bilgisayar Mühendisliği Bölümünden mezun oldu. 1987 yılında Teletaş firmasının karşılıksız bursunu kazandı. 1989 yılından itibaren İ.T.Ü. Bilgi İşlem Merkezi'nde Araştırma Görevlisi olarak çalışmaktadır.

Ayşe Olcay Akgün evlidir ve bir kızı vardır.

BÖLÜM 1. GİRİŞ

Ana çatı (Main Frame) türünden bilgisayarlarda karşılaşılan ve özellikle çok kullanıcı olmasının yarattığı sistem yönetimi karmaşasını azaltmada, işletim sistemini doğrudan yönlendiren yordamların önceden hazırlanmış olması ve gereğinde hemen kullanılabilmesi önemli rol oynamaktadır. Değişik sistemlerde bu işlevler basit işletim sistemi komutlarının peşpeşe yer aldığı "Batch" türü dosyaların yürütülmesi ile sağlanır. Bu işlevlerin uygun bir programlama dili çerçevesinde hazırlanması büyük ölçüde kolaylık, esneklik ve etkinlik sağlamaktadır. Böylece, bir kullanıcıya ait bir ya da daha fazla prosesin (görevin) yaratılması ve sistemden beklenen hizmetlerin özelliklerinin ve sırasının belirlenmesi ve işlemlerin yürütülüş şeklinin izlenmesi kolaylaşmaktadır. Sonuçta bilgisayar sisteminin yük akışının düzenlendiğini göz önüne alırsak bu uygun dili genel olarak Yük Akışı dili olarak adlandırabiliriz. Yük akış dilleri arasında, IBM sistemlerinin REXX yorumlayıcısını [1] ve UNISYS sistemlerinin WFL [2] derleyicisini sıralayabiliriz. Genel olarak bu dillerin yetenekleri hesaplama işlemlerinden daha çok, dosya işlemleri ve sistem işletmeye yönelik işlemler üzerinde yoğunlaşmaktadır. İşletim sistemi özelliklerine sıkı sıkıya bağlı olan bu diller, doğal olarak değişik türden (veya işletim sistemli) bilgisayarlar arasında taşınabilme özelliklerine sahip değildirler.

1.1 ÇALIŞMANIN AMACI

Bu çalışmanın amacı kişisel bilgisayar ya da iş istasyonu türünden bir Ana (host) bilgisayar ve ona

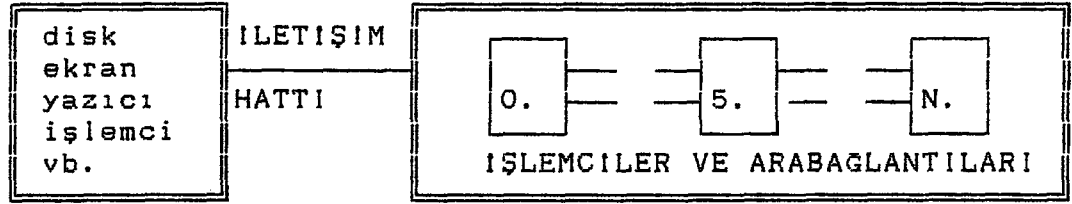
bağlı çok işlemcili sistemden oluşan yapılara [3][4][5] yönelik bir yük akış dilinin gerçekleşmesidir. MPWFL (Multiple Processor Work Flow Language) olarak adlandırılan bu dilin değişik türden ana bilgisayarlar arasında taşınabilir olması ve değişik mimarili çok işlemcili sistemler için kullanılabilir olması çalışmanın önemli hedefleri arasında sayılabilir.

MPWFL dilinde yazılan uygulama programı "C" diline çevrilerek tekrar derlenmektedir. Çevirme işlemini gerçekleyen programın ve diğer destek programlarının C dilinde hazırlanmış olması MPWFL uygulama programına taşınabilirlik özelliğini katmaktadır. Ayrıca ana bilgisayar ve çok işlemcili sistemin özelliklerine göre destek programlarının ve iletişim programlarının C dilinde kolaylıkla tekrar düzenlenebilir olması da bu çalışmayı desteklemektedir. MPWFL dili kapsamında benzer hizmetlerin farklı mimari yapılarda olan sistemlere uygulanmasını sağlayan özellikler yer almaktadır. Sonuç olarak, tekrar derlenerek veya destek programları üzerinde yeniden düzenleme yapılarak önceden hazırlanan bir uygulama programı, başka türden bir ana bilgisayar için ya da başka türden bir çoklu işlemci sistemi için değiştirilmeden kullanılabilir.

1.2 ÇALIŞMANIN KAPSAMI

Çalışmamız aşağıda tanıtılan türden sistemler ele alınarak yönlenmiştir. Hedef alınan sistemde ana bilgisayar; program geliştirme, dosya saklama, kullanıcı ile arabirim oluşturma türünden işlemleri yerine getirir. Aralarında belirli bir bağlantı mimarisi olan çok işlemcili sistemin temel işlevi ise yüksek hızda işlem gerçekleştirme olmaktadır (Şekil 1.1). Bu tür yapılarda çok işlemcili sistemin, ana bilgisayarın sahip olduğu disk, yazıcı, ekran gibi donanım yetenekleri olmayabilir. Çok işlemcili sistemin tüm işlemcileri veya

Yönetici özelliği verilmiş bir işlemcisi ile ana bilgisayar, bir iletişim ortamı üzerinden boyutları değişebilen mesajların aktarımı ile karşılıklı haberleşerek çalışmayı kotarırlar. Yönetici işlemcili uygulamada verilerin çok işlemcili sisteme yayılması, yönetici ve diğer işlemcilerin mevcut arabaglantı şebekesi üzerinden haberleşmesi ile gerçekleşir.



ANA BİLGİSAYAR

ÇOK İŞLEMCİLİ SİSTEM

Şekil 1.1 Sistemin genel yapısı

Çok işlemcili sistemi yönetme, çalıştırma için gerekli olan temel işlevler aşağıda sıralanmıştır:

- 1- Program kodu ve veri aktarma
- 2- Sistemi denetleme
- 3- Sistemi gözlemleme

Çok işlemcili sisteme aktarılacak olan programlar ve veriler genellikle dosya halinde saklı tutulmaktadır. Yine çok işlemcili sistemin ürettiği sonuçlar alınarak dosya halinde saklanması istenir. Böylece bu tür işlemler kısaca dosya işlemleri adı altında toplanabilir. Ancak özellikle veri üretme aşamaları kullanıcının ana bilgisayarda hazırladığı bir uygulama programı içinde gerçekleşebilir ve dosya şeklinde saklanmadan çok işlemcili sisteme aktarılması istenebilir. Bu türden çalışmaların desteklenmesi ile

esneklik kazanılmaktadır. Yük akış dilinin, aynı program kodunun birden fazla işlemci birimine aynı anda gönderilmesini sağlayan özelliklere de sahip olması beklenebilir.

Çok işlemcili sistemi denetleyen genel yapıllı komutlar kısa formatlı mesajlara dönüştürülerek aktarılır. Mesaj dönüştürme işleminin çok işlemcili sistemin yapısına bağlı olarak düzenlenmesi gerekmektedir. Çok işlemcili sistemin, sistem nesneleri gözlenerek sistemin genel çalışması hakkında bilgi edinilir. Sistem nesneleri türleri ve özellikleri hakkında çoklu işlemcili sistemler arasında farklılıklar vardır. Bu nedenle sistemi gözleme için kullanılacak olan mesajların formatları çok farklı olabilir. Değişik türden bilgileri bir araya getiren mesajları oluşturmada ve bu mesajları çözmede etkin bir yöntem izlenmelidir.

Yukarıda üç madde halinde özetlenen işlevlerin kolaylıkla ve esneklikle kotarılması için MPWFL dilinde, bilinen veri tipleri ve komutların yanısıra, özel işlevlere sahip veri tipleri ve komutlar yer almaktadır. Örneğin, program kodu ve veri bloklarını bir ya da birden fazla işlemciye göndermede Veri Tamponu değişkenleri, Görev değişkenleri, İş Sahibi değişkenleri kullanılır. Sistemi denetlemede ve gözlemlenmede iş sahibi değişkenleri, görev değişkenleri, grup değişkenleri kullanılmaktadır.

Bölüm 2'de ana bilgisayar ve çok işlemcili sistem'den oluşan bir yapı için geliştirilmiş olan bir dağınık işletim sistemi uygulaması, çalışma şekli açısından önemli bir örnek teşkil ettiğinden detaylıca ele alınmaktadır.

Bölüm 3'te MPWFL dilinin yapısı ve özellikleri tanıtılmaktadır. Bölüm 4'te Çevirme işleminin gerçekleştirilmesinde kullanılan Tarayıcılar ve

Ayrıştırıcılar konusu kısaca anlatılmıştır.

Bölüm 5'te ise MPWFL dilinin grameri verilmektedir. Bölüm 6'da ise gerçekleşmesinde izlenen yöntemler ve hazırlanan programlar hakkındaki açıklamalar yer almaktadır. Bölüm 7 ise örnek programları ve C diline çevrilen kodları kapsamaktadır.



BÖLÜM 2. FADOS DAĞINIK GERÇEK ZAMANLI İŞLETİM SİSTEMİ

Bu bölümde, FAMP (Fast Amsterdam Multiprocessor) çok işlemcili sistemi için geliştirilmiş olan FADOS dağınık gerçek zamanlı işletim sistemi [3] incelenecektir. Böylece, hedeflenen sistemlerin çalışma şekli özetlenecek ve tanıtılan sistemin özellikle temel birimleri arasındaki ilişkiler ele alınarak, MPWFL dilinin, bu türden hizmetlerin gerçekleştirilmesine yönelik, işlevler kümesi belirlenebilecektir. MPWFL dilinde bu işlevlerin gerçekleştirilmesinde değişik türden mimarisi olan diğer çok işlemcili sistemlerde de uygulanabilecek esneklikte olmasını sağlayan yöntemlerin geliştirilmesi, bu çalışmanın temel konusudur.

FADOS işletim sistemi, HOST-TARGET ilişkisinde çalıştırılan iki temel birimden oluşan sistem üzerinde kurulmaktadır. İki temel birimden biri UNIX işletim sistemli bir ana bilgisayardır (HOST). Diğeri ise MC68000 mikroişlemcisi yer alan işlemci kartlarından oluşturulmuş çok işlemcili bir sistemdir (TARGET). FAMP çok işlemcili sistemi, iki kapılı belleklerle (dual ported memory) işlemci modüllerinin bağlantısının sağlandığı çok seviyeli bir mimari yapıya sahiptir.

İşletim sisteminin temel işlevleri arasında; HOST ile TARGET işlemcilerinde bulunan prosesler arasında veri iletişimini, TARGET bilgisayarındaki programlar aracılığıyla HOST bilgisayarında bulunan dosyalara erişimini, TARGET bilgisayarında yürütülen programların HOST bilgisayarından gözlenmesini ve hata ayıklanabilmesini (debugging) sağlamak sayılabilir. İşletim sistemi hizmetleri, MC68000 tabanlı çok

işlemcili sistem ile UNIX ana bilgisayar tarafından ilişkili olarak bereberce gerçekleştirilmektedir.

2.1 İŞLETİM SİSTEMİNİN ÖZELLİKLERİ

HOST-TARGET yapısı aşağıdaki nedenlerle tercih edilmiştir:

1-Özel amaçlarla tasarlanmış çok işlemcili sistemlerde, zaman paylaşımli bir sistemin kurulmasında pek çok sorun sözkonusu olmaktadır.

2-Bellek yönetimi, terminal, disk gibi, gerçek zaman uygulamalarının çoğunda gerek duyulmayan, sadece program geliştirilmesi aşamasında kullanılacak olan kimi donanım öğeleri, program geliştirilmesi işi sonlandığında, atıl durumda kalacaktır.

HOST-TARGET yapısındaki bu sistemden ve gerçek zamanlı işletim sisteminden beklenenler aşağıda maddelendirilmiştir:

1-Gerçek zamanlı işletim sistemi, farklı TARGET bilgisayarlarında kolaylıkla kurulabilmelidir. Burada, TARGET için gerekli olabilecek donanım değişikliği ya da donanımın geliştirilmesi bir engel oluşturmamalıdır. TARGET bilgisayarlar, özel amaçla tasarlanan bir sistemde farklı mikroişlemcilerin kullanımına karar verilebilmesi gibi, oldukça sık olarak değişikliklere uğrayabilecektir. İşte bu nedenle, işletim sisteminin değişik TARGET bilgisayarlarına kurulabiliyor olması önem taşımaktadır.

2-TARGET bilgisayar üzerinde yürütülen programların, zaman paylaşımli bir sistem aracılığıyla izlenmesi ve işletilmesi mümkün olması gerekir. İyi bir kullanıcı arayüzü oluşturmak yeterince zordur. Kullanıcı

arayüzünü, gerçek zamanlı bir sistem yerine zaman paylaşımli bir sistemde gerçekleştirmek daha elverişlidir. Eger, kullanıcı arayüzü zaman paylaşımli bir sistemde gerçekleşirse, bu işe uygun birçok yazılım aracı kullanılabilecek ve daha iyi bir arayüz ortaya çıkabilecektir. Gerçek zamanlı sistemin gözlenmesi için, işletmenin kullanacağı arayüzün bir başka bilgisayar tarafından gerçekleşmesi ile, birden fazla bağımsız sistemin aynı anda gözlenebilmesini sağlayan bir işletmen arayüzü oluşturulabilir ve gerçek zamanlı işletim sisteminin arayüzle ilgili prosesler nedeniyle artacak olan karmaşıklığı minumuma indirgenmiş olur.

3-TARGET bilgisayarında yürütülen programların, HOST üzerindeki dosyalara, verilere erişebilmesinin sağlanması gerekir. Çünkü bu programlar, HOST bilgisayarında bulunan kimi giriş bilgilerini kullanarak, yine HOST üzerinde saklanacak olan çıkış bilgilerini üretmektedirler. Yerel bir disk kullanımı, TARGET sistemin güvenilirliğini azaltacağından, HOST bilgisayarında yer alan dosyalara erişim yeterli hızla sağlanamıyorsa düşünülebilir.

4-işletim sistemi, birçok yüksek seviyeli programlama dilini desteklemelidir.

5-Hata ayıklama (debugging) işlemi için özel yazılım araçları eklenmelidir. Gerçek zaman programlarının, duraksama noktaları (break-points) kullanılarak hata ayıklama işlemleri gerçekleşirken, her proses için duraksama noktaları oluşturulur ve böylece, zaman kritik olmayan proseslerin, zaman kritik olanları bozmaksızın hata ayıklanmaları mümkün olur.

6-Program geliştirme için ayrı bir sistem kullanımı, hem fiziksel, hem de lojik olarak iyi bir HOST - TARGET bağlantısını gerektirmektedir.

Bunun içinse, yüksek hızlı ve mesaj kaynaklı bir bağlantı gerekmektedir. Karakter kaynaklı bağlantılar, düşük hızları nedeniyle tercih edilmemektedir. Çözüm olarak, HOST bilgisayarındaki işlemciden herhangi bir destek almaksızın çok büyük miktarlardaki veriyi, yüksek bir hızda transfer edebilen LAN (yerel bilgisayar ağıları) kullanılmıştır.

2.2 İŞLETİM SİSTEMİNİN GERÇEKLENMESİ

Sistem prosesleri arasındaki iletişim, mesaj alışverişi yardımıyla gerçekleştirilir. Prosesler arasındaki iletişimde, HOST - TARGET arayüzü saydamdır. Bunun başlıca iki yararı vardır:

1-Belirli bir görevin hangi işlemci tarafından (HOST ya da TARGET) yürütüleceğinin belirlenmesinde, geniş bir özgürlük vardır.

2-TARGET bilgisayarı için gereken kullanıcı arayüzü, tamamen HOST bilgisayarındaki proseslerce sağlanabilir.

Proseslerin iletişimi esasına dayalı bir işletim sisteminin en önemli olumlu tarafı, tasarımının basitliğidir. Ancak, prosesler arası iletişimin getirdiği önışlem (overhead) süresi olumsuz yönüdür. İşletim sistemi için oluşturulan tasarımın yapısını büyük oranda belirleyen HOST - TARGET arasındaki 3 ana etkileşim alanı şunlardır:

1-HOST bilgisayarında oluşturulan programlar, TARGET bilgisayarına aktarılabilmelidir. TARGET veya HOST bilgisayarlarında yürütülen programlar, değişik işlemcilerden bir grup prosesi yükleyebilmelidir.

2-TARGET bilgisayarındaki programlar, HOST bilgisayarındaki dosyalara erişebilmelidir.

3-TARGET bilgisayarında yürütülen programların simgesel hata ayıklanması işlemi için, HOST bilgisayarındaki derleyicinin ürettiği adlara ilişkin bilginin ve bellek görüntüsünün yorumunun edinilmesi daha uygundur. Böylece, TARGET üzerindeki bir prosesin sadece, fiziksel adreslere duraksama noktası (break-point) vermek ve HOST bilgisayarındaki sembolik hata ayıklayıcıya bellek görüntüsünü aktarmak gibi alt düzeydeki işlemleri kotarması yeterli olacaktır.

İşletim sisteminin, TARGET bilgisayarının her bir işlemcisinde yer alan parçaları iki ana gruba ayrılır:

a-Kullanıcı ve sistem proseslerine kesmeleri ileten ve proses iletişimini sağlayan bir çekirdek yazılım (nucleus).

b-Sistem prosesleri:

i) Bir kullanıcı prosesini belleğe yükleyen ve canlandıran Yükleyici yazılımı (load process).

ii) Duraksama noktalarını (break-points) ve adresleme hatalarını kotaran Sistem Gözlemeleme yazılımı (debug monitor).

İşletim sisteminin HOST bilgisayarında yer alan parçaları:

a-Bir komut yorumlayıcı (command interpreter)

b-Bir hata ayıklayıcı (debugger)

c-Bir dosya hizmet birimi (file server)

Bir prosesin, henüz yüklenmemiş olan diğer bir prosesle iletişim kurmak istemesi ile ortaya çıkabilen sorunları engellemek için, yükleme işleminin iki aşamalı yapılmasına karar verilmiştir:

1-Prosesleri işlemcilerin çekirdek yazılımlarına tanıtmak. Böylece, bu ilk aşamanın sonunda, tüm

prosesler varolacaklardır.

2-ilk aşamada tanıtılan proseslerin adresleri belirlenir.

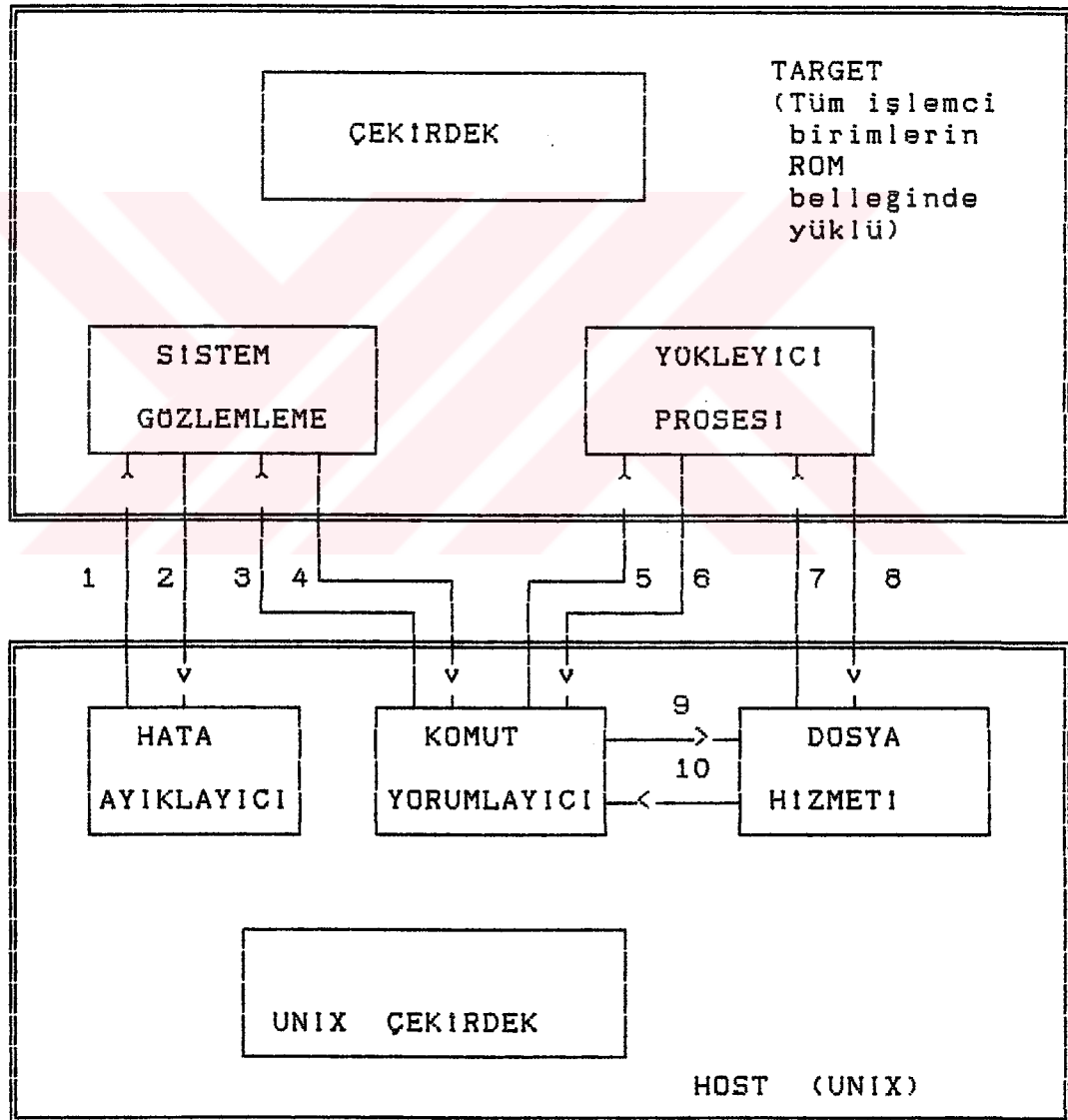
2.3 UYGULAMA

UNIX işletim sisteminde, yürütülebilir dosyalar listesini giriş olarak alan bir komut yorumlayıcı mevcuttur (Şekil 2.1). Her dosya ile, gerekli yığın alanı, donanım ve yazılım öncelikleri (priority) ve dosyanın yükleneceği işlemci belirtilir. Komut yorumlayıcı, yüklenecek her proses için ilgili işlemciye "load process" mesajını gönderir. Daha önceden belirtildiği gibi, her işlemcide mevcut olan bu Yükleme prosesi, yüklenecek dosyanın adı, gerekli yığın alanı ve önceliği bilgilerini içeren komut mesajını bekler. Yüklenecek dosya, işlemcinin belleğine okunur ve çekirdeğe bir sistem çağrısı yapılarak yeni bir proses yaratılır. Bu proses yaratılır yaratılmaz hazır kuyruğuna eklenir ve yürütülmeye başlatılabilir. Her prosesin ilk yapacağı iş, diğer işlemcilerin adreslerini ve standart giriş/çıkış ve hata (standard input output, error) için dosya tanımlayıcılarını (file descriptor) içeren mesajı beklemek olacaktır. Tüm prosesler bu şekilde başlatıldıktan ve herbiri mesaj bekliyorken, komut yorumlayıcı tarafından, diğer proseslerin adreslerinin ve o prosesler için açılmış olan dosyaların tanımlayıcılarını içeren mesaj gönderilir. Şekil 2.1'de verilen temel işlemler aşağıda sıralanmıştır:

- 1-Hata ayıklayıcıdan komut aktarımı
- 2-Debug monitöründen yanıt mesajları

3-Bir prosesi durdurma (stop) ya da sonlandırma (kill) istekleri

- 4-Sistem gözleme modülünden onay mesajları
- 5-Bir prosesin başlatılmasını (start) isteme
- 6-Başlatılan prosesin onayı
- 7-Prosesin program kodunu dosyadan yükleme
- 8-Verinin dosya olarak saklanması
- 9-Komut yorumlayıcısının standart giriş, standart çıkış ve hata için açılış isteği
- 10-Dosya tanımlama bilgilerinin aktarımı



Şekil 2.1 Sistemin yapısı ve işlem türleri

2.4 MPWFL DİLİNE YÖNELİK BELİRLENEN İŞLEVLER

Bu bölümün pragraflarında özetlenen FADOS işletim sisteminin incelenmesi sonucunda MPWFL dili için temel oluşturan işlevler belirlenmiştir. Genel olarak ifade edilirse, Ana bilgisayar (HOST) ile çok işlemcili sistem (TARGET) arasında sabit uzunluklu özel bilgi taşıyan mesajlar ve değişken uzunlukta veri blokları karşılıklı olarak aktarılmaktadır. Mesajlar genellikle uygulama programı içinde ya da komut yorumlayıcı üzerinden yaratılır. Veri blokları ise genellikle dosya işlemlerine (kaynak veya varış olarak) ilişik olmaktadır. Mesajları ve veri bloklarını şu şekilde sınıflandırabiliriz:

1-Proseslerin işletilmesine (başlatma, sonlandırma gibi) yönelik sabit formatta kısa mesajlar ve yanıtları.

2-Sistemin (ve sistem nesnelerinin) yönetilmesine (hata ayıklama, gözlemlene gibi) yönelik değişken formatta kısa mesajlar ve yanıtları.

3-Standart işlevli ve değişken boyutlu veri blokları (proses program kodu gibi) ve yanıtları.

4-Uygulama ile belirlenen ve değişken boyutlu veri blokları (çok işlemcili sisteme işlenecek veri gönderme ve üretilen sonuçları alma gibi) ve yanıtları.

MPWFL dili için, farklı yapılı çok işlemcili sistemlerde benzerlik taşıyan özel işlevli kısa mesajlar (görevlerin yönetilmesine yönelik) belirlenerek, kullanımda kolaylık sağlayan basit yapılı komutlarla bunların işlenmesi öngörülmüştür (Bölüm 3). Veri bloklarının yapısı ve özellikle sistemin yönetilmesini sağlayan mesajların yapısı değişik sistemler arasında oldukça farklılık göstermektedir. Bu amaçla, mesajları ve veri bloklarını oluşturan\çözen ve kullanıcıya etkin ve esnek bir ortam

sunan bir çalışma şekli tasarlanmıştır (Bölüm 3). Uygulama programında kullanıcı, mesajları veya veri bloklarını oluşturan bilgileri değişkenler yardımıyla işlemekte ve haberleşme paketlerini istediği formatta kolaylıkla oluşturabilmektedir.



BÖLÜM 3. MPWFL DİLİ

Bu tezin konusu olan MPWFL dili, ana bilgisayar ve çok işlemcili sistemden oluşan veri işleme sistemleri [3], [4], [5] için sistem yönetimi işlevlerine sahip özel amaçlı işlemlere yönelik bir dil olarak tasarlanmıştır. Hedeflenen özel amaçlı işlemlerin yürütülmesi için kullanıcının kolaylıkla ve esneklikle program yazmasını sağlayan veri yapıları ve işlevler MPWFL dilinde tanımlanmaktadır. Bu bölümde, MPWFL dilinin yapısı, özellikleri ve komutları tanıtan paragraflar yer almaktadır (Bölüm 3.2). Ana bilgisayar ile çok işlemcili sistem arasındaki iletişimin yapısı ve veri formatları, özel işlev komutları ile ilişkili olarak Bölüm 3.3'te tanıtılmaktadır.

3.1 MPWFL DİLİNİN YETENEKLERİ

Hedef çok işlemcili sistem ve Ana bilgisayar arasında gerçekleştirilen dosya aktarımları ve sistemin denetlenmesi\gözetlenmesinde kullanılan mesaj oluşturma, aktarma ve bu mesajların yorumlanması belli bir deneyim gerektirmektedir. Bu türden sistemler için ana bilgisayar tarafında yüksek seviyeli dillerde (çoğunlukla C dilinde) hazır altprogramlar sağlanmaktadır [5]. Bu biçimde hazırlanan bir uygulama programında, bilinen türdeki işlemlerin yanı sıra yukarıda tanımlanan işlemlerin kullanım şekli karmaşık ve anlaşılması zor bir görüntü vermektedir. Ana bilgisayarın kullanım amacı aynı zamanda çok işlemcili sistem ile kullanıcı arasında bir arabirim olmasıdır. O halde, bu amaçlara yönelik tasarlanmış bir programlama

dili ile, bir uygulama programı geliştirmek daha uygun olacaktır. MPWFL dilinin özel işlev komutlarını gerçekleyen yordamların C dilinde hazırlanması sonucunda tasarımcı, gereğinde bu işlevlerin gerçekleştirilme şeklini değiştirebilir ya da hedef sistem için hazır olarak verilen altprogramları kullanabilir. Bölüm 7'de verilen örnek programların ve C dili karşılıklarının incelenmesiyle iki uygulama arasındaki fark daha rahat görülebilir.

Dilin gerçekleştirilmesindeki karmaşıklığı ve zorlukları indirgemek için, genel amaçlı programlama dillerinin (örneğin FORTRAN, C gibi) bazı temel veri yapıları ve yetenekleri (ondalıklı sayılar, trigonometrik ve matematik arşiv fonksiyonları gibi) bizim çalışmamızda gerekli olmadığı gerekçesi ile kapsamamaktadır. Bazı bilinen yük akış dillerinde de benzer düşüncelerle genel programlama dillerinin tüm yetenekleri yoktur [1], [2]. MPWFL dilinin tasarımında öncelikle, kullanıcının çok işlemcili bir sisteme yönelik olarak yürütülmesini isteyebileceği yük akış işlemleri belirlenmiştir. Belirlenen yük akış işlemlerinin kolaylıkla ve esneklikle yürütülmesini sağlayan komutlar ve veri yapıları ortaya çıkarılmıştır. Çalışmada tespit edilen özel amaçlı komutlar ve veri yapılarının yanı sıra genel amaçlı programlama dilleri komutlarının ve veri yapılarının hangilerinin kapsammasının yeterli olabileceği belirlenmiştir. Tüm komutların ve veri yapılarının rahat kullanımını sağlayan ilişkileri de gözetilerek dilin grameri ve komut kümesi geliştirilmiştir.

MPWFL dilinde örneğin, tek boyutlu ve çok boyutlu dizi yapıları, ondalıklı sayılar, giriş veya çıkış parametrelerine sahip altprogram tanımları gibi bir genel programlama dilinden beklentiler yer almamaktadır. Ancak hazırlanan uygulama şekli gereğinde bu tür eksikleri de ortadan kaldıracılabilmektedir. Hazırlanan bir

Çevirici program ile MPWFL dilindeki program, C dili koduna dönüştürülmektedir. Program içinde "main" bölümünün kapsanması sonucu C kodu derlenerek kendi başına yürütülebilen bir hedef dosya (kişisel bilgisayarda EXE program) üretilmektedir. Ancak, "main" dışında sadece hizmet altprogramlarından oluşan bir MPWFL dili programı, C diline çevrilerek ve derlenerek "Object" kodu üretilebilir. Diğer bir C dili programında (diğer yüksek seviyeli programlama dilleri de kullanılabilir) çok işlemcili sisteme yönelik işlemleri yürüten bu hizmet altprogramları çağrılır. Sonuç olarak her iki programın "object" kodları Bağlayıcı (Linker) tarafından birleştirilerek yürütülebilen bir program kodu (EXE) üretilir.

Böylece, kullanıcı için C dilinde oldukça karmaşık görünen çok işlemcili sisteme yönelik işlevler daha elverişli olan MPWFL dilinde gerçekleştirilebilir. Çok işlemcili sistemde işlenecek olan verilerin bir dosyada saklanması yerine, genel amaçlı dilde yazılmış bir uygulama programı içinde üretildiği ve çok işlemcili sistemle etkileşimli çalışıldığı durumlarda bu türden çok dilde programlama (multi-language) daha uygun olacaktır. Hazırlanan çevirici programı ile, bu tür çalışmayı kolaylaştıran ve MPWFL dilindeki programda kullanılan tüm değişken ve altprogram tanımlarını içeren "include" [5] dosyasının üretilmesi de sağlanmaktadır.

3.2 MPWFL DİLİNİN ÖZELLİKLERİ

Bu bölümde MPWFL dilinin özellikleri ve komut kümesi tanıtılmaktadır. Öncelikle program yapısı verilmekte ve izleyen bölümlerde veri yapıları ve komutlar yer almaktadır.

3.2.1 PROGRAM YAPISI

Bu bölümde programın genel yapısı tanıtılarak modülleri açıklanacaktır. Ayrıca veri ve değişken türleri sunularak her biri için detaylı açıklamalar yer alacaktır.

3.2.1.1 PROGRAMIN GENEL YAPISI

Program, tanımlama bloğu ve altprogram bloğu (blokları) olmak üzere iki temel bloktan oluşmaktadır.

Tanımlama bloğunda, program içinde kullanılacak tüm değişkenler ve altprogramlar tanımlanır. Altprogram bloğunda ise, altprogram gövdeleri yer alır. Ana program da, bir altprogram görüntüsünde sunulur.

program :

tip tanımlayıcısı değişken adı;	]	tanımlama bloğu
tip tanımlayıcısı değişken adı,değişken adı;		
altprogram tanımlama;		
SUBBODY altprogram adı	]	altprogram bloğu (blokları)
SUBBEGIN		
komut(lar)		
SUBEND		

Program yapısını daha iyi tanıtabilmek amacıyla işlevsel olarak çok anlamlı olması gözetilmeyen bir örnek aşağıda verilmiştir. Örnek uygulama programları Bölüm 7'de verilmiştir.

Örnek program:

```

INTEGER sayi, sonuc;
STRING yanit;
BOOLEAN bayrak;
SUBROUTINE sorgulama, main;
SUBBODY sorgulama
SUBBEGIN
    yanit=ACCEPT('Toplama yapılınsın mı? (E/H)');
    bayrak=yanit SEQL 'E';
SUBEND
SUBBODY main
SUBBEGIN
    sayi=5;
    sonuc=6;
    sorgulama;
    IF (bayrak) THEN
        sonuc=sonuc+sayi;
        yanit=ASCII(sonuc);
        DISPLAY('Toplama sonucu=');
        PLINE(yanit);
    ELSE
        DISPLAY('Toplama yapılmadi..');
    ENDIF;
SUBEND

```

3.2.1.2 DEĞİŞKEN TANIMLAMA

Değişken tanımlama bloğunda, programda yer alan değişkenlerin tümünün tanımlanması gerekir. Değişken adları ve altprogram adları birbirinden farklı olmalıdır. Önceden tanımlanmış olan anahtar sözcükleri (IF, REPEAT, ..), değişken ve altprogram adı olarak kullanılamaz. Program derlenmeden önce bütün karakterler büyük harfe çevrildiğinden, değişken adları büyük-küçük harfleri karışık bulundurabilirler. Dolayısıyla büyük-küçük harf farkı, değişkenleri birbirinden ayırtetmekte kullanılamaz.

Aşağıda listelenen tip tanımlayıcıları,

- 1-BOOLEAN : Mantıksal değişkenler için,
- 2-INTEGER : Tamsayı değişkenleri için,
- 3-STRING : Katar değişkenleri için,

- 4-TASK : Görev değişkenleri için,
- 5-BUFFER : Veri tamponu değişkenleri için,
- 6-OWNER : İş sahibi değişkenleri için,
- 7-GROUP : Grup değişkenleri için,
- 8-SUBROUTINE : Altprogram adları için,

kullanılır. MPWFL dilinde, standart değişken tipleri kullanımı ile yeni değişken tipleri yaratılamaz.

3.2.1.3 ALTPROGRAMLAR

C dilinde altprogram parametreleri yığın üzerinden aktarılır ve altprogramda tanımlanan değişkenler yine yığın üzerinde yer alır. İşlevleri izleyen bölümde tanımlanan Grup veri yapısının üyeleri olan değişkenlerin yığın üzerinde tanımlanması, programın bütünü içinde yürütülme sırasında denetlenemeyen ve sonuçları belirsiz işlemlere neden olabilirler. Bu nedenle, MPWFL dili için altprogramlar parametresiz olarak tanımlanmış ve tüm değişkenlerin Global olması istenmiştir. Ayrıca değişik yüksek seviyeli dillerle karmaşık programlama yapıldığında, bu özellikler programları birleştirme işlemi için kolaylıklar sağlamaktadır.

Altprogramlar parametresiz olduklarından, belirlenen değişkenlere parametre aktarımı yapıldıktan sonra, altprogramlar çağrılabilir. Bir altprogram başka bir altprogramı çağırabilir. Altprogram içinde yeni bir altprogram tanımlanamaz. MPWFL dilinde bir altprogram gibi tanımlanan "main" bölümü aslında C dili kodu için ana program olmaktadır ve çevirme işlemi bunu gözeterek çalışmaktadır.

3.2.1.4 VERİ TIPLERİ

3.2.1.4.1 MANTIKSAL VERİ TİPİ

Mantıksal değişkenler, BOOLEAN anahtar sözcüğü ile tanımlanır. Bu değişkenler, TRUE veya FALSE değerini alabilirler. C diline çevrildiğinde "unsigned int" tipi olarak 16 bitlik yer kaplarlar. TRUE için 1, FALSE için 0 değerini alırlar. Mantıksal değişkenler için, NOT, AND, OR ve IMP mantıksal operatörleri kullanılabilir.

3.2.1.4.2 TAMSAYI VERİ TİPİ

Tamsayı değişkenler, INTEGER anahtar sözcüğü ile tanımlanır. Tamsayı değişkenler, -32768 ile +32767 aralığındaki değerlerden birini alabilirler. C diline çevrildiklerinde, "int" tipi olarak 16 bitlik yer kaplarlar. Tamsayı değişkenler için, +, -, *, / ve MOD operatörleri veya LSS, GTR, EQL, LEQ, GEQ, NEQ gibi karşılaştırma operatörleri kullanılabilir.

3.2.1.4.3 KATAR VERİ TİPİ

Katar değişkenleri, STRING anahtar sözcüğü ile tanımlanır.

C diline çevrildiklerinde, 80 sekizlik bir karakter dizisine ("char değişken_adı[80]") karşı düşer. Her türlü katar işlemlerinde, dizi uzunluğu 80'e sınırlandırılır. Karakter dizisi 0. dizi elemanından başlar ve "0x00" ile sonlandırılır (C dilindeki katar formatı değiştirilmeden kullanılmıştır).

3.2.1.4.4 VERİ TAMPONU DEĞİŞKEN TİPİ

Veri tamponu değişkenleri, BUFFER sözcüğü ile tanımlanır. C diline çevrildiklerinde, 3 alt alandan oluşan bir kayıt yapısına karşı düşer. Bu alt alanlardan ikisi işaretsiz tamsayı ("unsigned int") ve diğeri ise işaretsiz karakter işaretçisi ("unsigned char *pointer") tipindedir. Veri tamponu kayıt yapısı aşağıda tanımlanmıştır:

```
struct BUFFER{
    unsigned int COUNT;
    unsigned int MAX;
    unsigned char *POINTER;
}
```

Veri tamponu değişkenini oluşturan alanların taşıdıkları bilgi aşağıdaki gibidir:

COUNT : veri tamponunda mevcut eleman sayısı

MAX : veri tamponunun alabileceği maksimum eleman sayısı

POINTER : veri tamponunun başlangıç adresi

Veri tamponu işlemlerinde, POINTER alanı NULL değerini taşıyorsa yani, veri tamponu için sistemden bellek istenmemişse, işlem yürütülmez; MAX alanındaki değer kadar veri üzerinde işlem yürütülür. Atama işlemlerinde varış için, MAX alanındaki değer uzunluğunda yeni bir bellek alanı istenir ve COUNT alanındaki değer kadar veri kaynaktan varışa kopyalanır.

3.2.1.4.5 GÖREV DEĞİŞKEN TİPİ

Görev değişkenleri, TASK sözcüğü ile tanımlanır. C diline çevrildiklerinde, 3 alandan oluşan bir kayıt yapısına karşı düşer. Bu alanlar sırasıyla, 80 sekizli uzunluğunda bir karakter dizisi, bir tamsayı ve veri tamponu kayıt yapısıdır. Görev kayıt yapısı aşağıda

tanımlanmıştır:

```
struct TASK{
    char NAME[80];
    int FIELD;
    struct BUFFER POINTER;
}
```

Görev değişkenini oluşturan alanların taşıdıkları bilgi aşağıdaki gibidir:

NAME : görev değişkeninin adı

FIELD : görevle ilgili bilgi alanı

POINTER: görevin program kodunu içeren veri tamponunun başlangıç adresi

3.2.1.4.6 İŞ SAHİBİ DEĞİŞKENİ TİPİ

İş sahibi değişkenleri, OWNER sözcüğü ile tanımlanır. C diline çevrildiklerinde, char[16] boyutunda bir karakter dizisi olarak işlem görür. İşlemciler 0-127 arasında numaralandırılmış olarak kabul edilir. 16 sekizlik dizi, 128 bitlik bir sayı olarak yorumlanır. Bu sayıdaki bitler, en soldaki en küçük numaralı işlemciyi ve en sağdaki ise en büyük numaralı işlemciyi gösterecek şekilde sıralanmıştır. Buna göre, 0 numaralı işlemciyi işaret eden 0. bit, dizinin 0. karakterinin 7. bitidir. 127 numaralı işlemciyi işaret eden 127. bit ise dizinin 15. karakterinin 0. bitidir. TASK ve GROUP fonksiyonları, iş sahibi değişkeni ile tanımlanmış işlemciler için gerçekleştirir.

3.2.1.4.7 GRUP DEĞİŞKENİ TİPİ

Ana bilgisayar ile çok işlemcili sistem arasında kullanıcının belirleyebileceği formatta karşılıklı veri aktarımının sağlanması için Grup değişkenleri

tanımlanmaktadır. Kullanıcının belirlediği değişkenlerden oluşan bir grup için; veri gönderme işleminde bir iletişim paketini oluşturma ve veri alma işlemi için iletişim paketini çözümüleme işlemleri kullanıcıya yansıtılmadan kendiliğinden çözümlenmektedir. Grup değişkenleri, GROUP sözcüğü ile tanımlanır ve C diline çevrildiklerinde düğümlerden oluşan tek bağlantılı bir listeyi işaret eder. Listenin düğümleri C dilinde bir kayıt yapısı (structure) oluşturmaktadır. Bu kayıt yapısı aşağıda tanımlanmıştır:

```
struct GROUP{
    char *POINTER;
    unsigned int COUNT;
    unsigned int TYPE;
    struct GROUP *NEXT;
}
```

Grubu oluşturan düğümlerin kayıt alanlarının taşıdıkları bilgi aşağıdaki gibidir:

POINTER : Grup Üyesi değişkenin adresine işaret eder
COUNT: Grup üyesi değişkenin veri gönderme işlemi için kaç sekizlisinin anlamlı olduğunu gösterir
TYPE : Grup üyesi değişkenin tipi saklanır
NEXT : Listenin bir sonraki düğüme, yani bir sonraki grup üyesi değişkenine işaret eder (başka düğüm yoksa NULL)

Grup değişkeni ve ilgili komutlar kullanılarak, seçilecek tipten düğümlerle (değişkenlerle) bir liste yapısı oluşturulabilir. Bu listenin düğümleri (liste üyesi değişkenlerin içerikleri), iş sahibi değişkeni ile önceden belirlenen işlemcilerle gönderilebilir veya işlemcilerden bu düğümlere (liste üyesi değişkenlere) ilişkin bilgiler istenebilir. Çok işlemcili sistem ile program değişkenleri arasında üç tür parametre aktarma şekli olabilir. Bunlar; değişkenler üzerinden, sadece veri gönderme, sadece veri alma ve aynı değişken

Üzerinden veri gönderme\alma türleri olmaktadır.

Bir değişkenin veri gönderme işleminde kaç sekizlisinin anlamlı olduğu kayıt alanının COUNT bilgisi ile belirlenmektedir. Veri alma işleminde ise o değişkene kaç sekizlik verinin aktarılacağını çok işlemcili sistem bildirmektedir. Böylece, sadece veri gönderme ve veri gönderme\alma için düğümün COUNT alanına gereken sekizli sayısı yüklenir. Sadece alma işlemi için ise COUNT alanına 0 değeri yüklenmelidir. Burada aynı değişken için, gönderme\alma işleminde gönderilen sekizli sayısı ile alınan sekizli sayısı aynı olmak zorunda değildir. Veri alma durumunda hatalı işlemlerin engellenmesi için, alınan sekizli sayısının o değişkenin kapsayabileceği maksimum sekizli sayısını aşmaması sağlanmıştır.

3.2.2 KOMUT KOMESİ

3.2.2.1 ATAMA KOMUTLARI

3.2.2.1.1 MANTIKSAL DEĞİŞKENLERE ATAMA

Mantıksal değişkenlere atama işlemi aşağıda simgesel olarak gösterilmiştir. Öncül olarak gösterilen simge listelenen ifadelerden oluşabilir. Mantıksal işlem operatörü olarak; NOT (TOMLEME), AND (VE), OR (VEYA), IMP (ifade_a VE TOMLEME ifade_b) kullanılabilir.

mantıksal değişken = mantıksal ifade

```
mantıksal ifade : ---- Öncül -----
                  |NOT|      | AND |
                  |    |      | OR  |      Öncül
                  |    |      | IMP |
```

Öncül : mantıksal sabit
 mantıksal değişken
 aritmetik karşılaştırma
 katar karşılaştırma
 LOAD fonksiyonu
 SAVE fonksiyonu
 GETTASK fonksiyonu
 SENDTASK fonksiyonu
 RUNTASK fonksiyonu
 ABORTTASK fonksiyonu
 KILLTASK fonksiyonu
 GETGROUP fonksiyonu
 SENDGROUP fonksiyonu
 (mantıksal ifade)

Örnekler:

```
boolA = boolB AND TRUE;
boolA = integerA EQU integerB;
boolA = SENDTASK(ownerA,taskA);
```

3.2.2.1.2 TAMSAYI DEĞİŞKENLERE ATAMA

Tamsayı değişkenlere atama işlemi aşağıda simgesel olarak gösterilmiştir. Öncül olarak gösterilen simge listelenen ifadelerden oluşabilir. Tamsayı işlem operatörü olarak; - (eksileme), + (toplama), - (çıkarma), * (çarpma), / (bölme), MOD (tamsayı bölmede kalan) kullanılabilir.

tamsayı değişken = tamsayı ifade;

tamsayı ifade :	----	Öncül	-----
	-	+	
		-	
		*	Öncül
		/	
		MOD	

Öncül : tamsayı sabit
 tamsayı değişken
 DECIMAL fonksiyonu
 LENGTH fonksiyonu
 ISFILE fonksiyonu
 görev değişkeni [FIELD]
 (tamsayı ifade)

Örnekler:

```
integerC = 4+(integerA - integerB);
integerC = DECIMAL(stringA);
integerC = taskA[FIELD];
```

3.2.2.1.3 KATAR DEĞİŞKENLERİNE ATAMA

Tamsayı değişkenlere atama işlemi aşağıda simgesel olarak gösterilmiştir.

Katar değişkeni = katar ifadesi;

katar ifadesi : katar sabiti
 katar değişkeni
 ACCEPT fonksiyonu
 CONCAT fonksiyonu
 ASCII fonksiyonu
 TAKE fonksiyonu
 DROP fonksiyonu
 görev değişkeni [NAME]

Örnekler:

```
stringC = `fileA`;
stringC = CONCAT(stringA);
stringC = taskA[NAME];
```

3.2.2.1.4 GÖREV DEĞİŞKENLERİNE ATAMA

Görev değişkeninin, değişik tiplerde olan üç alt alanına ayrı ayrı atamalar yapılabilir. Bu atamalar aşağıda gösterilmiştir:

```
görevdeğişkeni[NAME= katar değişkeni veya katar sabiti];
görevdeğişkeni[FIELD=tamsayısabıt veya tamsayıdeğişken];
görevdeğişkeni[POINTER = veri tamponu değişkeni veya
görev değişkeni [POINTER]];
```

Örnekler:

```
taskC[NAME = 'fileA'];
taskC[FIELD = 12];
taskC[POINTER = bufferA];
```

3.2.2.1.5 VERİ TAMPONU DEĞİŞKENLERİNE ATAMA

Veri Tamponu değişkenlerine atama işlemi aşağıda simgesel olarak gösterilmiştir.

```
veri tamponu değişkeni = veri tamponu ifadesi;

veri tamponu ifadesi : veri tamponu değişkeni
                      ALLOC fonksiyonu
                      REVERSE fonksiyonu
                      PACK fonksiyonu
                      UNPACK fonksiyonu
                      görev değişkeni [POINTER]
```

Örnekler:

```
bufferC = bufferA;
bufferC = ALLOC(100);
bufferC = taskA[POINTER];
```

3.2.2.2 GRUP KOMUTLARI

3.2.2.2.1 GROUPNAME

Bu komut ile, grup listesine simgesel bir isim atanır.

Komut formatı :

GROUPNAME (parametre1, parametre2)

parametre1 = grup değişkeni adı

parametre2 = katar sabiti veya katar değişkeni adı

Açıklama:

Grup değişkeninin işaret ettiği bir liste yoksa (liste işaretçisi NULL ise), aşağıda tanımlanan işlemler yapılır; aksi halde, herhangi bir işlem yapılmaksızın geri dönlür. Başlangıçta sistemden, katarın uzunluğu kadar bellek istenir. Katar, bu bellek alanına kopyalanır. Grup değişkeninin işaret ettiği liste için ilk düğüm elemanı oluşturulur. Bu düğümde, katarın adresi, katarın uzunluğu ve düğüm tipi (tip=0) bilgileri yer alır.

Örnek:

```
GROUPNAME(group1,'gname');
```

3.2.2.2.2 GROUPMAKE

Bu komut ile, grup listesine yeni bir düğüm elemanı eklenir.

Komut formatı:

```
GROUPMAKE (parametre1, parametre2, parametre3)
```

```
parametre1 = grup değişkeni adı
parametre2 = düğüm değişkeni adı
parametre3 = tamsayı sabit veya tamsayı değişkeni
```

Açıklama:

Grup değişkeninin işaret ettiği listeye daha önce GROUPNAME komutu ile simgesel bir isim atanmış ise, aşağıdaki işlemler yapılır. Aksi halde herhangi bir işlem yapılmaksızın geri dönlür. Parametre2 ile verilen değişken, listenin yeni bir düğümünü oluşturur. Bu düğümün tipi, eğer değişkenin tipi:

```
BOOLEAN ise, 1;
INTEGER ise, 2;
STRING ise, 3;
BUFFER ise, 4;
```

olarak belirlenir.

Listenin son elemanı olan bu düğümün bilgileri, parametre2 ile verilen değişkenin adresi, değişkenin tipi ve parametre3 ile verilen sekizli sayısı (COUNT) olacaktır.

Örnek:

```
GROUPMAKE (groupA,varA,2);
```

3.2.2.3 GÖREV KOMUTLARI

3.2.2.3.1 TASKINIT

Bu komut, bir görev değişkenine ilk koşul değerlerini atar.

Komut Formatı :

```
TASKINIT (parametre)
```

```
parametre = görev değişkeni
```

Açıklama:

Görevin isim (NAME) alanına boş dizi, tamsayı (FIELD) alanına 0 değeri atanır. Veri tamponu (BUFFER) alanı eğer, bir bellek bölgesine işaret ediyorsa, bu bellek bölgesi, sistemin belleğine iade edilir ve veri tamponu alanına NULL atanır.

Örnek:

```
TASKINIT (task1);
```

3.2.2.3.2 TASKCOPY

Bu komut, kaynak görev değişkeninin içeriklerini varış görev değişkenine atar.

Komut Formatı:

TASKCOPY (parametre1, parametre2)

parametre1 = varış görev değişkeni
parametre2 = kaynak görev değişkeni

Açıklama:

Varış görev değişkeni için önce, TASKINIT işlemi yürütülür. Kaynak görev değişkeninin isim ve tamsayı alanları, varış görev değişkenine atanır. Kaynak görev değişkeninin veri tamponu alanı NULL değilse, sistemden gerekli büyüklükte tampon bellek istenerek kaynak veri tamponunun işaret ettiği bellek bölgesi bu bellek alanına kopyalanır. Tampon belleğin adresi, varış veri tamponunun işaretçisine (POINTER) atanır. Son olarak kaynak veri tamponunun MAX ve COUNT değerleri varışa atanır.

Örnek:

TASKCOPY (task1, task2);

3.2.2.4 GÖRONTOLEME KOMUTLARI

3.2.2.4.1 DISPLAY

Ekranda yeni bir satıra karakter dizisi görüntülenir.

Komut Formatı :

DISPLAY (parametre)

parametre = katar sabiti veya katar değişkeni

Açıklama:

Yeni bir satır için CR LF (satır başı, bir satır atla) işlemleri yürütülür. Parametre ile belirlenen karakter dizisi, büyük harflerle ekranda görüntülenir.

Örnek:

```
DISPLAY ('program sonu');
```

3.2.2.4.2 PLINE

Ekranda karakter dizisi görüntülenir.

Komut Formatı:

PLINE (parametre)

parametre = katar sabiti veya katar değişkeni

Açıklama:

Parametre ile verilen karakter dizisi, imleç'in (cursor) ekranda kaldığı yerden başlayarak, büyük harflerle görüntülenir.

Örnek:

```
PLINE (dizi1);
```

3.2.2.4.3 PCONTROL

Ekran kontrol karakterleri işlenir.

Komut Formatı:

PCONTROL (parametre)

parametre = katar sabiti

Açıklama:

Parametre ile verilen C dilindeki 'backslash' ekran kontrol kodları yürütülür. Parametrede kodlar büyük harfle verilirse, küçük harfe çevrilir.

Backslash kodları:

\b	geri silme (backspace)
\f	sayfa atla (form feed)
\n	yeni satır (carriage return, line feed)
\r	satır başı (carriage return)
\t	yatay tab (tab)

Örnek:

```
PCONTROL ('\f\n\n');
```

3.2.2.5 AKIŞ KONTROL KOMUTLARI

3.2.2.5.1 IF

If-then ve if-then-else kontrol yapısı.

Komut Formatı:

```
IF (mantıksal ifade) THEN komut(lar) ENDIF
veya,
IF (mantıksal ifade) THEN komut(lar) ELSE komut(lar)
ENDIF
```

Açıklama:

Mantıksal ifade sonucu doğru ise, THEN sözcüğünü izleyen komutlar yürütülür ve ENDIF'i izleyen komutlarla program akışı devam eder. Eğer mantıksal ifade sonucu yanlış ise ve ELSE sözcüğünü izleyen komutlar varsa bu komutlar yürütülür, aksi halde IF komutuna ilişkin işlem sonlanmış olur. İç içe if komutları kullanmak mümkündür.

Örnek:

```
IF (a GTR b) THEN a=1 ;
                ELSE a=2 ;
                ENDIF;
```

3.2.2.5.2 WHILE

While kontrol yapısı.

Komut Formatı:

```
WHILE (mantıksal ifade) BEGIN komut(lar) ENDWHILE
```

Açıklama:

Mantıksal ifade (ki burada sadece mantıksal değişken veya mantıksal sabit olabilir) doğru ise BEGIN sözcüğünü izleyen komut(lar) ENDWHILE sözcüğüne kadar yürütülür, ENDWHILE sözcüğünden sonra program akışı WHILE sözcüğüne geri döner. Mantıksal ifade yanlış ise, program akışı, ENDWHILE sözcüğünü izleyen komutla devam eder.

Örnek:

```
WHILE (bayrak) BEGIN
    a=a+b;
    IF (a GTR 50) THEN bayrak=FALSE;
    ENDIF;
ENDWHILE;
```

3.2.2.5.3 REPEAT

Repeat-Until kontrol yapısı.

Komut Formatı:

```
REPEAT komut(lar) UNTIL (mantıksal ifade)
```

Açıklama:

Program akışı, REPEAT sözcüğünü izleyen komutla sürer ve UNTIL sözcüğü ile verilen mantıksal ifade (ki burada sadece mantıksal sabit veya mantıksal değişken olabilir) sonucu eğer doğru ise, REPEAT sözcüğüne geri döner; yanlış ise, UNTIL sözcüğünden sonraki komutla devam eder.

Örnek:

```
REPEAT
  a=a+b;
  IF (a GTR 50) THEN bayrak=FALSE;
  ENDIF;
UNTIL (bayrak);
```

3.2.2.5.4 SELECT

Select kontrol yapısı.

Komut Formatı:

```
SELECT
  seçim listesi
ENDSELECT
```

veya,

```
SELECT
  seçim listesi
  OTHERWISE
  komutlar
ENDSELECT
```

seçim listesi: WHEN (mantıksal ifade) DO komutlar
ENDWHEN;

Açıklama:

SELECT gövdesi içinde yer alan WHEN sözcüğünü izleyen mantıksal ifade eğer doğru ise DO sözcüğünü izleyen komutlar yürütülür, aksi halde bir sonraki WHEN için işlem yinelenir. Eğer OTHERWISE sözcüğü var ise ve, WHEN mantıksal ifadelerinin hiçbirisi doğru değilse, OTHERWISE sözcüğünü izleyen komutlar yürütülecektir.

Örnek:

```
SELECT
  WHEN (a EQL 5) DO DISPLAY('a = 5') ENDWHEN;
  WHEN (a EQL 6) DO DISPLAY('a = 6') ENDWHEN;
  OTHERWISE
    DISPLAY('a 5 ve 6 değil!!');
ENDSELECT
```

3.2.2.6 İŞ SAHİBİ KOMUTLARI

3.2.2.6.1 RESETOWNER

İş sahibi değişkenine başlangıç değerinin atanması.

Komut Yapısı:

```
RESETOWNER (parametre1, parametre2);
```

parametre1 = iş sahibi değişkeni

parametre2 = tamsayı sabit veya tamsayı değişken

Açıklama:

Parametre1 ile verilen iş sahibinin; 0. işlemci numarasından, parametre2 ile verilen tamsayı bilgisine karşı düşen işlemci numarasına kadar olana işlemci bitleri LOJİK 1'lenir. Geri kalan işlemci bitleri ise LOJİK 0'lanır. Örneğin, parametre2'ye 0 verilmesi ile tüm bitler LOJİK 0'lanır, 1 verilmesi ile de, 0. bit LOJİK 1 olur, diğerleri ise LOJİK 0'lanır.

Örnek:

```
RESETOWNER (ownerA,4);
```

Bu komuttan sonra ownerA değişkeninin içeriği:

1 1 1 1 0 0 ... 0

olmaktadır.

3.2.2.6.2 ADDOWNER

İş sahibi değişkenine yeni bir iş sahibi ekleme.

Komut Yapısı:

```
ADDOWNER (parametre1, parametre2);
```

parametre1 = iş sahibi değişkeni

parametre2 = tamsayı sabit veya tamsayı değişken

Açıklama:

Parametre1 ile verilen iş sahibi değişkeninin, parametre2 ile belirtilen tamsayı bilgisine karşı düşen biti LOJİK 1'lenir. Diğer işlemci numaralarına (iş sahiplerine) karşı düşen bitler komuttan etkilenmez.

Örnek:

ADDDOWNER (ownerB, 2);

Komut yürütülmeden önce ownerB değişkeninin içeriği:

0 0 0 0 0

ise, komut yürütüldükten sonra ownerB değişkeninin içeriği:

0 0 1 0 0

olacaktır.

3.2.2.6.3 SUBOWNER

İş sahibi değişkeninden bir iş sahibinin çıkarılması.

Komut Yapısı:

SUBOWNER (parametre1, parametre2);

parametre1 = iş sahibi değişkeni

parametre2 = tamsayı sabit veya tamsayı değişken

Açıklama:

Parametre1 ile verilen iş sahibinin, parametre2 ile belirtilen tamsayı bilgisine karşı düşen biti LOJİK 0'lanır. Diğer işlemci numaralarına (iş sahiplerine) karşı düşen bitler komuttan etkilenmez.

Örnek:

SUBOWNER (ownerC, 2);

Komut yürütülmeden önce ownerC değişkeninin içeriği:

1 1 1 0 0

ise, komut yürütüldükten sonra ownerC değişkeninin içeriği:

1 1 0 0 0

olacaktır.

3.2.2.6.4 MOVEOWNER

İş sahibi değişkenine yeni iş sahiplerini ekleme.

Komut Yapısı:

MOVEOWNER (parametre1, parametre2);

parametre1 = iş sahibi değişkeni;

parametre2 = tamsayı sabit veya tamsayı değişken

Açıklama:

Parametre2 ile verilen tamsayı bilgisinin düşük anlamlı sekizlisi, parametre1 ile belirtilen iş sahibine yüklenmesi istenen bilgiyi; yüksek anlamlı sekizlisi ise, iş sahibinin hangi sekizlisine bu yüklemenin yapılacağı bilgisini gösterir. Böylece sekiz iş sahibi, tek bir komut yürütülerek bir iş sahibi değişkenine atanabilir.

Örnek:

MOVEOWNER (ownerD, 25);

25 tamsayısının ikili sayı sisteminde karşılığı:

00000000 00011001

olmaktadır. 0 halde bu değere karşı düşen işlemci numaraları ise:

3. 4. 7.

olur. Parametre2'de yüksek anlamlı sekizli 0 değerinde olduğundan, ownerD iş sahibi değişkeninin 0. sekizlisine, parametre2'nin düşük anlamlı sekizlisinde yer alan 25 sayısı yüklenecektir.

3.2.2.7 FONKSİYONLAR

3.2.2.7.1 KATAR FONKSİYONLARI

3.2.2.7.1.1 ACCEPT

Bir giriş/çıkış fonksiyonudur.

Fonksiyon Formatı:

ACCEPT (argüman);

argüman = katar sabiti veya katar değişkeni

Açıklama:

Argüman ile verilen katar dizisi ekrana DISPLAY edilir ve kullanıcıdan beklenen giriş dizisi alınarak, ilgili katar değişkenine atanır.

Örnek :

```
yanıt=ACCEPT ('programa devam? (E/H)');
```

3.2.2.7.1.2 CONCAT

İki katarı bitiştiren fonksiyon.

Fonksiyon formatı:

CONCAT (argüman);

argüman = katar sabiti veya katar değişkeni

Açıklama:

Atandığı dizi değişkeninin içeriğine, argümanda verilen katarı bitiştirir.

Örnek:

```
katar2='hoş';
```

```
katar2=CONCAT('geldiniz');
```

Son işlemden sonra katar2'in içeriği, 'hoşgeldiniz' olacaktır.

3.2.2.7.1.3 ASCII

Katara dönüştürme fonksiyonu.

Fonksiyon Formatı:

ASCII (argüman);

argüman = tamsayı sabit veya tamsayı değişken

Açıklama:

Tamsayı argüman, katara dönüştürülür.

Örnek:

```
katar3=ASCII(5);  
katar='Sonuc=';  
katar=CONCAT(katar3);  
DISPLAY(katar);
```

Bu komutların yürütülmesiyle ekrana "Sonuc=5" karakterleri çıkar.

3.2.2.7.1.4 TAKE

Katar dilimleme fonksiyonu.

Fonksiyon Formatı:

TAKE (argüman1, argüman2);

argüman1 = katar sabiti veya katar değişkeni

argüman2 = tamsayı sabit veya tamsayı değişken

Açıklama:

Argüman1'de verilen katarın, başından itibaren, argüman2 ile verilen tamsayı kadar karakteri alınır.

Örnek:

```
ilk=TAKE('C:\DOSYA.TXT',3);
```

Bu komuttan sonra ilk değişkeninin içeriği 'C:\' olacaktır.

3.2.2.7.1.5 DROP

Katar dilimleme fonksiyonu.

Fonksiyon Formatı:

```
DROP (argüman1, argüman2);
```

```
argüman1 = katar sabiti veya katar değişkeni
argüman2 = tamsayı sabiti veya tamsayı değişkeni
```

Açıklama:

Argüman1 ile verilen katarın, başından itibaren, argüman2 ile verilen tamsayı kadar karakteri atılır.

Örnek:

```
ilk=DROP('C:\DOSYA.TXT',3);
```

Bu komuttan sonra ilk değişkeninin içeriği 'DOSYA.TXT' olacaktır.

3.2.2.7.2 TAMSAYI FONKSİYONLARI

3.2.2.7.2.1 DECIMAL

Tamsayıya dönüştürme fonksiyonu.

Fonksiyon Formatı:

```
DECIMAL (argüman);
```

```
argüman = katar sabiti veya katar değişkeni
```

Açıklama:

Argüman ile verilen katarla belirtilen sayı, tamsayıya çevirilir.

Örnek:

```
integerA=DECIMAL('3');
```

3.2.2.7.2.2 LENGTH

Katar uzunluğunu veren fonksiyon.

Fonksiyon Formatı:

```
LENGTH (argüman);
```

argüman = katar sabiti veya katar değişkeni

Açıklama:

Argümanda verilen katarın kaç karakter olduğunu verir.

Örnek:

```
u=LENGTH('13579');
```

Bu komuttan sonra u değişkeninin içeriği, 5 olacaktır.

3.2.2.7.2.3 ISFILE

Dosya uzunluğunu veren fonksiyon.

Fonksiyon Formatı:

```
ISFILE (argüman)
```

argüman = katar sabiti veya katar değişkeni

Açıklama:

Argüman ile verilen dosya bulunamamışsa, 0;
dosya bulunmuşsa, dosyanın uzunluğunu verir.

Örnek:

```
dosya_uz=ISFILE('DOSYA.TXT');
```

3.2.2.7.3 VERİ TAMPONU FONKSİYONLARI

3.2.2.7.3.1 ALLOC

Bellek alanı isteyen fonksiyon.

Fonksiyon Formatı:

```
ALLOC (argüman);
```

argüman = tamsayı sabit veya tamsayı değişken

Açıklama:

Sistem belleğinden, argümanda verilen tamsayı kadar karakter (sekizli) uzunluğunda bellek istenir. ALLOC'un atanacağı tampon değişkeninin daha önceden sahiplendiği bellek varsa, sisteme geri verilerek yenisi atanır. ALLOC komutunun yürütülmesinden sonra, tampon değişkeninin alacağı değerler:

```
POINTER = yeni bellek alanının adresi
MAX      = fonksiyonda verilen tamsayı argüman
COUNT  = 0
olacaktır.
```

Örnek:

```
buf1=ALLOC(800);
```

3.2.2.7.3.2 REVERSE

Tampondaki veri sekizlilerinin sırasını değiştiren fonksiyon.

Fonksiyon formatı:

REVERSE (argüman1, argüman2, argüman3);

argüman1 = veri tampon değişkeni veya görev değişkeninin
işaretçi alanı

argüman2 = tamsayı sabit veya tamsayı değişken

argüman3 = tamsayı sabit veya tamsayı değişken

Açıklama:

Argüman1 ile verilen (veya işaret edilen) veri tamponu için, argüman2 eleman sayısını, argüman3 ise bir elemanın kaç sekizliden oluştuğunu gösterir. REVERSE fonksiyonunun yürütülmesi ile, argüman2 ile verilen sayıdaki eleman için, herbir elemanın sekizlileri ters sırada dizilirler. Argüman2 ile argüman3'un çarpımının sonucu eğer tampon değişkeninin COUNT değerinden büyük ise, işlemler COUNT adet sekizliye kadar yürütülür.

Örnek:

buf2=REVERSE(buf1, 2, 2);

Komutun yürütülmesinden sonra veri tamponlarının içerikleri aşağıda verilmiştir:

buf1 : 1 2 3 4

buf2 : 2 1 4 3

3.2.2.7.3.3 PACK

Veri tamponundaki elemanları sıkıştırılmış formatta (ASCII'den HEX'e dönüşüm) yeniden düzenleyen fonksiyon.

Fonksiyon Formatı:

PACK (argüman);

argüman = tampon değişkeni veya görev değişkeninin
işaretçisi

Açıklama:

Argümanda belirtilen veri tamponunun COUNT alanında verilen sayıda sekizlisi üzerinde PACK işlemi uygulanır. Sonuçta COUNT/2 kadar yeni sekizli oluşturulur.

Örnek:

```
buf2=PACK(buf1);
```

Komutun yürütümünden sonra veri tamponlarının içerikleri aşağıda verilmiştir:

```
buf1 : $31 $32 $35 $34
```

```
buf2 : $12 $54
```

3.2.2.7.3.4 UNPACK

Veri tamponu üzerinde önceden PACK edilmiş elemanların üzerinde UNPACK işlemini (HEX'den ASCII'ye dönüşüm) gerçekleyen fonksiyon.

Fonksiyon Formatı:

```
UNPACK (argüman);
```

argüman = tampon bellek değişkeni veya görev
değişkeninin işaretçisi

Açıklama:

Argümanda belirtilen tampon belleğin COUNT alanında verilen sayıdaki elemanı üzerinde UNPACK işlemi uygulanır. COUNT*2 kadar sayıda yeni eleman oluşturulur. Ancak, COUNT*2 sonucu, fonksiyonun atandığı tampon bellek değişkeninin MAX alanında belirtilen değeri aşamaz.

Örnek:

```
buf2=UNPACK(buf1);
```

Komutun yürütümünden sonra veri tamponlarının içerikleri aşağıda verilmiştir:

```
buf1 : $12 $54
```

```
buf2 : $31 $32 $35 $34
```

3.2.2.7.4 MANTIKSAL FONKSİYONLAR

3.2.2.7.4.1 LOAD

Veri tamponu alanına, dosya yükleyen fonksiyon.

Fonksiyon Formatı:

LOAD (argüman1, argüman2);

argüman1 = katar sabiti veya katar değişkeni
argüman2 = veri tamponu değişkeni veya görev
değişkeninin işaretçisi

Açıklama:

Argüman1 ile verilen katar, kaynak dosyanın adıdır. Argüman2 ile verilen veri tamponu, varış bellek alanıdır. Kaynak dosyanın uzunluğu, argüman2 ile verilen tampon değişkeninin MAX alanındaki değerden büyükse; dosyadan MAX uzunluğunda veri (sekizli), veri tamponu alanına taşınır. Aksi halde, dosyadaki tüm veriler tampon alanına aktarılır. Aktarım işlemi tamamlandığında, tampon değişkeninin COUNT alanına, aktarılan sekizli sayısı yüklenir ve, fonksiyon TRUE olarak geri döner.

Fonksiyonun FALSE dönme koşulları ise:

- i) dosya bulunamazsa,
- ii) veri tamponu işaretçisi, NULL değerini taşıyorsa.

Örnek:

```
bool1=LOAD ('OA.TEZ', buf1);
```

3.2.2.7.4.2 SAVE

Veri tamponu alanından bir dosyaya veri aktaran fonksiyon.

Fonksiyon Formatı:

SAVE (argüman1, argüman2, argüman3);

argüman1 = katar sabiti veya katar değişkeni
 argüman2 = tampon bellek değişkeni veya görev
 değişkeninin işaretçisi
 argüman3 = tamsayı sabit veya tamsayı değişken

Açıklama:

Argüman1 aktarım yapılacak varış dosyasının adını taşır.
 Argüman2 ise kaynak veri tamponu alanına işaret eder.
 Argüman3, aktarım yapılabilecek maksimum veri sekizlisi sayısını belirler.

Kaynak veri tamponundaki COUNT adet veri sekizlisi, varış dosyasına yazılır. Tampon değişkeninin COUNT değeri, argüman3 ile verilen tamsayı değerinden büyükse, argüman3 değeri kadar veri sekizlisi dosyaya yazılır. Aktarım işleminden sonra, fonksiyon TRUE değeri ile geri döner.

Fonksiyonun FALSE dönme koşulları:

- i) veri tamponu işaretçisi, NULL değerini taşıyorsa,
- ii) dosya işlemlerinde bir hata oluşursa.

Örnek:

```
bool2=SAVE(fileA, buf2, 2);
```

3.2.2.7.4.3 GETTASK

Belli bir işlemciden, belli bir görevin FIELD alanındaki verilerini isteyen fonksiyon.

Fonksiyon Formatı:

GETTASK (argüman1, argüman2);

argüman1 = iş sahibi değişkeni
 argüman2 = görev değişkeni

Açıklama:

İş sahibi değişkeni ile belirtilmiş işlemciden, görev değişkeninin NAME alanında verilen ad bilgisi gönderilerek, sözkonusu görevin FIELD alanındaki bilgi beklenir. İletişimde hata olmazsa, fonksiyon TRUE değeri ile geri döner. İş sahibi değişkeni, birden fazla işlemci numarası içerirse, en küçük numaralı işlemci için işlemler yürütülür.

Örnek:

```
bool3=GETTASK (owner3, task3);
```

3.2.2.7.4.4 SENDTASK

İş sahibi ile belirtilen tüm işlemcilere, görevin NAME, FIELD ve eğer varsa BUFFER alanındaki bilgiler gönderilir.

Fonksiyon Formatı:

```
SENDTASK (argüman1, argüman2);
```

```
argüman1 = iş sahibi değişkeni  
argüman2 = görev değişkeni
```

Açıklama:

İş sahibi değişkeni ile belirtilen işlemcilerin hepsine, görev değişkeninin NAME, FIELD, ve eğer varsa, BUFFER alanında yer alan bilgiler gönderilir. İletişimde bir hata olmazsa, fonksiyon TRUE değeri ile geri döner.

Örnek:

```
bool4=SENDTASK (owner4, task4);
```

3.2.2.7.4.5 RUNTASK

İş sahibi işlemcilerinde, belirli bir görevin yürütülmesi sağlanır.

Fonksiyon Formatı:

RUNTASK (argüman1, argüman2);

argüman1 = iş sahibi değişkeni
argüman2 = görev değişkeni

Açıklama:

İş sahibi ile belirtilen tüm işlemcilerden, görev adı (NAME) ile verilen görevin yürütülmesi istenir. İletişimde bir hata olmazsa fonksiyon, TRUE değeri ile döner.

Örnek:

bool5=RUNTASK(owner5, task5);

3.2.2.7.4.6 ABORTTASK

Belli bir görevin yürütülmesine ara veren fonksiyon.

Fonksiyon Formatı:

ABORTTASK (argüman1, argüman2);

argüman1 = iş sahibi değişkeni
argüman2 = görev değişkeni

Açıklama:

İş sahibi değişkeni ile belirtilen işlemcilerin hepsinden, görev değişkeninin NAME alanında verilen addaki görevin yürütülmesine ara verilmesi istenir. İletişimde hata olmazsa ve işlem başarıyla tamamlanırsa, fonksiyon TRUE değeri ile geri döner.

Örnek:

```
bool6=ABORTTASK(owner6, task6);
```

3.2.2.7.4.7 KILLTASK

Bir görevin varlığını sonlandıran fonksiyon.

Fonksiyon Formatı:

```
KILLTASK (argüman1, argüman2);
```

```
argüman1 = iş sahibi değişkeni  
argüman2 = görev değişkeni
```

Açıklama:

iş sahibi değişkeni ile belirtilen işlemcilerden, görev adı ile verilen görevin yok edilmesi istenir. İletişimde bir hata olmazsa ve işlem başarıyla tamamlanırsa fonksiyon, TRUE değeri ile geri döner.

Örnek:

```
bool7=KILLTASK(owner7, task7);
```

3.2.2.7.4.8 GETGROUP

Belli bir işlemciden, grup bilgilerini isteyen fonksiyon.

Fonksiyon Formatı:

```
GETGROUP (argüman1, argüman2);
```

```
argüman1 = iş sahibi değişkeni  
argüman2 = grup değişkeni
```

Açıklama:

iş sahibi değişkeni ile belirlenen işlemciden, grup

değişkeni ile adı verilen gruba ilişkin bilgiler istenir. Gelen bilgiler, grup listesindeki ilgili değişkenlerine atanır. İletişimde hata olmazsa ve işlem başarıyla tamamlanırsa, fonksiyon TRUE değeri ile geri döner. İş sahibi değişkeni, birden fazla işlemci numarası içerirse, en küçük numaralı işlemci için işlemler yürütülür.

Örnek:

```
bool8=GETGROUP(owner8, group8);
```

3.2.2.7.4.9 SENDGROUP

İş sahibi işlemcilerine, belirlenen grubun bilgilerini gönderen fonksiyon.

Fonksiyon Formatı:

```
SENDGROUP (argüman1, argüman2);
```

```
argüman1 = iş sahibi değişkeni  
argüman2 = grup değişkeni
```

Açıklama:

Grup değişkeni ile belirlenen grubun üyeleri ile bir veri paketi oluşturulur. İş sahibi değişkeni ile belirtilen işlemcilere, bu paket gönderilir. Eğer iletişimde bir hata oluşmaz ve işlem başarıyla tamamlanırsa, fonksiyon TRUE değeri ile geri döner.

Örnek:

```
bool9=SENDGROUP(owner9, group9);
```

3.2.2.7.5 BASIT KOMUTLAR

Bu grup içinde; sadece komut adı simgesinden oluşturulmuş, parametresi olmayan komutlar yer alır.

3.2.2.7.5.1 NOP

Bu komutta herhangi bir işlem yürütülmez.

3.2.2.7.5.2 RETURN

Altprogram metni sonlanmadan, altprogramdan dönüş için kullanılır. Altprogram metninin en sonunda bir geri dönüş komutu yer almasına gerek yoktur.

3.2.2.7.5.3 BREAK

Program akış kontrol komutu ile çerçevelenmiş blok içinden çıkış komutu.

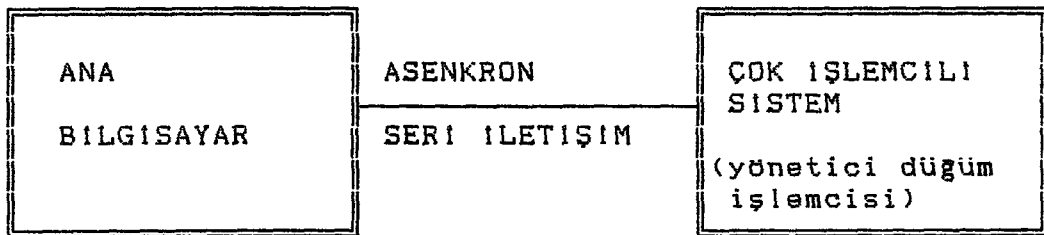
3.3 ANA BİLGİSAYAR VE ÇOK İŞLEMCİLİ SİSTEM ARASINDAKİ İLETİŞİM

3.3.1 İLETİŞİMİN YAPISI

Hedef alınan sistemde Ana (Host) bilgisayar ve Çok işlemcili sistem yer alır (Şekil 3.1). Ana bilgisayar kullanıcı ile sistemin doğrudan ilişkisini sağlar. Burada ana bilgisayar; program geliştirmede, programların ve işlenecek olan verilerin çok işlemcili sisteme yüklenmesinde, çok işlemcili sistemin denetlenmesinde ve çok işlemcili sistemin ürettiği sonuçların saklanması kullanılmaktadır. Ana bilgisayar olarak kişisel bilgisayarların kullanılacağı

düşünülmektedir. Ancak iş istasyonlarının da ana bilgisayar olarak kullanılabilmesi sağlanmaktadır. Ana bilgisayar ile çok işlemcili sistem arasındaki iletişimin yapısı ve hızının sistemin genel verimliliği üzerindeki etkisi önemli olabilir. Çoğunlukla var olması nedeni ile örnek uygulamada iki birim arasındaki iletişim ortamı olarak Asenkron Seri İletişim seçilmiştir.

Seri bağlantı, ana bilgisayar ile çok işlemcili sistemin seçilen bir yönetici düğüm işlemcisi arasında yapılmaktadır. Çok işlemcili sistemin diğer düğüm işlemcilerine veri aktarımı ise sistemin doğal yapısında var olan iletişim hatları ile seçilen yönetici düğüm işlemcisi üzerinden yapılmaktadır. Seri iletişim hızının 9600 baud seçilmesine rağmen büyük boyutlu verilerin aktarımı uzun sürebilir ve sonuçta sistemin genel verimi beklenen düzeye çıkamayabilir. Bu sakıncaları ortadan kaldırmak için iletişim ortamı ve yapısı kullanıcıya açık bırakılmıştır. Kullanıcı C dilinde hazırlanan destek programları değiştirebilir. Böylece, örneğin LAN bağlantısı kurularak ana bilgisayarın tüm düğüm işlemcilerine doğrudan ve hızlı olarak veri aktarması mümkün olabilir. Bu tür değişiklikler MPWFL dilinde hazırlanan iş programının değiştirilmesini gerektirmemektedir.



Şekil 3.1 Ana bilgisayar ve sistem bağlantısı

3.3.2 İLETİŞİM PROTOKOLLARI

Kullanıcının MPWFL dilinde yazdığı iletişime yönelik komutlar, destek programları ile tanımlanmış paket formatları haline dönüştürülürler, iletilirler ve yanıtlar tekrar çözülürler. Temel alınan iletişim protokolları, ana bilgisayar ve çok işlemcili sistem tarafından karşılıklı istek\yanıt şeklinde yürütülürler. Veri paketlerinin sonuna CRC hesaplanarak eklenir ve paketi alan birim tarafından CRC sınaması yapılır. CRC hesabı, veri paketinin tüm sekizlileri üzerinde EXOR işlemi yapılarak gerçekleştirilir.

Grup komutları dışındaki iletişim komutları sabit yapıda paket formatlarına sahiptir. Grup düğüm üyelerinin kullanıcı tarafından belirlenmesinin bir sonucu olarak, ana bilgisayar ile çok işlemcili sistem arasında kullanıcının istediği formatta veriler aktarılabilir. Aşağıdaki bölümlerde komutların sonucunda ana bilgisayar tarafından üretilen paketler ve çok işlemcili sistemin yanıtları açıklanmaktadır. Burada açıklanan protokoller ana bilgisayar ile çok işlemcili sistem arasında senkron seri bağlantı olmasına göre hazırlanmıştır. Gereğinde destek programları C dilinde tekrar düzenlenerek daha farklı paket formatları ve iletişim protokolları hazırlanabilir. Bu durumda, MPWFL dilinde hazırlanan iş programının değiştirilmesi gerekmemektedir.

3.3.2.1 SENDTASK İLETİŞİM FORMATI

Kullanıcının değişken adı ile belirlediği görev için ad (NAME), tamsayı (FIELD) ve program kodu (POINTER ile işaret edilen veri tamponu içeriği) bir paket haline dönüştürülür. İş sahibi işlemcilere bu paketin doğru olarak gönderilmesini sağlayan yönetici düğüm işlemcisiinden onay gelmesi durumunda komuttan TPAE

degeri döndürülür. Aksi halde komut FALSE degeri ile döner.

Paket formatı (Ana -> Yönetici):

2 sekizli : tüm paketteki sekizli sayısı
 1 sekizli : CRC(paketteki sekizli sayısı alanı için)
 2 sekizli : 'ST' sendtask format kodu
 16 sekizli : iş sahibi değişkenin içeriği
 2 sekizli : görev adı karakter sayısı (n)
 n sekizli : görev adı
 2 sekizli : görev tamsayı alanı
 2 sekizli : görev kodu sekizli sayısı (m)
 m sekizli : görev program kodu
 1 sekizli : CRC ('ST' format kodu ve sonrası için)

Yanıt formatı (Yönetici -> Ana):

1 sekizli :yanıt kodu ('O':işlem tamam,'F':işlem hatalı)

3.3.2.2 RUNTASK İLETİŞİM FORMATI

Kullanıcının değişken adı ile belirlediği görev için ad bir paket haline dönüştürülür. İş sahibi işlemcilere bu paketin doğru olarak gönderilmesini sağlayan yönetici düğüm işlemcisinden onay gelmesi durumunda komuttan TRUE degeri döndürülür. Aksi halde komut FALSE degeri ile döner.

Paket formatı (Ana -> Yönetici):

2 sekizli : tüm paketteki sekizli sayısı
 1 sekizli : CRC (paketteki sekizli sayısı alanı için)
 2 sekizli : 'RT' runtask format kodu
 16 sekizli : iş sahibi değişkenin içeriği
 2 sekizli : görev adı karakter sayısı (n)
 n sekizli : görev adı
 1 sekizli : CRC ('RT' format kodu ve sonrası için)

Yanıt formatı (Yönetici -> Ana):

1 sekizli :yanıt kodu ('O':işlem tamam,'F':işlem hatalı)

3.3.2.3 ABORTTASK İLETİŞİM FORMATI

Kullanıcının değişken adı ile belirlediği görev için ad bir paket haline dönüştürülür. İş sahibi işlemcilere bu paketin doğru olarak gönderilmesini sağlayan yönetici düğüm işlemcisinden onay gelmesi durumunda komuttan TRUE değeri döndürülür. Aksi halde komut FALSE değeri ile döner.

Paket formatı (Ana -> Yönetici):

2 sekizli : tüm paketteki sekizli sayısı
 1 sekizli : CRC (paketteki sekizli sayısı alanı için)
 2 sekizli : 'AT' aborttask format kodu
 16 sekizli : iş sahibi değişkenin içeriği
 2 sekizli : görev adı karakter sayısı (n)
 n sekizli : görev adı
 1 sekizli : CRC ('AT' format kodu ve sonrası için)

Yanıt formatı (Yönetici -> Ana):

1 sekizli :yanıt kodu ('O':işlem tamam,'F':işlem hatalı)

3.3.2.4. KILLTASK İLETİŞİM FORMATI

Kullanıcının değişken adı ile belirlediği görev için ad bir paket haline dönüştürülür. İş sahibi işlemcilere bu paketin doğru olarak gönderilmesini sağlayan yönetici düğüm işlemcisinden onay gelmesi durumunda komuttan TRUE değeri döndürülür. Aksi halde komut FALSE değeri ile döner.

Paket formatı (Ana -> Yönetici):

2 sekizli : tüm paketteki sekizli sayısı
 1 sekizli : CRC (paketteki sekizli sayısı alanı için)
 2 sekizli : 'KT' killtask format kodu
 16 sekizli : iş sahibi değişkenin içeriği
 2 sekizli : görev adı karakter sayısı (n)
 n sekizli : görev adı
 1 sekizli : CRC ('KT' format kodu ve sonrası için)

Yanıt formatı (Yönetici -> Ana):

1 sekizli :yanıt kodu ('0':işlem tamam,'F':işlem hatalı)

3.3.2.5 GETTASK İLETİŞİM FORMATI

Kullanıcının değişken adı ile belirlediği görev için ad bir paket haline dönüştürülür. İş sahibi işlemcisine bu paketin doğru olarak gönderilmesini sağlayan yönetici düğüm işlemcisinden onay gelmesi durumunda komuttan TRUE değeri ile parametre geri döndürülür. Aksi halde komut FALSE değeri ile döner.

Paket formatı (Ana -> Yönetici):

2 sekizli : tüm paketteki sekizli sayısı
 1 sekizli : CRC (paketteki sekizli sayısı alanı için)
 2 sekizli : 'GT' gettask format kodu
 1 sekizli : iş sahibinin numarası
 2 sekizli : görev adı karakter sayısı (n)
 n sekizli : görev adı
 1 sekizli : CRC ('GT' format kodu ve sonrası için)

Yanıt formatı (Yönetici -> Ana):

1 sekizli :yanıt kodu ('0':işlem tamam,'F':işlem hatalı)
 2 sekizli : görev tamsayı alanı (yanıt kodu '0' ise)
 1 sekizli : CRC (yanıt kodu '0' ise)

3.3.2.6 SENDGROUP İLETİŞİM FORMATI

Kullanıcının değişken adı ile belirlediği grup için ad ve grup listesinin üyelerinin içerikleri bir paket haline dönüştürülür. İş sahibi işlemcilere bu paketin doğru olarak gönderilmesini sağlayan yönetici düğüm işlemcisinden onay gelmesi durumunda komuttan TRUE değeri döndürülür. Aksi halde komut FALSE değeri ile döner.

Paket formatı (Ana -> Yönetici):

```

2 sekizli : tüm paketteki sekizli sayısı
1 sekizli : CRC (paketteki sekizli sayısı alanı için)
2 sekizli : 'SG' sendgroup format kodu
16 sekizli : iş sahibi değişkenin içeriği
2 sekizli : grup adı karakter sayısı (n)
n sekizli : grup adı
2 sekizli : 1. grup üyesinin sekizli sayısı (m1)
m1 sekizli : 1. grup üyesinin içerikleri
2 sekizli : k. grup üyesinin sekizli sayısı (mk)
mk sekizli : k. grup üyesinin içerikleri
1 sekizli : CRC ( 'SG' format kodu ve sonrası için)

```

Yanıt formatı (Yönetici -> Ana):

```

1 sekizli :yanıt kodu ('O':işlem tamam,'F':işlem hatalı)

```

3.3.2.7. GETGROUP İLETİŞİM FORMATI

Kullanıcının değişken adı ile belirlediği grup için ad ve grup listesinin üye sayısı bir paket haline dönüştürülür. İş sahibi işlemcisine bu paketin doğru olarak gönderilmesini sağlayan yönetici düğüm işlemcisinden onay gelmesi durumunda komuttan TRUE değeri ile grup üyelerine atanacak parametreler geri döndürülür. Aksi halde komut FALSE değeri ile döner.

Paket formatı (Ana -> Yönetici):

```

2 sekizli : tüm paketteki sekizli sayısı
1 sekizli : CRC (paketteki sekizli sayısı alanı için)
2 sekizli : 'GG' getgroup format kodu
1 sekizli : iş sahibinin numarası
1 sekizli : grup üyelerinin sayısı
2 sekizli : grup adı karakter sayısı (n)
n sekizli : grup adı
1 sekizli : CRC ( 'GG' format kodu ve sonrası için)

```

Yanıt formatı (Yönetici -> Ana):

```

1 sekizli :yanıt kodu ('O':işlem tamam,'F':işlem hatalı)
2 sekizli :tüm paketteki sekizli sayısı (yanıt : 'O' ise)
1 sekizli :CRC(paket sekizli sayısı için)(yanıt:'O' ise)
2 sekizli :1.grup üyesi sekizli sayısı(m1)(yanıt:'O'ise)
m1 sekizli:1. grup üyesinin içerikleri (yanıt : 'O' ise)
2 sekizli :k.grup üyesi sekizli sayısı(mk)(yanıt:'O'ise)
mk sekizli:k. grup üyesinin içerikleri (yanıt : 'O'ise)
1 sekizli :CRC(grup üyeleri için)(yanıt : 'O' ise)

```

BÖLÜM 4. DERLEYİCİ TASARIMI

Derleyici, yüksek düzeyli bir programlama dilinde yazılmış olan metni giriş olarak alan ve birleştirici dil (assembler) veya makine dili gibi alt düzeyde olan bir dilde metin ya da program kodu üreten bir programdır [6], [7], [8]. Derleyicilerin ilk geliştirildikleri yıllar olan 1940'lı yılların sonlarında derleyici yazılımı, zor ve karmaşık programlar olarak değerlendiriliyordu. Günümüzde, derleyici yazılımında kullanılan kimi yazılım geliştirme araçları (tools) mevcuttur [9], [10]. Bu bölümde derleyicilerin yapısına yönelik açıklamalar yer almaktadır. MPWFL dili için LR ayrıştırma yöntemi seçildiğinden LR ayrıştırıcılar bu bölümde ayrıca ele alınmaktadır. MPWFL dilinin gerçekleştirilmesinde YACC [9] destek programı kullanılmıştır. YACC destek programına yönelik açıklamalar da bu bölümde verilmektedir.

Derleyici bir kaynak programı giriş olarak alır ve çıkış olarak giriş koduna eşdeğer bir makine komut dizisi üretir. Derleyici bu işlemi, Şekil 4.1'de gösterildiği üzere, birçok evrede gerçekleştirir. Bu evreler sırasıyla, tarayıcı (lexical analyzer), ayrıştırıcı (parser), arakod üretimi (intermediate code generator), kod üretimi (code generation) ve kod optimizasyonudur (code optimization). Evrelerin işlevleri bu bölümün paragraflarında kısaca anlatılmaktadır.



Şekil 4.1 Derleme evreleri

4.1 TARAYICI

Tarayıcı, kaynak programı karakter karakter okur ve böylece en alt düzeydeki birimler olan sözcükler (token) belirlenir. Bir token, tek başına bilgi taşıyan en küçük karakter katarıdır. Token'ları iki gruba ayırabiliriz:

- i) Önceden tanımlı olan sözcükler (dile ilişkin anahtar sözcükler veya operatörler).
- ii) Programcı tarafından tanımlanabilen sözcükler (değişkenler, sabitler, vb.).

Bir token, token değeri ve token tipi olmak üzere iki bilgi alanından oluşur. Derleyicinin kaynak program üzerinde işlem yaptığı tek evresi tarayıcıdır. Bu evre, derleyicinin en çok zaman yitirdiği aşamasıdır. Ayırıştırıcı, yeni bir token'a gerek duyduğunda, bir alt program olan tarayıcıyı çağırır. Tarayıcı, sıradaki token'a ilişkin bilgiyi ayırıştırıcıya aktarır. Bunun

için tarayıcı sırasıyla,

- i)Kaynak programı karakter karakter okur.
- ii)Anlamsız boşluk karakterlerini ve boş satırları atlar.
- iii)Dile ilişkin token'ları saptar. Bunun için gerektiğinde tablolara başvuru (lookahead) işlemini uygular.
- iv)Sembol tablosuna eklemeler yapar.
- v)Hataları sezer ve haber verir (kaynak programın karakter kümesinde olmayan karakterleri sezdiğinde).

Tarayıcı işlevlerini yürüten destek programları mevcuttur. UNIX işletim sisteminin destek programı olan LEX bu programlara örnek olarak verilebilir [10].

4.2 AYRIŞTIRICI

Derleyicinin ikinci evresini oluşturan ayrıştırıcının iki temel görevi vardır:

- i) Token'ların kaynak dilin kurallarına uygun sırada bulunup bulunmadıklarını saptamak.
- ii) Ayrıştırıcıdan sonra görev yapacak olan evrelerin kod üretebilmesi için token'ları bir ağaç yapısına yerleştirmek.

Dilin kurallarına (gramer) uygun olarak üretilen ayrıştırma ağacı ile işlem önceliği belirlenir.

Ayrıştırma işlemi,

- 1) Yukarıdan aşağıya (top-down)
- 2) Aşağıdan yukarıya (bottom-up)

olmak üzere iki türlü yapılabilir.

Yukarıdan aşağıya ayrıştırma işleminde , gramerin başlangıç simgesi olan S'den başlayarak, kökten yapraklara doğru bir ayrıştırma ağacı oluşturulur. Yukarıdan aşağı ayrıştırma yöntemine örnek olarak LL ayrıştırıcılar verilebilir [6].

Aşağıdan yukarı ayrıştırmada ise, yapraklardan (yani tarayıcıdan alınan token'lerden) başlanıp yukarı (köke) doğru ayrıştırma ağacı oluşturulur ve gramerin başlangıç simgesi olan S'ye ulaşılmaya çalışılır. Aşağıdan yukarı ayrıştırma yöntemine örnek olarak LR ayrıştırıcılar verilebilir [6].

İki ayrıştırma işleminde de giriş üzerinde soldan sağa doğru ilerlenir ve her seferinde birer token işlem görür. Aşağıdan yukarıya ayrıştırıcılar, ötele-indirge (shift-reduce) ayrıştırıcılar adını da alır. Çünkü aşağıdan yukarıya ayrıştırmada yığının tepesinde bir üretimin (production) sağ tarafı görülene kadar giriş simgeleri yığına atılır. Bunun ardından üretimin sağ yanı, üretimin sol yanındaki sembol ile değiştirilir (indirgenir).

4.3 ARAKOD ÜRETİLMESİ

Bu aşamada, ayrıştırıcının ürettiği ağaç yapısından yararlanılarak kaynak dilin arakod gösterilimi elde edilir. Birçok arakod gösterim şekli vardır. Arakod gösteriminde, sadece koşulsuz dallanma veya tek koşullu dallanma ve aritmetik işlem komutları kullanılabilir. "WHILE-DO" veya "IF-THEN-ELSE" gibi yüksek düzeyli akış kontrol komutları sadece arakod gösteriminde kullanılabilen komutlar kullanılarak gerçekleştirilmektedir.

4.4 KOD ÜRETİMİ

Arakod, dördüncü evre olan kod üretimi aşamasında, bir dizi makine dili komutuna dönüştürülür. Bu evrede verilerin saklanması için gerekli bellek adreslerinin seçimi, verilere erişmek için gerekli komutların seçimi gibi dikkat edilmesi gereken işlemler yapılır. Arakodu doğrudan makine koduna dönüştürmek, gereksiz yere pekçok yükle ve sakla komutlarını üretecektir. Bunu önlemenin yolu, kod üretimi evresinde, özellikle saklayıcıların kullanımının izlenmesi ve sadece gerektiğinde yükleme ve saklama işlemlerinin yapılmasının sağlanmasıdır.

4.5 KOD OPTİMİZASYONU

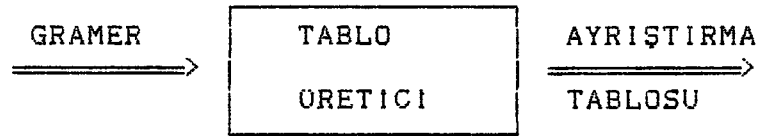
Kod optimizasyonu, bir derleyici için seçime bağlı bir evredir. Amacı, arakodu gözden geçirip iyileştirmektir. Üretilecek kodun daha kısa sürede yürütülebilmesi ve/veya bellekte daha az yer kaplamasını sağlayabilmek için, yerel optimizasyon, döngü optimizasyonu, indis hesapları optimizasyonu ve ortak alt ifadelerin kaldırılması gibi birçok optimizasyon yöntemini kullanır.

4.6 LR AYRIŞTIRICILAR

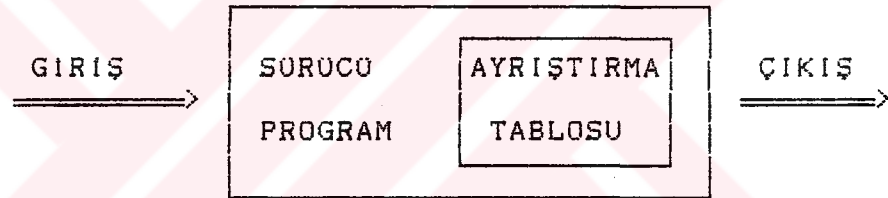
LR ayrıştırıcı, aşağıdan yukarıya ayrıştırma yönteminin genel ve etkin bir şeklidir. LR denmesinin nedeni, giriş katarını soldan sağa taramaları ancak, en sağdan başlayarak türetimleri gerçekleştirmeleridir. Yani giriş katarını en soldan tarar ve çıkışı en sağdan türetirler. Bu yöntemin olumlu özellikleri arasında şunlar sayılabilir:

- 1) Gerilemeye (backtracking) gerek duymaz.
- 2) LR ayrıştırıcı sözdizim yazım (syntax) hatalarını, girişin soldan sağa taranması sırasında hemen farkederek.

Bu yöntemin tek sorunu, bir programlama dili için elle bir LR ayrıştırıcı geliştirmenin oldukça kûlfetli bir iş olmasıdır. Bu nedenle ayrıştırma tablosunun oluşturulmasında bir tablo üretici program (LR Parser Generator) kullanılır (Şekil 4.2).



a) Tablo üretimi

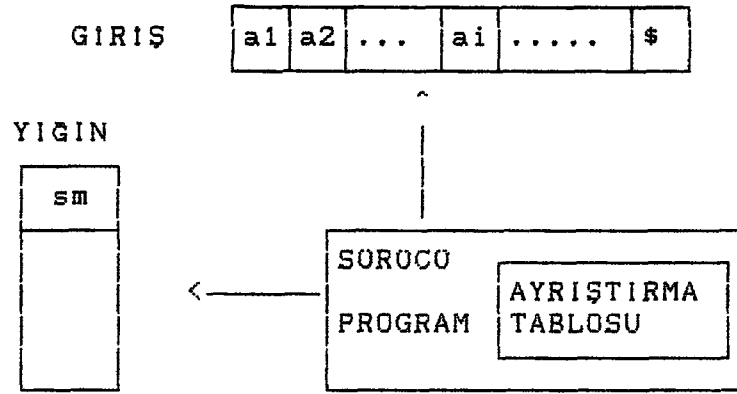


b) Ayrıştırma işlemleri

Şekil 4.2 LR ayrıştırıcının gerçekleştirilmesi

4.7 LR AYRIŞTIRICI PROGRAMI

Ayrıştırıcı, bir giriş katarı, bir yığın ve bir ayrıştırma tablosundan yararlanır. Giriş katarı soldan sağa ve her adımda bir karakter olmak üzere okunur (Şekil 4.3).



Şekil 4.3 LR ayrıştırma

Yığın, $SO \ X_1 \ S_1 \ X_2 \ S_2 \ \dots \ X_m \ S_m$, (S_m yığının tepesinde yer almak üzere) şeklinde bir katarı içerir. Burada her X_i bir giriş simgesine, her S_i ise bir durum bilgisine karşı düşer. Her durum bilgisi, yığında kendinden önce gelen diğer durum bilgilerini içerir ve ötele-indirge kararlarını vermede rehberlik yapar.

Ayrıştırma tablosu, bir ayrıştırma işlemi fonksiyonu olan ACTION ile bir durum numarası değiştirme fonksiyonu olan GOTO işlemlerinden oluşturulur.

LR ayrıştırıcı, ilk bileşeni yığının içeriği ve ikinci bileşeni ise giriş simgeleri olan bir yapıdadır:

$(SO \ X_1 \ S_1 \ X_2 \ S_2 \ \dots \ X_m \ S_m ; a_i \ a_{i+1} \ \dots \ a_n \ \$)$

Burada S_m yığının tepesindeki durumu, a_i ise güncel giriş karakterini simgeler. İş (S_m, a_i) , a_i girişi ve yığının tepesindeki S_m durumu için aşağıdaki dört değerden birini alabilir:

1) Ötele s

Burada işlemden sonra geçilecek yeni durum s ile simgelenmektedir. Öteleme işlemi gerçekleştirilir, yeni

konum

(S0 X1 S1 X2 S2 ... Xm Sm ai s ; ai+1 ... an \$) olur. Ayrıştırıcı ai'yi ve bir sonraki durum olan s'yi ($s=Y_d(sm, ai)$) yığına atar. Son olarak ai+1 yeni güncel giriş simgesi olur.

2) İndirge A $\rightarrow \beta$

Ayrıştırıcı bir indirgeme işlemini gerçekler ve yeni konum

(S0 X1 S1 X2 S2 ... Xm-r Sm-r A s ; ai ai+1 ... an \$) olur.

Burada,

$s = Y_d(Sm-r, A)$

$r = \beta$ 'nın boyu, üretimin sağ yanındaki elemanların sayısı

olarak verilmiştir. Ayrıştırıcı önce yığından Sm-r durumuna kadar 2r adet simge çeker (r durum simgesi ve r gramer simgesi olmak üzere) . Ayrıştırıcı bundan sonra, türetimin sol yanında yer alan nonterminali, yani A'yı ve $I_\beta(Sm-r, A)$ için tablodan bulunan S durumunu yığına atar. İndirgeme sonucu güncel giriş simgesi değişikliğe uğramaz.

3) Kabul

Ayrıştırmanın başarılı olduğunu gösterir ve ayrıştırma işlemi sonuçlanmıştır.

4) Hata

$I_\beta(Sm, ai)$ tanımsızdır, ayrıştırıcı hata verir.

Başlangıçta (S0 a1 a2 ... an \$) konumunda olan LR ayrıştırıcı, bir hata ya da kabul durumu ile karşılaşınca kadar ayrıştırmayı sürdürür. Buradaki S0 başlangıç durumunu, a1 a2 ... an ise ayrıştırılacak olan

simge katarını simgelemektedir. Bir hata durumu ile karşılaşıldığında, ayrıştırıcı bir hata mesajı verir ve derleme işlemi sonlanır.

4.8 LR AYRIŞTIRICI UYGULAMASI

Bu paragrafta bir LR grameri tanımlanarak buna ilişkin tablosu verilecektir [7]. Örnek bir giriş dizisinin LR yöntemine göre ayrıştırma işlem adımları izlenecektir.

Örnek gramer:

- 0. s --> e
- 1. e --> e + t
- 2. ! t
- 3. t --> NUM

Bu gramerde tamsayı sabitini simgeleyen "NUM" ve tamsayı işlem operatörünü simgeleyen "+" token'ları kullanılmıştır. Gramerde oluşturulan s, e, t nonterminalleri yardımıyla, tamsayıların ve işlem operatörlerinin birden fazla kullanılması ile geçerli bir dizi yaratılması sağlanmaktadır. Örnek gramere ilişkin üretilen ayrıştırma tablosu Şekil 4.4'de verilmiştir. Giriş dizisi bir karakter (token) ile sonlanmalıdır. Bu token tabloda "#" simgesi ile gösterilmiştir. Tabloda "-" simgesi hatalı işlem (kurallarla tanımlanmamış işlemler) olduğunu belirtmektedir.

	ACTION			GOTO	
	#	NUM	+	e	t
0	-	s1	-	2	3
1	r3	-	r3	-	-
2	kabul	-	s4	-	-
3	r2	-	r2	-	-
4	-	s1	-	-	5
5	r1	-	r1	-	-

Şekil 4.4 Örnek ayrıştırma tablosu

Giriş dizisi olarak 1+2+3 ele alındığında yürütülen ayrıştırma işlemleri Şekil 4.5'te verilmiştir. Giriş dizisinin sonlandığını # karakteri simgelemektedir. Başlama durumu olarak yığına 0 durumu atılır ve tarayıcıdan token'ların alınması ile işlemler adım adım gramer kurallarına göre (tabloya başvurularak) yürütülür. Burada yapılan işlemlerin açıklanması olarak ilk bir kaç işlem adımı verilmektedir. Örnek dizi için diğer işlemler Şekil 4.5'te gösterildiği gibi benzer şekilde yürütülmektedir. İlk NUM token'ın (1 tamsayısı) alınması, 0. durumda işlem olarak tablodan 1 numaralı durumun ötelenmesi (s1) gerektiği çıkar. Giriş dizisinin görüntüsünün +2+3# olması ve yığının tepesindeki durumun 1 olması, 3 numaralı üretim sonucunu veren bir indirgeme işleminin (r3) yapılmasına neden olur. Öncelikle 1 numaralı durum yığından alınır, 3 numaralı durum gramerde $t \rightarrow \text{NUM}$ kuralını tanımladığından t için tablonun GOTO alanına bakılır.

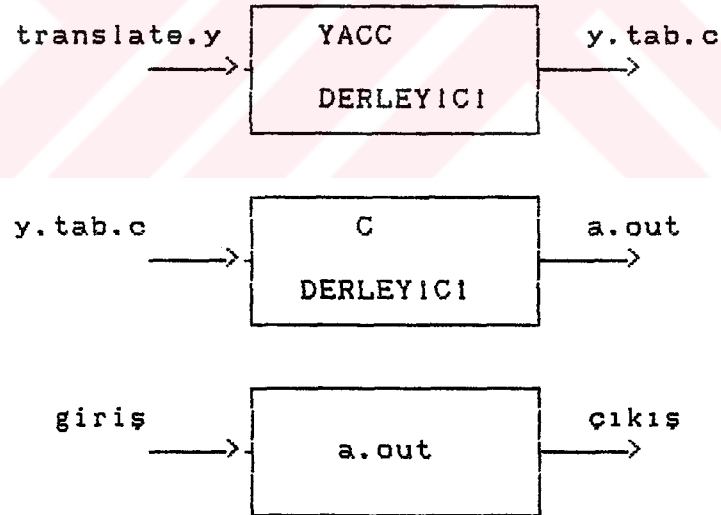
YIGIN	SIMGE	GİRİŞ	AÇIKLAMA
0	\$	1+2+3#	Başlangıç durumu yığına
01	\$NUM	+2+3#	Ötele NUM
0	\$	+2+3#	İndirge $t \rightarrow \text{NUM}$
03	\$t	+2+3#	
0	\$	+2+3#	İndirge $e \rightarrow t$
02	\$e	+2+3#	
024	\$e+	2+3#	Ötele +
0241	\$e+NUM	+3#	Ötele NUM
024	\$e+	+3#	İndirge $t \rightarrow \text{NUM}$
0245	\$e+t	+3#	
024	\$e+	+3#	İndirge $e \rightarrow e + t$
02	\$e	+3#	
0	\$	+3#	
02	\$e	+3#	
024	\$e+	3#	Ötele +
0241	\$e+NUM	#	Ötele NUM
024	\$e+	#	İndirge $t \rightarrow \text{NUM}$
0245	\$e+t	#	
024	\$e+	#	İndirge $e \rightarrow e + t$
02	\$e	#	
0	\$	#	Kabul

Şekil 4.5 1+2+3 giriş dizisi için ayrıştırma işlemleri

Yığının tepesinde bu anda 0 durumu olması ile, 3 numaralı durumun yığına atılmasını gerektirdiği tabloya bakılarak öğrenilmektedir. Kabul işlem adımının gerçekleşmesi (ayrıştırmanın sonlanması), derleyicinin kod üretme aşaması yöntemine bağlı olduğundan farklılıklar gösterebilir.

4.9 YACC DESTEK PROGRAMI

YACC, LALR(1) tipinde ayrıştırıcı üreten bir UNIX işletim sistemi destek programıdır [9]. LALR, LR türünden bir ayrıştırıcıdır. Temel farkı ayrıştırıcı tablosu üretilmesi üzerindedir [6]. Bir giriş/çıkış dönüştürücüsü, YACC kullanılarak Şekil 4.6'da görüldüğü gibi oluşturulabilir.



Şekil 4.6 YACC ile dönüştürücü oluşturma

İlk olarak dönüştürücünün YACC gösteriliminde kurallarını içeren bir dosya, örneğin "translate.y" dosyası hazırlanır. Ardından bir UNIX işletim sistemi komutu olan aşağıdaki ifade verilerek:

```
yacc translate.y
```

translate.y dosyası, LALR yöntemini kullanan y.tab.c adında bir C programına [11] dönüştürülür. Burada üretilen y.tab.c programı aslında, C dilinde yazılmış LALR ayrıştırıcı ve kullanıcı tarafından hazırlanmış C altprogramlarını içermektedir. y.tab.c programı, LR ayrıştırıcı programını içeren ly kütüphanesi de eklenerek aşağıdaki komut verilerek derlenir.

```
cc y.tab.c -ly
```

Bu komut yürütüldüğünde istenen hedef program a.out elde edilir. a.out programı istenen dönüştürme işlemini (translate) gerçekler. Gerekli başka altprogramlar varsa, y.tab.c ile bunlar derlenip yüklenebilir.

Bir YACC kaynak programı üç bölümden oluşmaktadır. Bu bölümler aşağıda verilmiştir. Bu bölümün açıklamaları izleyen paragraflarda yer almaktadır.

tanımlamalar (declarations)

%%

dönüştürme kuralları (translation rules)

%%

kullanıcı tarafından hazırlanmış C altprogramları

4.9.1 TANIMLAMA BÖLÜMÜ

Bir YACC programının tanımlamalar kısmında iki bölüm vardır. İlk bölümde, bildiğimiz C tanımlamaları "%" işaretleri arasında verilmektedir. Bu bölümde,

- i)YACC kaynak programlarını dönüştürme kuralları;
- ii)C altprogramları bölümünde yer alan dönüştürme kuralları;
- iii)C altprogramlarının kullanacağı tampon değişkenlerin

tanımlamaları verilir.

Tanımlama kısmında bulunan ikinci bölümde ise, gramerdeki token'ların tanımlamaları yer almaktadır. Bu bölümde tanımlamaları yapılan token'lar, YACC programının diğer dönüştürme kuralları ve C altprogramları bölümlerinde kullanılabilirler.

4.9.2 DÖNÜŞTÜRME KURALLARI BÖLÜMÜ

Bir YACC kaynak programındaki ilk "%" işaret çiftini izleyen program bölümü, dönüştürme kurallarını içerir. Her kural bir gramer türetim kuralını ve buna ilişkin işlemleri (semantic actions) belirtir. Bu bölümde uyulan kurallar maddeler halinde özetlenecek olursa:

- i) Sağ tarafında birden fazla seçenek olan türetim kurallarında, sağ taraflar birbirlerinden "!" karakteri ile ayrılırlar.
- ii) Her türetim kuralının sonunda ";" işareti kullanılır.
- iii) Yazılan ilk türetim kuralına ilişkin sol taraf, gramerin başlangıç simgesidir.
- iv) Türetim kuralında tek tırnak işaretleri arasında verilen tek karakterler birer token olarak değerlendirilir. Token olarak tanımlanmamış ve tırnak işaretleri arasında verilmeyen karakter katarları ise birer nonterminal olarak alınır.
- v) Bir YACC işlemi (semantic action), bir dizi C komutundan oluşmaktadır.

Bu komutlarla verilen, "\$\$" karakter çifti, türetim kuralının sol tarafındaki nonterminale ilişkin ; "\$i" gösterilimi ise, türetim kuralının sağ tarafındaki i'nci gramer simgesine (terminal ya da nonterminal) ilişkin sözde değişkendir.

4.9.3 ALTPROGRAMLAR BÖLÜMÜ

yylex() adındaki tarayıcı program ve seçimlik olarak da hata kotarma altprogramları bu bölümde yer alır. yylex() tarayıcı programı, bir token ve o token'a ilişkin "attribute" değerinden oluşan veri çiftlerini üretir.

4.9.4 ÖRNEK YACC PROGRAMI

Burada, basit bir hesap makinesi işlemlerini yapabilen bir uygulama verilecektir [6]. Bu örneğin verilmesindeki amaç bir YACC programının görüntüsünü sunmaktır. Uygulamanın tasarımına ve gerçekleşmesine ilişkin tüm açıklamalar konu dışında olduğundan verilmemiştir. Amaç program, aritmetik ifadeleri almakta, yürütmekte ve sayısal sonucu ekrana çıkarmaktadır. Amaç programın üretilme aşamaları olarak; gramerin belirlenmesi, gramerin YACC dilinde uygulanması ve yukarıda açıklanan derleme işlemlerinin yürütülmesi adımları verilebilir. Aşağıda gramer ve YACC programı listelenmiştir. Gramerde, digit token'ı 0 ile 9 arasındaki tek basamaklı tamsayıları simgelemektedir.

Gramer:

```
E -> E + T | T
T -> T * F | F
F -> (E) | digit
```

YACC program:

```
%{
#include <ctype.h>
%}
%token DIGIT

%%
line    : expr '/n'  {printf("%d\n", $1);}
        ;
expr    : expr '+' term {$$ = $1 + $3; }
        | term
        ;
term    : term '*' factor {$$ = $1 * $3;}
        | factor
        ;
factor  : '(' expr ')'  {$$ = $2;}
        | DIGIT
        ;
%%
yylex() {
    int c;
    c = getchar();
    if (isdigit (c)){
        yylval = c - '0';
        return DIGIT;
    }
    return c;
}
```

BÖLÖM 5. MPWFL DİLİ GRAMERİ

Gramer, terminaller (token) ve nonterminaller olarak iki farklı türden simgeler kullanarak LALR gramer kurallarına uygun biçimde oluşturulmuştur. Gramer yazımında büyük harfler ile gösterilen terminaller, Tarayıcı tarafından program satırları taranarak her token (anahtar sözcükler ve değişkenler) için ayrı bir kod halinde üretilirler. Küçük harflerle gösterilen nonterminaller ise Ayrıştırıcının kurallar kümesini oluştururlar. Gramerin bir kural satırı ";" karakteri ile sonlanır.

MPWFL dili grameri işlevsel bölümlerinin açıklamaları, izleyen bölümlerde yer almaktadır. Bölüm 5.3 ise gramerin topluca bir dökümünü içermektedir.

5.1 PROGRAM GÖVDESİ

Burada programın iki temel yapısı yer alır. Bunlar Tanımlama alanı ve altprogram alanıdır. Bir program, en az bir değişken tanımlama ve en az bir altprogram gövdesi ile program metninin sonlandığını gösteren EOF (dosya sonu) token'ından oluşur. Tanımlama alanında tip tanımlayıcısı ile buna ilişkin değişken adı veya adları bulunmaktadır. Birden fazla değişken tanımlama yapmak mümkündür. Altprogram tanımlama, altprogram adı ve bu altprograma ilişkin program satırlarından oluşmaktadır.

```

program : program_body EOF;

program_body : declaration_list sub_program_list;

declaration_list : declaration
                  !declaration_list declaration;

declaration : DECLARATION_SPECIFIER identifier_list
              STOP;

identifier_list : ID
                !identifier_list COMMA ID;

sub_program_list : sub_program_body
                  !subprogram_list subprogram_body;

subprogram_body : subprogram_specifier SUBBEGIN
                  statement_list SUBEND;

subprogram_specifier : SUBBODY SUBPROGRAM_ID;

```

5.2 KOMUTLAR

Komut listesi olarak incelenen satırlarda en az bir geçerli komut yer almalıdır. Komut satırı STOP ile sonlanır. Komutlar işlevlerine göre gruplanarak aşağıdaki bölümlerde açıklanmıştır.

```

statement_list : statement_line
                !statement_list statement_line;

statement_line : statement STOP;

statement : assignment_statement
            !groupname_statement
            !groupmake_statement
            !taskcopy_statement
            !taskinit_statement
            !owner_statement
            !display_statement
            !pline_statement
            !pcontrol_statement
            !SIMPLE_STATEMENT_NAME
            !if_statement
            !free_statement
            !while_statement
            !repeat_statement
            !select_statement
            !SUBPROGRAM_ID;

```

5.2.1 ÖZEL İŞLEV KOMUTLARI

Bu grupta parametreleri sabit ve özel işlev gerçekleyen komutlar (altprogram çağırma görüntüsünde) yer alır. Aynı türden parametrelere sahip özel işlev komutları bir grup oluştururlar. Tarayıcı bu grubun üyeleri için işlev_NAME token'ını üretir. Kod üretme aşamasında komutun gerçek ismi kullanılmaktadır.

```
groupname_statement : GROUPNAME L_PARAN GROUP_ID COMMA
                    string_sub_expression R_PARAN;
```

```
groupmake_statement : GROUPMAKE L_PARAN GROUP_ID COMMA
                    sub_identifier_list COMMA
                    integer_sub_primary R_PARAN;
```

```
sub_identifier_list : INTEGER_ID
                    !STRING_ID
                    !BOOLEAN_ID
                    !BUFFER_ID;
```

```
taskcopy_statement : TASKCOPY L_PARAN TASK_ID COMMA
                    TASK_ID R_PARAN;
```

```
taskinit_statement : TASKINIT L_PARAN TASK_ID R_PARAN;
```

```
owner_statement : OWNER_STATEMENT_NAME L_PARAN OWNER_ID
                 COMMA integer_sub_primary R_PARAN;
```

```
display_statement : DISPLAY L_PARAN
                  string_sub_expression R_PARAN;
```

```
pline_statement : PLINE L_PARAN string_sub_expression
                 R_PARAN;
```

```
pcontrol_statement : PCONTROL L_PARAN STRING_CONSTANT
                   R_PARAN;
```

```
free_statement : FREE L_PARAN buffer_sub_expression
               R_PARAN;
```

5.2.2 AKIŞ KONTROL KOMUTLARI

Bu grupta program akışını denetleyen komutlar yer alır. Gramerde, belirsizlik (ambiguous) durumunu kaldıracak önlemler alınmıştır.

```

if_statement : IF L_PARAN boolean_expression R_PARAN
              THEN statement_list ENDIF
              !IF L_PARAN boolean_expression R_PARAN
              THEN statement_list ELSE statement_list
              ENDIF;

while_statement : WHILE L_PARAN boolean_expression
                  R_PARAN BEGIN statement_list
                  ENDWHILE;

repeat_statement : REPEAT statement_list UNTIL L_PARAN
                  boolean_sub_expression R_PARAN;

select_statement : SELECT when_statement_list ENDSELECT
                  !SELECT when_statement_list OTHERWISE
                  statement_list ENDSELECT;

when_statement_list:when_statement
                  !when_statement_list when_statement;

when_statement : WHEN L_PARAN boolean_expression R_PARAN
                DO statement_list ENDWHEN;

```

5.2.3 ATAMA KOMUTLARI

Değişken tiplerine göre farklı özelliklere sahip atamalar vardır. Bu grupta parametreleri sabit ve özel işlev gerçekleyen fonksiyonlar yer alır. Aynı türden parametrelere sahip özel işlev fonksiyonları için tek işlev_NAME token'ı kullanılır. Kod üretme aşamasında fonksiyonun gerçek ismi kullanılmaktadır.

```

assignment_statement : boolean_assignment_statement
                      !integer_assignment_statement
                      !string_assignment_statement
                      !task_assignment_statement
                      !buffer_assignment_statement;

```

5.2.3.1 Mantıksal atama komutları

Burada mantıksal değişkenlere yapılabilen atama biçimleri sıralanmaktadır.

```

boolean_assignment_statement : BOOLEAN_ID EQUAL
                               boolean_expression;

boolean_expression : boolean_expression boolean_operator
                   boolean_term
                   !boolean_term;

boolean_term : NOT boolean_primary
              !boolean_primary;

boolean_operator : AND
                 !OR
                 !IMP;

boolean_primary : boolean_sub_expression
                 !arithmetic_comparasion
                 !string_comparasion
                 !load_function
                 !save_function
                 !task_function
                 !group_function
                 !L_PARAN boolean_expression R_PARAN;

boolean_sub_expression : BOOLEAN_ID
                       !BOOLEAN_CONSTANT;

arithmetic_comparasion : integer_sub_primary RELOP
                        integer_sub_primary;

string_comparasion : STRING_ID STRING_RELOP
                    string_sub_expression;

load_function : LOAD L_PARAN string_sub_expression COMMA
                 buffer_sub_expression R_PARAN;

save_function : SAVE L_PARAN string_sub_expression COMMA
                buffer_sub_expression COMMA
                integer_sub_primary R_PARAN;

task_function : TASK_FUNCTION_NAME L_PARAN OWNER_ID
               COMMA TASK_ID R_PARAN;

group_function : GROUP_FUNCTION_NAME L_PARAN OWNER_ID
                COMMA GROUP_ID R_PARAN;

```

5.2.3.2 TAMSAYI ATAMA KOMUTLARI

Burada tamsayı değişkenlerine yapılabilen atama biçimleri sıralanmaktadır.

```

integer_assignment_statement : INTEGER_ID EQUAL
                              integer_expression;

integer_expression : integer_expression integer_operator
                    term
                    !term;

term : MINUS integer_primary
      !integer_primary;

integer_operator : PLUS
                  !MINUS
                  !STAR
                  !DIV
                  !MOD;

integer_primary : integer_sub_primary
                 !integer_function
                 !task_field_attribute_primary
                 !L_PARAN integer_expression R_PARAN;

integer_sub_primary : INTEGER_CONSTANT
                    !INTEGER_ID;

integer_function : INTEGER_FUNCTION_NAME L_PARAN
                  string_sub_expression R_PARAN;

task_field_attribute_primary : TASK_ID L_BRACKET
                              TASK_FIELD_ATTRIBUTE
                              R_BRACKET;

```

5.2.3.3 KATAR ATAMA KOMUTLARI

Burada katar değişkenlerine yapılabilen atama biçimleri sıralanmaktadır.

```

string_assignment_statement : STRING_ID EQUAL
                             string_expression;

string_expression : string_sub_expression
                   !string_function
                   !ascii_function
                   !take_drop_function
                   !task_name_attribute_primary;

string_sub_expression : STRING_CONSTANT
                      !STRING_ID;

string_function : STRING_FUNCTION_NAME L_PARAN
                 string_sub_expression R_PARAN;

```

```

ascii_function : ASCII L_PARAN integer_sub_primary
                R_PARAN;

task_name_attribute_primary : TASK_ID L_BRACKET
                             TASK_NAME_ATTRIBUTE
                             R_BRACKET;

take_drop_function : TAKE_DROP_FUNCTION_NAME L_PARAN
                    string_sub_expression COMMA
                    integer_sub_primary R_PARAN;

```

5.2.3.4 VERİ TAMPONU ATAMA KOMUTLARI

Burada veri tamponu değişkenlerine yapılabilen atama biçimleri sıralanmaktadır.

```

buffer_assignment_statement : BUFFER_ID EQUAL
                             buffer_expression;

buffer_expression : buffer_sub_expression
                  !alloc_function
                  !pack_unpack_function
                  !reverse_function;

buffer_sub_expression : BUFFER_ID
                      !task_buffer_attribute_primary;

task_buffer_attribute_primary : TASK_ID L_BRACKET
                               TASK_BUFFER_ATTRIBUTE
                               R_BRACKET;

alloc_function : ALLOC L_PARAN integer_sub_primary
                R_PARAN;

pack_unpack_function : PACK_UNPACK_FUNCTION_NAME L_PARAN
                     buffer_sub_expression R_PARAN;

reverse_function : REVERSE L_PARAN buffer_sub_expression
                  COMMA integer_sub_primary COMMA
                  integer_sub_primary R_PARAN;

```

5.2.3.5 GÖREV ATAMA KOMUTLARI

Burada görev değişkenlerine yapılabilen atama biçimleri sıralanmaktadır.

```

task_assignment_statement : TASK_ID L_BRACKET
                           task_assignment_attribute
                           R_BRACKET;

task_assignment_attribute : TASK_NAME_ATTRIBUTE EQUAL
                           string_sub_expression
                           !TASK_FIELD_ATTRIBUTE EQUAL
                           integer_sub_primary
                           !TASK_BUFFER_ATTRIBUTE EQUAL
                           buffer_sub_expression;

```

5.3 GRAMER

MPWFL dili gramerinin topluca dökümü aşağıda verilmektedir.

```

program : program_body EOF;

program_body : declaration_list sub_program_list;

declaration_list : declaration
                 !declaration_list declaration;

declaration : DECLARATION_SPECIFIER identifier_list STOP;

identifier_list : ID
                !identifier_list COMMA ID;

sub_program_list : sub_program_body
                 !subprogram_list subprogram_body;

subprogram_body : subprogram_specifier SUBBEGIN statement_list SUBEND;

subprogram_specifier : SUBBODY SUBPROGRAM_ID;

statement_list : statement_line
               !statement_list statement_line;

statement_line : statement STOP;

statement : assignment_statement
           !groupname_statement
           !groupmake_statement
           !taskcopy_statement
           !taskinit_statement
           !owner_statement
           !display_statement
           !pline_statement
           !pcontrol_statement
           !SIMPLE_STATEMENT_NAME
           !if_statement

```

```

!free_statement
!while_statement
!repeat_statement
!select_statement
!SUBPROGRAM_ID;

groupname_statement : GROUPNAME L_PARAN GROUP_ID COMMA string_sub_expression R_PARAN;

groupmake_statement : GROUPMAKE L_PARAN GROUP_ID COMMA sub_identifier_list COMMA
                    integer_sub_primary R_PARAN;

sub_identifier_list : INTEGER_ID
                    !STRING_ID
                    !BOOLEAN_ID
                    !BUFFER_ID;

taskcopy_statement : TASKCOPY L_PARAN TASK_ID COMMA TASK_ID R_PARAN;

taskinit_statement : TASKINIT L_PARAN TASK_ID R_PARAN;

owner_statement : OWNER_STATEMENT_NAME L_PARAN OWNER_ID COMMA integer_sub_primary R_PARAN;

display_statement : DISPLAY L_PARAN string_sub_expression R_PARAN;

pline_statement : PLINE L_PARAN string_sub_expression R_PARAN;

pcontrol_statement : PCONTROL L_PARAN STRING_CONSTANT R_PARAN;

if_statement : IF L_PARAN boolean_expression R_PARAN THEN statement_list ENDIF
              !IF L_PARAN boolean_expression R_PARAN THEN statement_list ELSE statement_list
              ENDIF;

free_statement : FREE L_PARAN buffer_sub_expression R_PARAN;

while_statement : WHILE L_PARAN boolean_expression R_PARAN BEGIN statement_list ENDWHILE;

repeat_statement : REPEAT statement_list UNTIL L_PARAN boolean_sub_expression R_PARAN;

select_statement : SELECT when_statement_list ENDSELECT
                 !SELECT when_statement_list OTHERWISE statement_list ENDSELECT;

when_statement_list : when_statement
                    !when_statement_list when_statement;

when_statement : WHEN L_PARAN boolean_expression R_PARAN DO statement_list ENDWHEN;

assignment_statement : boolean_assignment_statement
                    !integer_assignment_statement
                    !string_assignment_statement
                    !task_assignment_statement
                    !buffer_assignment_statement;

boolean_assignment_statement : BOOLEAN_ID EQUAL boolean_expression;

integer_assignment_statement : INTEGER_ID EQUAL integer_expression;

```

```

string_assignment_statement : STRING_ID EQUAL string_expression;

buffer_assignment_statement : BUFFER_ID EQUAL buffer_expression;

task_assignment_statement : TASK_ID L_BRACKET task_assignment_attribute R_BRACKET;

task_assignment_attribute : TASK_NAME_ATTRIBUTE EQUAL string_sub_expression
                           !TASK_FIELD_ATTRIBUTE EQUAL integer_sub_primary
                           !TASK_BUFFER_ATTRIBUTE EQUAL buffer_sub_expression;

boolean_expression : boolean_expression boolean_operator boolean_term
                   !boolean_term;

boolean_term : NOT boolean_primary
              !boolean_primary;

boolean_operator :      AND
                    !OR
                    !IMP;

boolean_primary : boolean_sub_expression
                 !arithmetic_comparasion
                 !string_comparasion
                 !load_function
                 !save_function
                 !task_function
                 !group_function
                 !L_PARAN boolean_expression R_PARAN;

boolean_sub_expression : BOOLEAN_ID
                       !BOOLEAN_CONSTANT;

arithmetic_comparasion : integer_sub_primary RELOP integer_sub_primary;

string_comparasion : STRING_ID STRING_RELOP string_sub_expression;

load_function : LOAD L_PARAN string_sub_expression COMMA buffer_sub_expression R_PARAN;

save_function : SAVE L_PARAN string_sub_expression COMMA buffer_sub_expression COMMA
                integer_sub_primary R_PARAN;

task_function : TASK_FUNCTION_NAME L_PARAN OWNER_ID COMMA TASK_ID R_PARAN;

group_function : GROUP_FUNCTION_NAME L_PARAN OWNER_ID COMMA GROUP_ID R_PARAN;

integer_expression : integer_expression integer_operator term
                   !term;

term : MINUS integer_primary
      !integer_primary;

integer_operator : PLUS
                 !MINUS
                 !STAR
                 !DIV
                 !MOD;

```

```

integer_primary : integer_sub_primary
                !integer_function
                !task_field_attribute_primary
                !L_PARAN integer_expression R_PARAN;

integer_sub_primary : INTEGER_CONSTANT
                    !INTEGER_ID;

integer_function : INTEGER_FUNCTION_NAME L_PARAN string_sub_expression R_PARAN;

task_field_attribute_primary : TASK_ID L_BRACKET TASK_FIELD_ATTRIBUTE R_BRACKET;

string_expression : string_sub_expression
                  !string_function
                  !ascii_function
                  !take_drop_function
                  !task_name_attribute_primary;

string_sub_expression : STRING_CONSTANT
                      !STRING_ID;

string_function : STRING_FUNCTION_NAME L_PARAN string_sub_expression R_PARAN;

ascii_function : ASCII L_PARAN integer_sub_primary R_PARAN;

task_name_attribute_primary : TASK_ID L_BRACKET TASK_NAME_ATTRIBUTE R_BRACKET;

take_drop_function : TAKE_DROP_FUNCTION_NAME L_PARAN string_sub_expression COMMA
                    integer_sub_primary R_PARAN;

buffer_expression : buffer_sub_expression
                  !alloc_function
                  !pack_unpack_function
                  !reverse_function;

buffer_sub_expression : BUFFER_ID
                      !task_buffer_attribute_primary;

task_buffer_attribute_primary : TASK_ID L_BRACKET TASK_BUFFER_ATTRIBUTE R_BRACKET;

alloc_function : ALLOC L_PARAN integer_sub_primary R_PARAN;

pack_unpack_function : PACK_UNPACK_FUNCTION_NAME L_PARAN buffer_sub_expression R_PARAN;

reverse_function : REVERSE L_PARAN buffer_sub_expression COMMA integer_sub_primary COMMA
                  integer_sub_primary R_PARAN;

```

BÖLÜM 6. ÇEVİRİCİNİN GERÇEKLENMESİ

Bölüm 5.3'de verilen gramerin metni token'ların da eklenmesi ile UNIX işletim sisteminin destek programı olarak verilen YACC programına uygun biçime dönüştürülür. YACC programı, adı giriş parametresi olarak verilen dosyayı kendi kurallarına göre inceleyerek, LALR yöntemine göre çözülmüş, C dilinde bir kod üretir. Gerçekte YACC'ı kullanma nedenimiz üretilen bu kod (dosya) değildir. Çalışma şeklimizin getirdiği eklemelerin ve düzenlemelerin YACC'ın ürettiği C dili kodu üzerinde işlenmesi güçlük doğurmaktadır. YACC, gramerin bir diğer sonucu olarak Ayrıştırma tablosunu (dosyasını) da üretir. Üretilen ayrıştırma tablosu temel alınarak bu bölümde sıralanan aşamalarla Çevirici programı hazırlanmıştır.

6.1 AYRIŞTIRMA TABLOSU

YACC'ın ürettiği ayrıştırma tablosunun başlangıcından bir kısmı aşağıda listelenmiştir.

```
state 0
    $accept : _program $end
    DECLARATIONSPECIFIER shift 5
    . error

    program goto 1
    programbody goto 2
    declarationlist goto 3
    declaration goto 4
```

```

state 1
    $accept : program_$end
    $end accept
    .error

state 2
    program : programbody_EOF
    EOF shift 6
    .error

state 3
    programbody : declarationlist_subprogramlist
    declarationlist : declarationlist_declaration

    SUBBODY shift 11
    DECLARATIONSPECIFIER shift 5
    .error

    subprogramlist goto 7
    declaration goto 8
    subprogrambody goto 9
    subprograms specifier goto 10

state 4
    declarationlist : declaration_
    .reduce 3

```

Bu tabloda aşağıda listelenen türde bilgiler yer alır:

1-Bir duruma verilen numara (state 0 gibi).

2-Numarası verilen durumun, gramerin hangi satırının hangi aşamasına ait olduğu (program : programbody_EOF gibi). Burada "_" karakteri o satırın hangi terminal veya nonterminaline ulaşıldığını göstermektedir.

3-Geçerli token geldiğinde hangi durum numarasına "shift" yapılacağı (EOF shift 6 gibi).

4-Geçerli token gelmediğinde hata olduğu belirtilmesi (.error gibi).

5-Nonterminallerin hangi durum numarasına "reduce" işlemi yapılacağı (reduce 3 gibi).

6-Reduce işleminden sonra güncel token'a hangi durumun numarasının verileceği (program goto 1 gibi).

6.2 GERÇEKLEME AŞAMALARI

Uygulamada bir tarayıcı hazırlanarak, giriş metninden yeni token'ları üretme, değişkenlerin yönetimi gibi gramerde tanımlanmayan işlevler bu altprograma bırakılmıştır. Ayrıştırma tablosuna ilişkin temel program adımları şu aşamalardan oluşur:

a-Yeni bir token alındığında güncel duruma ve o token'a göre hangi durum numarasına "shift" veya "reduce" edileceği belirlenir. Programda bu işlev için (C dilinde) bir "switch" ve "case" yapısı, Ayrıştırma tablosu doğrudan kullanılarak kurulmaktadır. "Shift" veya "reduce" işleminden önce genellikle (bazı özel durumlar hariç) güncel token'a ilişkin C dili kod üretilmektedir. Bu çalışma şekli genel derleyici tasarımı yöntemlerine uygun düşmemektedir. Makina dili hedef alınarak kod üretme yerine, bir başka yüksek seviyeli dile çevirme işlemi yapıldığından en basit ve en hızlı sonuç üreten yöntem seçilmiştir.

b-Reduce işlemleri için, ayrıştırma tablosuna bakılarak, durum numarasına karşı düşen bilgi alanından kaç adet token'ın yığından çekileceği tesbit edilebilmektedir.

c-Reduce işleminden sonra incelenen nonterminalin hangi durum numarasında bulunuyorsa ona ilişkin "goto" numarası elde edilmesi gerekmektedir. Bu işlem için programda yine "switch" ve "case" yapısı kullanılmıştır. Ancak burada "case" için veriler (a) şıkkında olduğu gibi bir arada değildir ve bu bilgiler tüm ayrıştırma tablosu metnine yayılmıştır. Program geliştirme aşamasında bu güçlüğü ortadan kaldırmak için bir TABLO.C (EK D.) destek programı hazırlanmıştır. Destek programı

ayrıştırma tablosu metni üzerinde nonterminalleri işleyerek doğrudan "case" ifadelerini üretmektedir.

Çeviri programının "switch-case" ifadelerini basitleştiren tablolar gibi yöntemleri de kimi yerde kullanarak, hata mesajlarını içeren, değişik türden bilgi veren dosyaları da üreten daha etkin bir kod gerçeklemeye çalışılmıştır. Program hakkında detaylı bilgi izleyen bölümlerde verilmiştir.

6.3 ÇEVİRİCİ PROGRAMININ GENEL YAPISI

Çevirici programın (EK A.) genel yapısı işlem adımları biçiminde ve altprogramlar ve işlevleri şeklinde aşağıda açıklanmaktadır. Programda kullanılan bazı önemli Veri yapıları ve Tablolar ise Bölüm 6.3.3'te verilmiştir. MPWFL çevirici programının kullanım şekli ve argümanları aşağıda verilmiştir:

MPWFL giriş_dosyası çıkış_dosyası [include_dosya] [durum_dosyası]

Giriş dosyası, MPWFL dilinde hazırlanan kaynak metin dosyasıdır. Çıkış dosyası, C dilinde üretilen metin dosyasıdır ve bir sonraki aşama olarak bu dosya C derleyicisinde derlenmektedir. Destek programlarının yer aldığı UTIL.H dosyası da (EK B.) derleme işlemlerine katılır. Diğer iki dosya adı ise isteğe göre argüman olarak verilebilir. Çevirici diğer bir C dili programına yönelik olarak, çevirilen programda kullanılan tüm değişkenlerin ve altprogramların tanımlarını Include dosyasına yazar. Durum dosyasında ise çevirme işleminin tüm aşamaları (tarama ve ayrıştırma) saklanmaktadır. Durum dosyası incelenerek çevirme yöntemi hakkında bilgi edinmek mümkündür.

6.3.1 ANA PROGRAM İŞLEM ADIMLARI

1-ERRMSG.TXT (EK C.) hata dosyası eğer bulunamazsa, hata ekrana çıkarılır ve programdan çıkılır. Aksi halde, hata dosyası 2 boyutlu bir diziye (matrise) okunur.

2-MPWFL dilinde yazılmış programın bulunduğu giriş dosyası ile, C dilindeki kodun üretileceği çıkış dosyası açılır.

3-Eğer istenmiş ise (program argümanı olarak adları verilmiş ise) , INCLUDE ve DURUM TEST dosyaları açılır. Include dosyası, başka bir programdan, çevrilen programın altprogramlarının çağrılması istenildiğinde yararlı olmaktadır. Burada, çevrilen programda kullanılan tüm değişkenlerin tipleri ile birlikte "extern" olarak ve altprogramların tanımları yeralır. Durum test dosyasında ise çevirme işleminde işleme giren token'lar ve tüm "shift", "reduce", "push", "pop", "goto" türü işlemler yer alır.

4-Giriş dosyası okunur ve aşağıdaki işlemler yapılarak, belleğe yazılır (filtreleme ve ön işlemler). Bu işlemlerden sonra çevirme işlemi için karakterler bu bellek alanından (Giriş dizisi) alınır. Bu işlemler aşağıda sıralanmıştır:

- i)Tüm alfabetik karakterler büyük harfe çevrilir.
- ii){} arasında yer alan açıklama (comment) alanları atılır.
- iii)Satır başı (Carriage Return, 0x0d) yerine EOLN sabiti yerleştirilir. Bunun haricindeki tüm kontrol karakterleri (0x00'dan 0x1f'e kadar) yerine boşluk karakteri yerleştirilir.
- iv)Giriş dosyasının karakter sayısı MAX INPUT sabitinden (20000 sekizli) daha büyük ise, programdan çıkılır.
- v)Giriş dosyasının sonuna gelindiğinde, son karakter

olarak 0x00 bilgisi (EOF) yazılır ve dosya sonu işaretlenmiş olur.

5-INITFILE altprogramı çağırılır.

6-PARSE altprogramı çağırılır.

7-PARSE altprogramından dönüşte hata var ise hata mesajı, hata yoksa işlem bitti mesajı ekrana çıkarılır. Giriş ve çıkış dosyaları kapatılarak programdan çıkılır.

6.3.2 ALTPROGRAMLAR

6.3.2.1 INITFILE

Çıkış dosyası ve eğer varsa INCLUDE dosyasına gerekli açılış satırlarını yazar.

6.3.2.2 ISFUNC

Güncel token ile parametre olarak verilen dizi karşılaştırılır. Karşılaştırma sonucunda iki dizi birbirinin aynı ise, altprogram TRUE değeri ile geri döner.

6.3.2.3 ISDELIM

Parametre olarak verilen karakterin sonlandırma elemanı (delimiter) olup olmadığını sınar. Eğer karakter bir delimiter ise, altprogram TRUE değeri ile geri döner.

6.3.2.4 IS_IN

Verilen karakterin, verilen dizide bulunup bulunmadığını sınar. Eğer karakter dizide bulunuyorsa, TRUE değerini alarak geri döner.

6.3.2.5 NEXTTOKEN

Giriş dizisinden yeni bir token getiren ayrıştırıcıdır. Yeni token, kurallara uygun bir token ise, güncel token kodu olarak geri döner. Aksi halde geçersiz token kodu olarak geri döner.

Bu altprogramda sırasıyla:

- 1-Token'ların arasındaki boşluklar atılır.
- 2-Dosya sonu (End of File, 0x00) sinaması yapılır. Eğer dosya sonu ise, dosya sonu token'ı olarak geri döner.
- 3-Satır sonu (EOLN) sinaması yapılır. Eğer satır sonu ise satır sayacı artırılır ve aşağıdaki işlemlerle altprograma devam edilir.
- 4-Yeni token'ın ilk karakterinin bir rakam olup olmadığı sınanır. Eğer ilk karakter bir rakam ise, sayı oluşturulana kadar token üzerinde (giriş dizisinde) ilerlenir ve sayının son rakamına da ulaşılinca, elde edilen sayı, tamsayı sabit olarak geri döner.
- 5-Yeni token'ın ilk karakterinin tek tırnak olup olmadığı sınanır. Eğer tek tırnaksa, token üzerinde (giriş dizisinde) ikinci bir tek tırnak işaretine raslanıncaya kadar ilerlenir ve iki tek tırnak arasındaki karakter katarı bir katar sabiti olarak geri döner.
- 6-Yeni token'ın ilk karakteri alfabetikse, bir anahtar sözcük olup olmadığı sınanır. Eğer bir anahtar sözcükse, anahtar sözcük olarak geri döner.
- 7-Yeni token bir anahtar sözcük değilse, değişkenler tablosunda aranır. Değişkenler tablosunda bulunursa,

değişkenin tip koduyla geri döner. Değişkenler tablosunda bulunamazsa, yeni bir değişken olarak tabloya eklenir (MAKE-ID altprogramı ile).

8-Yeni token'ın ilk karakteri alfanumerik değilse, bir sonlandırma elemanı (delimiter) olup olmadığına bakılır. Eğer sonlandırma elemanı da değilse, altprogram hata kodu ile geri döner. Aksi halde sonlandırma elemanı kodu ile geri döner.

6.3.2.6 PRINT_TOKEN

Güncel token'a ilişkin bilgileri durum bilgisi dosyasına yazar.

6.3.2.7 PUSH

Token'ı yığına atar.

6.3.2.8 POP

Yığından token'ı çeker.

6.3.2.9 GOTOSTATE

Parametre olarak verilen giriş koduna (production) ilişkin GOTO kodunu verir. Parametre ile verilen giriş koduna (production) göre, "product table"dan bir nonterminal alınır. Yığının tepesindeki durum ve bu nonterminal yardımıyla, dönüş parametresi değeri belirlenir.

6.3.2.10 SHIFT

Parametre olarak verilen durum için öteleme (shift) işlemi yapılır. Parametre olarak verilen durum bilgisi, güncel token'ın durum bilgisi alanına atanır. Güncel token, PUSH altprogramı ile yığına atılır ve NEXTTOKEN altprogramı çağrılarak yeni bir güncel token belirlenir.

6.3.2.11 REDUCE

Parametre olarak verilen giriş kodu (reduction) için, "reduce-num-table" dan bulunan sayıda POP altprogramı çağrılır. Aynı parametre kullanılarak GOTOSTATE altprogramı çağrılır, bu altprogramın dönüş parametresi ise, güncel token'ın durum bilgisi alanına atanır. Bundan sonra güncel token, PUSH altprogramı ile yığına atılır.

6.3.2.12 MAKE_ID

Parametre olarak verilen ve bir değişken olan güncel token için, çıkış ve include dosyalarına, bu değişkene ilişkin tanımlama bilgilerini yazar.

6.3.2.13 PARSE

Dosya sonu (EOF) ya da hata oluşuncaya kadar ayrıştırma (parsing) işlemini gerçekleştirir. Bunun için:

1-Yığın ilk koşullarına getirilir ve başlangıç durumu yığına atılır.

2-Yeni token, NEXTTOKEN altprogramı ile alınır.

3-

- a)Yığının tepesindeki durum, indirgeme (reduce) işlemini gerektiriyorsa, indirgeme işlemi yapılır.
- b)Aksi halde, token'a ilişkin C dili kodu üretilir.
- c)Yığının tepesindeki duruma göre shift, reduce işlemleri yapılır.

4-Ayrıştırma işlemi tamamlanmamışsa ve hata oluşmadıysa 3 numaralı işleme geri dönülür. Aksi halde altprogramdan geri dönülür.

6.6.3. VERİ YAPILARI VE TABLOLAR

6.6.3.1 Veri Yapıları

Tarayıcının ve Ayrıştırıcının kullandığı token'lar program içinde aşağıdaki gibi tanımlanmıştır:

```
#define COMMA 5
#define STOP 6
#define PLUS 7
```

Böylece program içinde token'ların sayısal olarak işlenmesi sağlanmaktadır. Aslında YACC destek yazılımı token'lar için benzer biçimde token'ları tanımlamaktadır. Ancak hazırlanan programa uygun gelecek bir sıralama yaptık.

Token için programda aşağıda gösterilen bir yapı kullanılmaktadır:

```
struct TOKENREC
{
    int type;
    int value;
    int state;
    char name[80];
}
```

Bu yapıda; "type" alanında token'ın kodu, "value" alanında token bir tamsayı değeri anlamını taşıyorsa bu değer, "state" alanında bu token için üretilen durum değeri, "name" alanında ise token'ın özgün katarı saklanır. TOKENREC yapısında elamanlara sahip yığın için "stack" tek boyutlu dizisi kullanılmaktadır.

Değişkenler için aşağıda gösterilen bir yapı kullanılmaktadır.

```
struct IDREC
{
    int type;
    char name[80];
}
```

Bu yapıda, "type" alanında değişkenin türü, "name" alanında ise değişkenin özgün katarı saklanır. IDREC yapısında elemanları olan değişkenler tablosu için "id_list" tek boyutlu dizisi kullanılmaktadır.

Hata mesajları ana programın başında bir dosyadan okunur ve mesaj katarları "errmsg" iki boyutlu dizisinde saklanır.

6.6.3.2 Tablolar

Programda ayrı işlevlere sahip dört adet tablo (tek boyutlu dizi) yer alır. Bu tablolar aşağıda açıklanmıştır:

Hata mesajları için kullanılan "errno_table" tablosunun elemanları, hata mesajı katarlarını saklayan "errmsg" iki boyutlu dizisinin indisleridir. Hata mesajı oluşturabilen bir ayrıştırıcı durumu, "errno_table" içinde indis olarak kullanılır ve karşı düşen değer okunur. Bu değer ayrıştırıcı durumuna ait hata mesajının indisidir.

Ayrıştırma işleminin başında, o anda yapılacak olan işlemin bir "reduce" işlemi olup olmadığı sınılanır. Sınama işlemi için "reduce_table" tablosu kullanılmaktadır. Bu tablo ayrıştırıcı durumunun değeri ile indislenir. Karşı düşen değerin "0" olması "reduce" işleminin yapılmayacağını gösterir. 0'dan farklı değer ise bir "reduce" işleminin yapılacağını gösterir. Tablodan elde edilen sayının değeri aynı zamanda kaç numaralı duruma "reduce" yapıldığını bildirir. Bir durum için bu tabloda "reduce" işlemi yok anlamına gelen 0 değeri olsa da ayrıştırıcının bundan sonraki işlem adımlarında, özellikle C kodu üretildikten sonra, tekrar "reduce" işlemi yapılabilir.

Ayrıştırıcı, "reduce_num_table" tablosuna erişerek bir duruma ait "reduce" işleminde kaç defa "pop" işlemi yapılacağını öğrenmektedir.

Ayrıştırıcının "gotostate" altprogramında, verilen giriş parametresinin hangi numaralı "nonterm" e karşı düştüğü "product_table" tablosuna erişilerek bulunmaktadır. Altprogramda, elde edilen bu "nonterm" numarasına ve yığının tepesindeki duruma göre yeni "goto" durum numarası saptanmaktadır.

BÖLÖM 7. UYGULAMA PROGRAMLARI

Bu bölümde; MPWFL dilinde hazırlanmış iki örnek programın metni ve bunların C diline çevrilen metinleri verilmektedir. Örnek programlarda tanımlanan işlemlerin yapılmasını sınamak amacıyla bir deney düzeni oluşturulmuştur.

Deney düzeninde iki adet kişisel bilgisayar yer almaktadır. Kişisel bilgisayarlar seri haberleşme hattı ile bağlantılıdır. Kişisel bilgisayarlardan biri ana bilgisayar olarak kullanılmakta, diğeri ise çok işlemcili sisteminin benzetimi işlevini yürütmektedir. Benzetim için kişisel bilgisayara bir program (EK E.) hazırlanmıştır. Benzetim programı ile iki bilgisayar arasındaki mesaj paketleri izlenilmekte ve ana bilgisayar için mesaj üretilmektedir.

Birinci örnek program, derlendiğinde tek başına bir yürütebilir kod üretilmesine yönelik uygulamayı göstermektedir.

İkinci örnek program ise, ayrı bir C dili uygulama programı ile birlikte kullanım şeklini gösteren çok dilde programlama örneğidir. Burada, iki dilde yazılan uygulama programları ayrı ayrı derlenerek "object" kodları üretilmekte ve bağlayıcı (Linker) tarafından tek bir yürütülebilir kod haline getirilmektedir. Bu uygulamada, birinci örnek programın alt programları korunarak ve sadece "main" bölümü çıkarılarak ikinci MPWFL dili örnek programı oluşturulmuştur. Bu altprogramları kullanan C dili ana program örneği hazırlanmıştır.

7.1 BİRİNCİ UYGULAMA PROGRAMI

Dört adet işlemci biriminden oluşan sisteme yönelik bir uygulama programı hazırlanmıştır. Burada kullanıcının belirlediği bir göreve ait program kodu dosyadan okunarak çok işlemcili sisteme gönderilmektedir. Programda her bir işlemci için ayrı ayrı saklanan işlenecek veri dosyaları alınarak ilişkin işlemci birimlerine gönderilmektedir. Çok işlemcili sistemden adı verilen görevin (dört işlemci biriminde de aynı simgeli görev) yürütülmesi istenmektedir. Son olarak çok işlemcili sistemin ürettiği sonuçlar, her bir işlemci biriminden ayrı ayrı alınarak dosya halinde saklanmaktadır. İşlenecek olan verilerin tamsayı karakterlerinden (0-9) oluşmakta ve dosyada ASCII kodlarıyla saklanmaktadır. Bu veriler çok işlemcili sisteme sıkıştırılarak (PACK işlemi) aktarılmaktadır. Benzer şekilde çok işlemcili sistemin ürettiği sonuçlar tekrar ASCII koduna dönüştürülerek (UNPACK işlemi) dosyada saklanmaktadır.

Birinci uygulama programı, üç altprogram ve bir ana programdan oluşmaktadır. Bu altprogramlar sırasıyla şu işlemleri yürütürler:

1-Sorgu alt programında; kullanıcıdan görev kodu dosya adı, kaynak veri dosyalarının taban adı, görev adı ve grup adı alınmaktadır. Görev kodu dosyası bir Veri tampounu değişkeninde aktarıldıktan sonra altprogram sonlanmaktadır.

2- Dosya oku alt programında, kaynak dosya adı işlemci numarası ile bitleştirilerek bir işlemciye ilişkin veri dosya adı belirlenmektedir. Kaynak dosya adı ile dosyaya erişilir ve dosya içerikleri bir Veri Tamponu değişkenine aktarılır. Grup değişkeni üzerinden veri dosyaları ayrı ayrı işlemcilere gönderilir. Tüm işlemler hatasız gerçekleştiğinde çok işlemcili sistemden

görevlerin yürütülmesi istenir.

3- Dosya yaz alt programında; grup değişkeni üzerinden, her bir işlemciden varış dosyası adı, sonuç verilerinin boyutu ve sonuç verileri alınır. İşlem hatasız gerçekleşti ise varış dosyasında alınan veriler saklanır.

7.1.1 BİRİNCİ UYGULAMA PROGRAMININ MPWFL DİLİ DÖKÜMÜ

Birinci uygulama programının MPWFL dili dökümü aşağıda verilmiştir:

{ ÖRNEK 1 PROGRAMI }

```

BOOLEAN bool1,bool2;
INTEGER uzunluk,say;
STRING ek,uz,Kaynak,Varis,yaniti,yanit2,dosyal;
STRING GorevAdi,GrupAdi;
BUFFER buf1,buf2,buf3;
TASK TaskA;
OWNER OwnerA,OwnerB;
GROUP GroupA;

```

```

SUBROUTINE DosyaOku,DosyaYaz,Sorgu,main;

```

```

SUBBODY Sorgu

```

```

SUBBEGIN

```

```

dosyal=ACCEPT('Gorev dosyasi adini girin :');

```

```

uzunluk=ISFILE(dosyal);

```

```

uz=ASCII(uzunluk);

```

```

yaniti=dosyal;

```

```

yaniti=CONCAT(' dosyasi boyutu=');

```

```

yaniti=CONCAT(uz);

```

```

DISPLAY(yaniti);

```

```

uzunluk=uzunluk+10;

```

```

buf1=ALLOC(uzunluk);

```

```

{veri tamponu için bellek alındı}

```

```

yanit2=ACCEPT('dosyayi oku? (E/H)');
bool1= yanit2 SEQU 'E';
IF (bool1) THEN bool1=LOAD(dosyal,buf1); ENDIF;
IF (bool1) THEN
    DISPLAY('Gorev dosyasi okundu');
    kaynak= ACCEPT('Kaynak veri dosyasi adini girin:');
    GorevAdi= ACCEPT('Gorev adini girin:');
    GrupAdi= ACCEPT('Grup adini girin:');
ELSE
    DISPLAY('Gorev dosyasi okunmadi !!');
ENDIF;
SUBEND { sorgu altprogrami }

```

SUBBODY DosyaOku

SUBBEGIN

```

ek='.GIR';
TASKINIT(TaskA);
TaskA[POINTER=buf1];           {görev değişkeninde program kodu kapsandı}
TaskA[NAME=GorevAdi];         {görev adı verildi}
RESETOWNER(OwnerA,4);         { 0.1.2.3. no lu işlemciler iş sahibi yapıldı}
bool1=SENDTASK(OwnerA,TaskA);  {görev dört işlemciye gönderildi}
FREE(buf1);                   {buf1 in işlevi sonlandı}
buf2=ALLOC(2000);             {diğer veri tamponları için bellek istendi}
buf3=ALLOC(1000);
GROUPNAME(GroupA,GrupAdi);    {dört üyeli bir grup için ad verildi}
GROUPMAKE(GroupA,buf2,2000);   {1. üye, çıkış, veri tamponu}
GROUPMAKE(GroupA,Varis,0);     {2. üye, giriş, katar}
GROUPMAKE(GroupA,uzunluk,0);   {3. üye, giriş, tamsayı}
GROUPMAKE(GroupA,buf3,0);     {4. üye, giriş, veri tamponu}
say=0;
bool2=bool1;
WHILE (bool2)
    BEGIN
        dosya1=kaynak;         {işlenecek verilerin dosyadan okunması}
        uz=ASCII(say);
        dosya1=CONCAT(uz);
        dosya1=CONCAT(ek);     {dosya adı: örnek0.gir, örnek1.gir ..}
        uzunluk=ISFILE(dosya1) {dosya var mı ?}
    END

```

```

bool1= uzunluk NEQ 0;
IF (bool1) THEN bool1=LOAD(dosya1,buf2); ENDIF;
IF (bool1) THEN
    dosya1=CONCAT(' dosyasi okundu');
    DISPLAY(dosya1);
    buf2=PACK(buf2);                      {veri boyutunu indirge }
    RESETOWNER(OwnerB,0);
    ADDOWNER(OwnerB,say);                  {sadece say no lu işlemciye gönder}
    bool1=SENDGROUP(OwnerB,GroupA);
ENDIF;
say=say+1;
bool2=bool1 AND (say NEQ 4);
ENDWHILE;
IF(bool1) THEN
    bool1= RUNTASK(OwnerA,TaskA);          {dört işlemcide dört görevi başlat}
    DISPLAY('Dosya okuma ve Gorev Baslatma tamamlandi');
ENDIF;
SUBEND { Dosya oku }

SUBBODY DosyaYaz
SUBBEGIN
say=0;
bool2=TRUE;
WHILE(bool2)
BEGIN
    RESETOWNER(OwnerB,0);                  {say no lu işlemciden işlenen verileri alma}
    ADDOWNER(OwnerB,say);
    bool1=GETGROUP(OwnerB,GroupA);          {işlenen veriler buf3 te }
    IF(bool1) THEN                          {dosya adı Varis ta}
        buf3=UNPACK(buf3);                  {veriler açılır}
        yanit1='C: ';
        yanit1=CONCAT(Varis);
        bool1=SAVE(yanit1,buf3,uzunluk);     {dosyada saklama}
        yanit1=CONCAT(' dosyasi saklandi');
        DISPLAY(yanit1);
    ENDIF;
    say=say+1;
    bool2=bool1 AND (say NEQ 4);

```

```
ENDWHILE;
```

```
IF(bool1) THEN
```

```
    bool1= KILLTASK(OwnerA.TaskA);           (dört işlemcide görevi)
```

```
    ENDIF;                                   (sonlandır)
```

```
SUBEND (Dosya Yaz )
```

```
SUBBODY main
```

```
(ana program)
```

```
SUBBEGIN
```

```
Sorgu;
```

```
IF(bool1) THEN
```

```
    DosyaOku;
```

```
    IF(bool1) THEN
```

```
        DosyaYaz;
```

```
    ENDIF;
```

```
ENDIF;
```

```
SUBEND { main }
```

7.1.2 BİRİNCİ UYGULAMA PROGRAMININ C DİLİ DÜKÖMÜ

Birinci uygulama programının C diline çevrilmesiyle elde edilen metin aşağıda listelenmiştir. Bu metin TurboC [12] derleyicisinde derlenerek yürütülebilir program kodu (EXE) elde edilmektedir.

```
#include <util.h>
```

```
unsigned int BOOL1=FALSE;
```

```
unsigned int BOOL2=FALSE;
```

```
int UZUNLUK=0;
```

```
int SAY=0;
```

```
char EK[80]="";
```

```
char UZ[80]="";
```

```
char KAYNAK[80]="";
```

```
char VARIS[80]="";
```

```
char YANIT1[80]="";
```

```
char YANIT2[80]="";
```

```
char DOSYA1[80]="";
```

```

char GOREVADI[80]="";
char GRUPADI[80]="";
struct BUFFER BUF1={NULL,0,0};
struct BUFFER BUF2={NULL,0,0};
struct BUFFER BUF3={NULL,0,0};
struct TASK TASKA={"",0,NULL,0,0};
struct OWNER OWNERA={0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
struct OWNER OWNERB={0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
struct GROUP GROUPA={NULL,0,0,NULL};
void DOSYADOKU(void);
void DOSYAYAZ(void);
void SORGU(void);
void SORGU (void)
{

strcpy(DOSYA1,ACCEPT("GOREV DOSYASI ADINI GIRIN :") );

UZUNLUK=ISFILE(DOSYA1) ;

strcpy(UZ,ASCII(UZUNLUK) );

strcpy(YANIT1,DOSYA1);

strcpy(YANIT1,CONCAT(YANIT1," DOSYASI BOYUTU=") );

strcpy(YANIT1,CONCAT(YANIT1,UZ) );

printf("\n%s",YANIT1) ;

UZUNLUK=UZUNLUK + 10;

BUFF_COPY(&BUF1,ALLOC(UZUNLUK));

strcpy(YANIT2,ACCEPT("DOSYAYI OKU? (E/H)") );

BOOL1=!strcmp(YANIT2,"E");

```

```

if(BOOL1) {

    BOOL1=LOAD(DOSYA1,&BUF1) ;

} ;

if(BOOL1) {

    printf("\n%s","GOREV DOSYASI OKUNDU") ;

    strcpy(KAYNAK,ACCEPT("KAYNAK VERİ DOSYASI ADINI GIRIN:") );

    strcpy(GOREVADI,ACCEPT("GOREV ADINI GIRIN:") );

    strcpy(GRUPADI,ACCEPT("GRUP ADINI GIRIN:") );

} else {
    printf("\n%s","GOREV DOSYASI OKUNMADI !!") ;
} ;

}

void DOSYAOKU (void)
{

    strcpy(EK,".GIR");

    TASKINIT(&TASKA) ;
    BUFF_COPY(&TASKA.POINTER,&BUF1);
    strcpy(TASKA.NAME,GOREVADI);

    RESETOWNER(&OWNER,4) ;

    BOOL1=SENDTASK(&OWNER,&TASKA) ;

    FREE(&BUF1) ;

    BUFF_COPY(&BUF2,ALLOC(2000));

```

```

BUFF_COPY(&BUF3,ALLOC(1000));

GROUPNAME(&GROUPA,GRUPAD1) ;

GROUPMAKE(&GROUPA,&BUF2,4,2000) ;

GROUPMAKE(&GROUPA,&VARIS,3,0) ;

GROUPMAKE(&GROUPA,&UZUNLUK,2,0) ;

GROUPMAKE(&GROUPA,&BUF3,4,0) ;

SAY=0;

BOOL2=BOOL1;

while(BOOL2) {

strcpy(DOSTA1,KAYNAK);

strcpy(UZ,ASCII(SAY) );

strcpy(DOSTA1,CONCAT(DOSTA1,UZ) );

strcpy(DOSTA1,CONCAT(DOSTA1,EK) );

UZUNLUK=ISFILE(DOSTA1) ;

BOOL1=UZUNLUK != 0;

if(BOOL1) {

BOOL1=LOAD(DOSTA1,&BUF2) ;

} ;

if(BOOL1) {

```

```

strcpy(DOSTA1,CONCAT(DOSTA1," DOSYASI OKUNDU" ) );

printf("\n%s",DOSTA1) ;

BUFF_COPY(&BUF2,PACK(&BUF2));

RESETOWNER(&OWNERB,0) ;

ADDDOWNER(&OWNERB,SAY) ;

BOOL1=SENDGROUP(&OWNERB,&GROUPA) ;

} ;

SAY=SAY + 1;

BOOL2=BOOL1 && (SAY != 4) ;

} ;

if(BOOL1) {

BOOL1=RUNTASK(&OWNERB,&TASKA) ;

printf("\n%s","DOSYA OKUMA VE GOREV BASLATMA TAMAMLANDI" ) ;

} ;

}

void DOSYATAZ (void)
{

SAY=0;

BOOL2=TRUE;

while(BOOL2) {

```

```

RESETOWNER(&OWNERB,0) ;

ADDDOWNER(&OWNERB,SAY) ;

BOOL1=GETGROUP(&OWNERB,&GROUPA) ;

if(BOOL1) {

    BUFF_COPY(&BUF3,UNPACK(&BUF3));

    strcpy(YANIT1,"C:");

    strcpy(YANIT1,CONCAT(YANIT1,VARIS) );

    BOOL1=SAVE(YANIT1,&BUF3,UZUNLUK) ;

    strcpy(YANIT1,CONCAT(YANIT1," DOSYASI SAKLANDI" ) );

    printf("\n%s",YANIT1) ;

} ;

SAY=SAY + 1;

BOOL2=BOOL1 && (SAY != 4) ;

} ;

if(BOOL1) {

    BOOL1=KILLTASK(&OWNERB,&TASKA) ;

} ;

}

main ()
{

```

```
SORGU();
```

```
if(BOOL1) {
```

```
DOSYAKU();
```

```
if(BOOL1) {
```

```
DOSYAYAZ();
```

```
};
```

```
};
```

```
}
```

7.2 İKİNCİ UYGULAMA PROGRAMI

İkinci uygulamanın biri MPWFL dilinde diğeri ise C dilinde iki ayrı bölümü vardır. MPWFL dili bölümü çevrilerek C dili metin elde edilmektedir. Her iki C dili metin ayrı ayrı TurboC derleyicisinde derlenerek amaç program kodu (OBJ) elde edilmektedir. TurboC derleyicisinin Bağlayıcı (Linker) programı ile iki amaç program kodu birleştirilerek yürütülebilir program kodu (EXE) üretilmektedir. İkinci uygulama programı, birinci uygulama programının "main" bölümünün çıkarılmasıyla elde edilmektedir. Çeviriciden "include" (ornek2.h) dosyasının da üretilmesi istenir. Bu dosya C dili programında kullanarak, altprogramları içeren programda kullanılan tüm değişkenlerin ve altprogramın bu programda kullanılması sağlanır. Ana programda birinci programa göre ek işlev olarak işlenecek veri dosyalarının kullanıcı tarafından oluşturması sağlanmaktadır. Böylece MPWFL dili programda yer alan değişkenlerin kullanım şekli gösterilmektedir.

Aşağıdaki paragraflarda tüm metin dosyaları listelenmiştir.

7.2.1 İKİNCİ UYGULAMA PROGRAMININ MPWFL DİLİ DÖKÜMÜ

İkinci uygulama programının MPWFL dili dökümü aşağıda verilmiştir:

(ÖRNEK 2 PROGRAMI)

```

BOOLEAN bool1,bool2;
INTEGER uzunluk,say;
STRING ek,uz,Kaynak,Varis,yaniti,yanit2,dosyal;
STRING GorevAdi,GrupAdi;
BUFFER buf1,buf2,buf3;
TASK TaskA;
OWNER OwnerA,OwnerB;
GROUP GroupA;

SUBROUTINE DosyaOku,DosyaYaz,Sorgu;

SUBBODY Sorgu
SUBBEGIN
dosyal=ACCEPT('Gorev dosyasi adini girin :');
uzunluk=LSFILE(dosyal);
uz=ASCII(uzunluk);
yaniti=dosyal;
yaniti=CONCAT(' dosyasi boyutu=');
yaniti=CONCAT(uz);
DISPLAY(yaniti);
uzunluk=uzunluk+10;
buf1=ALLOC(uzunluk);           {veri tamponu için bellek alındı}
yanit2=ACCEPT('dosyayi oku? (E/H)');
bool1= yanit2 SEQU 'E';
IF (bool1) THEN bool1=LOAD(dosyal,buf1); ENDIF;
IF (bool1) THEN
    DISPLAY('Gorev dosyasi okundu');

```

```

        kaynak= ACCEPT('Kaynak veri dosyasi adini girin:');
        GorevAdi= ACCEPT('Gorev adini girin:');
        GrupAdi= ACCEPT('Grup adini girin:');
    ELSE
        DISPLAY('Gorev dosyasi okunmadi !!');
    ENDIF;
SUBEND ( sorgu altprogrami )

SUBBODY DosyaOku
SUBBEGIN
ek='.GIR';
TASKINIT(TaskA);
TaskA[POINTER=buf1];           {görev değişkeninde program kodu kapsandı}
TaskA[NAME=GorevAdi];          {görev adı verildi}
RESETOWNER(OwnerA,4);          { 0.1.2.3. no lu işlemciler iş sahibi yapıldı}
bool1=SENDTASK(OwnerA,TaskA);   {görev dört işlemciye gönderildi}
FREE(buf1);                     {buf1 in işlevi sonlandı}
buf2=ALLOC(2000);               {diğer veri tamponları için bellek istendi}
buf3=ALLOC(1000);
GROUPNAME(GroupA,GrupAdi);      {dört üyeli bir grup için ad verildi}
GROUPMAKE(GroupA,buf2,2000);     {1. üye, çıkış, veri tamponu}
GROUPMAKE(GroupA,Varis,0);        {2. üye, giriş, katar}
GROUPMAKE(GroupA,uzunluk,0);      {3. üye, giriş, tamsayı}
GROUPMAKE(GroupA,buf3,0);         {4. üye, giriş, veri tamponu}
say=0;
bool2=bool1;
WHILE (bool2)
    BEGIN
        dosya1=kaynak;           {işlenecek verilerin dosyadan okunması}
        uz=ASCII(say);
        dosya1=CONCAT(uz);
        dosya1=CONCAT(ek);        {dosya adı: ornek0.gir, ornek1.gir ...}
        uzunluk=ISFILE(dosya1);   {dosya var mı ?}
        bool1= uzunluk NEQ 0;
        IF (bool1) THEN bool1=LOAD(dosya1,buf2); ENDIF;
        IF (bool1) THEN
            dosya1=CONCAT(' dosyasi okundu');
            DISPLAY(dosya1);

```

```

    buf2=PACK(buf2);                {veri boyutunu indirge }
    RESETOWNER(OwnerB,0);
    ADDOWNER(OwnerB,say);            {sadece say no lu işlemciye gönder}
    bool1=SENDGROUP(OwnerB,GroupA);
    ENDIF;
    say=say+1;
    bool2=bool1 AND (say NEQ 4);
ENDWHILE;
IF(bool1) THEN
    bool1= RUNTASK(OwnerA,TaskA);    {dört işlemcide dört görevi başlat}
    DISPLAY('Dosya okuma ve Gorev Baslatma tamamlandi');
    ENDIF;
SUBEND ( Dosya oku )

SUBBODY DosyaYaz
SUBBEGIN
say=0;
bool2=TRUE;
WHILE(bool2)
BEGIN
    RESETOWNER(OwnerB,0);            {say no lu işlemciden işlenen verileri alma}
    ADDOWNER(OwnerB,say);
    bool1=GETGROUP(OwnerB,GroupA);    {işlenen veriler buf3 te }
    IF(bool1) THEN                    {dosya adı Varis ta}
        buf3=UNPACK(buf3);            {veriler açılır}
        yanit1='C: ';
        yanit1=CONCAT(Varis);
        bool1=SAVE(yanit1,buf3,uzunluk);    {dosyada saklama}
        yanit1=CONCAT(' dosyasi saklandi');
        DISPLAY(yanit1);
        ENDIF;
        say=say+1;
        bool2=bool1 AND (say NEQ 4);
    ENDWHILE;

    IF(bool1) THEN
        bool1= KILLTASK(OwnerA,TaskA);    {dört işlemcide görevi}

```

```
ENDIF;
SUBEND (Dosya Yaz )
```

```
{sonlandır}
```

7.2.2 İKİNCİ UYGULAMA PROGRAMININ C DİLİ DÜKÖMÜ

İkinci uygulama programının C diline çevrilmesiyle elde edilen metin aşağıda listelenmiştir. Bu metin TurboC [11] derleyicisinde derlenerek amaç program kodu (OBJ) elde edilmektedir.

```
#include <util.h>
unsigned int BOOL1=FALSE;
unsigned int BOOL2=FALSE;
int UZUNLUK=0;
int SAY=0;
char EK[80]="";
char UZ[80]="";
char KAYNAK[80]="";
char VARIS[80]="";
char YANIT1[80]="";
char YANIT2[80]="";
char DOSYA1[80]="";
char GOREVADI[80]="";
char GRUPADI[80]="";
struct BUFFER BUF1={NULL,0,0};
struct BUFFER BUF2={NULL,0,0};
struct BUFFER BUF3={NULL,0,0};
struct TASK TASKA={"",0,NULL,0,0};
struct OWNER OWNERA={0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
struct OWNER OWNERB={0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
struct GROUP GROUPA={NULL,0,0,NULL};
void DOSYAOKU(void);
void DOSYAYAZ(void);
```

```

void SORGU(void);
void SORGU (void)
{

strcpy(DOSYA1,ACCEPT("GOREV DOSYASI ADINI GIRIN :") );

UZUNLUK=ISFILE(DOSYA1) ;

strcpy(UZ,ASCII(UZUNLUK) );

strcpy(YANIT1,DOSYA1);

strcpy(YANIT1,CONCAT(YANIT1," DOSYASI BOYUTU=") );

strcpy(YANIT1,CONCAT(YANIT1,UZ) );

printf("\n%s",YANIT1) ;

UZUNLUK=UZUNLUK + 10;

BUFF_COPY(&BUF1,ALLOC(UZUNLUK));

strcpy(YANIT2,ACCEPT("DOSYAYI OKU? (E/H)") );

BOOL1=!strcmp(YANIT2,"E");

if(BOOL1) {

BOOL1=LOAD(DOSYA1,&BUF1) ;

} ;

if(BOOL1) {

printf("\n%s","GOREV DOSYASI OKUNDU") ;

strcpy(KAYNAK,ACCEPT("KAYNAK VERİ DOSYASI ADINI GIRIN:") );

```

```
strcpy(GOREVADI,ACCEPT("GOREV ADINI GIRIN:") );
```

```
strcpy(GRUPADI,ACCEPT("GRUP ADINI GIRIN:") );
```

```
else{
```

```
printf("\n%s","GOREV DOSTASI OKUNMADI !!") ;
```

```
} ;
```

```
}
```

```
void DOSYAOKU (void)
```

```
{
```

```
strcpy(EK,".GIR");
```

```
TASKINIT(&TASKA) ;
```

```
BUFF_COPY(&TASKA.POINTER,&BUF1);
```

```
strcpy(TASKA.NAME,GOREVADI);
```

```
RESETOWNER(&OWNER,4) ;
```

```
BOOL1=SENDTASK(&OWNER,&TASKA) ;
```

```
FREE(&BUF1) ;
```

```
BUFF_COPY(&BUF2,ALLOC(2000));
```

```
BUFF_COPY(&BUF3,ALLOC(1000));
```

```
GROUPNAME(&GROUPA,GRUPADI) ;
```

```
GROUPMAKE(&GROUPA,&BUF2,4,2000) ;
```

```
GROUPMAKE(&GROUPA,&VAR1S,3,0) ;
```

```
GROUPMAKE(&GROUPA,&UZUNLUK,2,0) ;
```

```
GROUPMAKE(&GROUPA,&BUF3,4,0) ;
```

```

SAY=0;

BOOL2=BOOL1;

while(BOOL2) {

strcpy(DOSYA1,KAYNAK);

strcpy(UZ,ASCII(SAY) );

strcpy(DOSYA1,CONCAT(DOSYA1,UZ) );

strcpy(DOSYA1,CONCAT(DOSYA1,EK) );

UZUNLUK=ISFILE(DOSYA1) ;

BOOL1=UZUNLUK != 0;

if(BOOL1) {

BOOL1=LOAD(DOSYA1,&BUF2) ;

} ;

if(BOOL1) {

strcpy(DOSYA1,CONCAT(DOSYA1," DOSYASI OKUNDU" ) );

printf("\n%s",DOSYA1) ;

BUFF_COPY(&BUF2,PACK(&BUF2));

RESETOWNER(&OWNERB,0) ;

ADDOwner(&OWNERB,SAY) ;

BOOL1=SENDGROUP(&OWNERB,&GROUPA) ;

```

```

} ;

SAY=SAY + 1;

BOOL2=BOOL1 && (SAY != 4) ;

} ;

if(BOOL1) {

    BOOL1=RUNTASK(&OWNERB,&TASKA) ;

    printf("\n%s","DOSTA OKUMA VE GOREV BASLATMA TAMAMLANDI") ;

} ;

}

void DOSYAYAZ (void)
{

    SAY=0;

    BOOL2=TRUE;

    while(BOOL2) {

        RESETOWNER(&OWNERB,0) ;

        ADDOWNER(&OWNERB,SAY) ;

        BOOL1=GETGROUP(&OWNERB,&GROUPA) ;

        if(BOOL1) {

            BUFF_COPY(&BUF3,UNPACK(&BUF3));

            strcpy(YANIT1,"C:");

```

```

strcpy(YANIT1,CONCAT(YANIT1,VARIS) );

BOOL1=SAVE(YANIT1,&BUF3,UZUNLUK) ;

strcpy(YANIT1,CONCAT(YANIT1," DOSYASI SAKLANDI" ) );

printf("\n%s",YANIT1) ;

} ;

SAY=SAY + 1;

BOOL2=BOOL1 && (SAY != 4) ;

} ;

if(BOOL1) {

BOOL1=KILLTASK(&OWNER,&TASKA) ;

} ;

}

```

7.2.3 İKİNCİ UYGULAMA PROGRAMI EKİNİN C DİLİ DÜKÖMÜ

İkinci uygulama programının C diline çevrilmesiyle elde edilen ek metin (include dosyası, "ornek2.h") aşağıda listelenmiştir. Bu metin C dili ana program metninde kapsanmalıdır..

```

#define TRUE 1
#define FALSE 0
struct BUFFER{
char *POINTER;
unsigned int MAX;

```

```

unsigned int COUNT;
};
struct TASK{
char NAME[80];
int FIELD;
struct BUFFER POINTER;
};
struct GROUP{
char *POINTER;
unsigned int COUNT;
unsigned int TYPE;
struct GROUP *NEXT;
};
struct OWNER{
unsigned char OWN[16];
};
extern unsigned int BOOL1;
extern unsigned int BOOL2;
extern int UZUNLUK;
extern int SAY;
extern char EK[80];
extern char UZ[80];
extern char KAYNAK[80];
extern char VARIS[80];
extern char YANIT1[80];
extern char YANIT2[80];
extern char DOSYA1[80];
extern char GOREVADI[80];
extern char GRUPADI[80];
extern struct BUFFER BUF1;
extern struct BUFFER BUF2;
extern struct BUFFER BUF3;
extern struct TASK TASKA;
extern struct OWNER OWNERA;
extern struct OWNER OWNERB;
extern struct GROUP GROUPA;
extern void DOSYAOKU(void);
extern void DOSYAYAZ(void);

```

```
extern void SORGU(void);
```

7.2.3 C DİLİ ANA PROGRAMININ DÜKÜMÜ

C dilinde hazırlanan ana programın dükümü aşağıda verilmiştir:

```
#include <string.h>
#include <math.h>
#include <ctype.h>
#include <errno.h>
#include <stdio.h>
#include <ornek2.h> /* çevirici tarafından üretildi */

FILE *fp1;
char kaynak_dosya[80];
char kaynak_dosyal[80];

main()
{

char p;
char giris[80];
int i,l,j;
SORGU(); /* sorgulama altprogramı (MPWFL) */
if(BOOL1){ /* BOOL1 değişkeni (MPWFL) */
i=0;
for(i=0;i<4;i++){
itoa(i,kaynak_dosyal,10);
strcat(kaynak_dosyal, ".GIR");
strcpy(kaynak_dosya, KAYNAK); /* KAYNAK değişkeni (MPWFL) */
for(j=0;j<80;j++)
if(kaynak_dosya[j]==0x00)break;
for(l=0;l<80;l++){
kaynak_dosya[j++]=kaynak_dosyal[l];
if(kaynak_dosyal[l]==0x00)break;
}
}
```

```

printf("\n kaynak dosya=%s\n", kaynak_dosya);
if((fpl=fopen(kaynak_dosya, "w"))==0){
printf("\n dosya HATA var\n");
exit(1);
}
printf("\n dosya verilerini giriniz (0-9) cikis Q\n?");
p='1';
while(p!='Q')
{
p=getchar();
if(isdigit(p))putc(p, fpl);
else {p='Q';
break;
}
}
printf("\n dosya yazildi");
fclose(fpl);

}
DOSYAOKU(); /* DOSYAOKU altprogramı (MPWFL) */
if(BOOL1) DOSYAYAZ(); /* DOSYAYAZ altprogramı (MPWFL) */
}
} /* main */

```

SONUÇLAR VE ÖNERİLER

Bu çalışmada, kişisel bilgisayar veya iş istasyonu türünden bir ana bilgisayar ve ona bağlantılı çok işlemcili sistemden oluşan yapılara yönelik bir yük akış dili tasarlanmıştır. Tasarımının başlangıç evresinde, hedef alınan yapıya sahip sistemler incelenmiştir. Özellikle, ana bilgisayar ve çok işlemcili sistem arasındaki mesajlaşmanın türleri ve gerçekleşme şekli ortaya çıkarılmıştır. Bu evrenin sonucunda, MPWFL olarak adlandırdığımız dilin işlevler ve hizmetler kümesi belirlenmiştir.

İşlevler ve hizmetler kümesi taban alınarak, MPWFL dilinin grameri oluşturulmuştur. Gramerin tasarımında LR ayrıştırma kuralları uygulanmıştır. UNIX işletim sistemli bilgisayar üzerinde koşturulan YACC destek yazılımına giriş olarak gramerin metni verilmiştir. YACC programının çıkış olarak ürettiği Ayrıştırma Tablosu kullanılarak alınarak bir Çevirici programı hazırlanmıştır. Çevirici program giriş olarak MPWFL dilinde yazılmış kaynak programı almakta ve amaç program olarak C dili bir metin üretmektedir. Özel işlevler için hazırladığımız destek programları ile C dili metin birlikte derlenerek yürütülebilir bir kod elde edilmektedir.

Yürütülebilir program kodunun tanımlanan işlevleri yerine getirdiğini sınamak için bir deney düzeni oluşturulmuştur. Deney düzeninde, seri haberleşme hattı ile bağlantılanmış iki adet kişisel bilgisayar yer almaktadır. Kişisel bilgisayarlardan biri ana bilgisayar olarak kullanılmakta, diğeri ise çok işlemcili sistem

benzetimi işlevini üstlenmektedir. Benzetim işlemi için ikinci kişisel bilgisayara bir program hazırlanmıştır. Böylece, deney sistemi üzerinde üretilen programlarının denenmesi ile beklenen işlemlerin sonuçları izlenmektedir.

Yazılımların denenmesi için iki uygulama programı hazırlanmıştır. Birinci uygulama programı çevrilerek ve derlenerek yürütebilir program kodu üretilmektedir. İkinci uygulama programı ise çok dilde programlamaya örnek olarak hazırlanmıştır. Bu uygulamada; MPWFL dilinde yazılan altprogramları kapsayan bir metin ile C dilinde yazılan bir ana program metni yer almaktadır. MPWFL programı çevrilerek ve derlenerek, C programı derlenerek amaç program kodları üretilmiştir. Her iki amaç program birleştirilerek ikinci uygulamanın yürütülebilir program kodu elde edilmiştir. Uygulanan programların yürütülmesi sonucunda tüm yazılımların kendinden beklenen işlevleri gerçekleştirdiği gözlenmiştir.

C dilinde hazırlanan özel işlevler destek programları, seri bağlantılı bir sistem varsayımı altında hazırlanmıştır. Karakter tabanında yapılan bu tür haberleşmenin süresi uzun olmaktadır. Bir sonraki aşama olarak, yerel alan şebekesi (LAN) ile bağlantılı bir sisteme yönelik bu destek programlarının hazırlanması ve denenmesi verilebilir.

MPWFL dilinin gerçek uygulaması iki türde yapılabilir. Birinci tür uygulamada, ana bilgisayar üzerinde mevcut durum korunur ve ele alınan çok işlemcili sistem üzerinde bir yazılım geliştirilmesi yapılır. İkinci tür uygulamada ise, ele alınan çok işlemcili sistemin ana bilgisayar ile haberleşme yazılımı korunur ve bununla ilişkili olarak ana bilgisayar üzerindeki destek programları yeniden hazırlanır. Her iki uygulama türünde de, MPWFL dili üzerinde yeniden bir çalışma yapılması gerekmemektedir.

KAYNAKLAR

- [1] IBM. System Product Interpreter Reference (REXX).
- [2] UNISYS, A Series Work Flow Language (WFL) Programming Reference Manual. (1987).
- [3] TUYNMAN, F., HERTZBERGER, L.O., A Distributed Real-Time Operating System, Software Practice and Experience, Vol 16(5), pp. 425-441. May, (1986).
- [4] AKGON. B. T., Kare Piramit Yapılı Modüler Çok İşlemcili Bir Gerçek Zaman Sistemi, Doktora Tezi, İ.T.Ü. Fen Bilimleri Enstitüsü. (1991).
- [5] INTEL, iPCS (Hypercube) Manuel.
- [6] AHO, A.V., SETHI. R., ULLMAN. J. D., Compilers Principles, Techniques, and Tools, Addison-Wesley.(1986).
- [7] HOLUB, A. I., Compiler Design in C. Prentice-Hall. (1990).
- [8] HUNTER, R., Compilers their Design and Construction Using Pascal, John Wiley, (1985).
- [9] SINIX sistem notları, YACC dokümanı.
- [10] SINIX sistem notları, LEX dokümanı.
- [11] KERNIGHAN, B.W., RITCHIE, D. M., The C Programming Language, Prentice-Hall, (1978).
- [12] BORLAND INTERNATIONAL. TurboC Ver. 2.0 Reference Guide, (1988).

EKLER

EK A. MPWFL.C PROGRAMININ DOKÜMANI

```
#include <string.h>
#include <math.h>
#include <ctype.h>
#include <errno.h>
#include <stdio.h>

#define EQUAL 0
#define LB 1
#define RB 2
#define LP 3
#define RP 4
#define COMMA 5
#define STOP 6
#define PLUS 7
#define MINUS 8
#define STAR 9
#define DIV 10
#define MOD 11
#define AND 12
#define OR 13
#define IMP 14
#define NOT 15
#define EOLN 16
#define EOF 17
#define IC 18
#define BC 19
#define SC 20
#define LOAD 21
#define SAVE 22
#define ASCII 23
#define ALLOC 24
#define REVERSE 25
#define GROUPNAME 26
#define GROUPMAKE 27
#define TASKCOPY 28
#define TASKINIT 29
#define DISPLAY 30
#define PLINE 31
#define PCONTROL 32
#define FREE 33
#define IF 34
#define THEN 35
#define ENDIF 36
#define ELSE 37
```

```

#define WHILE      38
#define BEGIN      39
#define ENDWHILE   40
#define REPEAT     41
#define UNTIL      42
#define SELECT     43
#define OTHERWISE  44
#define ENDSELECT  45
#define WHEN       46
#define DO         47
#define ENDWHEN    48
#define SUBBODY    49
#define SUBBEGIN   50
#define SUBEND     51
#define BID        52
#define IID        53
#define SID        54
#define BUD        55
#define TID        56
#define OID        57
#define GID        58
#define SUD        59
#define ID         60
#define TNA        61
#define TFA        62
#define TBA        63
#define TFN        64
#define GFN        65
#define RELOP      66
#define IFN        67
#define SFN        68
#define TDFN       69
#define PFN        70
#define OSN        71
#define SSN        72
#define DECS       73
#define SRELOP     74

#define BAD        75
#define ERRIDLST   76
#define LSS 0
#define LEQ 1
#define GTR 2
#define GEQ 3
#define EQU 4
#define NEQ 5
#define PARSERSTACKSIZE 100
#define IDSIZE 100
#define MAXINPUT 20000
#define TRUE 1
#define FALSE 0

struct TOKENREC
(
    int type;
    int value;

```



```

    0, 0, 0, 0, 0, 0, 62, 0, 0, 0, 0, 0, 0, 0, 0,
    33, 34, 35, 36, 0, 0, 0, 66, 80, 83, 00, 0, 0, 0, 0,
    0, 0, 0, 49, 0, 89, 101, 0, 0, 0, 0, 0, 0, 0, 64,
    0, 0, 0, 0, 31, 0, 37, 39, 0, 0, 0, 0, 0, 123, 0,
    47, 0, 104, 105, 113, 114, 0, 115, 124, 125, 0, 0, 43, 0, 0,
    0, 0, 0, 46, 0, 0, 0, 32, 0, 85, 0, 87, 88, 52, 116,
    0, 44, 0, 0, 86, 126, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    };
    errno_table[]={
128, 0, 1, 2, 0, 3, 0, 4, 0, 0, 5, 6, 7, 0, 0,
    8, 0, 0, 9, 10, 0, 11, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 13,
    14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 29,
    28, 0, 0, 0, 0, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
    40, 41, 42, 0, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,
    54, 0, 0, 55, 56, 57, 0, 58, 0, 0, 0, 0, 0, 0, 0,
    0, 59, 0, 0, 60, 61, 62, 63, 64, 65, 0, 0, 66, 0, 0,
    67, 68, 69, 0, 70, 0, 71, 72, 73, 0, 74, 0, 0, 0, 0,
    75, 76, 77, 0, 0, 0, 0, 0, 0, 78, 79, 80, 81, 82, 83,
    84, 85, 0, 0, 0, 0, 0, 86, 87, 88, 89, 90, 91, 0, 92,
    0, 0, 0, 93, 94, 0, 0, 0, 0, 95, 96, 97, 98, 99, 100,
    101, 0, 102, 103, 104, 105, 106, 107, 0, 0, 0, 0, 0, 0, 108,
    109, 110, 111, 112, 113, 114, 0, 115, 116, 117, 118, 119, 120, 121, 122,
    0, 0, 0, 0, 123, 124, 125, 0, 0, 0, 0, 130, 131, 132, 133,
    134, 135, 126, 0, 127, 0, 0, 136, 137, 138, 139, 140, 141, 0, 0,
    0, 142, 143, 144, 0, 145, 0, 0, 146, 147, 148, 149, 150, 0, 151,
    0, 152, 0, 0, 0, 0, 153, 0, 0, 0, 154, 155, 0, 156, 157,
    158, 159, 160, 0, 161, 162, 163, 0, 164, 0, 165, 0, 0, 0, 0,
    166, 0, 167, 129, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    };

void init_file(void){
    fprintf(fp2, "\n#include <util.h>");

    if(fp3_flag) {

        fprintf(fp3, "\n#define TRUE 1");
        fprintf(fp3, "\n#define FALSE 0");
        fprintf(fp3, "\nstruct BUFFER(");
        fprintf(fp3, "\nchar *POINTER%c", chs);
        fprintf(fp3, "\nunsigned int MAX%c", chs);
        fprintf(fp3, "\nunsigned int COUNT%c", chs);
        fprintf(fp3, "\n)%c", chs);
        fprintf(fp3, "\nstruct TASK(");
        fprintf(fp3, "\nchar NAME[80]%c", chs);
        fprintf(fp3, "\nint FIELD%c", chs);
        fprintf(fp3, "\nstruct BUFFER POINTER%c", chs);
        fprintf(fp3, "\n)%c", chs);
        fprintf(fp3, "\nstruct GROUP(");
        fprintf(fp3, "\nchar *POINTER%c", chs);
        fprintf(fp3, "\nunsigned int COUNT%c", chs);
        fprintf(fp3, "\nunsigned int TYPE%c", chs);
        fprintf(fp3, "\nstruct GROUP *NEXT%c", chs);
        fprintf(fp3, "\n)%c", chs);
        fprintf(fp3, "\nstruct OWNER(");
        fprintf(fp3, "\nunsigned char OWN[16]%c", chs);
    }
}

```

```

fprintf(fp3, "\n%c", chs);

}

}

int isfunc(char *s)
{
    char *start=input;
    char name[80];
    int len = strlen(s);
    int i=0;
    while(!isdelim(*start))name[i++]=*start++;
    name[i]=0;
    if(strlen(name)!=len) return(FALSE);
    if (strcmp(s, input, len) == 0)
    {
        strcpy(curtoken.name, input, len);
        curtoken.namelen = 0;
        input += len;
        return(TRUE);
    }
    return(FALSE);
} /* isfunc */

int isdelim(c)
char c;
{
    if(is_in(c, " =,+*/<>[];'\"")||c==EOLN||c==0x00||c==' ')
        return 1;
    return 0;
}

int is_in(char c, s)
char cha,*s;
{
    while (*s) if(*s++==cha)
        return 1;
    return 0;
}

int nexttoken(void)
{
    char *start, numstring[80];
    int decimal, len, numlen, i;

    while (*input == ' ')
        input++;

    if (*input == 0)
    {
        input++;
        return(EOF);
    }
    if (*input == EOLN)
    {
        line_num++;
    }

```

```

        if(fp4_flag)fprintf(fp4,"\n%d INCI SATIR",line_num);
        input++;
        if((line_num%4)==0) printf("\r!");
        if((line_num%4)==1) printf("\r/");
        if((line_num%4)==2) printf("\r-");
        if((line_num%4)==3) printf("\r\\");
        return(EOLN);
    }
    if (is_in(*input,"0123456789"))
    {
        start = input;
        len = 0;
        while (isdigit(*input))
        {
            input++;
            len++;
        }

        strncpy(curtoken.name, start, len);
        curtoken.name[len]=0;
        curtoken.value = atoi(start);
        return(IC);
    }
    else if (is_in(*input,""))
    {
        input++;
        start = input;
        len = 0;
        while (!is_in(*input,""))
        {
            input++;
            len++;
        }
        input++;
        if(len>80)return(BAD);
        strncpy(curtoken.name, start, len);
        curtoken.name[len]=0;
        return(SC);
    }
    else if (isalpha(*input))
    {
        if(isfunc("AND")) return(AND);
        if(isfunc("OR")) return(OR);
        if(isfunc("IMP")) return(IMP);
        if(isfunc("NOT")) return(NOT);
        if(isfunc("LOAD")) return(LOAD);
        if(isfunc("SAVE")) return(SAVE);
        if(isfunc("MOD")) return(MOD);
        if(isfunc("ASCII")) return(ASCII);
        if(isfunc("ALLOC")) return(ALLOC);
        if(isfunc("REVERSE")) return(REVERSE);
        if(isfunc("GROUPNAME")) return(GROUPNAME);
        if(isfunc("GROUPMAKE")) return(GROUPMAKE);
        if(isfunc("TASKCOPY")) return(TASKCOPY);
        if(isfunc("TASKINIT")) return(TASKINIT);
        if(isfunc("DISPLAY")) return(DISPLAY);
    }

```

```

if(isfunc("PLINE")) return(PLINE);
if(isfunc("PCONTROL")) return(PCONTROL);
if(isfunc("FREE")) return(FREE);
if(isfunc("IF")) return(IF);
if(isfunc("THEN")) return(THEN);
if(isfunc("ENDIF")) return(ENDIF);
if(isfunc("ELSE")) return(ELSE);
if(isfunc("WHILE")) return(WHILE);
if(isfunc("BEGIN")) return(BEGIN);
if(isfunc("ENDWHILE")) return(ENDWHILE);
if(isfunc("REPEAT")) return(REPEAT);
if(isfunc("UNTIL")) return(UNTIL);
if(isfunc("SELECT")) return(SELECT);
if(isfunc("OTHERWISE")) return(OTHERWISE);
if(isfunc("ENDSELECT")) return(ENDSELECT);
if(isfunc("WHEN")) return(WHEN);
if(isfunc("DO")) return(DO);
if(isfunc("ENDWHEN")) return(ENDWHEN);
if(isfunc("SUBBODY")) { in_sub=TRUE;return(SUBBODY);}
if(isfunc("SUBBEGIN")) {in_sub=FALSE;return(SUBBEGIN);}
if(isfunc("SUBEND")) return(SUBEND);
if(isfunc("SEQU")) return(SRELOP);
if(isfunc("LSS")){curtoken.value=LSS; return(RELOP);}
if(isfunc("LEQ")){curtoken.value=LEQ; return(RELOP);}
if(isfunc("GTR")){curtoken.value=GTR; return(RELOP);}
if(isfunc("GEQ")){curtoken.value=GEQ; return(RELOP);}
if(isfunc("EQU")){curtoken.value=EQU; return(RELOP);}
if(isfunc("NEQ")){curtoken.value=NEQ; return(RELOP);}
if(isfunc("TRUE")) return(BC);
if(isfunc("FALSE")) return(BC);
if(isfunc("INTEGER")){in_decs=IID; return(DECS);}
if(isfunc("BOOLEAN")){in_decs= BID; return(DECS);}
if(isfunc("STRING")){in_decs=SID; return(DECS);}
if(isfunc("BUFFER")){in_decs=BUD; return(DECS);}
if(isfunc("TASK")){in_decs=TID; return(DECS);}
if(isfunc("SUBROUTINE")){in_decs=SUD; return(DECS);}
if(isfunc("OWNER")){in_decs=OID; return(DECS);}
if(isfunc("GROUP")){in_decs=GID; return(DECS);}
if(isfunc("NAME")) return(TNA);
if(isfunc("FIELD")) return(TFA);
if(isfunc("POINTER")) return(TBA);
if(isfunc("SENDTASK")) return(TFN);
if(isfunc("GETTASK")) return(TFN);
if(isfunc("RUNTASK")) return(TFN);
if(isfunc("ABORTTASK")) return(TFN);
if(isfunc("KILLTASK")) return(TFN);
if(isfunc("SENDGROUP")) return(GFN);
if(isfunc("GETGROUP")) return(GFN);
if(isfunc("ADDOOWNER")) return(OSN);
if(isfunc("SUBOWNER")) return(OSN);
if(isfunc("RESETOWNER")) return(OSN);
if(isfunc("DECIMAL")) return(IFN);
if(isfunc("LENGTH")) return(IFN);
if(isfunc("ISFILE")) return(IFN);
if(isfunc("ACCEPT")) return(SFN);
if(isfunc("CONCAT")) return(SFN);

```

```

if(isfunc("TAKE")) return(TDFN);
if(isfunc("DROP")) return(TDFN);
if(isfunc("PACK")) return(PFN);
if(isfunc("UNPACK")) return(PFN);
if(isfunc("NOP")) return(SSN);

for(i=0;i<id_num;i++)
{
    if(isfunc(id_list[i].name))
    {
        if(in_decs!=0)return(BAD);
        return(id_list[i].type);
    }
}
if(in_decs!=0) {
    if(id_num>=IDSIZE)return(ERRIDLST);
    i=0;
    start=input;
    while(!isdelim(*start))id_list[id_num].name[i++]=*start++;
    id_list[id_num].name[i]=0;
    id_list[id_num].type=in_decs;
    isfunc(id_list[id_num].name);
    id_num++;
    return(ID);
}

return(BAD);
}
else {
    curtoken.name[0]=*input;
    curtoken.name[1]=0;
    switch(*(input++))
    {

        case '=' : return(EQUAL);
        case ',' : return(COMMA);
        case '+' : return(PLUS);
        case '-' : return(MINUS);
        case '*' : return(STAR);
        case '/' : return(DIV);
        case '(' : return(LP);
        case ')' : return(RP);
        case '[' : return(LB);
        case ']' : return(RB);
        case ';' : in_decs=0;
                    return(STOP);
        default : return(BAD);
    } /* switch */
}
/* nexttoken */
void print_token(void)
{
    if(fp4_flag){
        if(curtoken.type==BAD){
            fprintf(fp4,"\n%d inci satirda HATA=%d ",line_num,curtoken.type);
            return;}

```

```

if(curtoken.type<LC){
if(curtoken.type==EOLN)
fprintf(fp4,"\n%d=EOLN",curtoken.type);
else
if(curtoken.type==EOF FILE)
fprintf(fp4,"\n%d=EOF FILE",curtoken.type);
else
fprintf(fp4,"\n%d=%s",curtoken.type,curtoken.name);
} else {
if(curtoken.type==(C)){

fprintf(fp4,"\n%d=%d.%s",curtoken.type,curtoken.value,curtoken.name);
} else

fprintf(fp4,"\n%d=%s",curtoken.type,curtoken.name);

}
}
}
void push(struct TOKENREC *token)
{
if (stacktop == PARSESTACKSIZE - 1)
{
printf("\n YIGIN DOLU\n");
error = TRUE;
}
else
stack[++stacktop] = *token;
}

struct TOKENREC pop(void)
{
return(stack[stacktop--]);
}

int gotostate(int production)
{
int state = stack[stacktop].state;
int nonterm;

nonterm=product_table[production];
if(fp4_flag)
fprintf(fp4,"\ngoto production=%d, nonterm=%d, state=%d",
production,nonterm,state);
switch(nonterm)
{
case 0: return (291);
case 1:
switch(state){

case 0: return(1);
default: return (291); }
case 2:
switch(state){

```

```

    case 0: return(2);
    default: return (291); }
case 3:
    switch(state){

        case 0: return(3);
        default: return (291); }
case 4:
    switch(state){

        case 0: return(4);
        case 3: return(8);
        default: return (291); }
case 5:
    switch(state){

        case 5: return(12);
        default: return (291); }
case 6:
    switch(state){

        case 3: return(7);
        default: return (291); }
case 7:
    switch(state){

        case 3: return(9);
        case 7: return(14);
        default: return (291); }
case 8:
    switch(state){

        case 3: return(10);
        case 7: return(10);
        default: return (291); }
case 9:
    switch(state){

        case 15: return(19);
        case 54: return(76);
        case 124: return(185);
        case 216: return(248);
        case 226: return(254);
        case 256: return(274);
        case 268: return(278);
        default: return (291); }
case 10:
    switch(state){

        case 15: return(20);
        case 19: return(63);
        case 54: return(20);
        case 76: return(63);
        case 124: return(20);
        case 185: return(63);
        case 216: return(20);

```

```

    case 226: return(20);
    case 248: return(63);
    case 254: return(63);
    case 256: return(20);
    case 268: return(20);
    case 274: return(63);
    case 278: return(63);
    default: return (291); }
case 11:
    switch(state){

        case 15: return(21);
        case 19: return(21);
        case 54: return(21);
        case 76: return(21);
        case 124: return(21);
        case 185: return(21);
        case 216: return(21);
        case 226: return(21);
        case 248: return(21);
        case 254: return(21);
        case 256: return(21);
        case 268: return(21);
        case 274: return(21);
        case 278: return(21);
        default: return (291); }
case 12:
    switch(state){

        case 15: return(23);
        case 19: return(23);
        case 54: return(23);
        case 76: return(23);
        case 124: return(23);
        case 185: return(23);
        case 216: return(23);
        case 226: return(23);
        case 248: return(23);
        case 254: return(23);
        case 256: return(23);
        case 268: return(23);
        case 274: return(23);
        case 278: return(23);
        default: return (291); }
case 13:
    switch(state){

        case 15: return(24);
        case 19: return(24);
        case 54: return(24);
        case 76: return(24);
        case 124: return(24);
        case 185: return(24);
        case 216: return(24);
        case 226: return(24);
        case 248: return(24);

```

```

    case 254: return(24);
    case 256: return(24);
    case 268: return(24);
    case 274: return(24);
    case 278: return(24);
    default: return (291); }
case 14:
    switch(state){

        case 161: return(209);
        default: return (291); }
case 15:
    switch(state){

        case 15: return(25);
        case 19: return(25);
        case 54: return(25);
        case 76: return(25);
        case 124: return(25);
        case 185: return(25);
        case 216: return(25);
        case 226: return(25);
        case 248: return(25);
        case 254: return(25);
        case 256: return(25);
        case 268: return(25);
        case 274: return(25);
        case 278: return(25);
        default: return (291); }
case 16:
    switch(state){

        case 15: return(26);
        case 19: return(26);
        case 54: return(26);
        case 76: return(26);
        case 124: return(26);
        case 185: return(26);
        case 216: return(26);
        case 226: return(26);
        case 248: return(26);
        case 254: return(26);
        case 256: return(26);
        case 268: return(26);
        case 274: return(26);
        case 278: return(26);
        default: return (291); }
case 17:
    switch(state){

        case 15: return(27);
        case 19: return(27);
        case 54: return(27);
        case 76: return(27);
        case 124: return(27);
        case 185: return(27);

```

```

    case 216: return(27);
    case 226: return(27);
    case 248: return(27);
    case 254: return(27);
    case 256: return(27);
    case 268: return(27);
    case 274: return(27);
    case 278: return(27);
    default: return (291); }
case 18:
    switch(state){

        case 15: return(28);
        case 19: return(28);
        case 54: return(28);
        case 76: return(28);
        case 124: return(28);
        case 185: return(28);
        case 216: return(28);
        case 226: return(28);
        case 248: return(28);
        case 254: return(28);
        case 256: return(28);
        case 268: return(28);
        case 274: return(28);
        case 278: return(28);
        default: return (291); }
case 19:
    switch(state){

        case 15: return(29);
        case 19: return(29);
        case 54: return(29);
        case 76: return(29);
        case 124: return(29);
        case 185: return(29);
        case 216: return(29);
        case 226: return(29);
        case 248: return(29);
        case 254: return(29);
        case 256: return(29);
        case 268: return(29);
        case 274: return(29);
        case 278: return(29);
        default: return (291); }
case 20:
    switch(state){

        case 15: return(30);
        case 19: return(30);
        case 54: return(30);
        case 76: return(30);
        case 124: return(30);
        case 185: return(30);
        case 216: return(30);
        case 226: return(30);

```

```
case 248: return(30);
case 254: return(30);
case 256: return(30);
case 268: return(30);
case 274: return(30);
case 278: return(30);
default: return (291); }
```

```
case 21:
```

```
switch(state){
```

```
case 15: return(32);
case 19: return(32);
case 54: return(32);
case 76: return(32);
case 124: return(32);
case 185: return(32);
case 216: return(32);
case 226: return(32);
case 248: return(32);
case 254: return(32);
case 256: return(32);
case 268: return(32);
case 274: return(32);
case 278: return(32);
default: return (291); }
```

```
case 22:
```

```
switch(state){
```

```
case 15: return(33);
case 19: return(33);
case 54: return(33);
case 76: return(33);
case 124: return(33);
case 185: return(33);
case 216: return(33);
case 226: return(33);
case 248: return(33);
case 254: return(33);
case 256: return(33);
case 268: return(33);
case 274: return(33);
case 278: return(33);
default: return (291); }
```

```
case 23:
```

```
switch(state){
```

```
case 15: return(34);
case 19: return(34);
case 54: return(34);
case 76: return(34);
case 124: return(34);
case 185: return(34);
case 216: return(34);
case 226: return(34);
case 248: return(34);
case 254: return(34);
```

```

    case 256: return(34);
    case 268: return(34);
    case 274: return(34);
    case 278: return(34);
    default: return (291); }
case 24:
    switch(state){

        case 15: return(35);
        case 19: return(35);
        case 54: return(35);
        case 76: return(35);
        case 124: return(35);
        case 185: return(35);
        case 216: return(35);
        case 226: return(35);
        case 248: return(35);
        case 254: return(35);
        case 256: return(35);
        case 268: return(35);
        case 274: return(35);
        case 278: return(35);
        default: return (291); }
case 25:
    switch(state){

        case 15: return(36);
        case 19: return(36);
        case 54: return(36);
        case 76: return(36);
        case 124: return(36);
        case 185: return(36);
        case 216: return(36);
        case 226: return(36);
        case 248: return(36);
        case 254: return(36);
        case 256: return(36);
        case 268: return(36);
        case 274: return(36);
        case 278: return(36);
        default: return (291); }
case 26:
    switch(state){

        case 55: return(77);
        default: return (291); }
case 27:
    switch(state){

        case 55: return(78);
        case 77: return(125);
        default: return (291); }
case 28:
    switch(state){

        case 15: return(22);

```

```

    case 19: return(22);
    case 54: return(22);
    case 76: return(22);
    case 124: return(22);
    case 185: return(22);
    case 216: return(22);
    case 226: return(22);
    case 248: return(22);
    case 254: return(22);
    case 256: return(22);
    case 268: return(22);
    case 274: return(22);
    case 278: return(22);
    default: return (291); }
case 29:
    switch(state){

        case 15: return(38);
        case 19: return(38);
        case 54: return(38);
        case 76: return(38);
        case 124: return(38);
        case 185: return(38);
        case 216: return(38);
        case 226: return(38);
        case 248: return(38);
        case 254: return(38);
        case 256: return(38);
        case 268: return(38);
        case 274: return(38);
        case 278: return(38);
        default: return (291); }
case 30:
    switch(state){

        case 15: return(39);
        case 19: return(39);
        case 54: return(39);
        case 76: return(39);
        case 124: return(39);
        case 185: return(39);
        case 216: return(39);
        case 226: return(39);
        case 248: return(39);
        case 254: return(39);
        case 256: return(39);
        case 268: return(39);
        case 274: return(39);
        case 278: return(39);
        default: return (291); }
case 31:
    switch(state){

        case 15: return(40);
        case 19: return(40);
        case 54: return(40);

```

```

    case 76: return(40);
    case 124: return(40);
    case 185: return(40);
    case 216: return(40);
    case 226: return(40);
    case 248: return(40);
    case 254: return(40);
    case 256: return(40);
    case 268: return(40);
    case 274: return(40);
    case 278: return(40);
    default: return (291); }
case 32:
    switch(state){

        case 15: return(42);
        case 19: return(42);
        case 54: return(42);
        case 76: return(42);
        case 124: return(42);
        case 185: return(42);
        case 216: return(42);
        case 226: return(42);
        case 248: return(42);
        case 254: return(42);
        case 256: return(42);
        case 268: return(42);
        case 274: return(42);
        case 278: return(42);
        default: return (291); }
case 33:
    switch(state){

        case 15: return(41);
        case 19: return(41);
        case 54: return(41);
        case 76: return(41);
        case 124: return(41);
        case 185: return(41);
        case 216: return(41);
        case 226: return(41);
        case 248: return(41);
        case 254: return(41);
        case 256: return(41);
        case 268: return(41);
        case 274: return(41);
        case 278: return(41);
        default: return (291); }
case 34:
    switch(state){

        case 83: return(148);
        default: return (291); }
case 35:
    switch(state){

```

```

    case 73: return(95);
    case 80: return(127);
    case 106: return(174);
    case 126: return(186);
    default: return (291); }

```

```

case 36:
    switch(state){

        case 73: return(96);
        case 80: return(96);
        case 106: return(96);
        case 126: return(96);
        case 169: return(217);
        default: return (291); }

```

```

case 37:
    switch(state){

        case 95: return(169);
        case 127: return(169);
        case 174: return(169);
        case 186: return(169);
        default: return (291); }

```

```

case 38:
    switch(state){

        case 73: return(98);
        case 80: return(98);
        case 97: return(173);
        case 106: return(98);
        case 126: return(98);
        case 169: return(98);
        default: return (291); }

```

```

case 39:
    switch(state){

        case 73: return(99);
        case 75: return(121);
        case 80: return(99);
        case 97: return(99);
        case 106: return(99);
        case 126: return(99);
        case 169: return(99);
        case 184: return(227);
        default: return (291); }

```

```

case 40:
    switch(state){

        case 73: return(100);
        case 80: return(100);
        case 97: return(100);
        case 106: return(100);
        case 126: return(100);
        case 169: return(100);
        default: return (291); }

```

```

case 41:
    switch(state){

```

```

    case 73: return(101);
    case 80: return(101);
    case 97: return(101);
    case 106: return(101);
    case 126: return(101);
    case 169: return(101);
    default: return (291); }
case 42:
    switch(state){

        case 73: return(102);
        case 80: return(102);
        case 97: return(102);
        case 106: return(102);
        case 126: return(102);
        case 169: return(102);
        default: return (291); }
case 43:
    switch(state){

        case 73: return(103);
        case 80: return(103);
        case 97: return(103);
        case 106: return(103);
        case 126: return(103);
        case 169: return(103);
        default: return (291); }
case 44:
    switch(state){

        case 73: return(104);
        case 80: return(104);
        case 97: return(104);
        case 106: return(104);
        case 126: return(104);
        case 169: return(104);
        default: return (291); }
case 45:
    switch(state){

        case 73: return(105);
        case 80: return(105);
        case 97: return(105);
        case 106: return(105);
        case 126: return(105);
        case 169: return(105);
        default: return (291); }
case 46:
    switch(state){

        case 81: return(128);
        case 135: return(194);
        default: return (291); }
case 47:
    switch(state){

```

```

    case 81: return(129);
    case 135: return(129);
    case 187: return(230);
    default: return (291); }
case 48:
    switch(state){

        case 128: return(187);
        case 194: return(187);
        default: return (291); }
case 49:
    switch(state){

        case 81: return(131);
        case 130: return(193);
        case 135: return(131);
        case 187: return(131);
        default: return (291); }
case 50:
    switch(state){

        case 73: return(109);
        case 80: return(109);
        case 81: return(132);
        case 97: return(109);
        case 106: return(109);
        case 126: return(109);
        case 130: return(132);
        case 135: return(132);
        case 164: return(215);
        case 169: return(109);
        case 175: return(219);
        case 187: return(132);
        case 198: return(235);
        case 203: return(239);
        case 205: return(241);
        case 245: return(266);
        case 261: return(275);
        case 265: return(276);
        case 280: return(287);
        case 285: return(288);
        default: return (291); }
case 51:
    switch(state){

        case 81: return(133);
        case 130: return(133);
        case 135: return(133);
        case 187: return(133);
        default: return (291); }
case 52:
    switch(state){

        case 81: return(134);
        case 130: return(134);
        case 135: return(134);

```

```

    case 187: return(134);
    default: return (291); }
case 53:
    switch(state){

        case 82: return(138);
        default: return (291); }
case 54:
    switch(state){

        case 70: return(90);
        case 71: return(93);
        case 82: return(139);
        case 160: return(208);
        case 176: return(220);
        case 177: return(221);
        case 178: return(222);
        case 195: return(232);
        case 197: return(234);
        case 199: return(236);
        case 202: return(238);
        default: return (291); }
case 55:
    switch(state){

        case 82: return(140);
        default: return (291); }
case 56:
    switch(state){

        case 82: return(141);
        default: return (291); }
case 57:
    switch(state){

        case 82: return(143);
        default: return (291); }
case 58:
    switch(state){

        case 82: return(142);
        default: return (291); }
case 59:
    switch(state){

        case 84: return(152);
        default: return (291); }
case 60:
    switch(state){

        case 74: return(117);
        case 84: return(153);
        case 204: return(240);
        case 206: return(242);
        case 207: return(243);
        case 249: return(269);

```

```

        case 250: return(270);
        default: return (291); }
case 61:
    switch(state){

        case 74: return(119);
        case 84: return(119);
        case 204: return(119);
        case 206: return(119);
        case 207: return(119);
        case 249: return(119);
        case 250: return(119);
        default: return (291); }
case 62:
    switch(state){

        case 84: return(154);
        default: return (291); }
case 63:
    switch(state){

        case 84: return(155);
        default: return (291); }
case 64:
    switch(state){

        case 84: return(156);
        default: return (291); }

    }
    return(291);
} /* gotostate */

void shift(int state)
{
    int token1;
    if(fp4_flag)
        fprintf(fp4, "\nshift state=%d", state);
    curtoken.state = state;
    push(&curtoken);
    token1=EOLN;
    while(token1==EOLN) token1=nexttoken();
    tokentype = token1;
    curtoken.type=tokentype;
    print_token();
} /* shift */

void reduce(int reduction)
{
    int i;
    if(fp4_flag)
        fprintf(fp4, "\nreduction state=%d", reduction);

    for(i=1; i<=reduce_num_table[reduction]; i++)
        curtok = pop();
    curtoken.state = gotostate(reduction);

```

```

if(fp4_flag)
    fprintf(fp4, "\n reduce push=%d", curtoken.state);
    push(&curtoken);
} /* reduce */
void make_id(void)
{
    switch (in_decs)
    {
        int len;
        case BID: fprintf(fp2, "\nunsigned int %s=FALSE;", curtoken.name);
                    if(fp3_flag)
                        fprintf(fp3, "\n extern unsigned int %s;", curtoken.name);
                    break;

        case IID: fprintf(fp2, "\nint %s=0;", curtoken.name);
                    if(fp3_flag)
                        fprintf(fp3, "\n extern int %s;", curtoken.name);
                    break;

        case SID: fprintf(fp2, "\nchar %s[80]=%c%c;", curtoken.name, ch, ch);
                    if(fp3_flag)
                        fprintf(fp3, "\n extern char %s[80];", curtoken.name);
                    break;

        case BUD: fprintf(fp2, "\nstruct BUFFER %s={NULL,0,0};", curtoken.name);
                    if(fp3_flag)
                        fprintf(fp3, "\n extern struct BUFFER %s;", curtoken.name);
                    break;

        case TID: fprintf(fp2, "\nstruct TASK %s={%c%c,0,NULL,0,0};",
                        curtoken.name, ch, ch);
                    if(fp3_flag)
                        fprintf(fp3, "\n extern struct TASK %s;", curtoken.name);
                    break;

        case OID: fprintf(fp2,
                        "\nstruct OWNER %s={0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,"
                        curtoken.name);
                        fprintf(fp2, "\n0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};");
                    if(fp3_flag)
                        fprintf(fp3, "\n extern struct OWNER %s;", curtoken.name);
                    break;

        case GID: fprintf(fp2, "\nstruct GROUP %s={NULL,0,0,NULL};", curtoken.name);
                    if(fp3_flag)
                        fprintf(fp3, "\n extern struct GROUP %s;", curtoken.name);
                    break;

        case SUD: len=strlen(curtoken.name);
                    if(strncmp("MAIN", curtoken.name, len)==0) break;
                    fprintf(fp2, "\nvoid %s(void);", curtoken.name);
                    if(fp3_flag)
                        fprintf(fp3, "\n extern void %s(void);", curtoken.name);
                    break;
    }
}

int parse(void)
{
    struct TOKENREC firsttoken, tok1, tok2, tok3, tok4, tok5;
    char accepted = FALSE;
    char copy[80];
    int token1, token2, token3, token4;

```

```

error = FALSE;
isformula = FALSE;
stacktop = -1;
firsttoken.state = 0;
firsttoken.value = 0;
push(&firsttoken);
token1=EOLN;
while(token1==EOLN)token1=nexttoken();
tokentype = token1;
curtoken.type=tokentype;
do
{
if((stack[stacktop].state>0)&&(stack[stacktop].state<=291))
{
    token2=reduce_table[stack[stacktop].state];
    if(token2)reduce(token2);
    else
    {
switch (stack[stacktop].state)
{
case 0 :
    if (tokentype == DECS)
        shift(5);
    else
        error = TRUE;
        break;
case 1 :
    if (tokentype == EOF1LE)
        accepted = TRUE;
    else
        error = TRUE;
        break;
case 2 :
    if (tokentype == EOF1LE)
        shift(6);
    else
        error = TRUE;
        break;
case 3 :
    if (tokentype == SUBBODY)
        shift(11);
    else
    if (tokentype == DECS)
        shift(5);
    else
        error = TRUE;
        break;
case 5 :
    if (tokentype == ID)
    { make_id();
      shift(13);
    }
    else
        error = TRUE;
        break;
case 7 :

```

```

if (tokentype == SUBBODY)
    shift(11);
else
    reduce(2);
break;
case 10 :
if (tokentype == SUBBEGIN) {
    fprintf(fp2, "\n{\n");
    shift(15);}
else
    error = TRUE;
break;
case 11 :
if (tokentype == SUD){
    if(strcmp("MAIN", curtoken.name)==0)
        fprintf(fp2, "\nmain ( )");
    else
        fprintf(fp2, "\nvoid %s (void) ", curtoken.name);
    shift(16);}
else
    error = TRUE;
break;
case 12 :
if (tokentype == COMMA)
    shift(18);
else
    if (tokentype == STOP)
        shift(17);
    else
        error = TRUE;
break;
case 278:
case 274:
case 268:
case 256:
case 254:
case 248:
case 226:
case 216:
case 185:
case 124:
case 76:
case 54:
case 19:
case 15:
    switch(tokentype)
    {
        case GROUPNAME :fprintf(fp2, "\nGROUPNAME");shift(43);break;
        case GROUPMAKE :fprintf(fp2, "\nGROUPMAKE");shift(44);break;
        case TASKCOPY :fprintf(fp2, "\nTASKCOPY");shift(45);break;
        case TASKINIT :fprintf(fp2, "\nTASKINIT");shift(46);break;
        case DISPLAY :fprintf(fp2, "\nprintf(%c%c%c", ch, chb, ch, ch);
                        shift(48);break;
        case PLINE :fprintf(fp2, "\nprintf(%c%c", ch, ch, ch);
                        shift(49);break;
        case PCONTROL :fprintf(fp2, "\nprintf");shift(50);break;
    }

```

```

case FREE      :fprintf(fp2,"FREE");shift(52);break;
case IF        :fprintf(fp2,"IF");shift(51);break;
case WHILE     :fprintf(fp2,"WHILE");shift(53);break;
case REPEAT    :fprintf(fp2,"REPEAT");shift(54);break;
case SELECT    :fprintf(fp2,"SELECT");shift(55);break;
case BID       :fprintf(fp2,"%s",curtoken.name);shift(56);break;
case IID       :fprintf(fp2,"%s",curtoken.name);shift(57);break;
case SID       :fprintf(fp2,"%s",curtoken.name);
                shift(58);break;
case BUD       :fprintf(fp2,"%s",curtoken.name);
                shift(60);break;
case TID       :/*fprintf(fp2,"%s",curtoken.name);*/
                shift(59);break;
case OSN       :fprintf(fp2,"%s",curtoken.name);shift(47);break;
case SSN       :fprintf(fp2,"%s",curtoken.name);shift(31);break;
case SUD       :fprintf(fp2,"%s",curtoken.name);shift(37);break;
case SUBEND    :if(stack[stacktop].state==19){fprintf(fp2,"%s",curtoken.name);
                shift(62);break;}
                else {error=TRUE;break;}
case UNTIL     :if(stack[stacktop].state==76){fprintf(fp2,"%s",curtoken.name);
                shift(122);break;}
                else { error=TRUE;break;}
case ENDSELECT :if(stack[stacktop].state==185){fprintf(fp2,"%s",curtoken.name);
                shift(228);break;}
                else { error=TRUE;break;}
case ENDIF     :if(stack[stacktop].state==248){fprintf(fp2,"%s",curtoken.name);
                shift(267);break;}
                if(stack[stacktop].state==278){fprintf(fp2,"%s",curtoken.name);
                shift(286);break;}
                else { error=TRUE;break;}
case ELSE      :if(stack[stacktop].state==248){fprintf(fp2,"%s",curtoken.name);
                shift(268);break;}
                else { error=TRUE;break;}
case ENDWHILE  :if(stack[stacktop].state==254){fprintf(fp2,"%s",curtoken.name);
                shift(273);break;}
                else { error=TRUE;break;}
case ENDWHEN   :if(stack[stacktop].state==274){fprintf(fp2,"%s",curtoken.name);
                shift(283);break;}
                else { error=TRUE;break;}

default:
    error = TRUE;
    break;
}
break;
case 18 :
    if (tokentype == ID)
        {make_id();
        shift(61);
        }
    else
        error = TRUE;
break;
case 21 :
    if (tokentype == STOP) {
        fprintf(fp2,"%s",curtoken.name);
        shift(64);}

```

```

    else
        error = TRUE;
    break;
case 43 :
case 44 :
case 45 :
case 46 :
case 47 :
case 48 :
case 49 :
case 50 :
case 51 :
case 52 :
case 53 :
case 79 :
case 111:
case 112:
case 113:
case 114:
case 122:
case 136:
case 144:
case 145:
case 146:
case 157:
case 158:
case 159:

if (tokentype == LP)
{
    switch (stack[stacktop].state)
    {
        case 43 : fprintf(fp2,"(");shift(65); break;
        case 44 : fprintf(fp2,"(");shift(66); break;
        case 45 : fprintf(fp2,"(");shift(67); break;
        case 46 : fprintf(fp2,"(");shift(68); break;
        case 47 : fprintf(fp2,"(");shift(69); break;
        case 48 : shift(70); break;
        case 49 : shift(71); break;
        case 50 : fprintf(fp2,"(");shift(72); break;
        case 51 : fprintf(fp2,"(");shift(73); break;
        case 52 : fprintf(fp2,"(");shift(74); break;
        case 53 : fprintf(fp2,"(");shift(75); break;
        case 79 : fprintf(fp2,"(");shift(126); break;
        case 111: fprintf(fp2,"(");shift(177); break;
        case 112: fprintf(fp2,"(");shift(178); break;
        case 113: fprintf(fp2,"(");shift(179); break;
        case 114: fprintf(fp2,"(");shift(180); break;
        case 122: fprintf(fp2,"(");shift(184); break;
        case 136: fprintf(fp2,"(");shift(195); break;
        case 144:
            tok1=pop();
            push(&tok1);
            if(strcmp("CONCAT",tok1.name)!=0)
                fprintf(fp2,"(");
            shift(197); break;
    }
}

```

```

        case 145: fprintf(fp2,"");shift(198); break;
        case 146: fprintf(fp2,"");shift(199); break;
        case 157: fprintf(fp2,"");shift(205); break;
        case 158: fprintf(fp2,"");shift(206); break;
        case 159: fprintf(fp2,"");shift(207); break;
    }
}
else
    error = TRUE;
break;
case 88 :
case 90 :
case 93 :
case 94 :
case 117:
case 121:
case 208:
case 214:
case 215:
case 227:
case 232:
case 234:
case 235:
case 241:
case 242:
case 266:
case 269:
case 271:
case 272:
case 275:
case 287:
case 288:

if (tokentype == RP)
{
    switch (stack[stacktop].state)
    {
        case 88 : fprintf(fp2,"");shift(163); break;
        case 90 : fprintf(fp2,"");shift(165); break;
        case 93 : fprintf(fp2,"");shift(166); break;
        case 94 : fprintf(fp2,"");shift(167); break;
        case 117: fprintf(fp2,"");shift(181); break;
        case 121: fprintf(fp2,"");shift(183); break;
        case 208: fprintf(fp2,"");shift(244); break;
        case 214: fprintf(fp2,"");shift(246); break;
        case 215: fprintf(fp2,"");shift(247); break;
        case 227: fprintf(fp2,"");shift(255); break;
        case 232: fprintf(fp2,"");shift(257); break;
        case 234: fprintf(fp2,"");shift(259); break;
        case 235: fprintf(fp2,"");shift(260); break;
        case 241: fprintf(fp2,"");shift(263); break;
        case 242: fprintf(fp2,"");shift(264); break;
        case 266: fprintf(fp2,"");shift(277); break;
        case 269: fprintf(fp2,"");shift(279); break;
        case 271: fprintf(fp2,"");shift(281); break;
        case 272: fprintf(fp2,"");shift(282); break;
    }
}

```

```

        case 275: fprintf(fp2, " "); shift(284); break;
        case 287: fprintf(fp2, " "); shift(289); break;
        case 288: fprintf(fp2, ")); shift(290); break;
    }
}
else
    error = TRUE;
break;
case 55 :
    if (tokentype == WHEN) { fprintf(fp2, "\nelse if");
        shift(79); }
    else
        error = TRUE;
    break;
case 56 :
    if (tokentype == EQUAL) { fprintf(fp2, "=");
        shift(80); }
    else
        error = TRUE;
    break;
case 57 :
    if (tokentype == EQUAL) { fprintf(fp2, "=");
        shift(81); }
    else
        error = TRUE;
    break;
case 58 :
    if (tokentype == EQUAL)
        shift(82);
    else
        error = TRUE;
    break;
case 59 :
    if (tokentype == LB)
        shift(83);
    else
        error = TRUE;
    break;
case 60 :
    if (tokentype == EQUAL) { /* fprintf(fp2, "="); */
        shift(84); }
    else
        error = TRUE;
    break;
case 65 :
    if (tokentype == GID) { fprintf(fp2, "%s", curtoken.name);
        shift(85); }
    else
        error = TRUE;
    break;
case 66 :
    if (tokentype == GID) { fprintf(fp2, "%s", curtoken.name);
        shift(86); }
    else
        error = TRUE;
    break;

```

```

case 67 :
if (tokentype == TID){fprintf(fp2,"%s",curtoken.name);
shift(87);}
else
error = TRUE;
break;
case 68 :
if (tokentype == TID){fprintf(fp2,"%s",curtoken.name);
shift(88);}
else
error = TRUE;
break;
case 69 :
if (tokentype == OID){fprintf(fp2,"%s",curtoken.name);
shift(89);}
else
error = TRUE;
break;
case 202:
case 199:
case 197:
case 195:
case 178:
case 177:
case 176:
case 160:
case 71 :
case 70 :
if (tokentype == SID){fprintf(fp2,"%s",curtoken.name);
shift(92);}
else
if (tokentype == SC){
fprintf(fp2,"%c%c",ch,curtoken.name,ch);
shift(91);}
else
error = TRUE;
break;
case 72 :
if (tokentype == SC){strlwr(curtoken.name);
fprintf(fp2,"%c%c",ch,curtoken.name,ch);
shift(94);}
else
error = TRUE;
break;
case 169:
case 126:
case 106:
case 97:
case 80:
case 73:
switch(tokentype)
(
case NOT      :if(stack[stacktop].state==97){error=TRUE;break;}
               else{fprintf(fp2,"!");shift(97);break;}
case LOAD     :fprintf(fp2,"LOAD");shift(111);break;
case SAVE     :fprintf(fp2,"SAVE");shift(112);break;

```

```

case LP      :fprintf(fp2,"(");shift(106);break;
case BID     :fprintf(fp2,"%s",curtoken.name);shift(107);break;
case IID     :fprintf(fp2,"%s",curtoken.name);shift(116);break;
case SID     :fprintf(fp2,"!strcmp(%s",curtoken.name);
               shift(110);break;
case BC      :fprintf(fp2,"%s",curtoken.name);shift(108);break;
case TFN     :fprintf(fp2,"%s",curtoken.name);shift(113);break;
case GFN     :fprintf(fp2,"%s",curtoken.name);shift(114);break;
case IC      :fprintf(fp2,"%s",curtoken.name);shift(115);break;

default:
    error = TRUE;
    break;
}
break;

case 204:

if (tokentype == BUD){fprintf(fp2,"%&s",curtoken.name);
  shift(118);}
else
if (tokentype == TID){fprintf(fp2,"%&s.",curtoken.name);
  shift(120);}
else
    error = TRUE;
break;
case 250:
case 249:
case 207:
case 206:
case 74 :
if (tokentype == BUD){fprintf(fp2,"%&s",curtoken.name);
  shift(118);}
else
if (tokentype == TID){fprintf(fp2,"%&s.",curtoken.name);
  shift(120);}
else
    error = TRUE;
break;
case 184:
case 75 :
if (tokentype == BID){fprintf(fp2,"%s",curtoken.name);
  shift(107);}
else
if (tokentype == BC){fprintf(fp2,"%s",curtoken.name);
  shift(108);}
else
    error = TRUE;
break;
case 77 :
if (tokentype == OTHERWISE){fprintf(fp2," else { ");
  shift(124);}
else
if (tokentype == ENDSELECT)
  shift(123);
else

```

```

if (tokentype == WHEN) {fprintf(fp2," else if ");
shift(79);}
else
error = TRUE;
break;
case 187:
case 135:
case 130:
case 81:
switch(tokentype)
{
case LP      :fprintf(fp2,"(");shift(135);break;
case MINUS   :if(stack[stacktop].state==130){error=TRUE;break;}
               else{fprintf(fp2,"-");shift(130);break;}
case IID     :fprintf(fp2,"%s",curtoken.name);shift(116);break;
case TID     :fprintf(fp2,"%s.",curtoken.name);shift(137);break;
case IC      :fprintf(fp2,"%s",curtoken.name);shift(115);break;
case IFN     :fprintf(fp2,"%s",curtoken.name);shift(136);break;

default:
error = TRUE;
break;
}
break;
case 82:
switch(tokentype)
{
case ASCII   :fprintf(fp2,"ASCII");shift(145);break;
case SID     :fprintf(fp2,"%s",curtoken.name);shift(92);break;
case TID     :fprintf(fp2,"%s.",curtoken.name);shift(147);break;
case SC      :fprintf(fp2,"%c%c",ch,curtoken.name,ch);shift(91);break;
case SFN     :
if(strcmp("CONCAT",curtoken.name)==0)
{
tok1=pop();
tok2=pop();
push(&tok2);
push(&tok1);
fprintf(fp2,"%s{%s",curtoken.name,tok2.name);
}
else
fprintf(fp2,"%s",curtoken.name);
shift(144);break;
case TDFN    :fprintf(fp2,"%s",curtoken.name);shift(146);break;

default:
error = TRUE;
break;
}
break;
case 83 :
tok1=pop();
tok2=pop();
push(&tok2);
push(&tok1);
if (tokentype == TNA){

```

```

fprintf(fp2, "strcpy(%s.%s, ", tok2.name, curtoken.name);
    shift(149);}
    else
    if (tokentype == TFA){
fprintf(fp2, "%s.%s", tok2.name, curtoken.name);
    shift(150);}
    else
    if (tokentype == TBA){
fprintf(fp2, "BUFF_COPY(&%s.%s, ", tok2.name, curtoken.name);
    shift(151);}
    else
    error = TRUE;
    break;
case 84:
    switch(tokentype)
    {
        case ALLOC      :fprintf(fp2, "ALLOC");shift(157);break;
        case REVERSE    :fprintf(fp2, "REVERSE");shift(159);break;
        case BUD        :fprintf(fp2, "&%s", curtoken.name);shift(118);break;
        case TID        :fprintf(fp2, "&%s.", curtoken.name);shift(120);break;
        case PFN        :fprintf(fp2, "%s", curtoken.name);shift(158);break;

        default:
            error = TRUE;
            break;
    }
    break;
case 85 :
    if (tokentype == COMMA){fprintf(fp2, ",");
    shift(160);}
    else
    error = TRUE;
    break;
case 86 :
    if (tokentype == COMMA){fprintf(fp2, ",");
    shift(161);}
    else
    error = TRUE;
    break;
case 87 :
    if (tokentype == COMMA){fprintf(fp2, ",");
    shift(162);}
    else
    error = TRUE;
    break;
case 89 :
    if (tokentype == COMMA){fprintf(fp2, ",");
    shift(164);}
    else
    error = TRUE;
    break;
case 186:
case 174:
case 127:
case 95:
    switch(tokentype)

```

```

{
case AND      :fprintf(fp2," && ");shift(170);break;
case OR       :fprintf(fp2," || ");shift(171);break;
case IMP      :fprintf(fp2," &&! ");shift(172);break;
case RP       :if(stack[stacktop].state==127){reduce(58);break;}
               if(stack[stacktop].state==174){fprintf(fp2," ");
               shift(218);break;}
               if(stack[stacktop].state==186){fprintf(fp2," ");
               shift(229);break;}
               else{fprintf(fp2," ");shift(168);break;}

default:
    if(stack[stacktop].state==127)reduce(58);
    else
        error = TRUE;
        break;
}
break;
case 109 :
if (tokentype == RELOP){
switch(curtoken.value){
case LSS:fprintf(fp2," < "); break;
case LEQ:fprintf(fp2," <="); break;
case GTR:fprintf(fp2," > "); break;
case GEQ:fprintf(fp2," >="); break;
case EQU:fprintf(fp2," == "); break;
case NEQ:fprintf(fp2," != "); break;
}
shift(175);}
else
error = TRUE;
break;
case 110 :
if (tokentype == SRELOP)
shift(176);
else
error = TRUE;
break;
case 120 :
if (tokentype == LB)
shift(182);
else
error = TRUE;
break;
case 194:
case 128:
switch(tokentype)
{
case RP      :if(stack[stacktop].state==128){error=TRUE;break;}
               else{fprintf(fp2," ");shift(231);break;}
case PLUS     :fprintf(fp2," + ");shift(188);break;
case MINUS    :fprintf(fp2," - ");shift(189);break;
case STAR     :fprintf(fp2," * ");shift(190);break;
case DIV      :fprintf(fp2," / ");shift(191);break;
case MOD      :fprintf(fp2," % ");shift(192);break;

```

```

        default:
            reduce(59);
            break;
    }
    break;
case 137 :
    if (tokentype == LB)
        shift(196);
    else
        error = TRUE;
    break;
case 138:
    fprintf(fp2,"");
    reduce(60);
    break;
case 147 :
    if (tokentype == LB)
        shift(200);
    else
        error = TRUE;
    break;
case 148 :
    if (tokentype == RB)
        shift(201);
    else
        error = TRUE;
    break;

case 149 :
    if (tokentype == EQUAL)
        shift(202);
    else
        error = TRUE;
    break;
case 150 :
    if (tokentype == EQUAL) {fprintf(fp2,"=");
        shift(203);}
    else
        error = TRUE;
    break;
case 151 :
    if (tokentype == EQUAL) {fprintf(fp2,"=");/*
        shift(204);}
    else
        error = TRUE;
    break;
case 161:
    switch(tokentype)
    {
        case BID      :fprintf(fp2,"%s,1",curtoken.name);shift(212);break;
        case IID      :fprintf(fp2,"%s,2",curtoken.name);shift(210);break;
        case SID      :fprintf(fp2,"%s,3",curtoken.name);shift(211);break;
        case BUD      :fprintf(fp2,"%s,4",curtoken.name);shift(213);break;

        default:
            error = TRUE;

```

```

        break;
    }
    break;
case 162 :
    if (tokentype == TID){fprintf(fp2,"%s",curtoken.name);
        shift(214);}
    else
        error = TRUE;
    break;
case 285:
case 280:
case 265:
case 261:
case 245:
case 205:
case 203:
case 198:
case 175:
case 164:
    if (tokentype == IID){fprintf(fp2,"%s",curtoken.name);
        shift(116);}
    else
    if (tokentype == IC){fprintf(fp2,"%s",curtoken.name);
        shift(115);}
    else
        error = TRUE;
    break;
case 160 :
    if (tokentype == THEN){fprintf(fp2,"(\n");
        shift(216);}
    else
        error = TRUE;
    break;

case 179 :
    if (tokentype == OID){fprintf(fp2,"%s",curtoken.name);
        shift(223);}
    else
        error = TRUE;
    break;
case 180 :
    if (tokentype == OID){fprintf(fp2,"%s",curtoken.name);
        shift(224);}
    else
        error = TRUE;
    break;
case 182 :
    if (tokentype == TBA){fprintf(fp2,"%s",curtoken.name);
        shift(225);}
    else
        error = TRUE;
    break;
case 183 :
    if (tokentype == BEGIN){fprintf(fp2,"(\n");
        shift(226);}

```

```

    else
        error = TRUE;
    break;

case 196 :
    if (tokentype == TFA){fprintf(fp2,"%s",curtoken.name);
        shift(233);}
    else
        error = TRUE;
    break;
case 200 :
    if (tokentype == TNA){fprintf(fp2,"%s",curtoken.name);
        shift(237);}
    else
        error = TRUE;
    break;
case 209 :
    if (tokentype == COMMA){fprintf(fp2,",");
        shift(245);}
    else
        error = TRUE;
    break;
case 220:
    fprintf(fp2,"");
    reduce(84);
    break;
case 221 :
    if (tokentype == COMMA){fprintf(fp2,",");
        shift(249);}
    else
        error = TRUE;
    break;
case 222 :
    if (tokentype == COMMA){fprintf(fp2,",");
        shift(250);}
    else
        error = TRUE;
    break;
case 223 :
    if (tokentype == COMMA){fprintf(fp2,",");
        shift(251);}
    else
        error = TRUE;
    break;
case 224 :
    if (tokentype == COMMA){fprintf(fp2,",");
        shift(252);}
    else
        error = TRUE;
    break;
case 225 :
    if (tokentype == RB){
        tok1=pop();
        tok2=pop();
        tok3=pop();
        tok4=pop();

```

```

    tok5=pop();
    push(&tok5);
    push(&tok4);
    push(&tok3);
    push(&tok2);
    push(&tok1);
    if(tok5.type==BUD)
        fprintf(fp2,"");
    shift(253);}
    else
        error = TRUE;
    break;
case 229 :
    if (tokentype == DO){fprintf(fp2,"( ");
        shift(256);}
    else
        error = TRUE;
    break;
case 233 :
    if (tokentype == RB)
        shift(258);
    else
        error = TRUE;
    break;
case 236 :
    if (tokentype == COMMA){fprintf(fp2,",");
        shift(261);}
    else
        error = TRUE;
    break;
case 237 :
    if (tokentype == RB)
        shift(262);
    else
        error = TRUE;
    break;
case 238:
    fprintf(fp2,"");
    reduce(63);
    break;
case 240:
    fprintf(fp2,"");
    reduce(65);
    break;
case 243 :
    if (tokentype == COMMA){fprintf(fp2,",");
        shift(265);}
    else
        error = TRUE;
    break;
case 251 :
    if (tokentype == TID){fprintf(fp2,"%&s",curtoken.name);
        shift(271);}
    else
        error = TRUE;
    break;

```

```

case 252 :
    if (tokentype == GID){fprintf(fp2,"%s",curtoken.name);
        shift(272);}
    else
        error = TRUE;
    break;
case 270 :
    if (tokentype == COMMA){fprintf(fp2,",");
        shift(280);}
    else
        error = TRUE;
    break;
case 276 :
    if (tokentype == COMMA){fprintf(fp2,",");
        shift(285);}
    else
        error = TRUE;
    break;
case 291 :
    default:
        error = TRUE;
    break;
} /* switch */
}
}
else error=TRUE;
}
while ((!accepted) && (!error));
if (error)
{
    return(0);
}
return(1);
} /* parse */

main(argc,argv)
int argc;
char *argv[];
{

char p;
int i,l,j;
if(argc<3){
    printf("\n kullanim:");
    printf("\nMPWFL giris_dosya cikis_dosya [include_dosya][durum_test_dosya\n");
    exit(1);
}
if((fp1=fopen("ERRMSG.TXT","r+b"))==0){
    printf("\n ERRMSG.TXT dosyasinda HATA var\n");
    exit(1);
}
i=0;
j=0;
while((p=getc(fp1))!=EOF)
{

```

```

if(p==ch){errmsg[jl[i++]=0x00;
    j++;
    i=0;
}
else
    errmsg[jl[i++]=p;

}
fclose(fp1);
if((fp1=fopen(argv[1],"r+b"))==0){
    printf("\n giris dosyasinda HATA var\n");
    exit(1);
}
if((fp2=fopen(argv[2],"w"))==0){
    printf("\n cikis dosyasinda HATA var\n");
    exit(1);
}
if(argc==4){
    fp3_flag=TRUE;
    if((fp3=fopen(argv[3],"w"))==0){
        printf("\n include dosyasinda HATA var\n");
        exit(1);
    }
}
if(argc==5){
    fp4_flag=TRUE;
    if((fp4=fopen(argv[4],"w"))==0){
        printf("\n durum_test dosyasinda HATA var\n");
        exit(1);
    }
}
i=0;
while((p=getc(fp1))!=EOF)
{
    if(p==' '){comment=TRUE;p=' ';}
    if(p=='\n'){comment=FALSE;p=' ';}
    if(p==0x0d)p=EOLN;
    else
        if(iscntrl(p))p=' ';
    if(comment){
        if(p==EOLN){
            if(i>MAXINPUT-2){
                printf("\n giris dosyasi %d karakterden büyük var\n",MAXINPUT);
                exit(0);
            }
            code[i++]=p;
            code[i++]=' ';
        }
    }
    else
    {
        code[i]=toupper(p);
        i++;
        if(i==MAXINPUT){
            printf("\n giris dosyasi %d karakterden büyük var\n",MAXINPUT);
            exit(0);
        }
    }
}

```

```
    }  
    }  
}  
codefil=0;  
input=code;  
fclose(fp1);  
  
init_file();  
if(parse()) printf("\nislem tamam");  
else {  
    printf("\n %d inci satirda islem hatali, state=%d, token=%d",  
        line_num, stack[stacktop].state, tokentype);  
    j=errno_table[stack[stacktop].state];  
    printf("\nHATA=%s",errmsg[j]);  
}  
fclose(fp2);  
if(fp3_flag)fclose(fp3);  
if(fp4_flag)fclose(fp4);  
}/*main*/
```

EK B. UTIL.H PROGRAMININ DOKÜMANI

```

#include <string.h>
#include <math.h>
#include <ctype.h>
#include <errno.h>
#include <stdio.h>
#include <mem.h>
#include <alloc.h>
#include <bios.h>
#include <dir.h>
#define TRUE 1
#define FALSE 0
struct BUFFER{
char *POINTER;
unsigned int MAX;
unsigned int COUNT;
};
struct TASK{
char NAME[80];
int FIELD;
struct BUFFER POINTER;
};
struct GROUP{
char *POINTER;
unsigned int COUNT;
unsigned int TYPE;
struct GROUP *NEXT;
};
struct OWNER{
unsigned char OWN[16];
};
char _GEC[255]="";
struct BUFFER _GECBUF=(NULL,0,0);
struct BUFFER _GECBUF1=(NULL,0,0);

void BUFF_COPY(struct BUFFER *buf1, struct BUFFER *buf2)
{
if(buf2->POINTER==NULL){
return;
}
if(_GECBUF.POINTER!=NULL)
free(_GECBUF.POINTER);
_GECBUF.POINTER=(char *)malloc(buf2->MAX);
if(_GECBUF.POINTER==NULL)return;
memcpy(buf2->POINTER, _GECBUF.POINTER, buf2->COUNT);
if(buf1->POINTER!=NULL)
free(buf1->POINTER);
buf1->POINTER=(char *)malloc(buf2->MAX);

```

```

    if(buf1->POINTER==NULL)return;
    buf1->MAX=buf2->MAX;
    buf1->COUNT=buf2->COUNT;
    memcpy(_GECBUF.POINTER,buf1->POINTER,buf2->COUNT);
}

```

```

struct BUFFER *ALLOC(unsigned int say)
{
    if(_GECBUF1.POINTER!=NULL)
        free(_GECBUF1.POINTER);
    _GECBUF1.POINTER=(char*)malloc(say);
    _GECBUF1.MAX=say;
    _GECBUF1.COUNT=0;
    return(&_GECBUF1);
}

struct BUFFER *PACK(struct BUFFER *gir)
{
    unsigned int i,ii;
    char *ara;
    char *sakla;
    char bir,iki;
    if(_GECBUF1.POINTER!=NULL)
        free(_GECBUF1.POINTER);
    i=gir->MAX;
    _GECBUF1.POINTER=(char*)malloc(i);
    _GECBUF1.MAX=i;
    i=gir->COUNT/2;
    _GECBUF1.COUNT=i;
    sakla=_GECBUF1.POINTER;
    ara=gir->POINTER;
    for(ii=0;ii<i;ii++)
    {
        bir=*(ara++);
        bir=bir<<4;
        iki=*(ara++);
        *(sakla++)=bir|(iki&0x0f);
    }
    return(&_GECBUF1);
}

struct BUFFER *UNPACK(struct BUFFER *gir)
{
    unsigned int i,ii;
    char *ara;
    char *sakla;
    char bir,iki;
    if(_GECBUF1.POINTER!=NULL)
        free(_GECBUF1.POINTER);
    i=gir->MAX;
    _GECBUF1.POINTER=(char*)malloc(i);
    _GECBUF1.MAX=i;
    i=gir->COUNT*2;
    if(i>(gir->MAX))i=gir->MAX;
    _GECBUF1.COUNT=i;
    sakla=_GECBUF1.POINTER;
    ara=gir->POINTER;
    for(ii=0;ii<i/2;ii++)

```

```

{
    bir=*(ara++);
    iki=bir>>4;
    *(sakla++)=0x30|(iki&0x0f);
    *(sakla++)=0x30|(bir&0x0f);
}
return(&_GECBUF1);
}

struct BUFFER *REVERSE(struct BUFFER *gir, unsigned int sayi, unsigned int uzun)
{
    unsigned int i,j,ii;
    char *ara;
    char *sakla;
    char *tampon;
    char *tampon1;
    char bir,iki;
    if(_GECBUF1.POINTER!=NULL)
        free(_GECBUF1.POINTER);
    i=gir->MAX;
    _GECBUF1.POINTER=(char*)malloc(i);
    tampon=(char*)malloc(uzun+1);
    _GECBUF1.MAX=i;
    i=sayi;
    if((sayi*uzun)>gir->COUNT)i=gir->COUNT/uzun;
    _GECBUF1.COUNT=i*uzun;
    sakla=_GECBUF1.POINTER;
    ara=gir->POINTER;
    for(ii=0;ii<i;ii++)
    {
        tampon1=tampon;
        for(j=0;j<uzun;j++)tampon1++;
        for(j=0;j<uzun;j++)*(--tampon1)=*(ara++);
        tampon1=tampon;
        for(j=0;j<uzun;j++)*(sakla++)=*(tampon1++);
    }
    free(tampon);
    return(&_GECBUF1);
}

unsigned int LOAD(char *adi,struct BUFFER *buf1)
{
    FILE *fp;
    unsigned int i;
    char p;
    char *pp;
    if(buf1->POINTER==NULL){
        buf1->COUNT=0;
        return(FALSE);
    }

    pp=(char *)buf1->POINTER;
    if((fp=fopen(adi,"r+b"))==0){
        buf1->COUNT=0;
        return(FALSE);
    }
    i=0;
    while(((p=getc(fp))!=EOF)&&(i<buf1->MAX))

```

```

{
    *pp=p;
    pp++;
    i++;
}
buf1->COUNT=i;
fclose(fp);
return(TRUE);
}

unsigned int SAVE(char *adi,struct BUFFER *buf1, unsigned int say)
{
    FILE *fp;
    unsigned int i;
    char p;
    char *pp;
    if(buf1->POINTER==NULL){
        return(FALSE);
    }
    pp=(char *)buf1->POINTER;
    if((fp=fopen(adi,"w+b"))==0){
        return(FALSE);
    }
    i=1;
    while((p=putc(*pp,fp))!=EOF)&&(i<buf1->COUNT)&&(i<say))
    {
        pp++;
        i++;
    }
    p=p;
    fclose(fp);
    return(TRUE);
}

void FREE(struct BUFFER *buf1)
{
    if(buf1->POINTER!=NULL)
        free(buf1->POINTER);
    buf1->POINTER=NULL;
    buf1->MAX=0;
    buf1->COUNT=0;
}

char *ACCEPT(char *gir)
{
    printf("\n%s",gir);
    if(gets(_GEC)==NULL)_GEC[0]=0x00;
    else
        if(strlen(_GEC)>80)_GEC[80]=0x00;
        strupr(_GEC);
    return(_GEC);
}

char *ASCII(int gir)
{
    itoa(gir,_GEC,10);
    return(_GEC);
}

char *CONCAT(char *gir1, char *gir2)
{

```

```

int i;
char p;
i=0;
p=*gir1;
while((i<80)&(p!=0x00)){
    _GEC[i]=p;
    i++;
    gir1++;
    p=*gir1;
}
p=*gir2;
while((i<80)&(p!=0x00)){
    _GEC[i]=p;
    i++;
    gir2++;
    p=*gir2;
}
_GEC[i]=0x00;
return(_GEC);
}
char *TAKE(char *gir, int say)
{
    int i;
    char p;
    i=0;
    p=*gir;
    while((i<80)&(p!=0x00)&&(i<say)){
        _GEC[i]=p;
        i++;
        gir++;
        p=*gir;
    }
    _GEC[i]=0x00;
    return(_GEC);
}
char *DROP(char *gir, int say)
{
    int i;
    char p;
    i=0;
    p=*gir;
    while((i<80)&(p!=0x00)&&(i<say)){
        i++;
        gir++;
        p=*gir;
    }
    i=0;
    while((i<80)&(p!=0x00)){
        _GEC[i]=p;
        i++;
        gir++;
        p=*gir;
    }
    _GEC[i]=0x00;
    return(_GEC);
}

```

```

int DECIMAL (char * gir)
{
    return(atoi(gir));
}
int LENGTH (char * gir)
{
    return(strlen(gir));
}
int ISFILE (char * gir)
{
    struct ffbk ffbk;
    int stat;
    stat=findfirst(gir,&ffb,0);
    if(stat==-1)return(0);
    return(ffb.ff_fsize);
}
void GROUPNAME ( struct GROUP *gir, char *dizi)
{
    if(gir->POINTER!=NULL)return;
    gir->POINTER=(char *)malloc(strlen(dizi));
    gir->COUNT=strlen(dizi);
    strcpy(gir->POINTER,dizi);
}
void GROUPMAKE ( struct GROUP *gir, char *param,unsigned int tip, unsigned int say)
{
    struct GROUP *ara;
    struct GROUP *ekle;
    int i;
    if(gir->POINTER==NULL)return;
    ara=gir;
    while(ara->NEXT!=NULL)ara=ara->NEXT;
    ekle=(struct GROUP *)malloc(sizeof(struct GROUP));
    ekle->POINTER=param;
    if((tip==1)||tip==2){
        if(say>2)say=2;
    }
    if(tip==3){
        if(say>80)say=80;
    }
    if(tip==4){
        i=((struct BUFFER *)param)->MAX;
        if(say>i)say=i;
    }

    ekle->COUNT=say;
    ekle->TYPE=tip;
    ekle->NEXT=NULL;
    ara->NEXT=ekle;
}

void group_izle(struct GROUP *gir)
{
    int i,ii;
    struct GROUP *ara;
    char *dizi;
    i=0;

```

```

printf("\n group ize");
printf("\n %d inci eleman= ",i);
ara=gir;
while(ara->POINTER!=NULL){
dizi=ara->POINTER;
for(ii=0;ii<ara->COUNT;ii++){
printf("%x.",*dizi);
dizi++;
}
if(ara->NEXT==NULL){
printf("\nnext==null");
return;
}

```

```

i++;
printf("\n %d inci eleman= ",i);
ara=ara->NEXT;
}
}

```

```

unsigned int group_say(struct GROUP *gir)
{
unsigned int i,ii;
struct GROUP *ara;
char *dizi;
i=0;
ii=0;
ara=gir;
while(ara->POINTER!=NULL){
ii=ii+ara->COUNT;
i++;
if(ara->NEXT==NULL){
i=(i*2)+ii;
return(i);
}
ara=ara->NEXT;
}
i=(i*2)+ii;
return(i);
}

unsigned int group_bul(struct GROUP *gir)
{
unsigned int i,ii;
struct GROUP *ara;
char *dizi;
i=0;
ii=0;
ara=gir;
while(ara->POINTER!=NULL){
i++;
if(ara->NEXT==NULL){
return(i);
}
ara=ara->NEXT;
}
return(i);
}

```

```

}

void RESETOWNER( struct OWNER *gir, unsigned int say)
{
    unsigned int i,ii;
    unsigned char iii;
    if(say<127)
    for(i=0;i<16;i++) gir->OWN[i]=0x00;
    else
    {
    for(i=0;i<16;i++) gir->OWN[i]=0xff;
    return;
    }
    ii=say/8;
    for(i=0;i<ii;i++) gir->OWN[i]=0xff;
    ii=say%8;
    iii=0x80;
    for(ii=0;ii<(say%8);ii++){
    gir->OWN[ii]=gir->OWN[ii]&iii;
    iii=iii>>1;
    }
}

void ADDOWNER( struct OWNER *gir, unsigned int say)
{
    unsigned int i,ii;
    unsigned char iii;
    if(say>127)return;
    ii=say%8;
    iii=0x80;
    for(i=0;i<ii;i++)iii=iii>>1;
    i=say/8;
    gir->OWN[i]=gir->OWN[i]&iii;
}

void SUBOWNER( struct OWNER *gir, unsigned int say)
{
    unsigned int i,ii;
    unsigned char iii;
    if(say>127)return;
    ii=say%8;
    iii=0x80;
    for(i=0;i<ii;i++)iii=iii>>1;
    iii=~iii;
    i=say/8;
    gir->OWN[i]=gir->OWN[i]&iii;
}

void owner_izle( struct OWNER *gir)
{
    int i;
    printf("\n owner=");
    for(i=0;i<16;i++)printf("%x.",gir->OWN[i]);
}

unsigned int gonder(void)
{
    int i,j;
    char *bir;
    unsigned char abyte;

```

```

int stat;
unsigned char crc;
bir=_GECBUF.POINTER;

abyte=0xa0!0x00!0x03;
bioscom(0,abyte,0);
crc=bioscom(2,0,0);
crc=_GECBUF.COUNT;
stat=bioscom(3,0,0)&0x4000;

while(stat==0)stat=bioscom(3,0,0)&0x4000;
bioscom(1,crc,0);
abyte=_GECBUF.COUNT/256;
crc^=abyte;
stat=bioscom(3,0,0)&0x4000;

while(stat==0)stat=bioscom(3,0,0)&0x4000;
bioscom(1,abyte,0);
stat=bioscom(3,0,0)&0x4000;

while(stat==0)stat=bioscom(3,0,0)&0x4000;
bioscom(1,crc,0);
for(i=0;i<32000;i++)
    for(j=0;j<20;j++) stat=stat;
crc=0;
for(i=0;i<_GECBUF.COUNT;i++){
    stat=bioscom(3,0,0)&0x4000;

    while(stat==0)stat=bioscom(3,0,0)&0x4000;
    crc^=*bir;
    bioscom(1,*(bir++),0);
}
stat=bioscom(3,0,0)&0x4000;

while(stat==0)stat=bioscom(3,0,0)&0x4000;
bioscom(1,crc,0);
free(_GECBUF.POINTER);
stat=bioscom(3,0,0)&0x0100;
while(stat==0)stat=bioscom(3,0,0)&0x0100;
crc=bioscom(2,0,0);
if(crc=='0')
    return(TRUE);
return(FALSE);
}

unsigned int SENDTASK (struct OWNER *gir ,struct TASK *param)
{
    int i,ii;
    unsigned char *bir;
    unsigned char *uc;
    struct BUFFER *ara;
    ara=&(param->POINTER);
    _GECBUF.COUNT=24+strlen(param->NAME)+ara->COUNT;
    _GECBUF.POINTER=(char *)malloc(_GECBUF.COUNT);
    bir=_GECBUF.POINTER;
    *(bir++)='S';

```

```

*(bir++)='T';
for (i=0;i<16;i++)*(bir++)=gir->OWN[i];
ii=strlen(param->NAME);
*(bir++)=ii;
*(bir++)=ii>>8;
for(i=0;i<ii;i++)*(bir++)=param->NAME[i];
ii=param->FIELD;
*(bir++)=ii;
*(bir++)=ii>>8;
ii=ara->COUNT;
*(bir++)=ii;
*(bir++)=ii>>8;
uc=ara->POINTER;
for(i=0;i<ii;i++)*(bir++)=*(uc++);
i=gonder();
return(i);
}

unsigned int RUNTASK (struct OWNER *gir ,struct TASK *param)
{
int i,ii;
unsigned char *bir;
unsigned char *uc;
_GECBUF.COUNT=20+strlen(param->NAME);
_GECBUF.POINTER=(char *)malloc(_GECBUF.COUNT);
bir=_GECBUF.POINTER;
*(bir++)='R';
*(bir++)='T';
for (i=0;i<16;i++)*(bir++)=gir->OWN[i];
ii=strlen(param->NAME);
*(bir++)=ii;
*(bir++)=ii>>8;
for(i=0;i<ii;i++)*(bir++)=param->NAME[i];
i=gonder();
return(i);
}

unsigned int ABORTTASK (struct OWNER *gir ,struct TASK *param)
{
int i,ii;
unsigned char *bir;
unsigned char *uc;
_GECBUF.COUNT=20+strlen(param->NAME);
_GECBUF.POINTER=(char *)malloc(_GECBUF.COUNT);
bir=_GECBUF.POINTER;
*(bir++)='A';
*(bir++)='T';
for (i=0;i<16;i++)*(bir++)=gir->OWN[i];
ii=strlen(param->NAME);
*(bir++)=ii;
*(bir++)=ii>>8;
for(i=0;i<ii;i++)*(bir++)=param->NAME[i];
i=gonder();
return(i);
}

unsigned int KILLTASK (struct OWNER *gir ,struct TASK *param)
{
int i,ii;

```

```

unsigned char *bir;
unsigned char *uc;
_GECBUF.COUNT=20+strlen(param->NAME);
_GECBUF.POINTER=(char *)malloc(_GECBUF.COUNT);
bir=_GECBUF.POINTER;
*(bir++)='K';
*(bir++)='T';
for (i=0;i<16;i++)*(bir++)=gir->OWN[i];
ii=strlen(param->NAME);
*(bir++)=ii;
*(bir++)=ii>>8;
for(i=0;i<ii;i++)*(bir++)=param->NAME[i];
i=gonder();
return(i);
}

unsigned int GETTASK (struct OWNER *gir ,struct TASK *param)
{
int i,ii,stat,get_own;
unsigned char crc,crcl,get_char,maske;
unsigned char *bir;
unsigned char *uc;
get_own=256;
for(i=0;i<16;i++)
{
get_char=gir->OWN[i];
maske=0x80;
for(ii=0;ii<8;ii++){
if(get_char&maske){
get_own=(i*8)+ii;
break;
}
maske=maske>>1;
}
if(get_own!=256)break;
}
if(get_own==256)return(FALSE);
_GECBUF.COUNT=5+strlen(param->NAME);
_GECBUF.POINTER=(char *)malloc(_GECBUF.COUNT);
bir=_GECBUF.POINTER;
*(bir++)='G';
*(bir++)='T';
/*for (i=0;i<16;i++)*(bir++)=gir->OWN[i];*/
*(bir++)=get_own;
ii=strlen(param->NAME);
*(bir++)=ii;
*(bir++)=ii>>8;
for(i=0;i<ii;i++)*(bir++)=param->NAME[i];
i=gonder();
if(i==FALSE)
return(i);
stat=bioscom(3,0,0)&0x0100;
while(stat==0)stat=bioscom(3,0,0)&0x0100;
i=bioscom(2,0,0);
stat=bioscom(3,0,0)&0x0100;
while(stat==0)stat=bioscom(3,0,0)&0x0100;
ii=bioscom(2,0,0);

```

```

stat=bioscom(3,0,0)&0x0100;
while(stat==0)stat=bioscom(3,0,0)&0x0100;
crc=bioscom(2,0,0);
crc1=0;
crc1^=i;
crc1^=ii;
if(crc!=crc1)return(FALSE);
param->FIELD=(ii<<8)+(i&0x00ff);
return(TRUE);
}
void TASKINIT(struct TASK *param)
{
param->NAME[0]=0x00;
param->FIELD=0;
FREE(&(param->POINTER));
}
void TASKCOPY(struct TASK *param_d, struct TASK *param_s )
{
TASKINIT(param_d);
strcpy(param_d->NAME, param_s->NAME);
param_d->FIELD= param_s->FIELD;
BUFF_COPY(&(param_d->POINTER),&(param_s->POINTER));
}

unsigned int SENDGROUP (struct OWNER *gir ,struct GROUP *param)
{
int i,ii;
unsigned char *bir;
unsigned char *uc;
struct GROUP *ara;
i=group_say(param);
if(i==0)return(TRUE);
ara=param;
_GECBUF.COUNT=18+i;
_GECBUF.POINTER=(char *)malloc(_GECBUF.COUNT);
bir=_GECBUF.POINTER;
*(bir++)='S';
*(bir++)='G';
for (i=0;i<16;i++)*(bir++)=gir->OWN[i];
while(ara!=NULL){
ii=ara->COUNT;
if(ara->TYPE==4){
uc=ara->POINTER;
i=((struct BUFFER *)uc)->COUNT;
if(ii>i){
_GECBUF.COUNT=_GECBUF.COUNT-(ii-i);
ii=i;
}
}
*(bir++)=ii;
*(bir++)=ii>>8;
uc=ara->POINTER;
if(ara->TYPE==4)
uc=((struct BUFFER*)uc)->POINTER;

for(i=0;i<ii;i++)*(bir++)=*(uc++);
}

```

```

ara=ara->NEXT;
}
i=gonder();
return(i);
}
unsigned int GETGROUP (struct OWNER *gir ,struct GROUP *param)
{
int i,ii,stat,get_own;
unsigned char *bir;
unsigned char *uc;
struct GROUP *ara;
unsigned char crc,crc1,get_char,maske;
if(param->POINTER==NULL)return(TRUE);
get_own=256;
for(i=0;i<16;i++)
{
get_char=gir->OWN[i];
maske=0x80;
for(ii=0;ii<8;ii++){
if(get_char&maske){
get_own=(i*8)+ii;
break;
}
maske=maske>>1;
}
if(get_own!=256)break;
}
if(get_own==256)return(FALSE);
ara=param;
_GECBUF.COUNT=6+param->COUNT;
_GECBUF.POINTER=(char *)malloc(_GECBUF.COUNT);
bir=_GECBUF.POINTER;
*(bir++)='G';
*(bir++)='G';
/*for (i=0;i<16;i++)*(bir++)=gir->OWN[i];*/
*(bir++)=get_own;
*(bir++)=group_bul(param);
ii=ara->COUNT;
*(bir++)=ii;
*(bir++)=ii>>8;
uc=ara->POINTER;
for(i=0;i<ii;i++)*(bir++)=*(uc++);
i=gonder();
if(i==FALSE)
return(i);
stat=bioscom(3,0,0)&0x0100;
while(stat==0)stat=bioscom(3,0,0)&0x0100;
i=bioscom(2,0,0);
stat=bioscom(3,0,0)&0x0100;
while(stat==0)stat=bioscom(3,0,0)&0x0100;
ii=bioscom(2,0,0);
stat=bioscom(3,0,0)&0x0100;
while(stat==0)stat=bioscom(3,0,0)&0x0100;
crc=bioscom(2,0,0);
crc1=i;
crc1^=ii;

```

```

/* printf("\nrcrc=%x crc1=%x",crc,crc1);*/
if(crc!=crc1)return(FALSE);
_GECBUF.COUNT=(ii<<8)+(i&0x00ff);
_GECBUF.POINTER=(char *)malloc(_GECBUF.COUNT);
bir=_GECBUF.POINTER;
crc=0;
for(i=0;i<_GECBUF.COUNT;i++){
stat=bioscom(3,0,0)&0x0100;
while(stat==0)stat=bioscom(3,0,0)&0x0100;
*(bir)=bioscom(2,0,0);
/*printf("%.2x.",*(bir));*/
crc^=*(bir++);
}
stat=bioscom(3,0,0)&0x0100;
while(stat==0)stat=bioscom(3,0,0)&0x0100;
crc1=bioscom(2,0,0);

if(crc!=crc1){free(_GECBUF.POINTER);return(FALSE);}
bir=_GECBUF.POINTER;
while(ara->NEXT!=NULL){
ara=ara->NEXT;
uc=ara->POINTER;
stat=ara->COUNT;
if(ara->TYPE==4){
stat=((struct BUFFER*)uc)->MAX;
uc=((struct BUFFER*)uc)->POINTER;
}
i=*(bir++);
i=(i&0x00ff)+(*(bir++)<<8);
if((ara->TYPE==1)|| (ara->TYPE==2)){
if(i>2)stat=2;
else
stat=i;
}
if(ara->TYPE==3){
if(i>79)stat=79;
else
stat=i;
}
for(ii=0;ii<i;ii++)
{
if(ii<stat)*(uc++)=*(bir++);
else bir++;
}
if(ara->TYPE==4){
uc=ara->POINTER;
if(i>stat)i=stat;
((struct BUFFER *)uc)->COUNT=i;
}
if(ara->TYPE==3)*uc=0x00;
}
free(_GECBUF.POINTER);
return(TRUE);
}

```

EK C. ERRMSG.TXT DOSYASININ DOKÜMANI

TANIMLANMAMIŞ İŞLEM

"EOF bekleniyor
 "SUBBODY veya Degisken tanimi bekleniyor
 "Degisken bekleniyor
 "SUBBODY bekleniyor
 "SUBBEGIN bekleniyor
 "Altprogram Adi bekleniyor... SUBBODY ??
 "Virgul veya Noktali virgul bekleniyor DECS id_list ?
 "Komut bekleniyor... SUBBEGIN ??
 "Degisken adi bekleniyor
 "Komut veya SUBEND bekleniyor... SUBBEGIN komut(lar) ??
 "Noktali virgul bekleniyor
 "Parantez Ac bekleniyor... GROUPNAME ?
 "Parantez Ac bekleniyor... GROUPMAKE ?
 "Parantez Ac bekleniyor... TASKCOPY ?
 "Parantez Ac bekleniyor... TASKINIT ?
 "Parantez Ac bekleniyor... OWNER_ISLEM ?
 "Parantez Ac bekleniyor... DISPLAY ?
 "Parantez Ac bekleniyor... PLINE ?
 "Parantez Ac bekleniyor... PCONTROL ?
 "Parantez Ac bekleniyor... IF ?
 "Parantez Ac bekleniyor... FREE ?
 "Parantez Ac bekleniyor... WHILE ?
 "Komut bekleniyor... REPEAT ??
 "WHEN bekleniyor... SELECT ??
 "ESIT bekleniyor... boolean_id ?
 "ESIT bekleniyor... integer_id ?
 "ESIT bekleniyor... string_id ?
 "ESIT bekleniyor... buffer_id ?
 "Koseli parantez Ac bekleniyor... task_id ?
 "Group_id bekleniyor... GROUPNAME(??
 "Group_id bekleniyor... GROUPMAKE(??
 "Task_id bekleniyor... TASKCOPY(??
 "Task_id bekleniyor... TASKINIT(??
 "Owner_id bekleniyor... OWNER_ISLEM(??
 "String_id veya string_sabit bekleniyor... DISPLAY(??
 "String_id veya string_sabit bekleniyor... PLINE(??
 "String_sabit bekleniyor... PCONTROL(??
 "Boolean ifade bekleniyor... IF(??
 "Buffer_id veya task_id bekleniyor... FREE(??
 "Boolean_id veya boolean_sabit bekleniyor... WHILE(??
 "Komut bekleniyor
 "OTHERWISE veya ENDSELECT veya WHEN bekleniyor
 "Parantez Ac bekleniyor... WHEN ?
 "Boolean ifade bekleniyor... boolean_id= ??
 "Integer ifade bekleniyor... integer_id= ??

"String ifade bekleniyor... string_id= ??
 "Task alanı bekleniyor... task_id{ ??
 "Buffer ifade bekleniyor... buffer_id= ??
 "Virgül bekleniyor... GROUPNAME(group_id ?
 "Virgül bekleniyor... GROUPMAKE(group_id ?
 "Virgül bekleniyor... TASKCOPY(task_id ?
 "Parantez Kapa bekleniyor... TASKINIT(task_id ?
 "Virgül bekleniyor... OWNER_ISLEM(owner_id ?
 "Parantez Kapa bekleniyor... DISPLAY(string ?
 "Parantez Kapa bekleniyor... PLINE(string ?
 "Parantez Kapa bekleniyor... PCONTROL(string ?
 "Boolean operator veya parantez Kapa bekleniyor... IF(ifade ??
 "Boolean primary bekleniyor... NOT ??
 "Boolean ifade bekleniyor... (??
 "Karsilastirma operatoru bekleniyor... integer_id veya sabit ??
 "String karsilastirma operatoru bekleniyor... string_id ??
 "Parantez Ac bekleniyor... LOAD ?
 "Parantez Ac bekleniyor... SAVE ?
 "Parantez Ac bekleniyor... TASK_ISLEM ?
 "Parantez Ac bekleniyor... GROUP_ISLEM ?
 "Parantez Kapa bekleniyor... FREE(buffer_id ??
 "Koseli parantez Ac bekleniyor... task_id ?
 "Parantez Kapa bekleniyor... WHILE(boolean_ifade ??
 "Parantez Ac bekleniyor... REPEAT komut(lar) UNTIL ?
 "Komut bekleniyor... SELECT when_list OTHERWISE ??
 "Boolean ifade bekleniyor... WHEN(??
 "Lojik operator bekleniyor... boolean_ifade ?
 "Aritmetik operator bekleniyor integer_ifade ?
 "Integer primary bekleniyor... - ??
 "Integer ifade bekleniyor... (??
 "Parantez Ac bekleniyor... INTEGER_ISLEM ?
 "Koseli Parantez Ac bekleniyor... task_id ?
 "Parantez Ac bekleniyor... STRING_ISLEM ?
 "Parantez Ac bekleniyor... ASCII_ISLEM ?
 "Parantez Ac bekleniyor... TAKE_DROP_ISLEM ?
 "Koseli Parantez Ac bekleniyor... task_id ?
 "Koseli Parantez Kapa bekleniyor... task_id{task_attribute ?
 "Esit bekleniyor... TNA ?
 "Esit bekleniyor... TFA ?
 "Esit bekleniyor... TBA ?
 "Parantez Ac bekleniyor... ALLOC ?
 "Parantez Ac bekleniyor... PACK_UNPACK_ISLEM ?
 "Parantez Ac bekleniyor... REVERSE ?
 "String_id veya sabit bekleniyor... GROUPNAME(group_id, ??
 "ID (boolean,integer,string,buffer) bekleniyor... GROUPMAKE(group_id,??
 "Task_id bekleniyor... TASKCOPY(task_id, ??
 "Integer_id veya sabit bekleniyor... OWNER_ISLEM(owner_id, ??
 "THEN bekleniyor... IF(boolean_ifade) ??
 "Boolean ifade veya islem bekleniyor... boolean_ifade operator ??
 "Boolean operator veya parantez kapa bekleniyor... boolean_ifade ??
 "Integer_id veya sabit bekleniyor... integer_id karsilastirma_op ??
 "String_id veya sabit bekleniyor... string_id karsilastirma_op ??
 "String_id veya sabit bekleniyor... LOAD(??
 "String_id veya sabit bekleniyor... SAVE(??
 "Owner_id bekleniyor... TASK_ISLEM(??
 "Owner_id bekleniyor... GROUP_ISLEM(??

"TBA bekleniyor... task_idl ??
 "BEGIN bekleniyor... WHILE(ifade) ??
 "Boolean_id veya sabit bekleniyor... REPEAT komut(lar) UNTIL (??
 "Komut / ENDSELECT bekleniyor... komut(lar)?? veya OTHERWISE komut(lar) ??
 "Lojik operator... boolean_ifade ?? veya Parantez... WHEN(ifade ?
 "Integer term bekleniyor... integer_ifade operator ??
 "Integer operator... integer_ifade ?? veya parantez... (ifade ?
 "String_id veya sabit bekleniyor... INTEGER_ISLEM(??
 "TFA bekleniyor... task_idl ??
 "String_id veya sabit bekleniyor... STRING_ISLEM(??
 "Integer_id veya sabit bekleniyor... ASCII(??
 "String_id veya sabit bekleniyor... TAKE_DROP_ISLEM(??
 "TNA bekleniyor... task_idl ??
 "String_id veya sabit bekleniyor... TNA= ??
 "Integer_id veya sabit bekleniyor.. TFA= ??
 "Buffer_id veya task_id bekleniyor... TBA= ??
 "Integer_id veya sabit bekleniyor... ALLOC(??
 "Buffer_id veya task_id bekleniyor.. PACK_UNPACK_ISLEM(??
 "Buffer_id veya task_id bekleniyor.. REVERSE(??
 "Parantez Kapa bekleniyor... GROUPNAME(group_id,string ?
 "Virgul bekleniyor... GROUPMAKE(group_id,id ?
 "Parantez Kapa bekleniyor... TASKCOPY(task_id,task_id ?
 "Parantez Kapa bekleniyor... OWNER_ISLEM(owner_id,integer ?
 "Komut bekleniyor... THEN ??
 "Parantez Kapa bekleniyor... REPEAT komut(lar) UNTIL (boolean_ifade ?
 "DO bekleniyor... WHEN (boolean_ifade) ??
 "Degisken tanimi bekleniyor
 "Parantez Kapa bekleniyor... REVERSE(buffer,integer,integer ?
 "Virgul bekleniyor... LOAD(string ?
 "Virgul bekleniyor... SAVE(string ?
 "Virgul bekleniyor... TASK_ISLEM(owner_id ?
 "Virgul bekleniyor... GROUP_ISLEM(owner_id ?
 "Koseli parantez Kapa bekleniyor... task_id[TBA ?
 "Komut bekleniyor... BEGIN ??
 "Parantez Kapa bekleniyor... INTEGER_ISLEM (string ?
 "Koseli parantez Kapa bekleniyor... task_id[TFA ?
 "Parantez Kapa bekleniyor... STRING_ISLEM (string ?
 "Parantez Kapa bekleniyor... ASCII(integer ?
 "Virgul bekleniyor... TAKE_DROP_ISLEM(string ?
 "Koseli parantez Kapa bekleniyor... task_id[TNA ?
 "Parantez Kapa bekleniyor... ALLOC(integer ?
 "Parantez Kapa bekleniyor... PACK_UNPACK_ISLEM(buffer ?
 "Virgul bekleniyor... REVERSE(buffer ?
 "Integer_id veya sabit bekleniyor... GROUPMAKE(group_id,id ?)
 "Komut veya ENDIF/ELSE bekleniyor... komut(lar) ?? veya THEN komut(lar) ??
 "Buffer_id veya task_id bekleniyor... LOAD(string, ??
 "Buffer_id veya task_id bekleniyor... SAVE(string, ??
 "Task_id bekleniyor... TASK_ISLEM(owner_id, ??
 "Group_id bekleniyor... GROUP_ISLEM(owner_id, ??
 "Komut veya ENDWHILE bekleniyor... komut(lar) ?? veya BEGIN komut(lar)??
 "Komut bekleniyor... WHEN(boolean)DO ??
 "Integer_id veya sabit bekleniyor... TAKE_DROP_ISLEM(string, ??
 "Integer_id veya sabit bekleniyor... REVERSE(buffer, ??
 "Parantez Kapa bekleniyor... GROUPMAKE(group_id,id,integer ?
 "Komut bekleniyor... ELSE ??
 "Parantez Kapa bekleniyor... LOAD(string,buffer ?

```
"Virgul bekleniyor... SAVE(string,buffer ?  
"Parantez Kapa bekleniyor... TASK_ISLEM(owner_id,task_id ?  
"Parantez Kapa bekleniyor... GROUP_ISLEM(owner_id,group_id ?  
"Komut veya ENDWHEN bekleniyor... komut(lar)?? veya DO komut(lar)  
"Parantez Kapa bekleniyor... TAKE_DROP_ISLEM(string,integer ?  
"Virgul bekleniyor... REVERSE(buffer,integer ?  
"Komut veya ENDIF bekleniyor... komut(lar) ?? veya ELSE komut(lar) ??  
"Integer_id veya sabit bekleniyor... SAVE(string,buffer, ??  
"Integer_id veya sabit bekleniyor... REVERSE(buffer,integer, ??  
"Parantez Kapa bekleniyor... SAVE(string,buffer,integer ?  
"
```



EK D. TABLO.C PROGRAMININ DÜKÖMÖ

```
#include <string.h>
#include <math.h>
#include <ctype.h>
#include <errno.h>
#include <stdio.h>
```

```
#define PROG      1
#define PROGB     2
#define DECL      3
#define DEC       4
#define IDL       5
#define SPL       6
#define SPB       7
#define SPS       8
#define STATL     9
#define STATEM   10
#define STAT     11
#define GNS      12
#define GMS      13
#define SIL      14
#define TCS      15
#define TIS      16
#define OS       17
#define DS       18
#define PLS      19
#define PCS      20
#define IS       21
#define FS       22
#define WS       23
#define REP      24
#define SES      25
#define WSL      26
#define WST      27
#define ASS      28
#define BAS      29
#define IAS      30
#define SAS      31
#define BUAS     32
#define TAS      33
#define TAA      34
#define BEX      35
#define BOT      36
#define BOP      37
#define BPR      38
#define BSEX     39
#define ACOM     40
#define SCOM     41
```

```

#define LF      42
#define SF      43
#define TF      44
#define GF      45
#define IEX     46
#define TERM    47
#define IOP     48
#define IPR     49
#define ISP     50
#define IIF     51
#define TFAP    52
#define SEX     53
#define SSEX    54
#define SSF     55
#define AF      56
#define TNAP    57
#define TDF     58
#define BUEX    59
#define BUSEX   60
#define TBAP    61
#define ALF     62
#define PF      63
#define RF      64
#define EOFILE  65
#define STATE   66
#define REDUCE  67
#define IC      68
#define DOT     69
#define BAD     76
#define ERROR   76
#define IDSIZE  66
#define MAXINPUT 37000
#define TRUE 1
#define FALSE 0

```

```

struct TOKENREC
{
    int type;
    int value;
    char name[80];
};
struct IDREC
{
    int state;
    int gto;
};
struct TOKENREC curtoken;
char code[MAXINPUT];
struct IDREC id_list[IDSIZE][IDSIZE], curid;
int num_id[IDSIZE];
int table[300];
char ch=0x22;
char chb=0x5c;
char chc=0x25;
int statetop, tokentype, error, curtype;
int id_num=0;

```

```

char *input, isformula;
FILE *fp1,*fp2,*fp3,*fp4;
int fp3_flag=FALSE;
int fp4_flag=FALSE;
int isfunc(char *s)
{
    char *start=input;
    char name[80];
    int len = strlen(s);
    int i=0;
    while(!isdelim(*start))name[i++]=*start++;
    name[i]=0;
    if(strlen(name)!=len) return(FALSE);
    if (strcmp(s, input, len) == 0)
    {
        strncpy(curtoken.name, input, len);
        curtoken.name[len] = 0;
        input += len;
        return(TRUE);
    }
    return(FALSE);
} /* isfunc */

int isdelim(c)
char c;
{
    if(is_in(c," =,+* /()[];.$_'\"")||c==0x00||c==' ')
        return 1;
    return 0;
}

int is_in(char,s)
char cha,*s;
{
    while (*s) if(*s++==cha)
        return 1;
    return 0;
}

int nexttoken(void)
{
    char *start, numstring[80];
    int decimal, len, numlen,i;

    while (*input == ' ')
        input++;

    if (*input == 0)
    {
        input++;
        return(EOFILE);
    }

    if (is_in(*input,"0123456789"))
    {
        start = input;
        len = 0;
    }

```

```

while (isdigit(*input))
{
    input++;
    len++;
}

strncpy(curtoken.name, start, len);
curtoken.name[len]=0;
curtoken.value = atoi(start);
return(IC);
}

else if (isalpha(*input))
{
    if(isfunc("PROG")) return(PROG);
    if(isfunc("PROGB")) return(PROGB);
    if(isfunc("DECL")) return(DECL);
    if(isfunc("DEC")) return(DEC);
    if(isfunc("IDL")) return(IDL);
    if(isfunc("SPL")) return(SPL);
    if(isfunc("SPB")) return(SPB);
    if(isfunc("SPS")) return(SPS);
    if(isfunc("STATL")) return(STATL);
    if(isfunc("STATM")) return(STATM);
    if(isfunc("STAT")) return(STAT);
    if(isfunc("GNS")) return(GNS);
    if(isfunc("GMS")) return(GMS);
    if(isfunc("SIL")) return(SIL);
    if(isfunc("TCS")) return(TCS);
    if(isfunc("TIS")) return(TIS);
    if(isfunc("OS")) return(OS);
    if(isfunc("DS")) return(DS);
    if(isfunc("PLS")) return(PLS);
    if(isfunc("PCS")) return(PCS);
    if(isfunc("IS")) return(IS);
    if(isfunc("FS")) return(FS);
    if(isfunc("WS")) return(WS);
    if(isfunc("REP")) return(REP);
    if(isfunc("SES")) return(SES);
    if(isfunc("WSL")) return(WSL);
    if(isfunc("WST")) return(WST);
    if(isfunc("ASS")) return(ASS);
    if(isfunc("BAS")) return(BAS);
    if(isfunc("IAS")) return(IAS);
    if(isfunc("SAS")) return(SAS);
    if(isfunc("BUAS")) return(BUAS);
    if(isfunc("TAS")) return(TAS);
    if(isfunc("TAA")) return(TAA);
    if(isfunc("BEX")) return(BEX);
    if(isfunc("BOT")) return(BOT);
    if(isfunc("BOP")) return(BOP);
    if(isfunc("BPR")) return(BPR);
    if(isfunc("BSEX")) return(BSEX);
    if(isfunc("ACOM")) return(ACOM);
    if(isfunc("SCOM")) return(SCOM);
    if(isfunc("LF")) return(LF);

```

```

if(isfunc("SF")) return(SF);
if(isfunc("TF")) return(TF);
if(isfunc("GF")) return(GF);
if(isfunc("LEX")) return(LEX);
if(isfunc("TERM")) return(TERM);
if(isfunc("IOP")) return(IOP);
if(isfunc("IPR")) return(IPR);
if(isfunc("ISP")) return(ISP);
if(isfunc("IIF")) return(IIF);
if(isfunc("TFAP")) return(TFAP);
if(isfunc("SEX")) return(SEX);
if(isfunc("SSEX")) return(SSEX);
if(isfunc("SSF")) return(SSF);
if(isfunc("AF")) return(AF);
if(isfunc("TNAP")) return(TNAP);
if(isfunc("TDF")) return(TDF);
if(isfunc("BUEX")) return(BUEX);
if(isfunc("BUSEX")) return(BUSEX);
if(isfunc("TBAP")) return(TBAP);
if(isfunc("ALF")) return(ALF);
if(isfunc("PF")) return(PF);
if(isfunc("RF")) return(RF);
if(isfunc("STATE")) return(STATE);
if(isfunc("REDUCE")) return(REDUCE);
if(isfunc("ERROR")) return(ERROR);
i=0;
while(!isdelim(*input)){curtoken.name[i]=*input;
input++;
i++;
}
curtoken.name[i]=0;

return(BAD);
}
else {
curtoken.name[0]=*input;
curtoken.name[1]=0;
switch(*input++)
{

case '.' : return(DOT);
default : return(BAD);
} /* switch */
}
} /* nexttoken */
void print_token(void)
{
#ifdef TEST
if(curtoken.type>=BAD){
printf("\n%d inci satirda HATA=%d ",line_num,curtoken.type);
return;
}
if(curtoken.type<IC){
if(curtoken.type==EOLN)
printf("\n%d=EOLN",curtoken.type);
else

```

```

    if(curtoken.type==EOF)
printf("\nd=EOF",curtoken.type);
    else
printf("\nd=%s",curtoken.type,curtoken.name);
    } else {
    if(curtoken.type==LC){

printf("\nd=%d.%s",curtoken.type,curtoken.value,curtoken.name);
    } else

printf("\nd=%s",curtoken.type,curtoken.name);

    }
#endif
}

main(argc,argv)
int argc;
char *argv[];
{

char p;
unsigned int i,l;
unsigned int j=0;
int done=0;
if(argc<3){
printf("\n kullanim:\n LEX giris_dosyasi cikis_dosyasi [include_dosyasi]\n");
exit(1);
}
if((fp1=fopen(argv[1],"r+b"))==0){
printf("\n giris dosyasinda HATA var\n");
exit(1);
}
if((fp2=fopen(argv[2],"w"))==0){
printf("\n cikis dosyasinda HATA var\n");
exit(1);
}
if(argc>4){
fp3_flag=TRUE;
if((fp3=fopen(argv[3],"w"))==0){
printf("\n include dosyasinda HATA var\n");
exit(1);
}
}
if(argc==5){
fp4_flag=TRUE;
if((fp4=fopen(argv[4],"w"))==0){
printf("\n include1 dosyasinda HATA var\n");
exit(1);
}
}
for(i=0;i<300;i++)
table[i]=0;
for(i=0;i<IDSIZE;i++)
num_id[i]=0;
for(i=0;i<IDSIZE;i++)

```

```

{
    for(j=0;j<IDSIZE;j++)
        id_list[i][j].state=0;
        id_list[i][j].goto=0;
    }
    i=0;
    while((p=getc(fp1))!=EOF)
    {
        if(iscntrl(p))p=' ';
        codefil=toupper(p);
        i++;
        if(i>MAXINPUT){
            printf("\n giris dosyasi %d karakterden buyuk var\n",MAXINPUT);
            exit(0);
        }
    }

    codefil=0;
    fclose(fp1);
    input=code;
    curtoken.type=nexttoken();
    printf("\nbasla=%d=%s\n",curtoken.type,curtoken.name);
    for(;;){

        if(curtoken.type==EOFILE)break;
        if(curtoken.type==STATE){

            curtoken.type=nexttoken();
            statetop=curtoken.value;
            curtype=nexttoken();
            printf("\n state=%d, curtype=%d",statetop,curtype);
            if(fp4_flag)fprintf(fp4,"\nstate=%d",statetop);
            while(curtoken.type!=DOT)curtoken.type=nexttoken();
            done=1;
            while(done){
                curtoken.type=nexttoken();
                if((curtoken.type==STATE)
                ||(curtoken.type==EOFILE)){done=0;break;}
                else{
                    if(curtoken.type==REDUCE){
                        curtoken.type=nexttoken();
                        table[curtoken.value]=curtype;
                        if(curtoken.value==102)printf("\n!!!! %d",curtype);
                    }
                    else
                    if(curtoken.type<=RF){
                        tokentype=curtoken.type;
                        curtoken.type=nexttoken();
                        curtoken.type=nexttoken();

                        if(fp4_flag)fprintf(fp4,"\n token=%d , goto=%d",
                            tokentype,curtoken.value);
                        id_list[tokentype][num_id[tokentype]].state=statetop;
                        id_list[tokentype][num_id[tokentype]].goto=curtoken.value;
                        num_id[tokentype]++;
                    }
                }
            }
            /* else if(curtoken.type==ERROR)done=0;*/

```

```

/*else {curtoken.type=nexttoken();
printf("\ntoken1 %d=%s\n",curtoken.type,curtoken.name);
} */
}
}
else {curtoken.type=nexttoken();
printf("\ntoken %d=%x\n",curtoken.type,curtoken.name[0]);
}
}

```

```

for(i=0;i<15;i++){
fprintf(fp2,"\n");
for(j=0;j<20;j++)
fprintf(fp2,"%d,",table((i*20)+j));
}
fclose(fp2);
p=0x3b;
if(fp3_flag){
for(i=1;i<IDSIZE;i++)
{
fprintf(fp3,"\n default: return (291)%c }\ncase %d:\n switch(state){\n",p,i);
for(j=0;j<num_id[i];j++)
fprintf(fp3,"\n case %d: return(%d)%c",
id_list[i][j].state,id_list[i][j].gto,p);
}
fclose(fp3);}
if(fp4_flag) fclose(fp4);
}/*main*/

```

EK E. SIMU.C PROGRAMININ DOKÜMÜ

```

#include <string.h>
#include <math.h>
#include <ctype.h>
#include <errno.h>
#include <stdio.h>
#include <mem.h>
#include <alloc.h>
#include <bios.h>
#include <dir.h>
#define TRUE 1
#define FALSE 0
unsigned char gecbuf[32000];
unsigned int count,param;
/*
unsigned int gonder(void)
{
    int i;
    char *bir;
    unsigned char abyte;
    int stat;
    unsigned char crc;
    bir=gecbuf;

    abyte=0xa0!0x00!0x03;
    bioscom(0,abyte,0);
    crc=bioscom(2,0,0);
    crc=count;
    stat=bioscom(3,0,0)&0x4000;

    while(stat==0)stat=bioscom(3,0,0)&0x4000;
    bioscom(1,crc,0);
    abyte=count/256;
    crc^=abyte;
    stat=bioscom(3,0,0)&0x4000;

    while(stat==0)stat=bioscom(3,0,0)&0x4000;
    bioscom(1,abyte,0);
    stat=bioscom(3,0,0)&0x4000;

    while(stat==0)stat=bioscom(3,0,0)&0x4000;
    bioscom(1,crc,0);
    crc=0;
    for(i=0;i<count;i++){
        stat=bioscom(3,0,0)&0x4000;

        while(stat==0)stat=bioscom(3,0,0)&0x4000;
        crc^=*bir;

```

```

    bioscom(1,*(bir++),0);

}
stat=bioscom(3,0,0)&0x4000;

while(stat==0)stat=bioscom(3,0,0)&0x4000;
bioscom(1,crc,0);
free(gecbuf);
stat=bioscom(3,0,0)&0x0100;
while(stat==0)stat=bioscom(3,0,0)&0x0100;
crc=bioscom(2,0,0);
if(crc=='0')
return(TRUE);
return(FALSE);
} */

unsigned int paket_al(void)
{
    unsigned int i,j;
    char *bir;
    unsigned char abyte;
    int stat;
    unsigned char crc,crc1,kar1,kar2;
    unsigned int bekle;
    printf("\nPAKET ALMA (cikmak icin Q tusuna basin)");
    abyte=0xa0:0x00:0x03;
    bioscom(0,abyte,0);

    stat=bioscom(3,0,0)&0x0100;
    while(stat==0){
        if(kbhit())return(FALSE);
        stat=bioscom(3,0,0)&0x0100;
    }
    kar1=bioscom(2,0,0);

    stat=bioscom(3,0,0)&0x0100;
    while(stat==0)stat=bioscom(3,0,0)&0x0100;
    kar2=bioscom(2,0,0);

    stat=bioscom(3,0,0)&0x0100;
    while(stat==0)stat=bioscom(3,0,0)&0x0100;
    crc=bioscom(2,0,0);
    crc1=kar1;
    crc1^=kar2;
    if(crc1!=crc){
        printf("\n Hatali baslangic geldi\n iletisim sonu bekleniyor...");

    stat=bioscom(3,0,0)&0x0100;
    bekle=0xffff;
    while(bekle){
        stat=bioscom(3,0,0)&0x0100;
        if(stat==0){
            for(i=0;i<32000;i++)bekle=bekle;
            bekle--;

```

```

    }
    else{
        crc=bioscom(2,0,0);
        bekle=0xffff;
    }
}
crc='F';
stat=bioscom(3,0,0)&0x4000;
while(stat==0)stat=bioscom(3,0,0)&0x4000;
bioscom(1,crc,0);
printf("\n FALSE yaniti gonderildi");
return(FALSE);
}
count=256;
count=kar2*count+kar1;
printf("\n %d karakterlik paket aliniyor",count);
/*gecbuf=(char*)malloc(count);*/
bir=gecbuf;
crcl=0x00;
for(i=0;i<count;i++)
{
    stat=bioscom(3,0,0)&0x0100;
    while(stat==0)stat=bioscom(3,0,0)&0x0100;
    *(bir)=bioscom(2,0,0);
    /*printf("%x.",*(bir));*/
    crcl^=*(bir++);
}
/*printf("\ncrc bekleniyor");*/
stat=bioscom(3,0,0)&0x0100;
while(stat==0)stat=bioscom(3,0,0)&0x0100;
crc=bioscom(2,0,0);
if(crcl!=crc){
    printf("\n paket hatali geldi");
    crc='F';
    stat=bioscom(3,0,0)&0x4000;
    while(stat==0)stat=bioscom(3,0,0)&0x4000;
    bioscom(1,crc,0);
    printf("\n FALSE yaniti gonderildi");
    /*free(gecbuf);*/
    return(FALSE);
}
printf("\n paket hatasiz alindi , yanit gonderilmedi");
return(TRUE);
}

void paket_goster(int gir)
{
    unsigned int i,j;
    unsigned char kar1,kar2;
    unsigned char *bir;
    bir=gecbuf;
    kar1=*(bir++);
    kar2=*(bir++);
    if((gir==2)!(gir==3)){
        j=*(bir++);
        printf("%x",j);
    }
}

```

```

if(gir==3)bir++;
}
else
{

for(i=0;i<16;i++)printf("%x.",*(bir++));
}
kar1=*(bir++);
kar2=*(bir++);
j=256;
j=kar2*j+kar1;
printf("\nNAME=");
for(i=0;i<j;i++)printf("%c",*(bir++));
if(gir==1){
kar1=*(bir++);
kar2=*(bir++);
j=256;
j=kar2*j+kar1;
printf("\nTASKFIELD=%d",j);
kar1=*(bir++);
kar2=*(bir++);
j=256;
j=kar2*j+kar1;
printf("\nTASKBUFFER (%d karakter)",j);
printf("\n buffer dokumu icin tusa basin");
kar1=getch();
printf("\n buffer dokumu=");
for(i=0;i<j;i++)printf("%x.",*(bir++));
}
}

void paket_ac(void)
{
    unsigned int i,j,count1;
    unsigned char kar1,kar2;
    unsigned char *bir;
    bir=gecbuf;
    kar1=*(bir++);
    kar2=*(bir++);
    if((kar1=='S')&&(kar2=='T')){
        printf("\nSENDTASK ownerid=");
        paket_goster(1);
    }
    else
    if((kar1=='G')&&(kar2=='T')){
        printf("\nGETTASK owner no=");
        paket_goster(2);
    }
    else
    if((kar1=='R')&&(kar2=='T')){
        printf("\nRUNTASK ownerid=");
        paket_goster(0);
    }
    else
    if((kar1=='A')&&(kar2=='T')){
        printf("\nABORTTASK ownerid=");

```

```

paket_goster(0);
}
else
if((kar1=='K')&&(kar2=='T')){
printf("\nKILLTASK ownerid=");
paket_goster(0);
}
else
if((kar1=='S')&&(kar2=='G')){
printf("\nSENDGROUP ownerid=");
paket_goster(0);

for(i=0;i<16;i++)bir++;
kar1=*(bir++);
kar2=*(bir++);
j=256;
j=kar2*j+kar1;
for(i=0;i<j;i++)bir++;
count1=20+j;
param=1;
while(count1<count){
printf("\n%d inci parametre dokumu icin tusa basin",param);
kar1=getch();
kar1=*(bir++);
kar2=*(bir++);
j=256;
j=kar2*j+kar1;
printf("\nkarakter sayisi=%d dokum=",j);
for(i=0;i<j;i++)printf("%x.",*(bir++));
count1=count1+2+j;
param++;
}

}
else
if((kar1=='G')&&(kar2=='G')){
printf("\nGETGROUP owner no=");
paket_goster(3);
}
else
printf("\nTANIMSIZ paket geldi=%c%c\n",kar1,kar2);
}

main()
{
unsigned int i,j,ii,jj;
char *bir;
unsigned char abyte;
int stat;
unsigned char crc,crc1,kar1,kar2;
unsigned int bekle;
char giris[80];

abyte=0xa0:0x00:0x03;
bioscom(0,abyte,0);
kar1=bioscom(2,0,0);

```

```

printf("\nSIMULATOR HAZIR\n");
while(1){
    if(kbhit()){
        kar1=getch();
        if(kar1=='Q')exit(0);
    }
    if(paket_al()){
        paket_ac();
        printf("\n YANITINIZI GIRIN (O/F)=");
        kar1=getch();
        abyte=toupper(kar1);
        stat=bioscom(3,0,0)&0x4000;
        while(stat==0)stat=bioscom(3,0,0)&0x4000;
        bioscom(1,abyte,0);
        if(abyte=='F'){
            /*free(gecbuf);*/
        }
        else
        {
            bir=gecbuf;
            kar1=*(bir++);
            kar2=*(bir++);
            /*free(gecbuf);*/
            if((kar1=='G')&&(kar2=='T')){
                printf("\nGETTASK icin FIELD giriniz=");
                gets(giris);
                j=atoi(giris);
                crc=j;
                stat=bioscom(3,0,0)&0x4000;

                while(stat==0)stat=bioscom(3,0,0)&0x4000;
                bioscom(1,crc,0);
                abyte=j/256;
                crc^=abyte;
                stat=bioscom(3,0,0)&0x4000;

                while(stat==0)stat=bioscom(3,0,0)&0x4000;
                bioscom(1,abyte,0);
                stat=bioscom(3,0,0)&0x4000;

                while(stat==0)stat=bioscom(3,0,0)&0x4000;
                bioscom(1,crc,0);
            }
            if((kar1=='G')&&(kar2=='G')){
                kar1=*(bir++);
                param=*(bir++);
                param--;
                printf("\nGETGROUP icin %d adet parametre giriniz",param);
                /*gecbuf=(char*)malloc(32000);*/
                bir=gecbuf;
                count=0;
                for(i=1;i<=param;i++){
                    printf("\n %d inci parametre kac sekizli=",i);
                    gets(giris);
                    j=atoi(giris);

```

```

*(bir++)=j;
*(bir++)=(j/256);
count=count+2;
for(ii=1;ii<=j;ii++){
printf("\n%d inci parametrenin %d inci sekizlisini giriniz=",i,ii);
gets(giris);
jj=atoi(giris);
*(bir++)=jj;
count++;
}
} /*for*/

crc=count;
stat=bioscom(3,0,0)&0x4000;

while(stat==0)stat=bioscom(3,0,0)&0x4000;
bioscom(1,crc,0);
abyte=count/256;
crc^=abyte;
stat=bioscom(3,0,0)&0x4000;

while(stat==0)stat=bioscom(3,0,0)&0x4000;
bioscom(1,abyte,0);
stat=bioscom(3,0,0)&0x4000;
while(stat==0)stat=bioscom(3,0,0)&0x4000;
bioscom(1,crc,0);
for(i=0;i<3200;i++)
crc=0x00;
bir=gecbuf;
for(i=0;i<count;i++){
abyte=*(bir++);
crc^=abyte;
stat=bioscom(3,0,0)&0x4000;

while(stat==0)stat=bioscom(3,0,0)&0x4000;
bioscom(1,abyte,0);
}
stat=bioscom(3,0,0)&0x4000;
while(stat==0)stat=bioscom(3,0,0)&0x4000;
bioscom(1,crc,0);
}
}
}
}/*main*/

```

ÖZGEÇMİŞ

Ayşe Olcay Akgün 22 Ekim 1967'de İstanbul'da doğdu. Orta öğrenimini 1985 yılında Küçükçekmece Lisesinde tamamladı. 1989 yılında İ.T.Ü. Elektrik-Elektronik Fakültesi Kontrol ve Bilgisayar Mühendisliği Bölümünden mezun oldu. 1989 yılında İ.T.Ü. Bilgi İşlem Merkezi'nde Araştırma Görevlisi olarak çalışmaya başlamıştır. 1993 Ekim ayından itibaren TOBITAK Bilgisayar Ağları Ünitesinde tam zamanlı araştırmacı olarak çalışmaktadır. Ayşe Olcay Akgün evlidir ve bir kızı vardır.