BİR YAZILIM ÖLÇÜM SİSTEMİ NBv1.1 : C DİLİNDEKİ ÖĞRENCİ PROGRAMLARI ÖLÇÜMÜ TASARIM VE GERÇEKLEŞTİRİMİ

A SOFTWARE MEASUREMENT SYSTEM NBv1.1: DESIGN AND IMPLEMENTATION FOR MEASURING C LANGUAGE STUDENT PROGRAMS

ALPER YERTUTAN

YÜKSEK LİSANS TEZİ

2008

BİR YAZILIM ÖLÇÜM SİSTEMİ NBv1.1 : C DİLİNDEKİ ÖĞRENCİ PROGRAMLARI ÖLÇÜMÜ TASARIM VE GERÇEKLEŞTİRİMİ

A SOFTWARE MEASUREMENT SYSTEM NBv1.1: DESIGN AND IMPLEMENTATION FOR MEASURING C LANGUAGE STUDENT PROGRAMS

ALPER YERTUTAN

Başkent Üniversitesi
Lisansüstü Eğitim Öğretim ve Sınav Yönetmeliğinin
BİLGİSAYAR Mühendisliği Ana Bilim Dalı İçin Öngördüğü
YÜKSEK LİSANS TEZİ
olarak hazırlanmıştır.
2008

Fen Bilimleri Enstitüsü Müdürlüğü'ne,

Bu çalışma, jürimiz tarafından **BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI 'nda YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Başkan (Danışman) :

Prof. Dr. Ümit KARAKAŞ

Üye :

Doç. Dr. Serdar MÜLDÜR

Üye :

Yrd. Doç. Dr. Hasan OĞUL

ONAY

Bu tez 14/04/2008 tarihinde Enstitü Yönetim Kurulunca belirlenen yukarıdaki jüri üyeleri tarafından kabul edilmiştir.

...../...../2008

Prof.Dr. Emin AKATA

FEN BİLİMLERİ ENSTİTÜSÜ MÜDÜRÜ

TEŞEKKÜR EDERİM

Yazar, bu çalışmanın gerçekleşmesinde katkılarından dolayı, aşağıda adı geçen kişi ve kuruluşlara içtenlikle teşekkür eder.

Sayın Prof. Dr. Ümit KARAKAŞ 'a (Tez Danışmanı), çalışmanın sonuca ulaştırılmasında ve karşılaşılan güçlüklerin aşılmasında, ayırdığı değerli zamanın da her zaman yol gösterici olduğu için…

Bilgi İşlem Daire Başkanlığı'ndan Sayın Erdi ÇINAR 'a tezi hazırlamam da kullandığım sunucu ve sunucuda ki sorunları aşmamda ki yardımları için…

Araştırma Gör. Öykü Eren 'e tez çalışmamın kontrolünü yapmam için gerekli öğrenci programlarını sağladığı için...

Öğretim Gör. Muhammet Yorulmaz 'a öğrenci programlarının kontrolünü yapmam için gerekli öğrenci programlarını sağladığı için...

ÖZ

BİR YAZILIM ÖLÇÜM SİSTEMİ NBv1.1 :
C DİLİNDEKİ ÖĞRENCİ PROGRAMLARI ÖLÇÜMÜ TASARIM VE GERÇEKLEŞTİRİMİ

ALPER YERTUTAN
Başkent Üniversitesi Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Ana bilim Dalı

Bu çalışma, bilgisayar yazılımı öğreten kurumlara, bir destek aracı olarak geliştirilmiştir. Yazılım fiziği, profesyonel programlar için program boyu (programcının yazılım stilinden bağımsız olarak), yazılım emeği kestirimi gibi önemli ölçümleri oldukça duyarlı biçimde yapabilmektedir. Yine, yazılım fiziği parametrelerinin, öğrenci programları arasında yüksek oranda eşleşmesi, "değişken adı değiştirerek", yazılım fiziğinin, kopya tespit etme yöntemi olarak kullanılabileceğini göstermiştir. Bu çalışmada, Yazılım Fiziği (Software Physics) kuramı, derleyici gerçekleştiriminin önişleyici, tarayıcı ve kısmen sözdizim çözümleme kısımları gerçekleştirilerek C-1999 (ISO9899C programming language) standardındaki yazılımların ölçümü yapılmıştır. Geliştirme sırasında kısmen FLEX ve YACC yazılımları kullanılmış, bu çalışma için oluşturulan yazılıma entegre edilmiştir. Gerçekleştirilen tez yazılımı NB v.1.1, ikinci ve üçüncü sınıf bilgisayar öğrencilerinin yazmış olduğu C programları üzerinde sınanmıştır.

ANAHTAR SÖZCÜKLER: Yazılım Fiziği, derleyici gerçekleştirimi, yazılım ölçümü, FLEX, YACC

Danışman: Prof.Dr. Ümit KARAKAŞ, İstanbul Kültür Üniversitesi, Bilgisayar Mühendisliği Bölümü.

ABSTRACT

A SOFTWARE MEASUREMENT SYSTEM NBv1.1:
DESIGN AND IMPLEMENTATION FOR MEASURING C LANGUAGE STUDENT PROGRAMS

ALPER YERTUTAN
Başkent Üniversitesi Institute of Science
The Department of Computer Engineering

This thesis was developed as a supporting tool to teaching computer software institutes. Software physics can measure software parameters precisely for professional programs, like program length (independent from programmer' s programming style), programming effort, etc. The high correlation of software physics parameters of two students may be indication of cheating by changing variable names.

In this thesis, software physics theory is used. Preprocessor, scanner and syntax analysis parts are created with software physics measurement code. Student softwares that has C-1999 (ISO9899 C programming language) standarts is measured. For implemantation of the thesis, partly Flex and YACC programs are used and partly new program is written. Developed thesis software NB v.1.1 was checked with C programs that written by second and third class of computer engineering students.

KEY WORDS : Software Physics, compiler construction, software measurement, FLEX, YACC

Danışman: Prof.Dr. Ümit KARAKAŞ, İstanbul Kültür University, Computer Engineering section.

İÇİNDEKİLER LİSTESİ

ŞEKİLLER LİSTESİ

Sayfa

TABLOLAR LİSTESİ

Sayfa

1.GİRİŞ ve GENEL TANIMLAR

1.1 Çalışmanın Kapsamı ve Hedefi

Günümüzde bilgisayar programlama dilleri kullanılarak, pek çok yeni program oluşturulmaktadır. Gerek firma ortamında, gerekse akademik ortamda yazılımların ölçümü gereklidir. Bu ölçüm 2 temel amaca hizmet eder :

Bir; yazılmış olan programın çözdüğü soruna dayalı algoritma güçlüğü, boyutu ve emek kestirimi yapılabilmesi.

İki; ilk yazılmış programın asıl yapısını kullanan ama dışardan farklı bir program havası verilmiş bir kopya yazılım olup olmadığının kestirimidir. Yazılım ölçüm çalışmalarındaki en sağlam kuram(teori) Yazılım Fiziği (Software Physics) dir. [1]

Bu çalışmanın hedefi bilgisayar mühendisliği ikinci – üçüncü sınıf öğrencilerinin yazdığı / yazabildiği boyuttaki C dili (uluslararası standartlarda) [6] programlarının ölçümüdür. Bu ölçümlerin eşleşmesi ile kopya yakalama aşaması bu çalışmanın dışında tutulmuştur.

Bir programın kopya olup olmadığını yakalamak, o programda kullanılan program satır sayısının artmasıyla zorlaşmaktadır. Günümüzde kopya programları saptamak için çalışmalar yürütülmekte ve bir çoğu da geliştirilmektedir. Bu çalışmalardan biri 2005 yılında Hacettepe Üniversitesinde geliştirilmiştir. [5]

Bu çalışma çeşitli programlama dillerindeki Yazılım Mühendisliği parametrelerini ölçebilecek yetenekleri planlanan NB Yazılım Ölçüm Sistemi'nin 5 – 6 modülünden biri olup; üçüncü sınıf ve altındaki bilgisayar mühendisliği öğrencilerinin yazdığı programların, incelenmesinde kullanılacaktır. Öğrencilerin yazdıkları program kodlarının birimlere ayrılmasında yani tarayıcı (lexical analysis) kısmında Flex (fast lexical analyzer generator), kısmi söz çözümleme (partial syntax analysis) kısmında Yacc (Yet Another Compiler-Compiler) kullanılmıştır. Bu çalışma ile üçüncü sınıf ve altındaki bilgisayar mühendisliği öğrencilerinin yazdıkları programların, yazılım fiziği parametreleri açısından ölçülmesi hedeflenmiştir.

1.2 Yazılım Fiziği

1972 yılında, bilgisayar programlarının yazılıp yürütülmesinde ve bazı büyük miktarda teknik düz metinde, genel olarak uygulanabilen birçok sayısal ilişki görüldüğü saptanmıştır. Bu ilişkiler daha sonra bilimsel makalelerde, teknik kaynaklarda, konferanslarda ve bir çok doktora tezinde değişik başlıklar ile yayınlamıştır. Bu yeni tanımlanan ve geniş yer verilen öğreti ile ilgili kullanılan başlıklardan biri de teknik liderliğini Halstead' in yaptığı "yazılım fiziği" (Software Physics) olmuştur. [1a]

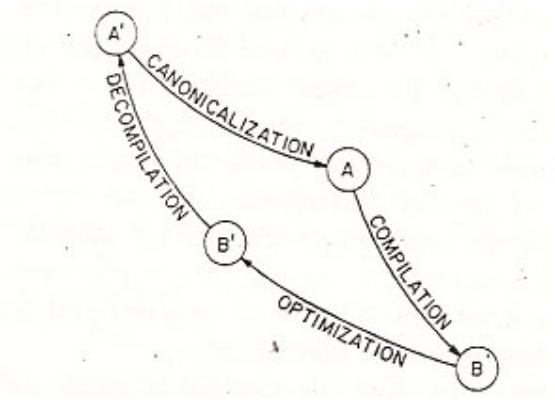
Yazılım fiziği, bilgisayar biliminin ve kısmen de yazılım mühendisliğinin yönlendirdiği fizik biliminin araştırmalarının bir dalı olarak bulunmuştur. Hem algoritmalar ve onların uygulanması ve bilgisayar programları hem de insanların iletişiminde kullanılan araçlar ile ilgilidir. Gelişmekte olan bu bilim dalı, sadece bir dilden başka bir dile çevrilirken, ölçülebilir özelliği olan algoritmalarda ya da programlarda kullanılmaktadır. Algoritmada veya bilgisayar programında kullanılan birimler sayesinde elimize ölçebileceğimiz veriler geçer. Yazılım fiziği de bu verilerin işlenmesi ile yazılım kalitesini arttırmak, yazılımı en etkili biçimde kullanmak veya kopya yazılımları ayırt etme gibi konulara yardımcı olacaktır.

Halstead [1b] ' in kitabına göre, bu işlemleri sadece bilgisayar programcıları değil, aynı zamanda hesaplamalı dilbilim (computational linguistics) araştırmacılarda kullanabilmektedir.

1.3 İdeal Yazılım Döngüsü

Halstead [1c], kitabında yazılım fiziği için gerekli parametrelerin bulunması konusunda, ideal yazılım döngüsüne değinmiştir. Konunun anlaşılabilmesi için bu noktaya da burada kısaca değinilmiştir. Prensipte bir algoritma tüm dillere uygulanabilmelidir. Yazılım döngüsü, program içinde kalite kaybı olmadan dört ayrı işleme bağlı kalarak şekil 2.1' de açıklamıştır. İlk aşamada, üst seviyedeki programlama dilindeki algoritma, alt seviyedekine (makine dili) çevrilir. İkinci aşamada alt seviyedeki programlama dilindeki algoritma en iyi kullanılacak hale getirilir. Üçüncü aşamada alt seviyedeki en iyi hale gelen programlama dilindeki algoritma, üst düzeye geri çevrilir. Çeşitli mühendislik dallarında geriye mühendislik (reverse engineering) vardır. Dördüncü aşamada algoritma döngülerin tekrardan yerleşmesi, prosedürlerin çağrılarını yapmasıyla ve en iyi halinden

geriye dönmesiyle, kuralları olan bir yapıya sonunda dönüşmektedir. Böylece bir algoritmanın uygulanabilirliği gösterilmektedir.



Şekil 1.1 İdeal Yazılım Döngüsü[1, s.4]

Kuramsal olarak, bu dört işlem sonunda algoritma her dönüşünde başlangıçtaki konumuna gelecektir. Aynı döngü farklı bir programlama dili ile yapılabilir. Bu döngüdeki A', B, B' noktalarında yapılabilecek ölçüm ve gözlemler, A noktasındaki algoritmanın ölçümü için ilişkiler (correlations) içermektedir.

Bu noktada şekil 1.1' nde önemli bir elemanın eksikliğinden bahsedilmektedir. XY koordinat düzlemi veya buna benzer sayım yapan bir düzlem. Çeşitli programlama dillerinin sayılabilir değişkenlere sahip olduğu göz önüne alınırsa, algoritmanın uygulanmasında bazı sayımların yapılabileceği, ayrıca makine dili (machine language- assembler) ve makine kodu (machine code) düzeyinde de sayım yapılmaktadır.

Algoritmanın her dilde uygulanmasında elde edilecek sayımlarda derleyicinin tanıdığı her bir birim (token) işleçler (operator) ve işlenenler (operand) şeklinde iki grup yapılmaktadır. İşleç (operator) ve işlenenlerin (operand) belirlenmesinden sonra herhangi bir algoritmanın herhangi bir çeşidinde ölçülebilir kümeler sunulabilmektedir.

1.4 Yazılım Fiziği Parametrelerinin Bulunmasında Dikkat Edilmesi Gereken Unsurlar

Halstead, bu teoriyi kitabında bilinen en eski algoritma (Euclid) örneği üzerinde açıklamıştır. Bu örnek ALGOL60 dilinde, belirtilen kitapta şöyle gösterilmiştir. [1d]

```
        IF (A=0)
LAST: BEGIN GCD := B; RETURN END;
        IF(B=0)
        BEGIN GCD:= A ; RETURN END;
HERE: G := A / B; R := A – B x G;
        IF (R=0) GO TO LAST;
        A := B; B := R; GO TO HERE
```

ALGOL60 örneği Clausen tarafından "hesaplayan makineler için birlikteliğin iletişimleri" (Communications of Association for Computing Machinery (CACM) [1d]) konferansında sunulmuştur.

Yukardaki örnekte:

1) "IF" bir eylemi gerçekleştiren olduğu için işleçtir(operator).
2) "(",")" parantezler eylemin ne olduğunu belirttiği için işleyendir (operator). parantez kapanmadan eylemin bitip bitmediği anlaşılamayacağından parantez çifti bir işleyen olarak sayılır.
3) "A" eylem içinde, eylemin uygulandığı tanıtıcı olduğundan işlenendir (operand).
4) "=" işareti IF eylemi içinde, eşitlik eylemi gerçekleştireceği için işleyendir. (operator)
5) "0" eşitlik eyleminde , eylem in kullandığı birim olduğundan işlenendir. (operand)
6) "LAST: " burada belirteçtir. Lakin "GO TO" eylemi tek başına işlevsizdir. "GO TO LAST" olarak bir eylem gerçekleştirir. Burada "LAST:" değil "GO TO LAST" eylem olduğundan işleyendir. (operator)

7) “HERE:” GO TO ile beraber kullanılacağı için “GO TO HERE:” işleyendir. (operator)
8) “BEGIN” eylemin başladığı “END” ise bittiği noktadır. Aynı parantez gibi “BEGIN...END” grubu tek bir eleman gibi sayılmaktadır ve işleyendir. (operator)
9) “GCD” eylem içinde eylemin uygulandığı değişken olduğundan işlenendir. (operand)
10) “:=” işareti atama eylemi gerçekleştirdiğinden işleyendir. (operator)
11) “;” işareti sonlandırma eylemi için kullanıldığından işleyendir. (operator)
12) “RETURN” çağırma eylemi gerçekleştirdiği için işleyendir. (operator)
13) “/”, “*”, “+”, “-“ işaretleri işlem eylemi gerçekleştirdikleri için işleyendir. (operator)
14) “G”, “B” ve “R” de aynı “A” gibi işlenendir. (operand)

Bu durumda yandaki programın işleç ve işlenen tabloları şöyle oluşmaktadır.

işleç	j	$f_{1,j}$
;	1	9
:=	2	6
( ),BEGIN..END	3	5
IF	4	3
=	5	3
/	6	1
*	7	1
-	8	1
GO TO HERE	9	1
GO TO LAST	10	1
	n1=10	N1= 31

Tablo 1.1 İşleç Tablosu

işlenen	i	$f_{2,i}$
B	1	6
A	2	5
0	3	3
R	4	3
G	5	2
GCD	6	2
	n2=6	N2=21

Tablo 1.2 İşlenen Tablosu

Bu örnekte; algol60 program yapısı üzerinde Euclid algoritmasının işleç (operator) ve işlenenlerinin (operand) nasıl bulunacağını kısmen göstermiş olduk.

1.5 Temel Yazılım Fiziği Parametreleri

Yukarıda da belirtildiği gibi algoritmadan ya da bilgisayar programından elde edilecek sayısal veri;

n_1 = uygulamada (programda) bulunan bağımsız veya eşsiz işleyenler (operator) sayısı.

n_2 = uygulamada (programda) bulunan bağımsız veya eşsiz işlenenler (operand) sayısı.

N_1 = uygulamada (programda) bulunan bağımsız veya eşsiz işleyenlerin (operator) toplam kullanım sayısı.

N_2 = uygulamada (programda) bulunan bağımsız veya eşsiz işlenenlerin (operand) toplam kullanım sayısı.

$f_{1,j}$ = uygulamada (programda) karşılaşılan işleyenin (operator) gözlenen sıklığı.

$f_{2,j}$ = uygulamada (programda) karşılaşılan işlenenin gözlenen sıklığı.

n = kelime dağarcığı. $n = n_1 + n_2$.

N = uygulama (program) uzunluğu. $N = N_1 + N_2$

Yukarıdaki parametreleri kullanarak , aşağıdaki şekil 1.2 elde edilmektedir.

$$N_1 = \sum_{j=1}^{j=\eta_1} f_{1,j}$$

$$N_2 = \sum_{j=1}^{j=\eta_2} f_{2,j}$$

$$N = \sum_{i=1}^{i=2} \sum_{j=1}^{j=\eta_i} f_{i,j}$$

Şekil 1.2 Temel Formüller

Yazılım fiziği, yukarıdaki sayısal verileri kullanarak, yazılım boyu kestirimi, yazılım emeği kestirimi ve yazılım amaç programı ile emek arasındaki ilişkileri de inceler.

1.6 Yazılım Boyu Kestirimi

Bir önceki başlık altında açıklanan temel yazılım fiziği parametrelerinden yararlanarak, yazılım boyu kestirimi yapılmaktadır. Bu bize daha karmaşık yapıları çözmemizde yardımcı olacaktır. “N”, yazılımın asıl uzunluğu olarak hesaplanmaktadır. “n”, yazılımda geçen eşsiz işleç ve işlenenlerin toplamı olarak hesaplanmaktadır. Farz edelimki “N” bir paragraf ve “n” de paragrafı oluşturan anlamlı cümle gruplarından biri olsun. “n” ‘lerin oluşturduğu grup, “N” yi ortaya çıkaracaktır. “n” lerin içeriği küçüldükçe, “N” yi oluşturan grubun eleman sayısı da artacaktır. “n” içeriği büyüdükçe, “N” yi oluşturan grubun eleman sayısı da azalacaktır. Buradan şu sonuca ulaşılmaktadır. “N” yazılım boyu için bir alt sınır (n<=N) “n” ler yardımı ile böyle belirlenmektedir. Üst sınır içinde “N” ve “n” yi aynı şekilde düşünelim. Bir “n”(kelime ya da kelime grubu) nin , “N”(paragraf) içinde kendini tekrar etmediğini varsayarsak, “N” / ”n” ile “n” lerin oluşturduğu ve “n” uzunlukta alt parçalara ulaşırız. “n” nin , “n” lerden oluşan kombinasyonu ise şöyledir.

n = 1	n =2	n=3		
A	AA	AAA	BAA	CAA
	AB	AAB	BAB	CAB
	BA	AAC	BAC	CAC
	BB	ABA	BBA	CBA
		ABB	BBB	CBB
		ABC	BBC	CBC
		ACA	BCA	CCA
		ACB	BCB	CCB
		ACC	BCC	CCC
$1 = 1^1$	$4 = 2^2$	$27 = 3^3$		

Tablo 1.3 Kombinasyon

Buradan üst limit için,

$$\text{“N”} <= \text{“n”}^{n+1} \tag{1.1}$$

diyebiliriz. Ama bu noktada önemli bir ayrıntı gelmektedir. İşleç (opreator) ve işlenen (operand). “n” ler içinde önce işleç sonra işlenen şeklinde oluşan yapıda

$$\text{“N”} <= \text{“n”} \times \text{“n1”}^{n1} \times \text{“n2”}^{n2}$$

formülüne ulaşıyoruz. “N” nin 2^N tane alt kümeye sahip olduğunu söyleyebiliriz.

$$2^N = n1^{n1} \times n2^{n2} \tag{1.2}$$

$$N = \log_2(n1^{n1} \times n2^{n2}) \tag{1.3}$$

$$N = \log_2 n1^{n1} + \log_2 n2^{n2} \tag{1.4}$$

Sonuç olarak;

$$N^{\wedge} = n1 \log_2 n1 + n2 \log_2 n2 \tag{1.5}$$

Daha önceki Euclid örneğindeki verileri kullanırsak.

n1 = 10 ,n2 = 6

$$N = N1 + N2 = 31 + 21 = 52$$

$$N^{\wedge} = 10 \log 210 + 6 \log 2\ 6 = 33 + 16 = 49$$

Örnek algoritma için, tahmin ettiğimiz uzunluk, programın hesaplanan uzunluk değerinin artı-eksi %10 luk sınırları içinde çıkmıştır. [1e]

1.7 Yazılım Emeği Kestirimi

Algoritmanın program içinde düzgün ve akıcı bir biçimde yerleştirilmesinde programı yazan kişinin programı oluştururken sarf ettiği emeği bulmak için daha önceden tanımladığımız yazılım fiziği parametrelerinden yararlanacağız.

Programcı için gerekli emek ile bu sayısal işlemler arasındaki ilişki 6 adımda gösterilmektedir.

Adım1) n adet elemanı olan kelime dağarcığından N tane seçimin yapılabildiği herhangi bir algoritma olduğunu farz edelim.

Adım2) Seçimlerin rasgele olmadığı ve sıralı olduğu farz edilerek, en uygun sıralama için ikili arama kullanıldığında her bir elemanın seçiminde $\log_2 n$ karşılaştırma yapılacaktır.

Adım3) Adım 1 ve 2 ye dayanarak N x $\log_2 n$ tane karşılaştırma olacaktır.

Adım4) Nlog2n aynı zamanda programın hacmini bize verecektir. Programın hacmi, bit değerinden programın büyüklüğüdür.[1f]

$$V = N\log 2n \qquad (1.6)$$

$$V = (N1 + N2) \log 2\ (n1 + n2) \qquad (1.7)$$

Hacim, program bir dilden başka bir dile değiştirilirken değişmektedir. Ama program sığası hep aynı kalmaktadır. Çünkü program sığası hesaplanması için temel parametreler kullanılmaktadır. İşleç için fonksiyon veya prosedür de

kullanılan işleç ve bir grubu sembol eden ya da işaret eden işleç; işlenen için de programa sadece girdi ve çıktı olarak gelen işlenenler, program sığası hesaplamasında kullanılır. İşleç (operator) ve işlenenler (operand) hem az sayıda hem de birbirinden farklı olduğundan, yeni "n" değerleri "N" değerleriyle aynı olur.

$$V^* = (n1^* + n2^*) \log2 (n1^* + n2^*) \quad (1.8)$$

Adım5) Her akla dayalı karşılaştırmanın , temel düzeyde akla dayalı çıkarsamalara ihtiyaç duyar. Bu çıkarsamaların miktarı programın zorluğunu(seviyesini) belirlemektedir. Adım 4'ten yararlanarak programın seviyesi hesaplanmaktadır.[1g]

$$L = V^*/ V \quad (1.9)$$

Eğer program sığası(V^*) değeri bilinmiyorsa, programın seviyesinin (L) hesaplanması için şöyle bir yöntem uygulanmaktadır. Algoritmadan aynısı olmayacak şekilde işleçler seçilir. Elimizdeki n1 değeri N1 ile aynı olur. Algoritmadaki tüm işlenenlerde seçilir.

$$L^\wedge = (n1^*/n1) \times (n2/N2) \quad (1.10)$$

Adım6) Diğer tüm adımlardan elde ettiğimiz veriler sonucunda akıldan yürütülen çıkarsamaların hesaplanması yani emeğin hesaplanması için aşağıdaki yöntem uygulanmaktadır.[1h]

$$E = V / L \quad (1.11)$$

$$L = V^* / V \quad (1.12)$$

$$E = V^2 / V^* \quad (1.13)$$

Daha önceki Euclid algoritma örneği üzerinde uyguladığımızda,

	Aldığı Değer	Kestirim Değeri
n1	10	/
n2	6	/
n	16	/
N1	31	/
N2	21	/
N	52	49
n1*	2	/
n2*	3	/
N1*	2	/
N2*	3	/
V	208	/
V*	11,6	/
L	0.0558	0.057
E		3.729.655

Tablo 1.4 Değerler

1.8 Program Zamanı

Temel düzeyde bilişsel ayırımlar (elementary mental discriminators) bir an içinde yapılan ayırımlardır. Öncelikle "an" (moments) açıklanmalıdır. John Stroud tarafından geliştirilen "an" insan beyninin temel işlemleri anlaması/algılaması için geçen zaman (saniye cinsiden) aralığıdır. Beynin veriyi alıp işleyip sonuçlandırmasında geçen süre bir an olarak ifade edilebilmektedir. Bu bulgu saniyede ($5 \leq S \leq 20$) temel düzeyde bilişsel ayırım, biçiminde de gösterilmektedir.[1j]

E= V / L eşitliğini kullanarak;

$$\check{T} = E / S = V / SL = V^2 / SV^* \qquad (1.14)$$

$$\check{T} = n1N2N\log 2n / 2Sn2 \qquad (1.15)$$

$$\check{T} = n1N2(n1\log_2 n1 + n2\log_2 n2)\log_2 n / 2Sn2 \qquad (1.16)$$

Eşitliğine ulaşılmaktadır. Bu eşitlik kullanılarak Communication ACM de tanımlanan on iki algoritma/program test edilmiştir. Yandaki tablo 1.4 de gösterilmektedir.

CACM No	Uygulama						Ortalama	
	birinci		ikinci		üçüncü			
	T	Ť	T	Ť	T	Ť	T	Ť
14	33	13	15	16	25	29	24	19
16	135	123	77	102	44	53	85	93
17	33	22	10	6	11	10	18	13
19	7	3	10	9	9	5	9	6
20	12	9	14	7	6	5	11	7
21	43	51	30	47	39	63	37	54
23	21	17	13	5	13	3	16	8
24	16	22	8	7	6	6	10	12
25	62	101	45	74	20	42	42	72
29	25	5	35	9	16	17	25	10
31	20	17	11	15	7	8	13	13
33	4	1	3	1	3	1	3	1
Toplamlar	411	384	271	298	199	242	293	309
Ortalama	34	32	23	25	17	20	24	26
Aralarındaki ilişki	0.92		0.92		0.94		0.94	

Tablo 1.5 Süre Sonuçları

Bu tabloda hesaplanan zaman stroud katsayısı 18 olarak alınmıştır. Buna göre elde edilen sonuçlarda hesaplanan zaman ile geçen gerçek zaman arasında anlamlı bir ilişki olduğunu söylenebilmektedir.

1.9 Programlama Dili Düzeyi

Verilen herhangi algoritma başka bir dile çevrilirken hacmi (V) büyürken program seviyesi düşmektedir. Herhangi bir algoritma için potansiyel hacim (V*) L x V çarpımına eşit olmaktadır. Eğer kullanılan dil sabit olup ta, algoritma çeşitlendirilir ise potansiyel hacim (V*) artarken program seviyesi (L) düşmektedir. L x V* da herhangi bir dil için sabit kalmaktadır. Buradan elde edilen ise şöyledir.

$$\lambda = LV^* \qquad (1.17)$$

Dil ilişkisinin ispatı için şöyle bir yöntem uygulanmıştır.

$$\lambda = L(L \times V) = L^2 V \qquad (1.18)$$

üstel 2 yerine bundan sonraki eşitliklerin daha rahat devam edebilmesi için "b" ifadesi kullanılmıştır.

$$\lambda = L^{b} V \quad (1.19)$$

$$\ln \lambda = \ln (L^{b} V) \quad (1.20)$$

$$\ln \lambda = b \ln L + \ln V \quad (1.21)$$

farz edilsin; $a = \ln \lambda$

$$X = \ln V$$

$$Y = -\ln L$$

Eşitlik bu hale dönüşmüştür, $X = a + b Y$ (1.22)

Bu da "b" yi X in Y' ye göre regresyonundaki eğim konumuna getirmiştir. Basit istatistiksel teknik ile b yi hesaplamak mümkün olmuştur.

Üstel b yi hesaplamak için şu yöntem uygulanmıştır.

$$b = \Sigma X Y - N \bar{Y} \bar{X} \; / \; \Sigma Y^2 - N \bar{Y}^2 \quad (1.23)$$

N verilen örnekteki sınıf aralığı.

Sonuçta elde edilen tablo aşağıdaki gibidir.

Language	sample	n	b
Algol 58	Halstead	12	2,62
Algol 58	Bulut	12	1,96
PL/I	Elshoff	34	2,57
PL/I(restr.)	Elshoff	120	1,68
Fortran	Bohrer	13	2,13
Fortran	Bulut	14	1,84
Compass	Bulut	7	3,40

Tablo 1.5 Sonuçlar

Tablo 1.5 e göre ağırlıklaştırılmış ortalama 1,987 olarak hesaplanmaktadır. Bu da "b" nin 2 ye yakın olması ile $\lambda = LV^*$ eşitliğinin doğruluğunu göstermektedir. Yazılım emeği kestirimi için de λ kullanılarak şöyle bir formül çıkarılabilir.

$$E = V^{*3} / \lambda^2 \qquad (1.24)$$

Bu formül, programlama emeğinin, temel bilişsel ayırımlar düzeyinde “en az program sığası V* in küpü” ile doğru orantılı, ama programlama dili düzeyinin karesi ile ters orantılı olduğunu belirtmektedir.[1k]

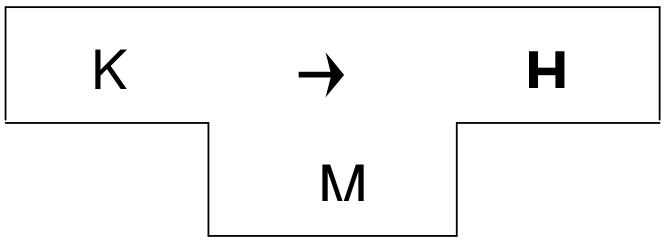
2. DERLEYİCİ

2.1 Derleyici Tanımı

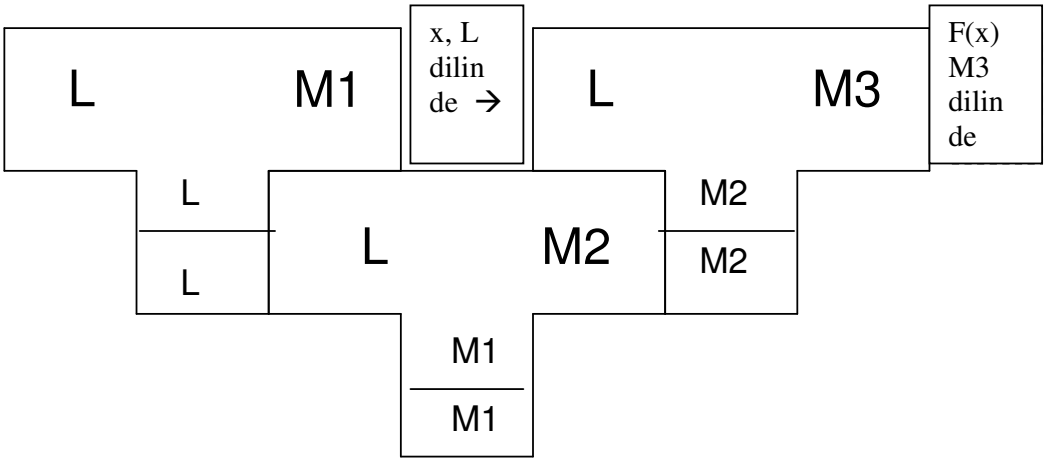
Bu tez çalışmasında bir derleyici geliştirilmemiş fakat derleyicinin önişleyici (pre-processor), tarayıcı (scanner) ve söz dizim çözümleme (syntax analysis) kısmı açık yazılım ortamından kazanılanlar ile sentezlenerek hazırlanmıştır. Bu tez çalışmasının gerçekleştirilmesi için derleyicinin anlamsal çözümleme (semantic analysis) kısmı gerekmemektedir. Bu nedenle bu bölümün ilerleyen sayfalarında bu kavrama değinilmemiştir. Bu nedenle ileriki kısımlarda anlatılacakların iyi anlaşılması için derleyici önceden özetlenmiştir.

Derleyici hazırlanan kaynak kodu, o an çalışmakta olduğu makinenin (bilgisayarın) anlayacağı hedef koda dönüştürmekte kullanılan programdır. Kaynak kod makinede(bilgisayarda) çalıştığında, yapılmak istenilen işlemler derleyicinin yardımıyla makinenin (bilgisayarın) anlayacağı makine dilinde çalışmış olur. Birden fazla programlama diline bağlı olarak bir çok derleyici vardır. Derleyici (compiler), temel bir yazılımdır. Bir çok makineye taşınması gerekmektedir. Taşınabilirlik (portability) ve yeniden kurulabilirlik (configuration management) özelliklerinin çok iyi ve anlaşılabilir olması gerekmektedir.

Bunu destekleyecek araçlardan biri T-çizelgeleri (T-diagrams) dır. T-çizelgeleri sol kenarında girdi dilini (input language / kaynak kodu), sağ ucunda çıktı dilini (output language / hedef kodu) ve alt tabanda yazılım dilini (writing language / makine kodu) içerir. T-çizelgeleri literatürde sıkça kullanılmıştır.[9, 10, 11a] Bazen, bir programlama dili derleyicisi, ilk modül olarak makine dili (assembler) 'de yazılan başlangıç birimi (örneğin C_0 derleyicisi) hariç olmak üzere gerçekleştireceği dilde adım adım (C_1, C_2, C_3, C_4 – full) yazılır. Bu tür kendi üzerinde oluşturma (bootstrap) çalışmalarında T-çizelgeleri [10, 11b] çok yararlıdır.



Şekil 2.1 T-Çizelge Modeli



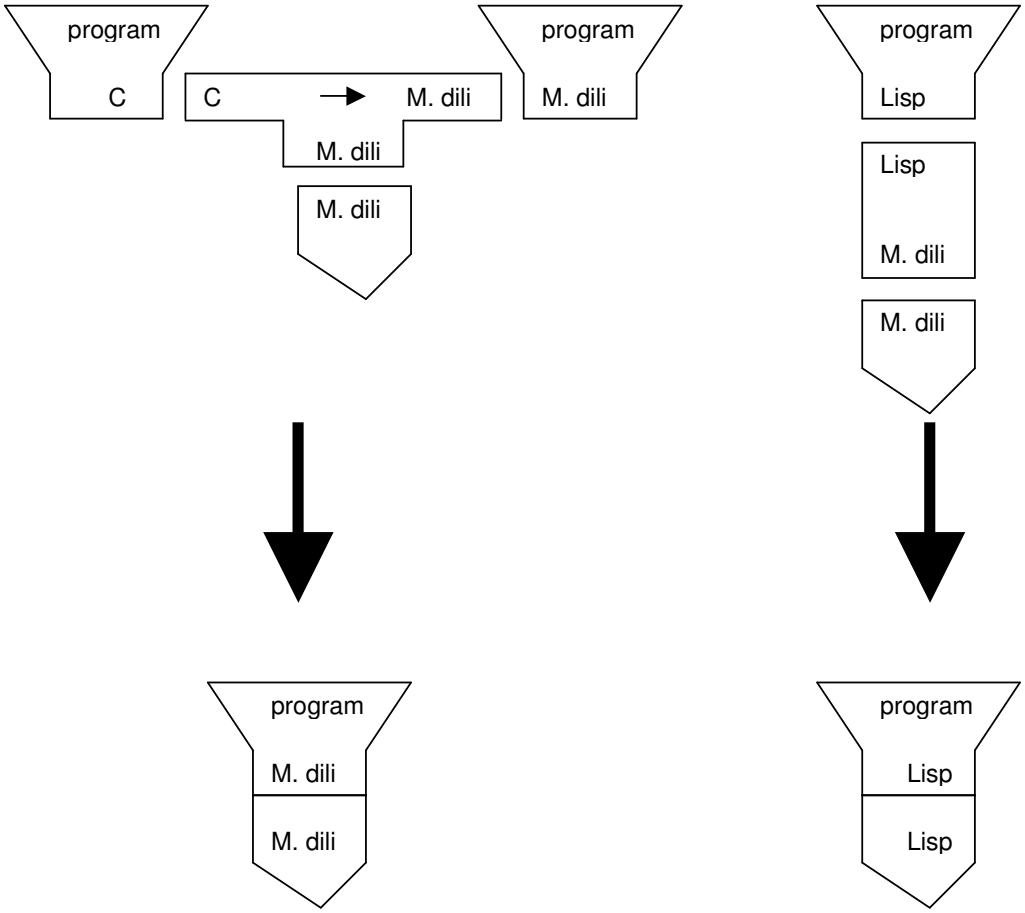
Şekil 2.2 Önyükleme (Bootstrap) Modeli

2.2 Derleyici ve Yorumlayıcı

T-çizelgesi düşey kolonu ile var olan bir yazım diline ve bu yazım dilini çalıştırabilecek bir assembler / bilgisayara ihtiyaç duyar. Bazen basit biçimde aşağıdaki gibi gösterilse de, asıl yapı sonraki şekilde olduğu gibidir.

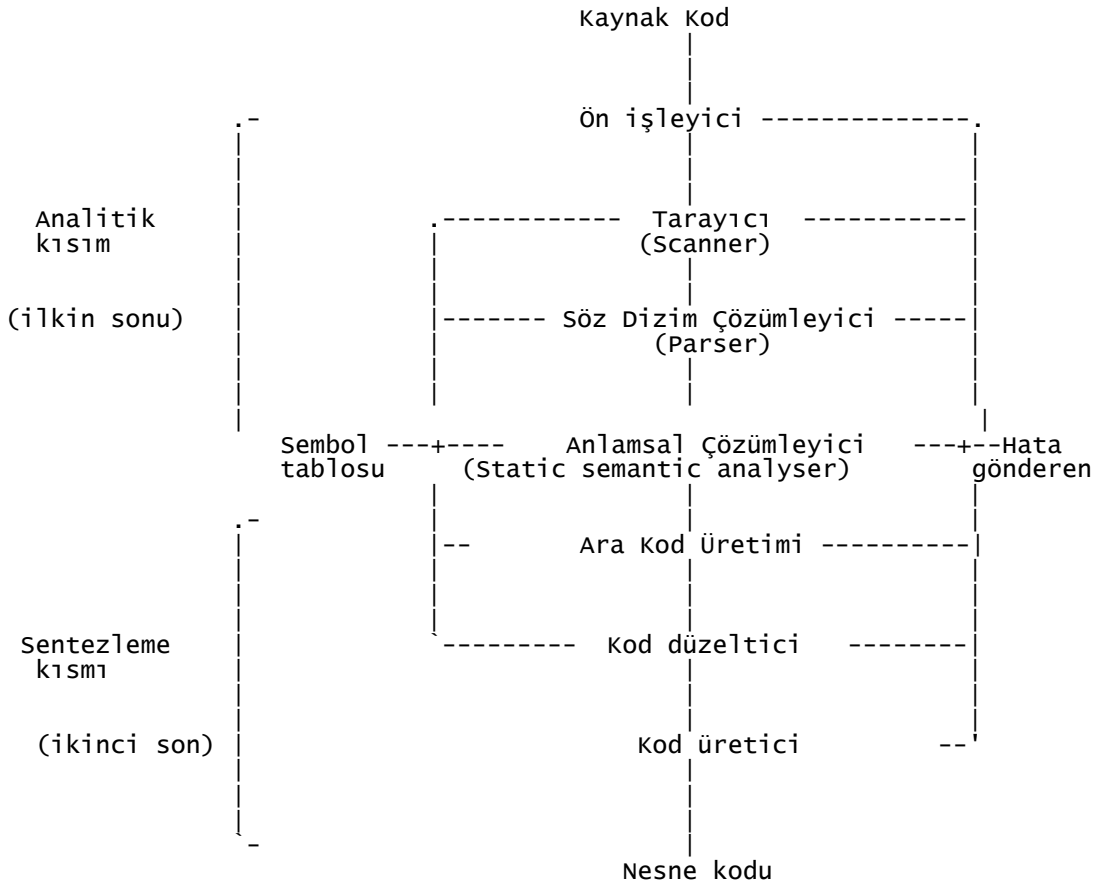
Şekil 2.3 T-diyagramları

Derleyici, kaynak (C dili) kodunu hedef (makine dili) koda dönüştürerek, programın makine dilinde çalışmasını sağlarken, yorumlayıcı kaynak kod u hedef koda dönüştürmeden ikisi arasında veri taşıması yapmaktadır. Aşağıdaki ikinci örnekde program, Lisp dili üzerinde çalışmaktadır.



Şekil 2.4 T-diyagramları [11]

Tezde T-çizelgeleri derleyicinin çalışması hakkında genel bir bilgi vermek için kullanılmıştır. Bundan sonra derleyicinin işlemesinde kullanılan aşamalarına bakmamız gerekmektedir. Derleyici temelde beş katmandan oluşmaktadır. Bu katmanlar sırasıyla ön işleyici, tarayıcı, söz dizim çözümleyici, anlamsal çözümleyici ve kod üretmedir.



Şekil 2.5 Derleyici Yapısı

Yukarıdaki şekil http://www.scifac.ru.ac.za/compilers/cha02g.htm kaynaktan sentezlenmiştir.

Günümüz derleyici gerçekleştirimi (compiler construction) derslerinin temel kaynağı AHO, Alfred V, SETHI, Ravi, ULLMAN, jeffrey D. İsimli yazarların yazdığı "Compilers" isimli kitaptır[2]. Aşağıdaki bilgiler önemli ölçüde bu kitabın örnekleri ile hazırlanmıştır. Ayrıca, üniversitelerin derleyici gerçekleştirimi konusuna destek olmak için hazırlanmış olan SALUS Peter H. İsimli yazarın, serinin baş editörlüğünü yaptığı "Handbook of Programming Languages Volume III" [7] kitabı da Hacettepe Üniversitesi Bilgisayar Mühendisliği bölüm kitaplığından bulunup okunmuştur.

Derleyici, tek geçişli ve çok geçişli derleyici şeklinde ikiye ayrılmaktadır. Tek geçişli derleyici, bilgisayar programındaki değişken/değişmezler ve bilgilerini ilk geçişinde elde ettiği şeklinde ileriki aşamalara aktarır ve makine kodunu o verilere göre düzenler. Çok geçişli derleyici ise ilk geçişinden sonra bir veya birden

fazla geçiş yapmaktadır. Bunun en önemli nedeni program içindeki ileriye dönük referansların ve sapmaların daha kolay ele alınmasıdır. Derleyici programı parçalar halinde çalıştığından, birkaç program parçası programın tekrar üzerinden geçmektedir. Böylece ilk aşamada aldığı verileri hem kontrol ettiği gibi hem de makine kodunu ayrıntılı bir şekilde oluşturmaktadır. Tek geçişli derleyicinin, çok geçişli derleyiciye göre avantajı, tek sefer geçiş yaptığı için çok hızlı sonuç oluşturmasıdır. Dezavantajı ise tek geçişli derleyicinin fonksiyon prototipi vb. ön tanımlar ile desteklenmesi gereğidir. Çok geçişli derleyici her program parçasının geçişi (bazı program parçaları bir kere geçerken, bazıları da birkaç kere geçmektedir) ile bilgisayar program koduna baktığından daha iyi bir makine kodu çıkarmaktadır.

2.3 Ön İşleyici

Kaynak kod derleme işleminde, yardımcı programlar kullanılmaktadır. Ön işleyicide bunlardan biridir. Yüksek seviyeli dillerde kullanılan ve onlar için tanımlanmış bir üst seviyeli programdır. Amaçları kaynak kod içinde özel sembolleri belirleyerek, onları ayıklaması, kullanıcı tanımlı makroları çalıştırması, kaynak kod da önceden çalıştırılması gerekli programları çağırmaktır. Böylece derleyiciye onların neler olduğu belirlenmiş ve ayıklanmış olarak kaynak koda aktarılır. Dört önişleme çeşidi vardır.[2a]

1) Makro işleme; kullanıcının tanımladığı uzun kod yapıları içinde kısa yöntemler olarak kullanılmaktadır.

   Ör: #define Max= 500

2) Dosya ekleme; asıl programdan önce çalışması gereken programların belirtildiği ve asıl programın başına yazılan ifadelerdir.

   Ör: #include <stdio.h>

3) Mantıklı ön işleyici; bu işleyiciler eski versiyon programlama dillerine yeni işlevler kazandırmak için oluşturulmuştur. Programlama dilinde olmayan ama sonradan eklenerek çalıştırılan bir yapıdır.

   Ör: ifdef, endif, elseif

4) Dil uzantıları; bunlar orijinal programa dili başka yapılar ile

birleştirerek kullanmak için oluşturulmuş ön işleyici kısımlarıdır.

Ör: __BORLANDC__ >> Borland C++

__XLC__ >> AIX Compiler(AIX derleyicisi)

__GNUG__ >> Gnu C++ on any platform

Bu tez çalışmasında, ön işleyicinin işlevleri tez için gerçekleştirilen yazılıma dahildir.

2.4 Tarayıcı

Kaynak kodun ilk olarak parçalanıp birimlerine ayrıldığı ve bu birimlerin sembol tablosuna yerleştirildiği kısımdır. Birimler boşluklara önem verilmeden kaynak koddan ayıklanır. Sonra birimin ismi, tipi, boyutu gibi özellikleri ile birlikte sembol tablosuna aktarılır. Sonra tarayıcı, birimi, gerekli yerde kullanması için söz dizim çözümlemeye gönderir.

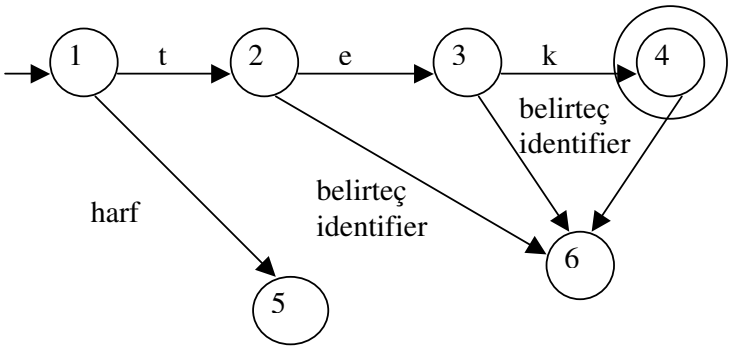
Programdan gelecek birimler için önceden kurallar belirlenir. Bu kurallar yardımı ile programdan birimler belirlenir.

```
Harf = [a - zA - Z]
sayı = [0 - 9]
belirteç = harf (harf | sayı)*

işaret = + | - | ϵ

D.sayı = işaret  (0 | [1 - 9]sayı*)
Ondalık= D.sayı . sayı*
Gerçek  sayı = (D.sayı | ondalık) E işaret sayı+
```

Önceki örnekte olduğu gibi kurallar belirlenir ve belirlenimci sonlu otomata kullanılarak birimlere ayrılır.



Şekil 2.6 Durum Diyagramı

Bir programlama dili için yüzlerce durum yazılmaktadır, böylece istenilen tüm birimler bulunabilmektedir. [2b, 4a, 3, 12, 13, 14]

Tez çalışmasında tarayıcının işlevleri tez için gerçekleştirilen yazılıma dahildir. Oluşturulan yapı tasarım kısmında verilecektir.

2.5 Sözdizim Çözümleme

Tarayıcıdan gelen birimler sözdizimin önceden belirlediği yerleşim kurallarına göre yerleştirilmeye çalışılır. Çünkü hatalı olursa, sözdizim çözümleme (syntax analysis) kaynak kodun daha ilk başta hata içerdiğini bu noktada belirlemektedir. Böylece derleyici daha fazla ilerlemeden programcıya geri bildirim gönderebilmektedir. Kaynak kod tüm kuralları hatasız geçtiğinde, anlamsal çözümlemeye girme sırası gelmiştir.

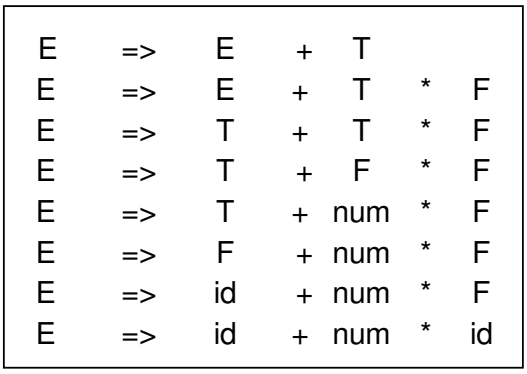
Söz dizimde belirlenen kurallar dilin dilbilgisini oluşturmaktadır. Tarayıcıdan gönderilen birimler burada dilbilgisine göre yerleşip yerleşmediği göre kontrol edilmektedir. Hem yukardan aşağı hem de aşağıdan yukarıya doğru söz dizim çözümleme yapılabilmektedir.

```
0)  S :: = E $
1)  E ::= E + T
2)      | E - T
3)      | T
4)  T ::= T * F
5)      | T / F
6)      | F
7)  F ::= num
8)    | id
```

Şekil 2.7 Dilbilgisi Kuralları

	Stack	rest-of-the-input	Action
1)		id - num * id $	shift
2)	id	- num * id $	reduce by rule 8
3)	F	- num * id $	reduce by rule 6
4)	T	- num * id $	reduce by rule 3
5)	E	- num * id $	shift
6)	E -	num * id $	shift
7)	E - num	* id $	reduce by rule 7
8)	E - F	* id $	reduce by rule 6
9)	E - T	* id $	shift
10)	E - T *	id $	shift
11)	E - T * id	$	reduce by rule 8
12)	E - T * F	$	reduce by rule 4
13)	E - T	$	reduce by rule 2
14)	E $		shift
15)	S		accept (reduce by rule 0)

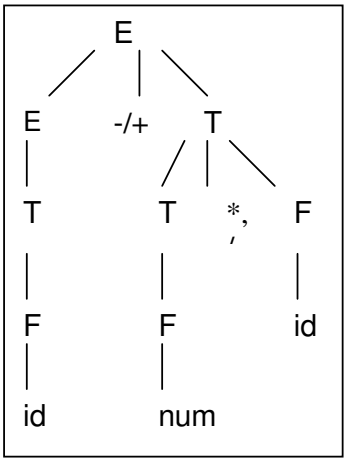
Şekil 2.8 Aşağıdan Yukarıya
Sözdizim Çözümleme Örneği



Şekil 2.9 Yukarıdan Aşağıya
Sözdizim Çözümleme Örneği

Yandaki şekil 2.7 da, BNF(Backus Naur Form) dan daha basit bir üst dil ile (http://lambda.uta.edu/cse5317/notes/node17.html) sayfasından yararlanarak bir aritmetik ifade için "söz dizim kuralları örneği" (dilbilgisi kuralları) verilmiştir.

Yanda şekil 2.7 da verilen kurallara dayanarak "x-2*y$" örneği için aşağıdan yukarıya indirgeme (http://lambda.uta.edu/cse5317/notes/node17.html) şekil 2.8 de, yine aynı kurallar ve "x+2*y$" örneği için yukarıdan aşağıya indirgeme (http://lambda.uta.edu/cse5317/notes/node12.html) şekil 2.9 de ve yanda sanal olarak oluşan söz-dizim ağacı (SyntaxTree) (http://lambda.uta.edu/cse5317/notes/node12.html) şekil 2.10 da görülmektedir.

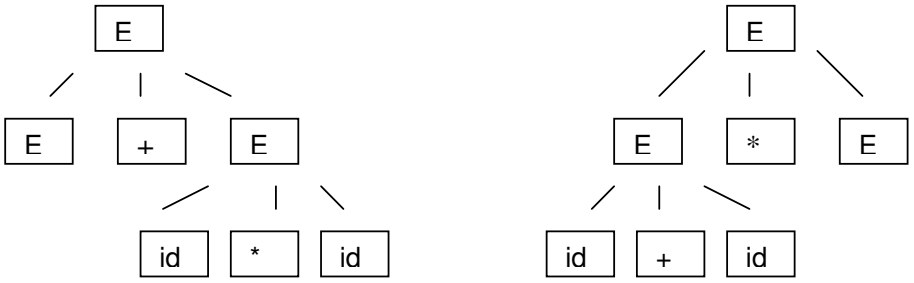


Şekil 2.10 Söz-dizim Ağacı

2.5.1 Belirsizlik (Ambiguity)

Belirsizlik söz-dizimde karşılaşılan ve tezin gerçekleştirim kısmında bahsi geçeceğinden kısaca özetlenmiştir. Kurallar tanımlanırken, aynı anda birden fazla söz-dizim ağacı oluşması ve bunun sonucunda birden fazla cevabın ortaya çıkma durumudur. Örnek [2c] olarak "id + id * id" ifadesi kullanılmıştır.

```
E => E + E                E => E * E
  => id + E                  => E + E * E
  => id + E * E              => id + E * E
  => id + id * E             => id + id * E
  => id + id * id            => id + id * id
```



Şekil 2.11 Belirsizlik

Bu belirsizlikten ya belirsizliğe neden olan kısmı yeniden yazarak ya da operatör önceliği (operator presedence) belirterek kurtarılmaktadır. Operatör önceliği belirtilen operatörlerden hangisinin önce işlemesi gerektiğini gösterir. Şekil 2.7 da verilen kurallar, çarpma ve bölme aritmetik ifadelerinin, toplama ve çıkarma aritmetik ifadelerine göre önceliği olduğunu göstermektedir. Şekil 2.10 için verilen örneği, şekil 2.7 dakı kurallara göre yeniden yaparsak, aşağıdaki gibi çıkacaktır.

```
E => E + T
  => E + T * T
  => T + T * F
  => T + F * F
  => F + F * id
  => id + id * id
```

Şekil 2.12 Operatör Önceliği

Tez çalışmasında söz-dizim çözümleme, açık yazılım ortamında bulunan (http://www.lysator.liu.se/c/ANSI-C-grammar-y.html) yazılımın kısmen düzeltilmesi (partial modification) ile yapılmıştır. 1985 yılında oluşturulan yazılım, günümüzde kullanılan C dilinin atası konumunda olduğundan, günümüzdekine uygun hale getirilmeye çalışılmıştır. Kullanılan yazılım ve gerekli düzeltmeler tez yazısının "tasarım ve gerçekleştirimi" kısmında açıklanacaktır.

2.6 Anlamsal Çözümleme

Bu aşamada, kullanıcının belirlemiş olduğu kurallara göre söz-dizim çözümleyicinin oluşturduğu söz-dizim ağacı, dilin ya da bilgisayar programının birimlerini yerli yerine yerleştirmiştir. Bu yerleştirmenin anlamsal açıdan da kontrolü gerekmektedir. Örneğin "Ayşe etek giyer" ve "Ali etek giyer" ifadeleri Türkçe deki özne-nesne-yüklem yapısına uygundurlar ama "Ali" ifadesi ile "etek" ifadesi anlamsal açıdan uyumsuzdur. "Ali" bir erkek ismi olmasına karşın, etek kadınların giydiği bir kıyafettir[1]. Bu nedenle ağaçtaki "Ali" ya da "etek" ifadelerinden biri değiştirilmelidir. Bilgisayar program kodunda aritmetik ifadelerde eğer hatalı ise bu şekilde düzeltilir. Hata ile kastedilen, bir tamsayı (integer) ile başka bir tamsayı (integer) herhangi bir aritmetik işlem sonucunda yeni bir tam sayı ortaya çıkarırlar. Bir tam sayı ile bir ondalıklı sayı aritmetik işleme giremezler. Çünkü bilgisayardaki makine kodu böyle bir işlemi yapamaz. Makine kodu verinin cinsine göre tamsayı ve ondalıklı sayıyı farklı belleklerde tutmaktadır. Ondalık sayı hem tamsayı hem de kesirli kısma sahiptir. Ondalık sayı ile işlem yapabilmek için tamsayının da ondalıklı hale getirilmesi yani virgülden sonra kaç sıfır gerekiyorsa eklenmesi gerekmektedir. Ama bundan önce makine kodunun elindeki veriyi tamsayı değil de ondalıklı sayı olarak kabul etmesi için anlamsal çözümleme kısmında tamsayı ifadesi, ondalıklı sayı olarak değiştirilmelidir.

Üretilcekler	Anlamsal Çözümleme Kuralları
E -> num	E.type := integer
E -> num . num	E.type := real
E -> id	E.type := lookup(id.entry)
E -> E1 op E2	E.type := if E1.type = integer and E2.type = integer then integer else if E1.type = integer and E2.type = real then real else if E1.type = real and E2.type = integer then real else if E1.type = real and E2.type = real then real else type_error

Şekil 2.13 Tip Değişimi[2d]

Peki anlamsal çözümleme için değişkenin tamsayı mı? Yoksa ondalıklı mı? Nasıl karar verilecektir? Bu noktada sembol tablosu (symbol table) işin içine girmektedir. Sembol tablosu birimlere ayırma (tarayıcı/scanner) sırasında değişkenler ile ilgili bilgileri tabloya eklemektedir. Bu bilgiler verinin adı, tipi, adres bilgisi, sembol tablosundaki sırası gibi verilerden oluşmaktadır. Bu veriler tarayıcı da belirlendikten sonra söz-dizim ve söz-dizim ağacının oluşmasında, anlamsal çözümlemede tip kontrolünde ve kod üretmede tipine göre değer aktarımında kullanılmaktadır.

Anlamsal çözümlemede tip düzenlemesi yapıldıktan sonra sanal olarak oluşan sözdizim ağacı üçlüler (Triples) ve dörtlüler (Quadruples) biçimine dönüştürülür. Buna ara kod üretimi de denir. Bu şekilde makine diline kolaylıkla adapte olmaları sağlanmaktadır. Üçlü, basitçe “x= -y” ifadesidir. “x” ilk değer, “-” ikinci değer ve “y” de üçüncü değer olacak şekil de üçlüyü oluşturmaktadır. Bu bir atama ifadesidir. “x” ifadesine “ y ifadesini atamaktadır. “-“ ise opsiyonsal bir durumdur. “-“ ifadesi olmadığında onun yeri boş bırakılır.[1,s.470]. Dörtlü, basitçe “x=y+z” ifadesidir. “x” ilk değeri, “y” ikinci değeri, “+” üçüncü değeri, “z” dördüncü değeri alır. Bu bir işlemin yapılıp “x” değerine aktarılmasıdır. “+” yerine diğer aritmetik işlemlerde gelebilmektedir.

	op	arg 1	arg 2	result
0	uminus	c		t1
1	*	b	t1	t2
2	uminus	c		t3
3	*	b	t3	t4
4	+	t2	t4	t5
5	:=	t5		a

Şekil 2.14 Dörtlü Gösterimi

	op	arg 1	arg 2
0	uminus	c	
1	*	b	0
2	uminus	c	
3	*	b	2
4	+	1	3
5	:=	a	4

Şekil 2.15 Üçlü Gösterimi

"a := b * - c + b * - c" ifadesi için yandaki şekil 2.14 dörtlü örneği iken yukardaki şekil 2.15 de üçlü örneği olarak verilmiştir.

Tezde anlamsal çözümleme (semantic analysis) kullanılmamıştır. Sadece derleyici kısmında derleyicinin önemli bir parçalarından biri olduğu için açıklanmıştır. Anlamsal çözümlemede açıklanan sembol tablosu (symbol table) yazılım parametrelerinin bilgileri için kullanılmıştır. Sembol tablosu tez yazısının "tez tasarım ve gerçekleştirimi" kısmın da ayrıntılı ele alınacaktır.

2.7 Kod Üretme

Anlamsal çözümlemeden sonra elde edilen ara koddaki üçlü ve dörtlüler asıl makine diline dönüştürülmeden önce indirgenirler. "x = 3 + d – 5" aritmetik ifadesi "x = d – 2" şekline indirgenmektedir. "d" değişken olarak başka bir yerdeki üçlü ya da dörtlü gösterim olarak ya bulunmaktadır, ya da sabit hafızada kayıtlıdır ve sonunda buraya eklenecektir. Bu son kısımdır. Buradan sonra derleyici işini

bitirecektir. Makine diline çevrilecek üçlü ve dörtlü ifadeler değişken, adres, gösterici (pointer), dizi gibi elemanlara sahiptirler. Bu elemanlar hafızaya yerleştirildikten sonra makine kodu oluşturulur.

"x := y + z" ifadesinin elemanları değişmeyen veri (static data) olarak hafıza bulunmaktadır. Makine kodu da aşağıdaki gibi olmaktadır.[1,s.516]

MOV y, R0

ADD z, R0

MOV R0, x

"MOV", "y" değerini "R0" yazmacına göndermektedir. "ADD", "z" değerini "R0" yazmacındaki "y" değerine eklemektedir. Yine "MOV" komutu ile "R0" yazmacındaki toplanmış değer "x" e gönderilmektedir. Burada basitçe değişmez hafızadaki veriler üzerinden makine kodu ile işlem yapılmaktadır. IF, FOR, karşılaştırmalar vb. için de makine kodu yazılabilmektedir. Aşağıdaki şekilde MS-DOS ortamında çalışan dört farklı derleyicinin makine kodu örneği gösterilmektedir. (http://www.scifac.ru.ac.za/compilers/cha02g.htm)

Borland C++ 3.1 (47 bytes)

```
CS:A0 BBB702      MOV   BX,02B7
CS:A3 C746FE5100 MOV  WORD PTR[BP-2],0051
CS:A8 EB07       JMP  B1
CS:AA 8BC3       MOV  AX,BX
CS:AC 0346FE     ADD  AX,[BP-2]
CS:AF 8BD8       MOV  BX,AX
CS:B1 83FB01      CMP   BX,1
CS:B4 7E05       JLE  BB
CS:B6 B80100     MOV  AX,1
CS:B9 EB02       JMP  BD
CS:BB 33C0       XOR  AX,AX
CS:BD 50         PUSH AX
CS:BE 83FB09     CMP  BX,9
CS:C1 7D05       JGE  C8
CS:C3 B80100       MOV   AX,1
CS:C6 EB02          JMP   CA
```

Turbo Pascal (46 bytes)
(with no short circuit evaluation)

```
CS:09 833E3E0009 CMP   WORD PTR[003E],9
CS:0E 7C04       JL   14
CS:10 B000       MOV  AL,0
CS:12 EB02       JMP  16
CS:14 B001       MOV  AL,1
CS:16 8AD0       MOV  DL,AL
CS:18 833E3E0001 CMP   WORD PTR[003E],1
CS:1D 7F04       JG   23
CS:1F B000       MOV  AL,0
CS:21 EB02       JMP  25
CS:23 B001       MOV  AL,01
CS:25 22C2       AND  AL,DL
CS:27 08C0       OR   AL,AL
CS:29 740C       JZ   37
CS:2B A13E00       MOV AX,[003E]
CS:2E 03064000    ADD AX,[0040]
```

```
       CS:C8 33C0          XOR   AX,AX                    CS:32 A33E00       MOV
[003E],AX
       CS:CA 5A         POP  DX                  CS:35 EBD2       JMP  9
       CS:CB 85D0       TEST DX,AX
       CS:CD 75DB       JNZ  AA
```

JPI TopSpeed Modula-2 (29 bytes) **Stony Brook QuickMod (24 bytes)**

```
       CS:19 2E         CS:                      CS:69 BB2D00     MOV  BX,2D
       CS:1A 8E1E2700   MOV  DS,[0027]           CS:6C B90200     MOV  CX,2
       CS:1E 833E000001 CMP  WORD PTR[0000],1    CS:6F E90200     JMP  74
       CS:23 7E11       JLE  36                  CS:72 01D9       ADD  CX,BX
       CS:25 833E000009 CMP  WORD PTR[0000],9    CS:74 83F901     CMP  CX,1
       CS:2A 7D0A       JGE  36                  CS:77 7F03       JG   7C
       CS:2C 8B0E0200   MOV  CX,[0002]           CS:79 E90500     JMP  81
       CS:30 010E0000   ADD  [0000],CX           CS:7C 83F909     CMP  CX,9
       CS:34 EBE3       JMP  19                  CS:7F 7CF1       JL   72
```

Şekil 2.16 Kod Üretimi

Bu tez çalışmasında son aşama olan kod üretme yapılmamıştır. Sadece derleyici ile ilgili bilgi vermek için kısaca açıklanmıştır.

3. TEZ ÇALIŞMASININ TASARIMI

3.1Tasarım Kararları ve Sınırları

Bu çalışma ikinci sınıfı bitirmiş bilgisayar mühendisliği öğrencilerinin yazabileceği boyuttaki C programlarının Yazılım Fiziği yönünden incelenmesidir. Önceki bölümlerde Yazılım Fiziği (Software Physics) ile ilgili tanımlar (Bölüm 1) ve Derleyici Gerçekleştirimi önbilgileri (Bölüm 2) verilmişti :

1) Bu çalışmada bilgisayar mühendisliği ikinci sınıf öğrencilerinin C programlama dilinde hazırladığı programların kullanılması.

2) Bu öğrenci programlarının yazılım fiziği parametrelerinin bulunması için gerekli olan C dili derleyicisinin (compiler), tarayıcı (scanner) ve kısmi sözdizim çözümleyici (parser) kısımlarının oluşturulması.

3) Tarayıcıda kullanılmak üzere yüz adet civarında birimlerin (Token) belirlenmesi.

4) Hazırlanan tez çalışması Yazılım Ölçüm Sistemi'nin 5 – 6 modülünden biri olduğundan, bu çalışmadan sonra devam edilebilmesi için birimlerin (Token) tanımlanıp numaralandırılması.

5) Sembol tablosunun birimler (Token) için yerleşim planının yapılması.

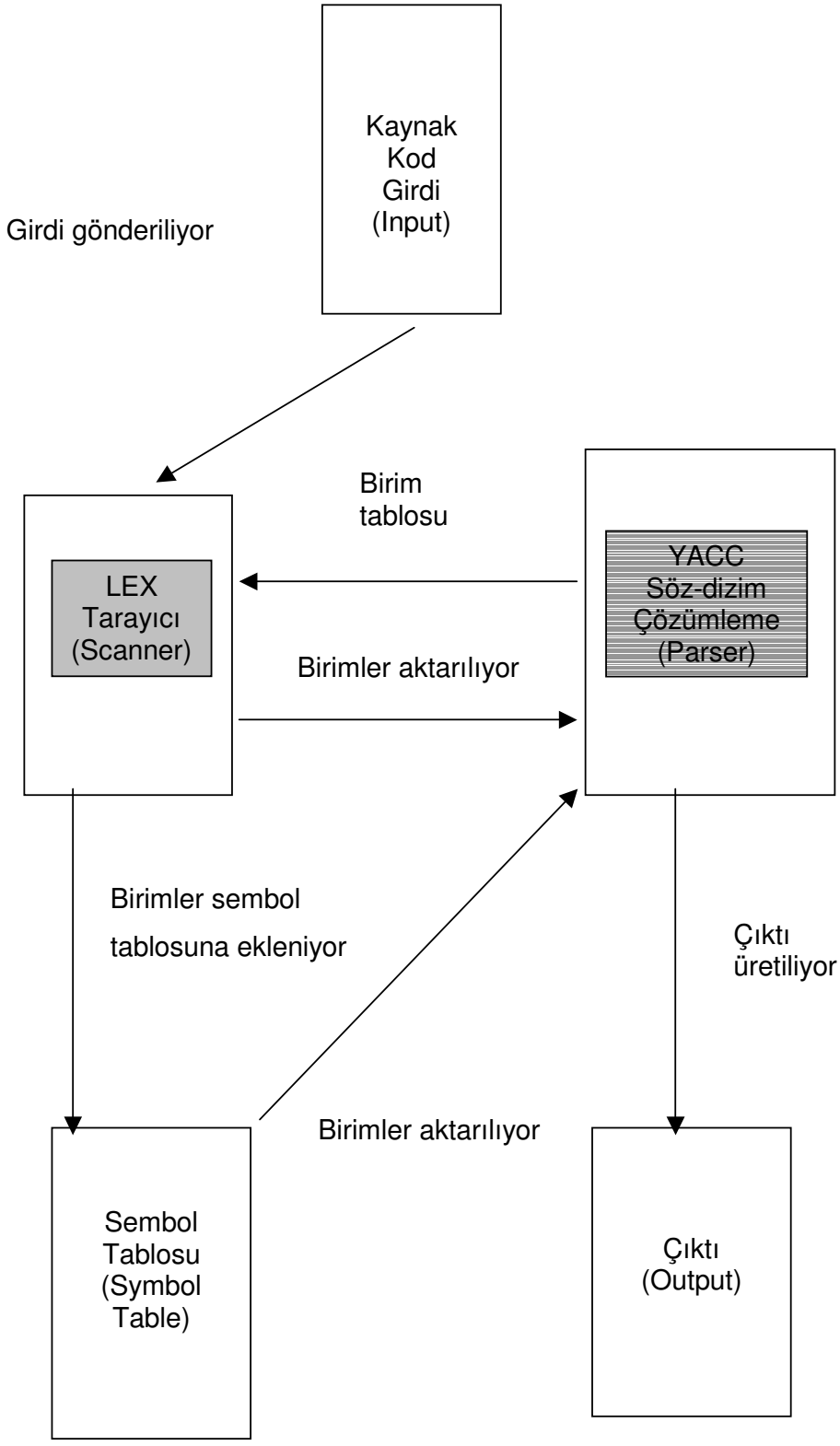
3.2 Tez Çalışmasının Blok Şeması

Bu blok şemasında iş akışı gri taralı alanlarda bulunan

LEX (http://www.lysator.liu.se/c/ANSI-C-grammar-l.html) ve

YACC (http://www.lysator.liu.se/c/ANSI-C-grammar-y.html) açık kaynak ortamında bulunarak kısmen tamir edilmiş ve eksiklikleri giderilmiştir. (Sorunlar ve çözümleri bölüm 4' de verilmiştir). Sembol tablosu kuruluşu ve sembol tablosuna kullanım sayaçları tarafımdan geliştirilmiştir.

Yanda tez çalışmasının temel mimarisi verilmiştir. Gri tonlu bölgeler (FLEX, YACC) açık yazılım ortamında bulunup, tamir edilerek kullanılan yazılım kesimleridir.



Şekil 3.1 NB v.1.1 Temel Yazılım Bölümleri

3.3 Gerçekleştirilen Tarayıcı (Scanner) Kısmı

Tarayıcı (scanner) ile ilgili (bölüm 2) derleyici bölümlerini anlatırken değinilmişti. Burada tarayıcının (scanner) nasıl çalıştığı açıklanacaktır.

Tarayıcı kendisine gönderilen bilgisayar program kodunu karakterler biçiminde okuyarak, kullanıcının belirtmiş olduğu kurallar doğrultusunda birimlere (Token) dönüştürür. Bazı birimler anahtar kelimelerdir (keywords). Bu anahtar kelimeler derleyicisi yapılan programlama dilinin kendine has kelimeleridir ve ayrılmış kelimelerdir (reserved words). Bu kelimeler kullanıcı tarafından yeniden tanımlanamamakta ve kendi görevleri dışında kullanılamamaktadır. “IF”, “FOR”, “RETURN” gibi kelimeler anahtar kelimelerdir. Kullanıcı “ifdir”, “forum”, “redurn” şeklinde belirteçleri (identifier) tanımlama hakkına sahiptir. Tarayıcıda anahtar kelimeler (keywords) ile kullanıcının tanımladığı bu belirteçleri (identifier) bilgisayar programından tanıyıp ayrıştırmaktadır. Aşağıdaki ve yandaki tablolarda LEX in kurallarının tanımlanmasında kullanılan ifadeler ve bu ifadelerin ne anlama geldiği açıklanmıştır. (http://epaperpress.com/lexandyacc/)

Meta karakter	**Karşılığı**
.	Yeni satır hariç herhangi bir karakter
\n	Yeni satır
*	İfadenin ya hiç ya da birden çok kullanılması
+	İfadenin ya bir ya da birden çok kullanılması
?	İfadenin ya hiç ya da bir kere kullanılması
^	Satır başlangıcı
$	Satır sonu
a|b	**a** veya **b**
(ab)+	**ab** ifadesinin(grup olarak) ya bir ya da birden çok kullanılması
"a+b"	"**a+b**" ifadesi
[ ]	Karakter sınıfı

Şekil 3.2 LEX de önceden tanımlamış karakterler

İfadeler	Karşılığı
abc	**abc**
abc*	**ab, abc, abcc, abccc, ...**
abc+	**abc, abcc, abccc, ...**
a(bc)+	**abc, abcbc, abcbcbc ...**
a(bc)?	**a, abc**
[abc]	**a**, **b**, **c** den sadece biri
[a-z]	a-z arasından herhangi bir harf
[a\-z]	**a**, **-**, **z** üçünden sadece biri
[-az]	**-**, **a**, **z** üçünden sadece biri
[A-Za-z0-9]+	Ya rakam veya harften biri ya rakam grubu ya harf grubu ya da hem harf hem de rakam grubu
[\t\n]+	boşluk
[^ab]	**a**, **b** dışında herhangi bir şey
[a^b]	**a**, **^**, **b** a veya b dışında herhangi bir şey
[a|b]	**a**, |, **b** üçünden sadece biri
a|b	**a**, **b** den sadece biri

Şekil 3.3 LEX de kullanılabilecek ifade karşılıkları

Şekil 3.3 de tanımlanan ifadeleri kullanarak kurallar oluşturulmaktadır. Bunlardan yararlanarak yandaki şekil 3.4 de gözüktüğü gibi kurallar tanımlanmaktadır.

harf → **[a-zA-Z]**

kelime → **HARF(HARF)+**

rakam → **[0-9]**

tam sayı → **RAKAM (RAKAM)***

belirteç → **HARF(HARF|RAKAM)***

değişmez → **HARF?(^\'\\\n)***

int→ **İNT**

float→ **FLOAT**

char→ **CHAR**

for→ **FOR**

Şekil 3.4 LEX Kurallarına Örnek

LEX; 1970 yılında YACC (söz-dizim çözümleyici) a yardımcı olması için Mike Lesk ve Eric Schimdt tarafından Bell laboratuarların da yapılmıştır. Amacı C dilinde yazılmış kaynak kodunu küçük parçalar halinde alarak, kullanıcı tarafından önceden belirlenmiş yapıları parçalar içinde belirlemektir. Belirlenen bu yapılar daha sonra YACC a aktarılarak, YACC ın işlevini gerçekleştirmesine yardımcı olurlar. Lex Unix ortamı için geliştirilmiş ve daha sonraki yıllarda diğer sistemlerde de kullanılabilir hale gelmiştir.[3a]

Bu tez çalışmasında LEX birimler (token) tablosuna ihtiyaç duymaktadır. Bu tablo sembol tablosuna gönderilen birimler için anahtar görevi görecektir. Bu tablo da yüz civarında önceden belirlenmiş ve daha sonradan yeni gelenler ile güncellenmiş birimler (token) ve kodları bulunmaktadır. Bu tablo bir numarada " identifier " ile başlayıp, elli iki numaraya kadar işaretlerden, elli iki numara ile altmış beş numara arası, sonradan eklenebilecek birimlere ve altmış beşten numaradan, yüz altı numaraya kadar anahtar kelimelerden oluşmaktadır.

Bu yüz civarındaki birimleri sembol tablosuna yerleştirmek aşağıdaki şekil 3.5 deki planlanmıştır. Bu plana uygun olarak da sembol tablosu oluşturulmuştur.

Şekil 3.5 Birim Yerleşim Planı

Şekil 3.6 da ise bu çalışmada belirlenen 106 adet birimin (token) kodları ayrıntılı bir biçimde tabloda gösterilmektedir.

#define	IDENTIFIER	1
#define	CONSTANT	2
#define	CONSTANT2	3
#define	SCOLON	4
#define	LCURL	5
#define	RCURL	6
#define	LPARA	7
#define	RPARA	8
#define	LBRAC	9
#define	RBRAC	10
#define	ASSIGNMENT	11
#define	MINUS	12
#define	PLUS	13
#define	MULTIPLICATION	14
#define	DIVISION	15
#define	REMAINDER	16
#define	COMMA	17
#define	LT	18
#define	GT	19
#define	BXOR	20
#define	BAND	21
#define	LOG_NEGATION	22
#define	QM	23
#define	BOR	24
#define	COLON	25
#define	BNEGATION	26
#define	DIRECT_SEL	27
#define	AND_OP	28
#define	OR_OP	29

```
#define    LE_OP          30
#define    GE_OP          31
#define    EQ_OP          32
#define    NE_OP          33
#define    INC_OP         34
#define    DEC_OP         35
#define    PTR_OP         36
#define    RIGHT_OP       37
#define    LEFT_OP        38
#define    RIGHT_ASSIGN   39
#define    LEFT_ASSIGN    40
#define    ADD_ASSIGN     41
#define    SUB_ASSIGN     42
#define    MUL_ASSIGN     43
#define    DIV_ASSIGN     44
#define    MOD_ASSIGN     45
#define    AND_ASSIGN     46
#define    XOR_ASSIGN     47
#define    OR_ASSIGN      48
#define    ELLIPSIS       49
#define    HASH           50
#define    UMINUS         51
#define    ADR_OP         52

#define    DEFINE         65
#define    END            66
#define    EXTERN         67
#define    CHAR           68
#define    CONST          69
#define    REGISTER       70
#define    SIZEOF         71
#define    UNION          72
#define    VOID           73
#define    TYPEDEF        74
#define    STRUCT         75
#define    UNSIGNED       76
#define    SIGNED         77
#define    INT            78
#define    LONG           79
#define    SHORT          80
#define    FLOAT          81
#define    DOUBLE         82
#define    DO             83
#define    IF             84
```

#define	FOR	85
#define	WHILE	86
#define	GOTO	87
#define	ELSE	88
#define	SWITCH	89
#define	RETURN	90
#define	ENUM	91
#define	AUTO	92
#define	BREAK	93
#define	CASE	94
#define	CONTINUE	95
#define	DEFAULT	96
#define	STATIC	97
#define	INCLUDE	98
#define	VOLATILE	99
#define	LIBRARY	100
#define	S_L	101
#define	TYPENAME	102
#define	IFDEF	103
#define	IFNDEF	104
#define	ELIF	105
#define	ENDIF	106

Şekil 3.6 Birimler Tablosu

Bu tez çalışmasında kullanılan LEX in açık yazılım ortamından bulunan, çekirdek kısım üzerine kurulduğunu daha önce belirtmiştik. Bu çekirdek kısım açık yazılım ortamında, http://www.lysator.liu.se/c/ANSI-C-grammar-l.html bu adreste bulunmakla beraber, aşağıda şekil 3.7 de gösterilmiştir.

```
D                       [0-9]
L                       [a-zA-Z_]
H                       [a-fA-F0-9]
E                       [Ee][+-]?{D}+
FS                      (f|F|l|L)
IS                      (u|U|l|L)*

%{
#include <stdio.h>
```

```
#include "y.tab.h"

void count();
%}

%%
"/*"                    { comment(); }

"auto"                  { count(); return(AUTO); }
"break"                 { count(); return(BREAK); }
"case"                  { count(); return(CASE); }
"char"                  { count(); return(CHAR); }
"const"                 { count(); return(CONST); }
"continue"              { count(); return(CONTINUE); }
"default"               { count(); return(DEFAULT); }
"do"                    { count(); return(DO); }
"double"                { count(); return(DOUBLE); }
"else"                  { count(); return(ELSE); }
"enum"                  { count(); return(ENUM); }
"extern"                { count(); return(EXTERN); }
"float"                 { count(); return(FLOAT); }
"for"                   { count(); return(FOR); }
"goto"                  { count(); return(GOTO); }
"if"                    { count(); return(IF); }
"int"                   { count(); return(INT); }
"long"                  { count(); return(LONG); }
"register"              { count(); return(REGISTER); }
"return"                { count(); return(RETURN); }
"short"                 { count(); return(SHORT); }
"signed"                { count(); return(SIGNED); }
"sizeof"                { count(); return(SIZEOF); }
"static"                { count(); return(STATIC); }
"struct"                { count(); return(STRUCT); }
"switch"                { count(); return(SWITCH); }
```

```
"typedef"               { count(); return(TYPEDEF); }
"union"                 { count(); return(UNION); }
"unsigned"              { count(); return(UNSIGNED); }
"void"                  { count(); return(VOID); }
"volatile"              { count(); return(VOLATILE); }
"while"                 { count(); return(WHILE); }

{L}({L}|{D})*           { count(); return(check_type()); }

0[xX]{H}+{IS}?          { count(); return(CONSTANT); }
0{D}+{IS}?              { count(); return(CONSTANT); }
{D}+{IS}?               { count(); return(CONSTANT); }
L?'(\\ .|[^\\ '])+'     { count(); return(CONSTANT); }

{D}+{E}{FS}?            { count(); return(CONSTANT); }
{D}*"."{D}+({E})?{FS}?  { count(); return(CONSTANT); }
{D}+"."{D}*({E})?{FS}?  { count(); return(CONSTANT); }

L?\"(\\ .| [^\\ "])*\"  { count(); return(STRING_LITERAL); }

"..."                   { count(); return(ELLIPSIS); }
">>="                   { count(); return(RIGHT_ASSIGN); }
"<<="                   { count(); return(LEFT_ASSIGN); }
"+="                    { count(); return(ADD_ASSIGN); }
"-="                    { count(); return(SUB_ASSIGN); }
"*="                    { count(); return(MUL_ASSIGN); }
"/="                    { count(); return(DIV_ASSIGN); }
"%="                    { count(); return(MOD_ASSIGN); }
"&="                    { count(); return(AND_ASSIGN); }
"^="                    { count(); return(XOR_ASSIGN); }
"|="                    { count(); return(OR_ASSIGN); }
">>"                    { count(); return(RIGHT_OP); }
"<<"                    { count(); return(LEFT_OP); }
"++"                    { count(); return(INC_OP); }
```

```
"--"                    { count(); return(DEC_OP); }
"->"                    { count(); return(PTR_OP); }
"&&"                    { count(); return(AND_OP); }
"||"                    { count(); return(OR_OP); }
"<="                    { count(); return(LE_OP); }
">="                    { count(); return(GE_OP); }
"=="                    { count(); return(EQ_OP); }
"!="                    { count(); return(NE_OP); }
";"                     { count(); return(';'); }
("{"|"<%")              { count(); return('{'); }
("}"|"%>")              { count(); return('}'); }
","                     { count(); return(','); }
":"                     { count(); return(':'); }
"="                     { count(); return('='); }
"("                     { count(); return('('); }
")"                     { count(); return(')'); }
("["|"<:")              { count(); return('['); }
("]"|":>")              { count(); return(']'); }
"."                     { count(); return('.'); }
"&"                     { count(); return('&'); }
"!"                     { count(); return('!'); }
"~"                     { count(); return('~'); }
"-"                     { count(); return('-'); }
"+"                     { count(); return('+'); }
"*"                     { count(); return('*'); }
"/"                     { count(); return('/'); }
"%"                     { count(); return('%'); }
"<"                     { count(); return('<'); }
">"                     { count(); return('>'); }
"^"                     { count(); return('^'); }
"|"                     { count(); return('|'); }
"?"                     { count(); return('?'); }

[ \t\v\n\f]             { count(); }
```

```
.                    { /* ignore bad characters */ }

%%

yywrap()
{
return(1);
}
```

Şekil 3.7 Çekirdek LEX

Şekil 3.7 deki LEX kodu 1985 yılında C dili için oluşturulmuştur. Temelde günümüzde kullanılan C dili ile aynı olmakla beraber, hem eksiklikler, hem programın çalışması için gerekli düzeltmeler, hem de tez için gerekli kodlar eklenmiştir. Yapılan önemli düzeltmeler ve eklentiler "sorunlar ve çözümleri kısmında açıklanmıştır.

3.4 Gerçekleştirilen Söz-Dizim Çözümleme ve YACC

Bu kısımla ilgili derleyici (Bölüm 2) açıklanırken, bilgilendirme yapılmıştı. Söz dizim çözümleme (parser) kullanıcının göndermiş olduğu, bilgisayar program kodundan, birimleri (token) tarayıcı yardımıyla alır ve kullanıcının oluşturduğu kurallara göre yerleştirmeyi denemektedir. Eğer kullanıcının oluşturduğu kurallar doğrultusunda yerleştirme yapamıyorsa sözdizim hatası (syntax error) oluşmaktadır. Kuralların tekrardan gözden geçirilmesi gerekmektedir. Eğer kullanıcının oluşturduğu kurallar doğrultusunda yerleştirme yapılmış ise söz-dizim ağacı oluşmuştur. Bu tezde sözdizim çözümleme için YACC programı kullanılmıştır.

YACC; 1970 yılında Stephen C. Johson tarafından Bell laboratuarların da üretilmiştir. Lex den önce Yacc yapılmış ve Unix ortamı için hazırlanmıştır. Yacc da daha sonraki senelerde diğer sistemlerde de kullanılabilir hale gelmiştir. Yacc, C dilindeki kaynak koddan gelen birimleri, C dilinin kurallarına göre yerleştirmeyi yapar. Bu kurallar da kullanıcı tarafından belirlenmiştir. Günümüzde IEEE tarafından kabul edilen, 1999 yılında en son hali olarak kabul gören C dili standardı bulunmaktadır.[3a]

Aşağıda gri alan içinde YACC koduna örnek verilmiştir.

```
%{
int yylex(void);
void yyerror(char *);
%}
%token INTEGER
%%
program:
program expr '\n' { printf("%d\n", $2); }
|
;
expr:
INTEGER { $$ = $1; }
| expr '+' expr { $$ = $1 + $3; }
| expr '-' expr { $$ = $1 - $3; }
;
%%
void yyerror(char *s) {
fprintf(stderr, "%s\n", s);
return 0;
}
int main(void) {
yyparse();
return 0;
}
```

Şekil 3.8 YACC Modeli

Yukarıdaki şekil 3.8 deki YACC modelinde, toplama ve çıkarma aritmetik işlemleri için kurallar tanımlanmıştır. Girdi olarak iki sayının toplama ya da çıkarma işlemi girildiğinde veya sadece sayı girildiğinde, ekrana sonuç yazdırılmaktadır. Eğer aritmetik işlem veya sayı yerine başka bir şey girdi olarak girilirse o zaman da, hata ekrana yazdırılmaktadır.

Bu tezde LEX de olduğu gibi YACC içinde açık yazılım ortamından çekirdek bir kod bulunmuş ve o kod üzerinde düzeltme ve eksikler tamamlanmıştır. Aşağıdaki çekirdek YACC kodu http://www.lysator.liu.se/c/ANSI-C-grammar-y.html adresinden temin edilmiştir.

```
%token IDENTIFIER CONSTANT STRING_LITERAL SIZEOF
%token PTR_OP INC_OP DEC_OP LEFT_OP RIGHT_OP LE_OP GE_OP EQ_OP NE_OP
%token AND_OP OR_OP MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN ADD_ASSIGN
%token SUB_ASSIGN LEFT_ASSIGN RIGHT_ASSIGN AND_ASSIGN
%token XOR_ASSIGN OR_ASSIGN TYPE_NAME

%token TYPEDEF EXTERN STATIC AUTO REGISTER
%token CHAR SHORT INT LONG SIGNED UNSIGNED FLOAT DOUBLE CONST VOLATILE VOID
%token STRUCT UNION ENUM ELLIPSIS

%token CASE DEFAULT IF ELSE SWITCH WHILE DO FOR GOTO CONTINUE BREAK RETURN

%start translation_unit //başlangıç
%%

primary_expression // temel ifade
        : IDENTIFIER
        | CONSTANT
        | STRING_LITERAL
        | '(' expression ')'
        ;

postfix_expression // son-ek ifade
        : primary_expression
        | postfix_expression '[' expression ']'
```

```
        | postfix_expression '(' ')'
        | postfix_expression '(' argument_expression_list ')'
        | postfix_expression '.' IDENTIFIER
        | postfix_expression PTR_OP IDENTIFIER
        | postfix_expression INC_OP
        | postfix_expression DEC_OP
        ;

argument_expression_list
        : assignment_expression
        | argument_expression_list ',' assignment_expression
        ;

unary_expression // tekil ifade
        : postfix_expression
        | INC_OP unary_expression
        | DEC_OP unary_expression
        | unary_operator cast_expression
        | SIZEOF unary_expression
        | SIZEOF '(' type_name ')'
        ;

unary_operator // tekil işlemler
        : '&'
        | '*'
        | '+'
        | '-'
        | '~'
        | '!'
        ;

cast_expression
        : unary_expression
        | '(' type_name ')' cast_expression
```

```
        ;

multiplicative_expression // çarpım ifadeleri
        : cast_expression
        | multiplicative_expression '*' cast_expression
        | multiplicative_expression '/' cast_expression
        | multiplicative_expression '%' cast_expression
        ;

additive_expression // ekleme ifadeleri
        : multiplicative_expression
        | additive_expression '+' multiplicative_expression
        | additive_expression '-' multiplicative_expression
        ;

shift_expression // kaydırma ifadeleri
        : additive_expression
        | shift_expression LEFT_OP additive_expression
        | shift_expression RIGHT_OP additive_expression
        ;

relational_expression // bağlantı ifadeleri
        : shift_expression
        | relational_expression '<' shift_expression
        | relational_expression '>' shift_expression
        | relational_expression LE_OP shift_expression
        | relational_expression GE_OP shift_expression
        ;

equality_expression // eşitlik ifadeleri
        : relational_expression
        | equality_expression EQ_OP relational_expression
        | equality_expression NE_OP relational_expression
        ;
```

```
and_expression
        : equality_expression
        | and_expression '&' equality_expression
        ;

exclusive_or_expression
        : and_expression
        | exclusive_or_expression '^' and_expression
        ;

inclusive_or_expression
        : exclusive_or_expression
        | inclusive_or_expression '|' exclusive_or_expression
        ;

logical_and_expression
        : inclusive_or_expression
        | logical_and_expression AND_OP inclusive_or_expression
        ;

logical_or_expression
        : logical_and_expression
        | logical_or_expression OR_OP logical_and_expression
        ;

conditional_expression // koşul ifadesi
        : logical_or_expression
        | logical_or_expression '?' expression ':' conditional_expression
        ;

assignment_expression // atama ifadesi
        : conditional_expression
        | unary_expression_assignment_operator_assignment_expression
```

```
        ;

assignment_operator // atama işlemleri
        : '='
        | MUL_ASSIGN
        | DIV_ASSIGN
        | MOD_ASSIGN
        | ADD_ASSIGN
        | SUB_ASSIGN
        | LEFT_ASSIGN
        | RIGHT_ASSIGN
        | AND_ASSIGN
        | XOR_ASSIGN
        | OR_ASSIGN
        ;

expression // ifade
        : assignment_expression
        | expression ',' assignment_expression
        ;

constant_expression
        : conditional_expression
        ;

declaration // tanımlama
        : declaration_specifiers ';'
        | declaration_specifiers init_declarator_list ';'
        ;

declaration_specifiers //tanımlama özellikleri
        : storage_class_specifier
        | storage_class_specifier declaration_specifiers
        | type_specifier
```

```
        | type_specifier declaration_specifiers
        | type_qualifier
        | type_qualifier declaration_specifiers
        ;

init_declarator_list
        : init_declarator
        | init_declarator_list ',' init_declarator
        ;

init_declarator // temel tanımlanan
        : declarator
        | declarator '=' initializer
        ;

storage_class_specifier // hafıza
        : TYPEDEF
        | EXTERN
        | STATIC
        | AUTO
        | REGISTER
        ;

type_specifier // tipler
        : VOID
        | CHAR
        | SHORT
        | INT
        | LONG
        | FLOAT
        | DOUBLE
        | SIGNED
        | UNSIGNED
        | struct_or_union_specifier
```

```
        | enum_specifier
        | TYPE_NAME
        ;

struct_or_union_specifier // yapı veya ortak tanımlar
        : struct_or_union IDENTIFIER '{' struct_declaration_list '}'
        | struct_or_union '{' struct_declaration_list '}'
        | struct_or_union IDENTIFIER
        ;

struct_or_union
        : STRUCT
        | UNION
        ;

struct_declaration_list
        : struct_declaration
        | struct_declaration_list struct_declaration
        ;

struct_declaration // yapı tanımlaması
        : specifier_qualifier_list struct_declarator_list ';'
        ;

specifier_qualifier_list
        : type_specifier specifier_qualifier_list
        | type_specifier
        | type_qualifier specifier_qualifier_list
        | type_qualifier
        ;

struct_declarator_list
        : struct_declarator
        | struct_declarator_list ',' struct_declarator
```

```
        ;

struct_declarator // yapı tanımlanan
        : declarator
        | ':' constant_expression
        | declarator ':' constant_expression
        ;

enum_specifier
        : ENUM '{' enumerator_list '}'
        | ENUM IDENTIFIER '{' enumerator_list '}'
        | ENUM IDENTIFIER
        ;

enumerator_list
        : enumerator
        | enumerator_list ',' enumerator
        ;

enumerator
        : IDENTIFIER
        | IDENTIFIER '=' constant_expression
        ;

type_qualifier
        : CONST
        | VOLATILE
        ;

declarator // tanımlanan
        : pointer direct_declarator
        | direct_declarator
        ;
```

```
direct_declarator // direkt tanımlanan
        : IDENTIFIER
        | '(' declarator ')'
        | direct_declarator '[' constant_expression ']'
        | direct_declarator '[' ']'
        | direct_declarator '(' parameter_type_list ')'
        | direct_declarator '(' identifier_list ')'
        | direct_declarator '(' ')'
        ;

pointer // adresleme
        : '*'
        | '*' type_qualifier_list
        | '*' pointer
        | '*' type_qualifier_list pointer
        ;

type_qualifier_list
        : type_qualifier
        | type_qualifier_list type_qualifier
        ;


parameter_type_list
        : parameter_list
        | parameter_list ',' ELLIPSIS
        ;

parameter_list
        : parameter_declaration
        | parameter_list ',' parameter_declaration
        ;

parameter_declaration
```

```
        : declaration_specifiers declarator
        | declaration_specifiers abstract_declarator
        | declaration_specifiers
        ;

identifier_list
        : IDENTIFIER
        | identifier_list ',' IDENTIFIER
        ;

type_name
        : specifier_qualifier_list
        | specifier_qualifier_list abstract_declarator
        ;

abstract_declarator // parantezler için
        : pointer
        | direct_abstract_declarator
        | pointer direct_abstract_declarator
        ;

direct_abstract_declarator
        : '(' abstract_declarator ')'
        | '[' ']'
        | '[' constant_expression ']'
        | direct_abstract_declarator '[' ']'
        | direct_abstract_declarator '[' constant_expression ']'
        | '(' ')'
        | '(' parameter_type_list ')'
        | direct_abstract_declarator '(' ')'
        | direct_abstract_declarator '(' parameter_type_list ')'
        ;

initializer
```

```
        : assignment_expression
        | '{' initializer_list '}'
        | '{' initializer_list ',' '}'
        ;

initializer_list
        : initializer
        | initializer_list ',' initializer
        ;

statement // durumlar
        : labeled_statement
        | compound_statement
        | expression_statement
        | selection_statement
        | iteration_statement
        | jump_statement
        ;

labeled_statement
        : IDENTIFIER ':' statement
        | CASE constant_expression ':' statement
        | DEFAULT ':' statement
        ;

compound_statement
        : '{' '}'
        | '{' statement_list '}'
        | '{' declaration_list '}'
        | '{' declaration_list statement_list '}'
        ;

declaration_list
        : declaration
```

```
        | declaration_list declaration
        ;

statement_list
        : statement
        | statement_list statement
        ;

expression_statement
        : ';'
        | expression ';'
        ;

selection_statement
        : IF '(' expression ')' statement
        | IF '(' expression ')' statement ELSE statement
        | SWITCH '(' expression ')' statement
        ;

iteration_statement
        : WHILE '(' expression ')' statement
        | DO statement WHILE '(' expression ')' ';'
        | FOR '(' expression_statement expression_statement ')' statement
        | FOR '(' expression_statement expression_statement expression ')' statement
        ;

jump_statement
        : GOTO IDENTIFIER ';'
        | CONTINUE ';'
        | BREAK ';'
        | RETURN ';'
        | RETURN expression ';'
```

```
        ;

translation_unit // başlangış noktasının yeri
        : external_declaration
        | translation_unit external_declaration
        ;

external_declaration // dış tanımlamalar
        : function_definition
        | declaration
        ;

function_definition // fonksiyon tanımı
        : declaration_specifiers declarator declaration_list
compound_statement
        | declaration_specifiers declarator compound_statement
        | declarator declaration_list compound_statement
        | declarator compound_statement
        ;

%%
```

Şekil 3.9 Çekirdek YACC

Yukarıdaki şekil 3.9 daki çekirdek YACC da 1985 yılında oluşturulmuştur. Günümüzde kullanılan C dili ile benzerlik göstermesine rağmen, hem eksikliklerinin giderilmesine hem de düzeltmelerinin yapılmasına ihtiyaç bulunmaktadır. Yapılan düzeltme ve eklentiler tezin " sorunlar ve çözümler" bölümünde açıklanmıştır.

3.5 Geliştirme süreci ve Geliştirilen Yazılımın Kolay-Belgele Tablosu

Bu tez çalışmasına açık yazılım ortamında bulunan LEX 'i düzeltilerek başlanılmıştır. Daha sonra sembol tablosu kurulmuş ve birimler doğru şekilde aktarılmıştır. Ardından YACC düzeltilmiş ve eksikleri ISO/IEC 9899:1999 C dili standardına [6] göre giderilmiştir. Son hali ile temel Yazılım Fiziği parametrelerini

hesaplayabilecek hale getirilmiştir. Yapılanların daha rahat anlaşılabilmesi için kolay-belgele tablosu oluşturulmuştur.

Kolay-belgele tablosu[8] yazılımcının yazılım üretkenliğini ölçmekte kullanılan bir tablodur. Bu basit ve anlaşılabilir tablo, yazılıma başlanırken ve ya yazılım sonunda kullanılabilmektedir. Yazılımın devam eden sürecinde, yazılım ile ilgili kestirimler yapılmaktadır. 1980 'li yıllarda ortaya çıkmıştır.

Yordam No	Alt sistem ya da Yordam adı	Güçlük kodu	Kim yazdı /yazacak	Belgelemesiz komut sayısı	önerilen belgeleme yüzdesi	emek normalize değeri	tahmini süre/ oluşan süre(saat)
1	tüm program		FAR				
11	main()	1,3	FAR	**45**	**1**	59,2	4,5
12	start	1,5	FAR	**223**	**2**	335,9	23
13	primary_expr	1,3	FAR	**6**	**0**	7,8	0,3
14	postfix_expr	1,3	FAR	**12**	**0**	15,6	0,6
15	argument_expr_list	1,3	FAR	**4**	**0**	5,2	0,2
16	unary_expr	1,3	FAR	**8**	**0**	10,4	0,4
17	unary_operator	1,3	FAR	**8**	**0**	10,4	0,4
18	cast_expr	1,3	FAR	**4**	**0**	5,2	0,2
19	multiplicative_expr	1,3	FAR	**6**	**0**	7,8	0,3
110	additive_expr	1,3	FAR	**5**	**0**	6,5	0,25
111	shift_expr	1,3	FAR	**4**	**0**	5,2	0,2
112	relational_expr	1,3	FAR	**7**	**0**	9,1	0,35
113	equality_expr	1,3	FAR	**5**	**0**	6,5	0,25
114	and_expr	1,3	FAR	**4**	**0**	5,2	0,2
115	exclusive_or_expr	1,3	FAR	**4**	**0**	5,2	0,2
116	inclusive_or_expr	1,3	FAR	**4**	**0**	5,2	0,2
117	logical_and_expr	1,3	FAR	**4**	**0**	5,2	0,2
118	logical_or_expr	1,3	FAR	**4**	**0**	5,2	0,2
119	conditional_expr	1,3	FAR	**4**	**0**	5,2	0,2
120	assignment_expr	1,3	FAR	**4**	**0**	5,2	0,2
121	assignment_operator	1,3	FAR	**13**	**0**	16,9	0,65
122	expr	1,3	FAR	**4**	**0**	5,2	0,2
123	constant_expr	1,3	FAR	**3**	**0**	3,9	0,15
124	declaration	1,3	FAR	**4**	**0**	5,2	0,2
125	declaration_specifiers	1,3	FAR	**8**	**0**	10,4	0,4
126	init_declarator_list	1,3	FAR	**4**	**0**	5,2	0,2
127	init_declarator	1,3	FAR	**4**	**0**	5,2	0,2
128	storage_class_specifier	1,3	FAR	**7**	**0**	9,1	0,35

129	type_specifier	1,3	FAR	**14**	**0**	18,2	1,4
130	struct_or_union_specifier	1,3	FAR	**5**	**0**	6,5	0,5
131	struct_or_union	1,3	FAR	**4**	**0**	5,2	0,2
132	struct_declaration_list	1,3	FAR	**4**	**0**	5,2	0,2
133	struct_declaration	1,3	FAR	**3**	**0**	3,9	0,15
134	qualifier_specifier_list	1,3	FAR	**6**	**0**	7,8	0,3
135	struct_declarator_list	1,3	FAR	**4**	**0**	5,2	0,2
136	struct_declarator	1,3	FAR	**5**	**0**	6,5	0,25
137	enum_specifier	1,3	FAR	**7**	**0**	9,1	0,7
138	enumerator_list	1,3	FAR	**4**	**0**	5,2	0,2
139	enumerator	1,3	FAR	**4**	**0**	5,2	0,4
140	enumeration_constant	1,3	FAR	**3**	**0**	3,9	0,3
141	type_qualifier	1,3	FAR	**4**	**0**	5,2	0,2
142	declarator	1,3	FAR	**4**	**0**	5,2	0,2
143	declarator2	1,3	FAR	**16**	**0**	20,8	1,6
144	pointer	1,3	FAR	**6**	**0**	7,8	0,3
145	type_qualifier_list	1,3	FAR	**4**	**0**	5,2	0,2
146	parameter_identifier_list	1,3	FAR	**4**	**0**	5,2	0,4
147	identifier_list	1,3	FAR	**4**	**0**	5,2	0,2
148	parameter_type_list	1,3	FAR	**4**	**0**	5,2	0,2
149	parameter_list	1,3	FAR	**4**	**0**	5,2	0,2
150	parameter_declaration	1,3	FAR	**5**	**0**	6,5	0,25
151	type_name	1,3	FAR	**3**	**0**	3,9	0,15
152	abstract_declarator	1,3	FAR	**5**	**0**	6,5	0,25
153	abstract_declarator2	1,3	FAR	**13**	**0**	16,9	1,3
154	typedef_name	1,3	FAR	**3**	**0**	3,9	0,3
155	initializer	1,3	FAR	**5**	**0**	6,5	0,25
156	initializer_list	1,3	FAR	**6**	**0**	7,8	0,6
157	designation	1,3	FAR	**3**	**0**	3,9	0,3
158	designator_list	1,3	FAR	**4**	**0**	5,2	0,4
159	designator	1,3	FAR	**4**	**0**	5,2	0,4
160	statement	1,3	FAR	**8**	**0**	10,4	0,4
161	labeled_statement	1,3	FAR	**5**	**0**	6,5	0,25
162	compound_statement	1,3	FAR	**4**	**0**	5,2	0,4
163	block_item_list	1,3	FAR	**4**	**0**	5,2	0,4
164	block_item	1,3	FAR	**4**	**0**	5,2	0,4
165	expression_statement	1,3	FAR	**4**	**0**	5,2	0,2
166	selection_statement	1,3	FAR	**5**	**0**	6,5	0,5
167	iteration_statement	1,3	FAR	**16**	**0**	20,8	1,6
168	jump_statement	1,3	FAR	**7**	**0**	9,1	0,35
169	pp_file	1,3	FAR	**4**	**0**	5,2	0,4
170	file	1,3	FAR	**4**	**0**	5,2	0,4
171	external_definition	1,3	FAR	**4**	**0**	5,2	0,2
172	function_definition	1,3	FAR	**5**	**0**	6,5	0,25

173	declaration_list	1,3	FAR	**4**	**0**	5,2	0,4
174	identifier	1,3	FAR	**3**	**0**	3,9	0,3
175	preprocessing_file	1,3	FAR	**4**	**0**	5,2	0,4
176	group	1,3	FAR	**4**	**0**	5,2	0,4
177	group_part	1,3	FAR	**4**	**0**	5,2	0,4
178	if_section	1,3	FAR	**6**	**0**	7,8	0,6
179	if_group	1,3	FAR	**8**	**0**	10,4	0,8
180	elif_groups	1,3	FAR	**4**	**0**	5,2	0,4
181	elif_group	1,3	FAR	**4**	**0**	5,2	0,4
182	else_group	1,3	FAR	**4**	**0**	5,2	0,4
183	endif_line	1,3	FAR	**3**	**0**	3,9	0,3
184	control_line	1,3	FAR	**8**	**0**	10,4	0,8
185	replacement_list	1,3	FAR	**6**	**0**	7,8	0,6
2	yywrap()	1,3	FAR	**45**	**0**	58,5	4,5
21	struct node	1,5	FAR	**13**	**5**	23,0	1,5
22	typedef struct node	0,9	FAR	**1**	**0**	0,9	0,1
23	NODE *current	0,9	FAR	**1**	**0**	0,9	0,1
24	NODE *AddSym ()	1,5	FAR	**18**	**1**	27,7	2
25	NODE *FindSym ()	1,5	FAR	**24**	**1**	36,7	2,5
26	NODE *FindSym2 ()	1,5	FAR	**19**	**1**	29,2	2
27	comment	1,1	FAR	**1**	**0**	1,1	0,2
28	comment	1,3	FAR	**21**	**0**	27,3	2,5
29	auto	1,5	FAR	**38**	**1**	57,7	4
210	break	1,5	FAR	**36**	**1**	54,7	4
211	case	1,5	FAR	**36**	**1**	54,7	4
212	char	1,5	FAR	**36**	**1**	54,7	4
213	const	1,5	FAR	**37**	**1**	56,2	4
214	continue	1,5	FAR	**37**	**1**	56,2	3,5
215	default	1,5	FAR	**36**	**1**	54,7	3,5
216	do	1,5	FAR	**36**	**1**	54,7	3,5
217	double	1,5	FAR	**36**	**1**	54,7	3,5
218	else	1,5	FAR	**51**	**1**	77,2	5
219	enum	1,5	FAR	**36**	**1**	54,7	3,6
220	extern	1,5	FAR	**36**	**1**	54,7	3,5
221	float	1,5	FAR	**36**	**1**	54,7	3,5
222	for	1,5	FAR	**36**	**1**	54,7	3,5
223	goto	1,5	FAR	**36**	**1**	54,7	3,5
224	if	1,5	FAR	**36**	**1**	54,7	3,5
225	int	1,5	FAR	**36**	**1**	54,7	3,5
226	long	1,5	FAR	**36**	**1**	54,7	3,5
227	register	1,5	FAR	**36**	**1**	54,7	3,5
228	return	1,5	FAR	**36**	**1**	54,7	3,5
229	short	1,5	FAR	**36**	**1**	54,7	3,5
230	signed	1,5	FAR	**36**	**1**	54,7	3,5

231	sizeof	1,5	FAR	**36**	**1**	54,7	3,5
232	static	1,5	FAR	**36**	**1**	54,7	3,5
233	struct	1,5	FAR	**40**	**1**	60,7	4
234	switch	1,5	FAR	**36**	**1**	54,7	3,5
235	typedef	1,5	FAR	**38**	**1**	57,7	4
236	union	1,5	FAR	**36**	**1**	54,7	3,5
237	unsigned	1,5	FAR	**36**	**1**	54,7	3,5
238	void	1,5	FAR	**37**	**1**	56,2	3,5
239	volatile	1,5	FAR	**36**	**1**	54,7	3,5
240	while	1,5	FAR	**36**	**1**	54,7	3,5
241	include	1,5	FAR	**36**	**1**	54,7	3,5
242	define	1,5	FAR	**36**	**1**	54,7	3,5
243	ifdef	1,5	FAR	**36**	**1**	54,7	3,5
244	ifndef	1,5	FAR	**36**	**1**	54,7	3,5
245	elif	1,5	FAR	**36**	**1**	54,7	3,5
246	endif	1,5	FAR	**39**	**1**	59,2	4
247	LIBRARY	1,5	FAR	**38**	**1**	57,7	4
248	S_L	1,5	FAR	**37**	**1**	56,2	3,5
249	IDENTIFIER	1,5	FAR	**107**	**1**	161,2	11
250	CONSTANT	1,5	FAR	**37**	**1**	56,2	3,5
251	CONSTANT	1,5	FAR	**37**	**1**	56,2	3,5
252	CONSTANT	1,5	FAR	**37**	**1**	56,2	3,5
253	CONSTANT2	1,5	FAR	**37**	**1**	56,2	3,5
254	CONSTANT	1,5	FAR	**37**	**1**	56,2	3,5
255	CONSTANT	1,5	FAR	**37**	**1**	56,2	3,5
256	CONSTANT	1,5	FAR	**37**	**1**	56,2	3,5
257	ELLIPSIS	1,5	FAR	**36**	**1**	54,7	3,5
258	RIGHT_ASSIGN	1,5	FAR	**36**	**1**	54,7	3,5
259	LEFT_ASSIGN	1,5	FAR	**36**	**1**	54,7	3,5
260	ADD_ASSIGN	1,5	FAR	**36**	**1**	54,7	3,5
261	SUB_ASSIGN	1,5	FAR	**36**	**1**	54,7	3,5
262	MUL_ASSIGN	1,5	FAR	**36**	**1**	54,7	3,5
263	DIV_ASSIGN	1,5	FAR	**36**	**1**	54,7	3,5
264	MOD_ASSIGN	1,5	FAR	**36**	**1**	54,7	3,5
265	AND_ASSIGN	1,5	FAR	**36**	**1**	54,7	3,5
266	XOR_ASSIGN	1,5	FAR	**36**	**1**	54,7	3,5
267	OR_ASSIGN	1,5	FAR	**36**	**1**	54,7	3,5
268	RIGHT_OP	1,5	FAR	**36**	**1**	54,7	3,5
269	LEFT_OP	1,5	FAR	**36**	**1**	54,7	3,5
270	INC_OP	1,5	FAR	**36**	**1**	54,7	3,5
271	DEC_OP	1,5	FAR	**36**	**1**	54,7	3,5
272	PTR_OP	1,5	FAR	**36**	**1**	54,7	3,5
273	AND_OP	1,5	FAR	**36**	**1**	54,7	3,5
274	OR_OP	1,5	FAR	**36**	**1**	54,7	3,5

275	LE_OP	1,5	FAR	**36**	**1**	54,7	3,5
276	GE_OP	1,5	FAR	**36**	**1**	54,7	3,5
277	EQ_OP	1,5	FAR	**36**	**1**	54,7	3,5
278	NE_OP	1,5	FAR	**36**	**1**	54,7	3,5
279	SCOLON	1,5	FAR	**37**	**1**	56,2	3,5
280	LCURL	1,5	FAR	**38**	**1**	57,7	4
281	RCURL	1,5	FAR	**44**	**1**	66,7	4,5
282	COMMA	1,5	FAR	**37**	**1**	56,2	3,5
283	COLON	1,5	FAR	**36**	**1**	54,7	3,5
284	ASSIGNMENT	1,5	FAR	**38**	**1**	57,7	3,5
285	LPARA	1,5	FAR	**78**	**1**	117,7	8
286	RPARA	1,5	FAR	**44**	**1**	66,7	4,5
287	LBRAC	1,5	FAR	**37**	**1**	56,2	4
288	RBRAC	1,5	FAR	**44**	**1**	66,7	4,5
289	DIRECT_SEL	1,5	FAR	**36**	**1**	54,7	3,5
290	BAND	1,5	FAR	**36**	**1**	54,7	3,5
291	LOG_NEGATION	1,5	FAR	**36**	**1**	54,7	3,5
292	BNEGATION	1,5	FAR	**36**	**1**	54,7	3,5
293	UMINUS	1,5	FAR	**83**	**1**	125,2	8,5
294	*MINUS	1,5	FAR				
295	PLUS	1,5	FAR	**36**	**1**	54,7	3,5
296	MULTIPLICATION	1,5	FAR	**103**	**1**	155,2	10,5
297	*ADR_OP	1,5	FAR				
298	DIVISION	1,5	FAR	**36**	**1**	54,7	3,5
299	REMAINDER	1,5	FAR	**36**	**1**	54,7	3,5
2100	LT	1,5	FAR	**36**	**1**	54,7	3,5
2101	GT	1,5	FAR	**36**	**1**	54,7	3,5
2102	BXOR	1,5	FAR	**36**	**1**	54,7	3,5
2103	BOR	1,5	FAR	**36**	**1**	54,7	3,5
2104	QM	1,5	FAR	**36**	**1**	54,7	3,5
2105	HASH	1,5	FAR	**36**	**1**	54,7	3,5
2106	/n	1,1	FAR	**1**	**0**	1,1	0,1
2107	[\t\v\f]	1	FAR	**1**	**0**	1,0	0,1
2108	**.**	1	FAR	**1**	**0**	1,0	0,1
				4575			441,45

Tablo 3.1 Kolay Belgele Tablosu

Yukarıdaki tablo 3.1 de, açık renkli olan alanlar çekirdek koddaki halleri ile kullanılmıştır. Orta renkli olan alanlar ya biraz düzeltilmiş ya da biraz eklenti yapılmıştır. Koyu renkli olan alanlar, ya çoğu değiştirilmiş ya da baştan oluşturulmuştur.

“MINUS“ ve “ADR_OP” özel durum olup, bir üstlerindeki satır içinde belirtilen kod içinde yaratılıp kullanılmışlardır. Bunun gibi bir özel durum da “TYPENAME “ için bulunmaktadır ve “ INTEGER “ içinde oluşturulup kullanılmıştır.

4.SORUNLAR VE ÇÖZÜMLER

4.1 Sembol Tablosu

Sembol tablosu, liste yapısından oluşmaktadır. Listeye eleman ekleme, YACC' ın belirlediği birimleri (token), LEX teki kontrol noktaları ve veri onaylaması ile gerçekleşmiştir. Birimler (token) eğer sembol tablosuna ilk kez yerleşecekse, yeni bir düğüm oluşturularak, gerekli bilgilerin o düğüme yazılması ve listeye eklenmesi sağlanmıştır. Eğer sembol tablosunda önceden eklenmiş bir birim (token), kod içinde tekrar bulunmuş ise, önceden oluşturulmuş o birime ait düğüm bulunarak, gerekli bilgiler güncellenmiştir.

Sembol tablosuna eklenen veri aynı zamanda ekran çıktısı olarak ta ekrana yazdırılmaktadır.

sira	tid	index	adet	tname	token
1	50	0	1	#	HASH
2	98	0	1	include	INCLUDE
3	100	3	1	<stdio.h>	LIBRARY
		...			

Tablo 4.1 Sembol Tablosu Ekran Çıktısı

4.2 LEX Sorunları ve Çözümleri

Öncelikle, açık yazılım ortamından gelen çekirdek LEX, tahmin edilebileceği gibi hemen çalışmaya hazır değildi. Tüm sorunları düzeltildiğinde bile tez için gerekli birimleri sembol tablosuna aktarmak için üzerine yeni modüller yazılması gerekmiştir.

Sorunlardan birincisi, kontrol edilen koda ait açıklamaların (comment) , ya bulunamaması, ya da kodun bir parçası biçiminde gözüküp, program hatası olarak sorun oluşturmasıdır.

Birinci sorun için çözüm şu şekildedir. Açık yazılım ortamında bulunan LEX açıklama (comment) işlevini yerine getiremiyorken, düzeltme sonucunda düzgün bir şekilde çalışmıştır. İki şekilde açıklama (comment) yapılabilmektedir. " // " ve "

/* - */ “ ilki tek satırlık kısa açıklama yapmaktadır ve sadece düzeltme yapılmıştır. İkincisi için LEX yerine bir ara modül yazılması gerekmiştir. Bir çok kaynak araştırılmıştır.

Sorunlardan ikincisi; kontrol edilen kodda yeni tip belirtildiğinde, hazırlanan kodun yeni tipi de değişken (identifier) olarak görmesi ve hata vermesidir.

İkinci sorun için çözüm şu şekildedir. ”Typedef” yapısında kökten bir düzelteme yapılmıştır. Çekirdek kodlarda “identifier” olarak geri dönen yapı söz-dizim hatalarına sebep oluyordu. “identifier” yapısını koruyarak “identifier” ları kendi içinde ayrılmıştır. “typedef” li olanlar “type_name” olarak “identifier” altında toplanmaya başladı. YACC kısmına ilerde değinilecektir. “typedef” ve “struct” yapıları içinde kontrol noktaları konulmuştur. Böylece “typedef” ve “struct” kullanıldığında “identifier” değilde “type_name” olduğu belirlenmiş olmaktadır. Sembol tablosuna da bu şekilde aktarılmaktadır.

Sorunlardan üçüncüsü; Yazılım Fiziğinde parantezlerin işlevleri gereği çift olarak sayılması gerekmektedir.

Üçüncü sorun için çözüm şu şekildedir. Üç parantez tipi için, ayrı düzenleme yapılmıştır. Parantezin açıldığı yerde sayaç bir artarken, parantezin kapandığı yerde bir azaltılmıştır. Sayaç LEX aşamasında uygulanmıştır. Ama kontrol YACC aşamasında yapılmaktadır.

Sorunlardan dördüncüsü; çıkarma işlemi ile negatif sayıyı belirtmek için kullanılan, “ - “ işaretinin ayrı olarak belirlenmesi gerekmektedir.

Dördüncü sorun için çözüm şu şekildedir. “ - “ işareti, hem “minus“ hem de ”uminus” olarak kullanıldığı için sorun oluşturmaktaydı. Aslında LEX in bu konuda çözüm sunmaktadır, fakat yazılım fiziği parametrelerini bulmak için başka bir yol kullanılması gerekmiştir. “ - “ işareti içine kontrol noktalarından gelen veriye göre “minus” veya “uminus” olduğuna karar veren bir yapı kurulmuştur. Böylece her 2 si içinde sayım yapılabilmesine olanak sağlanmıştır.

Sorunlardan beşincisi; çarpma işlemi ile adresi belirtmek için kullanılan, “ * “ işaretinin ayrı olarak belirlenmesi gerekmektedir.

Beşinci sorun için çözüm şu şekildedir. “ * ” işareti de “ - “ işareti gibi sorunludur. Hem “multiplication” hem de “adr_op” şeklinde kullanılmaktadır. Yine LEX in önerdiği çözüm yerine, her ikisinin de sayılabileceği bir yapı kurulmuştur. Kontrol noktalarından gelen bilgiye göre hangisi olduğuna karar verilip düzgün sayılması sağlanmıştır.

4.3 YACC Sorunları ve Çözümleri

Sorunlardan birincisi: “if ve if – else“ yapında oluşan “shift - shift “ ve “ shift – reduce “ hatalarının oluşmasıdır.

Birinci sorun için çözüm şu şekildedir. “ if ve if – else ” yapısı sorunu için bir çok çözüm yolu denenmiştir. İlk yapılan çözüm yolu, merdiven (katman) (kaynak olarak internet ortamındaki bir e-derginin forumlarından yararlanılmıştır.) yoluydu. Çözüm olmayınca Aho et al [2,s.175] in kitabındaki çözümü denenmiştir. İlk başta sorun gözlenmemiş, lakin farklı “if” yapılarını denenince, sorun olduğunu ve çözüm için öncelik (precedence) verilmesi gerektiğine karar verilmiştir. Sorun da bu şekilde çözülmüştür.

Sorunlardan ikincisi; LEX sorunları içinde belirtilmişti.

İkinci sorun için çözüm şu şekildedir. LEX de belirttiğim “typedef” sorununun YACC kısmında, “typedef” için belirtilen yapı günümüzde kullanılan yapıya uygun değildi. Tekrardan inceleyerek, yeni bir birim gerektiğine karar verilmiştir. Lakin, aynı zamanda “identifier” olacak bir birim. “type_name” ile hem YACC daki kural hatalarını hem de sayım için gerekli birimler oluşturulmuştur. Sözdizim “Struct” yapısıyla da uyumlu çalışır hale getirilmiştir. Oluşturulan çözüm dört yapıdan üçünü şuan için yakalayabilmektedir. Dördüncüsü için kural tanımlanmasına rağmen, hata ortaya çıkmaktadır. Hata çıkaran yapı “typedef struct name name” dir.

Sorunlardan üçüncüsü; günümüzde hazırlanan C dili ile yazılmış kodların zaman içinde farklı yapıların eklenmesi ile 1985-89 yıllarındaki yapıya göre gelişmiş olmasıdır.

Üçüncü sorun için çözüm şu şekildedir. Günümüzde kullanılan "preprocessing" kısmı 1985 yılında hazırlanan da olmadığından ISO/IEC 9899:1999 C dili standardından[6] yararlanarak, oluşturulmuş ve eklenmiştir.

5. SONUÇ

5.1 Sonuçlar

Bu çalışmadan iki sonuç elde edilmiştir. Birincisi; bu tez çalışmasında hedeflenen, ikinci ve üçüncü sınıf bilgisayar mühendisliği öğrencilerinin yapmış olduğu bilgisayar programlarının yazılım fiziği parametrelerini bulmaktı. Bu çalışma da mevcut olan LEX ve YACC programlarından yardım sağlanmıştır. LEX' in mevcut yapısı, C-99 yazılım dili standartlarına[6] göre tekrardan yapılandırılarak, C dilinde yazılmış kodlardaki birimleri belirleyip, listelemektedir. Bu liste için sembol tablosu oluşturulmuştur. Bu çalışmada anlamsal analiz yapılmadığı için sembol tablosu sadece birimler ile ilgili nitel ve nicel veri tutmuştur. Bu veriler işlenmesi için YACC' a gönderilmiştir. YACC' ın mevcut yapısında, C-99 yazılım dili standartlarına[6] göre düzeltmeler yapılarak, işleçleri ve işlenenleri belirleyebilecek biçimde kodlar eklenmiştir. Yapılanlar, ayrıntılı bir şekilde kolay belgele tablosunda gösterilmiştir[böl 3, s 56]. Gerçekleştirilen ve sentezlenen bu program, ikinci ve üçüncü sınıf öğrencilerinin hazırlamış olduğu bilgisayar programları ile kontrol edilmiştir. Ek 1 ve Ek 3' de elde edilen sonuçların dökümü mevcuttur. Ek 2' de Ek 1' deki ve Ek 4' de Ek 3' deki birimlerden elde edilen temel yazılım fiziği parametreleri gösterilmektedir.

Birinci sonuç olarak, bu tezin hedefi olan öğrenci programlarından temel yazılım fiziği parametreleri elde edilmesi başarıyla gösterilmiştir.

Bu tez, öğrenci programları arasında benzerliklerin bulunması, başka bir deyişle aşırmanın belirlenmesi kapsamında hazırlandığından, aynı ödev konusunda hazırlanan, on adet öğrenciye ait birer adet programdan, aralarında kopya olup olmadığına dair test yapılmıştır. Bu test ve sonuçları ikinci sonuç başlığı altında incelenmiştir.

İkinci olarak; aynı ödev konusunda, on öğrencinin hazırlamış olduğu birer

program, o dersin öğretmeni Öğretim Gör. Muhammet Yorulmaz' dan temin edilmiştir. Temin edilen on program arasında iki öğrencinin kopya çektiği bilinmekle beraber, hangi iki programın kopya olduğu bilinmemektedir. Bu koşullar altında, on adet programdan elde edilen temel yazılım fiziği parametreleri karşılaştırılarak, hangi iki adet programın kopya olduğunu gösteren bir test yapılmıştır.

On öğrenci programına ait "N", "N1", "N2", "n", "n1" ve "n2" temel yazılım fiziği parametreleri[böl 1, s 6], bu çalışmada hazırlanan kod yardımı ile elde edilmiştir. Aralarındaki ilişkinin, istatistiksel olarak gösterilip gösterilmediği incelenmiş ve şu sonuçlar elde edilmiştir. Öğrencilerin birer adet olan programları arasında istatistiksel olarak bir ilişki saptaması yapmak mümkün değildir. Çünkü her öğrenciye ait birer kod bulunduğundan, öğrenciler arasında doğrusal ya da doğrusal olmayan bir ilişki olduğuna dair hiçbir şey söylenememektedir. Ancak öğrencilere ait yapılmış ödev kodlamaları sayısı birden fazla olduğu taktirde, aralarında bir ilişki olup olmadığına dair istatistiksel bir çalışma yapılabilecektir.

Bu nedenle elimdeki test grubu için aralarında ilişki olup olmadığına dair istatistiksel bir çalışma yapılmamıştır. Onun yerine, frekanslar arasındaki fark kullanılarak uzaklık-yakınlık ilişkisi kurulmuştur. "N" program uzunluğu ve "n" de programda kullanılan kullanıcı tanımlı ve özel tanımlı elemanların sayısıdır. Bu iki değer, bizim pilot kod seçimlerimizi belirleyecektir.

Örneğin, o1 ogrenci1' e ait program kodu ve o2 de ogrenci2' e ait program kodu olduğu farz edilmektedir. Bu iki öğrenci koduna ait "N" ve "n" değerleri de birbirine oldukça yakın olduğu farz edilmektedir. Değerleri birbirine yakın olan öğrenci kodlarından "N1", "N2", "n1", "n2" değerleri seçilerek farklarına bakılacaktır.

Fark,N1[o1,o2] için, $-5 < 0 < 5$ arasında sonuç çıkarsa, "N2" içinde bakılacaktır. "N2" içinde (-5,+5) arasında sonuç çıkarsa, öğrenci kodlarının arasında yakınlık ilişkisi olduğu söylenecektir.

Fark,N1[o1,o2] için $-5 < 0 < 5$ dışında sonuç çıkarsa, öğrenci kodlarının arasında yakınlık ilişkisi olmadığı söylenecektir.

“N1” ve “N2” için sınırlar (-5,+5) arasındayken, “n1” ve “n2” için bu sınırlar (-1,+1) olarak belirlenmiştir.

Test grubumuzdan elde ettiğimiz parametreler yanda verilmiştir.

	o1	o2	o3	o4	o5	o6	o7	o8	o9	o10
N	2907	2140	2908	3188	2932	2306	5783	1888	2050	1237
N1	2345	1618	2346	2386	2185	1752	4420	1435	1555	941
N2	562	522	562	802	747	554	1363	453	495	296
n	117	72	116	76	72	99	106	56	75	109
n1	45	29	44	31	28	28	32	33	30	36
n2	72	43	72	45	44	71	74	23	45	73

Tablo 5.1 Test Sonuçları

Bu sonuçlardan “N”’ e bakıldığında o1, o3 ve o5 birbirine rakamsal olarak benzese de, “n” ye bakıldığında o1 ve o3’ ün birbirlerine daha yakın olduğu görülmektedir. bu nedenle o1 ve o3 için;

Fark N1 = 2345 – 2346 = -1,

Fark N2 = 562 – 562 = 0,

Fark n1 = 45 – 44 = 1,

Fark n2 = 72 – 72 = 0,

Bu farklar, “N1” ve “N2” için belirtilen (-5,+5) ve “n1” ve “n2” için belirtilen (-1,+1) aralıklarda bulunmaktadır. Sonuç olarak bu 2 öğrenci programı arasında, hem program uzunluğu, hem de programda kullanılan işleç ve işlenen sayısı bakımından göz ardı edilemeyecek bir yakınlık olduğunu söylenebilmektedir. Bu farkların yakınlığı sonucunda, bu iki öğrencinin aşırma (kopya) yaptığı söylenebilmektedir.

Bu test sonucu, programları aldığım Öğretim Görevlisi Muhammet Yorulmaz tarafından da incelenmiştir. Sonuçta tezimde hazırladığım bu programla elde ettiğim yazılım fiziği parametreleri ile ilgili 2 sonuca ulaşılmaktadır.

Birincisi, yazılım uzunluğu, yazılımın seviyesi, yazılımın sığası ve yazılım

için harcanan efor hesaplanabilmektedir.

İkincisi, ikinci sınıf bilgisayar mühendisliğinde okuyan öğrencilerin hazırlamış olduğu programlar arasında kopya olanın temel düzeyde tespitini yapabilmektedir.

Bu çalışma iki konuda yardım sağlayabilir. Birincisi, öğrenciye hazırlaması için verilecek programların takibinde yardımcı olabilir. İkincisi, öğrencilerin aynı ödev konusunda yaptığı kodlar arasında, kopya olan, olup olmadığını kontrol edebilir.

Asıl önemli olan bu çalışma bir sistemin başlangıç çalışmasıdır. Bu çalışma kullanılarak bundan sonra yapılacak diğer çalışmalarda ya da bu çalışmanın devamında yarar sağlanabilir.

KAYNAKLAR LİSTESİ

[1] HALSTEAD, Maurico H., Elements of Science, ISBN 0-444-00205-7, , a) s.1, b) s.3-4, c) s.4-5, d) s.6-8, e) s.9-18, f) s 19, g) s.25, h) s.46-47, j) s. 49, k) s.62-70

[2] AHO ,Alfred V, SETHI Ravi, ULLMAN,jeffrey D. , COMPILERS Principles,Techiques and Tools, International edition, ISBN 0-201-10194-7, a) s.16-17, b) s.83-157, c) s.171, d) s.360

[3] LEVINE, John R., MASON,Tony and BROWN, Doug , Lex & Yacc, ISBN 1-56592-000-7, 1994, a) s.xx

[4] BENNETT J. P., Introduction to Compiling Techniques, ISBN 007709221X, a) s.62-79

[5] KARAKAŞ, M. Ümit, KURT, Mücahit, Elektronik ortamda kopya ve aşırmaya karşı bir önlem : eDedektif s.0.4, 13s., 2005.

[6] ISO9899 1999-04, Programming languages — C, s29-162, 1999.

[7] SALUS Peter H., Handbook of Programming Languages Volume III, ISBN 1-57870-010-8, 1998

[8] KARAKAŞ, M. Ümit, Kolay Belgele s.2.0, 1986

[9] KOSTER C.H.A., CompilerTDiagramsNLcourse.pdf, http://www.cs.ru.nl/~kees/vbcourse/ivbstuff/dict012.pdf , s.8-14

[10] TERRY, P.D., Compiler Construction and Bootstrapping www.scifac.ru.ac.za/compilers/cha03g.htm

[11] T-diagrams, http://proglang.informatik.uni-freiburg.de/teaching/compilerbau/2004/T-diagrams.pdf **,** a) s.1-14 b) s.15-22

[12] Lesk, M . E ., Schmidt, E ., A Lexical Analyzer Generator, http://epaperpress.com/lexandyacc/download/lex.pdf

[13] NIEMANN, Tom, A COMPACT GUIDE TO LEX & YACC, http://epaperpress.com/lexandyacc/

[14] HUBERT, Bert, Lex and YACC primer/HOWTO, http://www.tldp.org/HOWTO/Lex-YACC-HOWTO.html#toc6

EK 1 ÖĞRENCİ PROGRAM ÖRNEĞİ

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <dos.h>
#include <string.h>
typedef char ListEntry;
typedef struct node {
                        struct node *next;
                        ListEntry entry;
                        }Node;
typedef struct L{
                Node* head;
                }List;
/*******************************************************
// link list kullanılarak ikincil belleğe getirilen kayıtlar sıralı dosya erisim
//performanslarını arttırabilmek icin yapılan move to front move to rear ve
//normal erisimlerin sürelerine göre kıyaslanması
*******************************************************/

//listeyi olusturan fonksiyon
void CreateList(List *list){
                list->head=NULL;
                }
Node * MakeNode(ListEntry x)
{
   Node *nodeptr;
        nodeptr=(Node *)malloc(sizeof(Node));
        if(nodeptr==NULL)
        printf("bellek yeterli deil");
        else{
                nodeptr->entry=x;
                nodeptr->next=NULL;
```

```
                }
                return nodeptr;
}
//listeye eleman ekleme
void AddList(ListEntry x, List *list){
Node *newnode, *temp;
newnode=MakeNode(x);
if(list->head==NULL)
                {
                newnode->next=list->head;
                list->head=newnode;
                }
else{
        temp=list->head;
        while(temp->next!=NULL)
                temp=temp->next;
                temp->next=newnode;
                }
}

//liste icindeki verilerin üzerinde gezinen ve kendisine gönderilen bir diger
//foksiyonun islevine göre listeyi dolasan fonksiyon
void TraverseList(List *list, void (*visit)(ListEntry))
{
Node *temp;
temp=list->head;
        while(temp!=NULL)
        {
    visit(temp->entry);
         temp=temp->next;
        }
}
//Traverse fonksiyonuna gönderilen fonksiyon
void print(ListEntry x)
```

```
{
    printf("%c ",x);
}

//move to front: dosya üzerinden son erisilen kayıtları yani en cok kukllanılan
//kayıtları basa atan fonksiyon
void movetofront(List *list,char x)
{
    Node * temp,*pre;
         temp=list->head;
         if(x!=temp->entry){
         while(temp->entry!=x && temp!=NULL)
         {
          pre=temp;
          temp=temp->next;
          }
        if(temp!=NULL){
          pre->next=temp->next;
          temp->next=list->head;
          list->head=temp;
          }
    }
}

//dosya olusturma kayıt ekleme islemi
void file1yaz(ListEntry x){
FILE *dp;
dp=fopen("final.txt", "a");
if(dp==NULL)
         printf("dosya olusturulamadi");
fputc(x,dp);
fclose(dp);
}
```

```
//move to front un olusturulduğu dosya
void filemyaz(ListEntry x){
FILE *dp;
dp=fopen("final_m.txt", "a");
if(dp==NULL)
        printf("dosya olusturulamadi");
fputc(x,dp);
fclose(dp);
}

//move to front: dosya üzerinden son erisilen kayıtlar bir daha kullanılmaz
//mantıgıyla kayıtları sona atan fonksiyon
void movetorear(List *list,char x){
Node *temp, *pre, *yedek;
    temp=list->head;
        if(x==temp->entry){
                yedek=temp;
                while(temp->next!=NULL){
          temp=temp->next;
                }
                list->head=yedek->next;
                yedek->next=NULL;
                temp->next=yedek;
              }
       else{
         while(temp->entry!=x){
       pre=temp;
                temp=temp->next;
                }

       if(temp->next!=NULL){
         pre->next=temp->next;
         yedek=temp;
```

```
        while(yedek->next!=NULL)
                yedek=yedek->next;

        yedek->next=temp;
        temp->next=NULL;
        }
        }
}


//dosya uzerinden erisim
void naccess(char x, int *a,char ad[10]){
FILE *dp;
char i;
        dp=fopen(ad,"r");
        if(dp==NULL)
                printf("dosya acilamadi");
        i=fgetc(dp);
        while(i!=x)
   {
                if(i!=x)
                *a=*a+1;
                i=fgetc(dp);
         }
         fclose(dp);
       }

int main(){

FILE *dp;
char i,dizi[20],dizi2[20],array[20];
int j,k,a=0;
double z;
List flist,list,list2;
```

```
char dosyaadi1[10],dosyaadi2[10],dosyaadi3[10];
struct time t;
CreateList(&flist);
CreateList(&list);

        dp=fopen("letters.txt","w");
        if(dp==NULL)
                printf("dosya olusturulamadi");

        printf("letters.txt dosyasina alfabe yaziliyor\n");
        for(i='a'; i<='z'; i++)
                {fputc(i,dp);
                }
        fclose(dp);
        fopen("letters.txt", "r");
      i=fgetc(dp);
        while(!feof(dp)){
                AddList(i,&flist);
                AddList(i,&list);
                i=fgetc(dp);
            }
                TraverseList(&flist,print);

        fclose(dp);
        printf("\n**************************A SIKKI**************************");
        printf("\nmove to front icin kelime girin:");
        scanf("%s", dizi);

        for(j=0; dizi[j]!='\0'; j++)
                {movetofront(&list,dizi[j]);

                }

        printf("\nmove to front sonrasi\n");
```

```
  TraverseList(&list,print);
  printf("\nfinal.txt dosyasina veriler yaziliyor\n");
  TraverseList(&list,file1yaz);

  CreateList(&list2);
  for(i='a'; i<='z'; i++)
        AddList(i,&list2);
 printf("**********************************B SIKKI*************************");
  printf("\n\nmove to rear icin kelime girin");
  scanf("%s",dizi2);

  for(k=0; dizi2[k]!='\0'; k++)

        {movetorear(&list2,dizi2[k]);
        }
  printf("\n\nmove to rear sonrasi:\n");
 TraverseList(&list2,print);
  printf("\nfinal_m.txt dosyasina veriler yaziliyor\n");
  TraverseList(&list2,filemyaz);

  printf("**********************************C SIKKI*************************");
  printf("\nFOR: orginal letter ordering....\n");
  printf("kelime girin:");
  scanf("%s", array);
  gettime(&t);
  printf("The current time is: %2d:%02d:%02d.%02d\n",t.ti_hour, t.ti_min,
t.ti_sec, t.ti_hund);
  a=0;
 strcpy(dosyaadi1,"letters.txt");
  for(z=0; z<=1000; z++){

  for(j=0; array[j]!='\0'; j++)
  {     naccess(array[j],&a,dosyaadi1);
        }
```

```
        }
        gettime(&t);
        printf("The current time is: %2d:%02d:%02d.%02d\n",t.ti_hour, t.ti_min,
t.ti_sec, t.ti_hund);
             a=0;
                 for(j=0; array[j]!='\0'; j++)
        {        naccess(array[j],&a,dosyaadi1);
                 a++;   }
         printf("Erisim sayisi:%d",a);


      a=0;
        printf("\n\nFOR: letter ordering after A....\n");
        printf("kelime girin:");
        scanf("%s", array);
        gettime(&t);
        printf("The current time is: %2d:%02d:%02d.%02d\n",t.ti_hour, t.ti_min,
t.ti_sec, t.ti_hund);
        strcpy(dosyaadi2,"final.txt");
      for(z=0; z<=1000; z++) {
        for(j=0; array[j]!='\0'; j++)
        {        naccess(array[j],&a,dosyaadi2);
                        }}

        gettime(&t);
        printf("The current time is: %2d:%02d:%02d.%02d\n",t.ti_hour, t.ti_min,
t.ti_sec, t.ti_hund);
         a=0;
         for(j=0; array[j]!='\0'; j++)
        {        naccess(array[j],&a,dosyaadi2);
                 a++;   }
         printf("Erisim sayisi:%d",a);

        a=0;
```

```
        printf("\n\nFOR: letter ordering after B....\n");
        printf("kelime girin:");
        scanf("%s", array);
        gettime(&t);
        printf("The current time is: %2d:%02d:%02d.%02d\n",t.ti_hour, t.ti_min,
t.ti_sec, t.ti_hund);
      strcpy(dosyaadi3,"final_m.txt");
      for(z=0; z<=1000; z++){
        for(j=0; array[j]!='\0'; j++)
        {       naccess(array[j],&a,dosyaadi3);
                        }}

        gettime(&t);
        printf("The current time is: %2d:%02d:%02d.%02d\n",t.ti_hour, t.ti_min,
t.ti_sec, t.ti_hund);
        a=0;
                 for(j=0; array[j]!='\0'; j++)
        {       naccess(array[j],&a,dosyaadi3);
                a++;   }
         printf("Erisim sayisi:%d",a);


getch();
return 0;
}
```

EK 2 EK 1 DEKİ ÖĞRENCİ PROGRAMININ ÇIKTI ÖRNEĞİ

sira	tid	index	adet	tname	token
1	50	0	1	#	HASH
2	98	0	1	include	INCLUDE
3	100	3	1	<stdio.h>	LIBRARY
4	50	0	2	#	HASH
5	98	0	2	include	INCLUDE
6	100	6	1	<conio.h>	LIBRARY
7	50	0	3	#	HASH
8	98	0	3	include	INCLUDE
9	100	9	1	<stdlib.h>	LIBRARY
10	50	0	4	#	HASH
11	98	0	4	include	INCLUDE
12	100	12	1	<dos.h>	LIBRARY
13	50	0	5	#	HASH
14	98	0	5	include	INCLUDE
15	100	15	1	<string.h>	LIBRARY
16	74	0	1	typedef	TYPEDEF
17	68	0	1	char	CHAR
18	1	18	1	ListEntry	IDENTIFIER
19	4	0	1	;	SCOLON
20	74	0	2	typedef	TYPEDEF
21	75	0	1	struct	STRUCT
22	1	22	1	node	IDENTIFIER
23	5	0	1	{	LCURL
24	75	0	2	struct	STRUCT
25	1	22	2	node	IDENTIFIER
26	52	0	1	adr_op	ADR_OP
27	1	27	1	next	IDENTIFIER
28	4	0	2	;	SCOLON
29	1	18	2	ListEntry	IDENTIFIER
30	1	30	1	entry	IDENTIFIER
31	4	0	3	;	SCOLON
32	6	0	1	}	RCURL
33	1	33	1	Node	IDENTIFIER
34	4	0	4	;	SCOLON
35	74	0	3	typedef	TYPEDEF
36	75	0	3	struct	STRUCT
37	1	37	1	L	IDENTIFIER
38	5	0	2	{	LCURL
39	1	33	2	Node	IDENTIFIER
40	52	0	2	adr_op	ADR_OP
41	1	41	1	head	IDENTIFIER
42	4	0	5	;	SCOLON
43	6	0	2	}	RCURL
44	1	44	1	List	IDENTIFIER
45	4	0	6	;	SCOLON

//listeyi olusturan fonksiyon

```
46    73    0     1     void        VOID
47    1     47    1     CreateList  IDENTIFIER
48    7     0     1     (           LPARA
49    1     44    2     List        IDENTIFIER
50    52    0     3     adr_op      ADR_OP
51    1     51    1     list        IDENTIFIER
52    8     0     1     )           RPARA
53    5     0     3     {           LCURL
54    1     51    2     list        IDENTIFIER
55    36    0     1     ->          PTR_OP
56    1     41    2     head        IDENTIFIER
57    11    0     1     =           ASSIGNMENT
58    1     58    1     NULL        IDENTIFIER
59    4     0     7     ;           SCOLON
60    6     0     3     }           RCURL
61    1     33    3     Node        IDENTIFIER
62    52    0     4     adr_op      ADR_OP
63    1     63    1     MakeNode    IDENTIFIER
64    7     0     2     (           LPARA
65    1     18    3     ListEntry   IDENTIFIER
66    1     66    1     x           IDENTIFIER
67    8     0     2     )           RPARA
68    5     0     4     {           LCURL
69    1     33    4     Node        IDENTIFIER
70    52    0     5     adr_op      ADR_OP
71    1     71    1     nodeptr     IDENTIFIER
72    4     0     8     ;           SCOLON
73    1     71    2     nodeptr     IDENTIFIER
74    11    0     2     =           ASSIGNMENT
75    7     0     3     (           LPARA
76    1     33    5     Node        IDENTIFIER
77    52    0     6     adr_op      ADR_OP
78    8     0     3     )           RPARA
79    1     79    1     malloc      IDENTIFIER
80    7     0     4     (           LPARA
81    71    0     1     sizeof      SIZEOF
82    7     0     5     (           LPARA
83    1     33    6     Node        IDENTIFIER
84    8     0     4     )           RPARA
85    8     0     5     )           RPARA
86    4     0     9     ;           SCOLON
87    84    0     1     if          IF
88    7     0     6     (           LPARA
89    1     71    3     nodeptr     IDENTIFIER
90    32    0     1     ==          EQ_OP
91    1     58    2     NULL        IDENTIFIER
92    8     0     6     )           RPARA
93    1     93    1     printf      IDENTIFIER
94    7     0     7     (           LPARA
95    101   0     1     string_literal  S_L
```

```
 96    8     0     7     )       RPARA
 97    4     0     10    ;       SCOLON
 98    88    0     1     else    ELSE
 99    5     0     5     {       LCURL
 100   1     71    4     nodeptr       IDENTIFIER
 101   36    0     2     ->      PTR_OP
 102   1     30    2     entry   IDENTIFIER
 103   11    0     3     =       ASSIGNMENT
 104   1     66    2     x       IDENTIFIER
 105   4     0     11    ;       SCOLON
 106   1     71    5     nodeptr       IDENTIFIER
 107   36    0     3     ->      PTR_OP
 108   1     27    2     next    IDENTIFIER
 109   11    0     4     =       ASSIGNMENT
 110   1     58    3     NULL    IDENTIFIER
 111   4     0     12    ;       SCOLON
 112   6     0     4     }       RCURL
 113   90    0     1     return  RETURN
 114   1     71    6     nodeptr       IDENTIFIER
 115   4     0     13    ;       SCOLON
 116   6     0     5     }       RCURL
 //listeye eleman ekleme
117    73    0     2     void    VOID
 118   1     118   1     AddList       IDENTIFIER
 119   7     0     8     (       LPARA
 120   1     18    4     ListEntry     IDENTIFIER
 121   1     66    3     x       IDENTIFIER
 122   17    0     1     ,       COMMA
 123   1     44    3     List    IDENTIFIER
 124   52    0     7     adr_op        ADR_OP
 125   1     51    3     list    IDENTIFIER
 126   8     0     8     )       RPARA
 127   5     0     6     {       LCURL
 128   1     33    7     Node    IDENTIFIER
 129   52    0     8     adr_op        ADR_OP
 130   1     130   1     newnode       IDENTIFIER
 131   17    0     2     ,       COMMA
 132   52    0     9     adr_op        ADR_OP
 133   1     133   1     temp    IDENTIFIER
 134   4     0     14    ;       SCOLON
 135   1     130   2     newnode       IDENTIFIER
 136   11    0     5     =       ASSIGNMENT
 137   1     63    2     MakeNode      IDENTIFIER
 138   7     0     9     (       LPARA
 139   1     66    4     x       IDENTIFIER
 140   8     0     9     )       RPARA
 141   4     0     15    ;       SCOLON
 142   84    0     1     if      IF
 143   7     0     10    (       LPARA
 144   1     51    4     list    IDENTIFIER
```

```
145  36   0    4    ->      PTR_OP
146  1    41   3    head    IDENTIFIER
147  32   0    2    ==      EQ_OP
148  1    58   4    NULL    IDENTIFIER
149  8    0    10   )       RPARA
150  5    0    7    {       LCURL
151  1    130  3    newnode IDENTIFIER
152  36   0    5    ->      PTR_OP
153  1    27   3    next    IDENTIFIER
154  11   0    6    =       ASSIGNMENT
155  1    51   5    list    IDENTIFIER
156  36   0    6    ->      PTR_OP
157  1    41   4    head    IDENTIFIER
158  4    0    16   ;       SCOLON
159  1    51   6    list    IDENTIFIER
160  36   0    7    ->      PTR_OP
161  1    41   5    head    IDENTIFIER
162  11   0    7    =       ASSIGNMENT
163  1    130  4    newnode IDENTIFIER
164  4    0    17   ;       SCOLON
165  6    0    6    }       RCURL
166  88   0    2    else    ELSE
167  5    0    8    {       LCURL
168  1    133  2    temp    IDENTIFIER
169  11   0    8    =       ASSIGNMENT
170  1    51   7    list    IDENTIFIER
171  36   0    8    ->      PTR_OP
172  1    41   6    head    IDENTIFIER
173  4    0    18   ;       SCOLON
174  86   0    1    while   WHILE
175  7    0    11   (       LPARA
176  1    133  3    temp    IDENTIFIER
177  36   0    9    ->      PTR_OP
178  1    27   4    next    IDENTIFIER
179  33   0    1    !=      NE_OP
180  1    58   5    NULL    IDENTIFIER
181  8    0    11   )       RPARA
182  1    133  4    temp    IDENTIFIER
183  11   0    9    =       ASSIGNMENT
184  1    133  5    temp    IDENTIFIER
185  36   0    10   ->      PTR_OP
186  1    27   5    next    IDENTIFIER
187  4    0    19   ;       SCOLON
188  1    133  6    temp    IDENTIFIER
189  36   0    11   ->      PTR_OP
190  1    27   6    next    IDENTIFIER
191  11   0    10   =       ASSIGNMENT
192  1    130  5    newnode IDENTIFIER
193  4    0    20   ;       SCOLON
194  6    0    7    }       RCURL
```

```
195   6     0     8     }           RCURL
 //liste icindeki verilerin üzerinde gezinen ve kendisine gönderilen bir diger
//foksiyonun islevine göre listeyi dolasan fonksiyon
196   73    0     3     void        VOID
197   1     197   1     TraverseList  IDENTIFIER
198   7     0     12    (           LPARA
199   1     44    4     List        IDENTIFIER
200   52    0     10    adr_op          ADR_OP
201   1     51    8     list        IDENTIFIER
202   17    0     3     ,           COMMA
203   73    0     4     void        VOID
204   7     0     13    (           LPARA
205   52    0     11    adr_op          ADR_OP
206   1     206   1     visit       IDENTIFIER
207   8     0     12    )           RPARA
208   7     0     14    (           LPARA
209   1     18    5     ListEntry       IDENTIFIER
210   8     0     13    )           RPARA
211   8     0     14    )           RPARA
212   5     0     9     {           LCURL
213   1     33    8     Node        IDENTIFIER
214   52    0     12    adr_op          ADR_OP
215   1     133   7     temp        IDENTIFIER
216   4     0     21    ;           SCOLON
217   1     133   8     temp        IDENTIFIER
218   11    0     11    =           ASSIGNMENT
219   1     51    9     list        IDENTIFIER
220   36    0     12    ->          PTR_OP
221   1     41    7     head        IDENTIFIER
222   4     0     22    ;           SCOLON
223   86    0     2     while       WHILE
224   7     0     15    (           LPARA
225   1     133   9     temp        IDENTIFIER
226   33    0     2     !=          NE_OP
227   1     58    6     NULL        IDENTIFIER
228   8     0     15    )           RPARA
229   5     0     10    {           LCURL
230   1     206   2     visit       IDENTIFIER
231   7     0     16    (           LPARA
232   1     133   10    temp        IDENTIFIER
233   36    0     13    ->          PTR_OP
234   1     30    3     entry       IDENTIFIER
235   8     0     16    )           RPARA
236   4     0     23    ;           SCOLON
237   1     133   11    temp        IDENTIFIER
238   11    0     12    =           ASSIGNMENT
239   1     133   12    temp        IDENTIFIER
240   36    0     14    ->          PTR_OP
241   1     27    7     next        IDENTIFIER
242   4     0     24    ;           SCOLON
```

```
243   6     0     9     }     RCURL
244   6     0     10    }     RCURL
 //Traverse fonksiyonuna gönderilen fonksiyon
245   73    0     5     void  VOID
246   1     246   1     print IDENTIFIER
247   7     0     17    (     LPARA
248   1     18    6     ListEntry     IDENTIFIER
249   1     66    5     x     IDENTIFIER
250   8     0     17    )     RPARA
251   5     0     11    {     LCURL
252   1     93    2     printf      IDENTIFIER
253   7     0     18    (     LPARA
254   101   0     2     string_literal  S_L
255   17    0     4     ,     COMMA
256   1     66    6     x     IDENTIFIER
257   8     0     18    )     RPARA
258   4     0     25    ;     SCOLON
259   6     0     11    }     RCURL
 //move to front: dosya üzerinden son erisilen kayıtları yani en cok kukllanılan
//kayıtları basa atan fonksiyon
260   73    0     6     void  VOID
261   1     261   1     movetofront       IDENTIFIER
262   7     0     19    (     LPARA
263   1     44    5     List  IDENTIFIER
264   52    0     13    adr_op      ADR_OP
265   1     51    10    list  IDENTIFIER
266   17    0     5     ,     COMMA
267   68    0     2     char  CHAR
268   1     66    7     x     IDENTIFIER
269   8     0     19    )     RPARA
270   5     0     12    {     LCURL
271   1     33    9     Node  IDENTIFIER
272   52    0     14    adr_op      ADR_OP
273   1     133   13    temp  IDENTIFIER
274   17    0     6     ,     COMMA
275   52    0     15    adr_op      ADR_OP
276   1     276   1     pre   IDENTIFIER
277   4     0     26    ;     SCOLON
278   1     133   14    temp  IDENTIFIER
279   11    0     13    =     ASSIGNMENT
280   1     51    11    list  IDENTIFIER
281   36    0     15    ->    PTR_OP
282   1     41    8     head  IDENTIFIER
283   4     0     27    ;     SCOLON
284   84    0     1     if    IF
285   7     0     20    (     LPARA
286   1     66    8     x     IDENTIFIER
287   33    0     3     !=    NE_OP
288   1     133   15    temp  IDENTIFIER
289   36    0     16    ->    PTR_OP
```

290	1	30	4	entry	IDENTIFIER
291	8	0	20	)	RPARA
292	5	0	13	{	LCURL
293	86	0	3	while	WHILE
294	7	0	21	(	LPARA
295	1	133	16	temp	IDENTIFIER
296	36	0	17	->	PTR_OP
297	1	30	5	entry	IDENTIFIER
298	33	0	4	!=	NE_OP
299	1	66	9	x	IDENTIFIER
300	28	0	1	&&	AND_OP
301	1	133	17	temp	IDENTIFIER
302	33	0	5	!=	NE_OP
303	1	58	7	NULL	IDENTIFIER
304	8	0	21	)	RPARA
305	5	0	14	{	LCURL
306	1	276	2	pre	IDENTIFIER
307	11	0	14	=	ASSIGNMENT
308	1	133	18	temp	IDENTIFIER
309	4	0	28	;	SCOLON
310	1	133	19	temp	IDENTIFIER
311	11	0	15	=	ASSIGNMENT
312	1	133	20	temp	IDENTIFIER
313	36	0	18	->	PTR_OP
314	1	27	8	next	IDENTIFIER
315	4	0	29	;	SCOLON
316	6	0	12	}	RCURL
317	84	0	2	if	IF
318	7	0	22	(	LPARA
319	1	133	21	temp	IDENTIFIER
320	33	0	6	!=	NE_OP
321	1	58	8	NULL	IDENTIFIER
322	8	0	22	)	RPARA
323	5	0	15	{	LCURL
324	1	276	3	pre	IDENTIFIER
325	36	0	19	->	PTR_OP
326	1	27	9	next	IDENTIFIER
327	11	0	16	=	ASSIGNMENT
328	1	133	22	temp	IDENTIFIER
329	36	0	20	->	PTR_OP
330	1	27	10	next	IDENTIFIER
331	4	0	30	;	SCOLON
332	1	133	23	temp	IDENTIFIER
333	36	0	21	->	PTR_OP
334	1	27	11	next	IDENTIFIER
335	11	0	17	=	ASSIGNMENT
336	1	51	12	list	IDENTIFIER
337	36	0	22	->	PTR_OP
338	1	41	9	head	IDENTIFIER
339	4	0	31	;	SCOLON

```
 340   1       51      13      list    IDENTIFIER
 341   36      0       23      ->      PTR_OP
 342   1       41      10      head    IDENTIFIER
 343   11      0       18      =       ASSIGNMENT
 344   1       133     24      temp    IDENTIFIER
 345   4       0       32      ;       SCOLON
 346   6       0       13      }       RCURL
 347   6       0       14      }       RCURL
 348   6       0       15      }       RCURL
 //dosya olusturma kayıt ekleme islemi
349    73      0       7       void    VOID
 350   1       350     1       file1yaz        IDENTIFIER
 351   7       0       23      (       LPARA
 352   1       18      7       ListEntry       IDENTIFIER
 353   1       66      10      x       IDENTIFIER
 354   8       0       23      )       RPARA
 355   5       0       16      {       LCURL
 356   1       356     1       FILE    IDENTIFIER
 357   14      0       1       *       MULTIPLICATION
 358   1       358     1       dp      IDENTIFIER
 359   4       0       33      ;       SCOLON
 360   1       358     2       dp      IDENTIFIER
 361   11      0       19      =       ASSIGNMENT
 362   1       362     1       fopen   IDENTIFIER
 363   7       0       24      (       LPARA
 364   101     0       3       string_literal  S_L
 365   8       0       24      )       RPARA
 366   4       0       34      ;       SCOLON
 367   84      0       3       if      IF
 368   7       0       25      (       LPARA
 369   1       358     3       dp      IDENTIFIER
 370   32      0       3       ==      EQ_OP
 371   1       58      9       NULL    IDENTIFIER
 372   8       0       25      )       RPARA
 373   1       93      3       printf  IDENTIFIER
 374   7       0       26      (       LPARA
 375   101     0       4       string_literal  S_L
 376   8       0       26      )       RPARA
 377   4       0       35      ;       SCOLON
 378   1       378     1       fputc   IDENTIFIER
 379   7       0       27      (       LPARA
 380   1       66      11      x       IDENTIFIER
 381   17      0       7       ,       COMMA
 382   1       358     4       dp      IDENTIFIER
 383   8       0       27      )       RPARA
 384   4       0       36      ;       SCOLON
 385   1       385     1       fclose  IDENTIFIER
 386   7       0       28      (       LPARA
 387   1       358     5       dp      IDENTIFIER
 388   8       0       28      )       RPARA
```

```
 389   4       0       37      ;       SCOLON
 390   6       0       16      }       RCURL
 //move to front un olusturulduğu dosya
391    73      0       8       void    VOID
 392   1       392     1       filemyaz        IDENTIFIER
 393   7       0       29      (       LPARA
 394   1       18      8       ListEntry       IDENTIFIER
 395   1       66      12      x       IDENTIFIER
 396   8       0       29      )       RPARA
 397   5       0       17      {       LCURL
 398   1       356     2       FILE    IDENTIFIER
 399   14      0       2       *       MULTIPLICATION
 400   1       358     6       dp      IDENTIFIER
 401   4       0       38      ;       SCOLON
 402   1       358     7       dp      IDENTIFIER
 403   11      0       20      =       ASSIGNMENT
 404   1       362     2       fopen   IDENTIFIER
 405   7       0       30      (       LPARA
 406   101     0       5       string_literal  S_L
 407   8       0       30      )       RPARA
 408   4       0       39      ;       SCOLON
 409   84      0       4       if      IF
 410   7       0       31      (       LPARA
 411   1       358     8       dp      IDENTIFIER
 412   32      0       4       ==      EQ_OP
 413   1       58      10      NULL    IDENTIFIER
 414   8       0       31      )       RPARA
 415   1       93      4       printf  IDENTIFIER
 416   7       0       32      (       LPARA
 417   101     0       6       string_literal  S_L
 418   8       0       32      )       RPARA
 419   4       0       40      ;       SCOLON
 420   1       378     2       fputc   IDENTIFIER
 421   7       0       33      (       LPARA
 422   1       66      13      x       IDENTIFIER
 423   17      0       8       ,       COMMA
 424   1       358     9       dp      IDENTIFIER
 425   8       0       33      )       RPARA
 426   4       0       41      ;       SCOLON
 427   1       385     2       fclose  IDENTIFIER
 428   7       0       34      (       LPARA
 429   1       358     10      dp      IDENTIFIER
 430   8       0       34      )       RPARA
 431   4       0       42      ;       SCOLON
 432   6       0       17      }       RCURL
 //move to front: dosya üzerinden son erisilen kayıtlar bir daha kullanılmaz
//mantıgıyla kayıtları sona atan fonksiyon
433    73      0       9       void    VOID
 434   1       434     1       movetorear      IDENTIFIER
 435   7       0       35      (       LPARA
```

436	1	44	6	List	IDENTIFIER
437	52	0	16	adr_op	ADR_OP
438	1	51	14	list	IDENTIFIER
439	17	0	9	,	COMMA
440	68	0	3	char	CHAR
441	1	66	14	x	IDENTIFIER
442	8	0	35	)	RPARA
443	5	0	18	{	LCURL
444	1	33	10	Node	IDENTIFIER
445	52	0	17	adr_op	ADR_OP
446	1	133	25	temp	IDENTIFIER
447	17	0	10	,	COMMA
448	52	0	18	adr_op	ADR_OP
449	1	276	4	pre	IDENTIFIER
450	17	0	11	,	COMMA
451	52	0	19	adr_op	ADR_OP
452	1	452	1	yedek	IDENTIFIER
453	4	0	43	;	SCOLON
454	1	133	26	temp	IDENTIFIER
455	11	0	21	=	ASSIGNMENT
456	1	51	15	list	IDENTIFIER
457	36	0	24	->	PTR_OP
458	1	41	11	head	IDENTIFIER
459	4	0	44	;	SCOLON
460	84	0	5	if	IF
461	7	0	36	(	LPARA
462	1	66	15	x	IDENTIFIER
463	32	0	5	==	EQ_OP
464	1	133	27	temp	IDENTIFIER
465	36	0	25	->	PTR_OP
466	1	30	6	entry	IDENTIFIER
467	8	0	36	)	RPARA
468	5	0	19	{	LCURL
469	1	452	2	yedek	IDENTIFIER
470	11	0	22	=	ASSIGNMENT
471	1	133	28	temp	IDENTIFIER
472	4	0	45	;	SCOLON
473	86	0	4	while	WHILE
474	7	0	37	(	LPARA
475	1	133	29	temp	IDENTIFIER
476	36	0	26	->	PTR_OP
477	1	27	12	next	IDENTIFIER
478	33	0	7	!=	NE_OP
479	1	58	11	NULL	IDENTIFIER
480	8	0	37	)	RPARA
481	5	0	20	{	LCURL
482	1	133	30	temp	IDENTIFIER
483	11	0	23	=	ASSIGNMENT
484	1	133	31	temp	IDENTIFIER
485	36	0	27	->	PTR_OP

486	1	27	13	next	IDENTIFIER
487	4	0	46	;	SCOLON
488	6	0	18	}	RCURL
489	1	51	16	list	IDENTIFIER
490	36	0	28	->	PTR_OP
491	1	41	12	head	IDENTIFIER
492	11	0	24	=	ASSIGNMENT
493	1	452	3	yedek	IDENTIFIER
494	36	0	29	->	PTR_OP
495	1	27	14	next	IDENTIFIER
496	4	0	47	;	SCOLON
497	1	452	4	yedek	IDENTIFIER
498	36	0	30	->	PTR_OP
499	1	27	15	next	IDENTIFIER
500	11	0	25	=	ASSIGNMENT
501	1	58	12	NULL	IDENTIFIER
502	4	0	48	;	SCOLON
503	1	133	32	temp	IDENTIFIER
504	36	0	31	->	PTR_OP
505	1	27	16	next	IDENTIFIER
506	11	0	26	=	ASSIGNMENT
507	1	452	5	yedek	IDENTIFIER
508	4	0	49	;	SCOLON
509	6	0	19	}	RCURL
510	88	0	3	else	ELSE
511	5	0	21	{	LCURL
512	86	0	5	while	WHILE
513	7	0	38	(	LPARA
514	1	133	33	temp	IDENTIFIER
515	36	0	32	->	PTR_OP
516	1	30	7	entry	IDENTIFIER
517	33	0	8	!=	NE_OP
518	1	66	16	x	IDENTIFIER
519	8	0	38	)	RPARA
520	5	0	22	{	LCURL
521	1	276	5	pre	IDENTIFIER
522	11	0	27	=	ASSIGNMENT
523	1	133	34	temp	IDENTIFIER
524	4	0	50	;	SCOLON
525	1	133	35	temp	IDENTIFIER
526	11	0	28	=	ASSIGNMENT
527	1	133	36	temp	IDENTIFIER
528	36	0	33	->	PTR_OP
529	1	27	17	next	IDENTIFIER
530	4	0	51	;	SCOLON
531	6	0	20	}	RCURL
532	84	0	5	if	IF
533	7	0	39	(	LPARA
534	1	133	37	temp	IDENTIFIER
535	36	0	34	->	PTR_OP

```
536   1     27    18    next    IDENTIFIER
537   33    0     9     !=      NE_OP
538   1     58    13    NULL    IDENTIFIER
539   8     0     39    )       RPARA
540   5     0     23    {       LCURL
541   1     276   6     pre     IDENTIFIER
542   36    0     35    ->      PTR_OP
543   1     27    19    next    IDENTIFIER
544   11    0     29    =       ASSIGNMENT
545   1     133   38    temp    IDENTIFIER
546   36    0     36    ->      PTR_OP
547   1     27    20    next    IDENTIFIER
548   4     0     52    ;       SCOLON
549   1     452   6     yedek   IDENTIFIER
550   11    0     30    =       ASSIGNMENT
551   1     133   39    temp    IDENTIFIER
552   4     0     53    ;       SCOLON
553   86    0     6     while   WHILE
554   7     0     40    (       LPARA
555   1     452   7     yedek   IDENTIFIER
556   36    0     37    ->      PTR_OP
557   1     27    21    next    IDENTIFIER
558   33    0     10    !=      NE_OP
559   1     58    14    NULL    IDENTIFIER
560   8     0     40    )       RPARA
561   1     452   8     yedek   IDENTIFIER
562   11    0     31    =       ASSIGNMENT
563   1     452   9     yedek   IDENTIFIER
564   36    0     38    ->      PTR_OP
565   1     27    22    next    IDENTIFIER
566   4     0     54    ;       SCOLON
567   1     452   10    yedek   IDENTIFIER
568   36    0     39    ->      PTR_OP
569   1     27    23    next    IDENTIFIER
570   11    0     32    =       ASSIGNMENT
571   1     133   40    temp    IDENTIFIER
572   4     0     55    ;       SCOLON
573   1     133   41    temp    IDENTIFIER
574   36    0     40    ->      PTR_OP
575   1     27    24    next    IDENTIFIER
576   11    0     33    =       ASSIGNMENT
577   1     58    15    NULL    IDENTIFIER
578   4     0     56    ;       SCOLON
579   6     0     21    }       RCURL
580   6     0     22    }       RCURL
581   6     0     23    }       RCURL
//dosya uzerinden erisim
582   73    0     10    void    VOID
583   1     583   1     naccess       IDENTIFIER
584   7     0     41    (       LPARA
```

585 68 0 4 char CHAR
586 1 66 17 x IDENTIFIER
587 17 0 12 , COMMA
588 78 0 1 int INT
589 52 0 20 adr_op ADR_OP
590 1 590 1 a IDENTIFIER
591 17 0 13 , COMMA
592 68 0 5 char CHAR
593 1 593 1 ad IDENTIFIER
594 9 0 1 [LBRAC
595 2 595 1 10 CONSTANT
596 10 0 1] RBRAC
597 8 0 41) RPARA
598 5 0 24 { LCURL
599 1 356 3 FILE IDENTIFIER
600 14 0 3 * MULTIPLICATION
601 1 358 11 dp IDENTIFIER
602 4 0 57 ; SCOLON
603 68 0 6 char CHAR
604 1 604 1 i IDENTIFIER
605 4 0 58 ; SCOLON
606 1 358 12 dp IDENTIFIER
607 11 0 34 = ASSIGNMENT
608 1 362 3 fopen IDENTIFIER
609 7 0 42 (LPARA
610 1 593 2 ad IDENTIFIER
611 17 0 14 , COMMA
612 101 0 7 string_literal S_L
613 8 0 42) RPARA
614 4 0 59 ; SCOLON
615 84 0 6 if IF
616 7 0 43 (LPARA
617 1 358 13 dp IDENTIFIER
618 32 0 6 == EQ_OP
619 1 58 16 NULL IDENTIFIER
620 8 0 43) RPARA
621 1 93 5 printf IDENTIFIER
622 7 0 44 (LPARA
623 101 0 8 string_literal S_L
624 8 0 44) RPARA
625 4 0 60 ; SCOLON
626 1 604 2 i IDENTIFIER
627 11 0 35 = ASSIGNMENT
628 1 628 1 fgetc IDENTIFIER
629 7 0 45 (LPARA
630 1 358 14 dp IDENTIFIER
631 8 0 45) RPARA
632 4 0 61 ; SCOLON
633 86 0 7 while WHILE
634 7 0 46 (LPARA

635	1	604	3	i	IDENTIFIER
636	33	0	11	!=	NE_OP
637	1	66	18	x	IDENTIFIER
638	8	0	46	)	RPARA
639	5	0	25	{	LCURL
640	84	0	7	if	IF
641	7	0	47	(	LPARA
642	1	604	4	i	IDENTIFIER
643	33	0	12	!=	NE_OP
644	1	66	19	x	IDENTIFIER
645	8	0	47	)	RPARA
646	14	0	4	*	MULTIPLICATION
647	1	590	2	a	IDENTIFIER
648	11	0	36	=	ASSIGNMENT
649	14	0	5	*	MULTIPLICATION
650	1	590	3	a	IDENTIFIER
651	13	0	1	+	PLUS
652	2	652	1	1	CONSTANT
653	4	0	62	;	SCOLON
654	1	604	5	i	IDENTIFIER
655	11	0	37	=	ASSIGNMENT
656	1	628	2	fgetc	IDENTIFIER
657	7	0	48	(	LPARA
658	1	358	15	dp	IDENTIFIER
659	8	0	48	)	RPARA
660	4	0	63	;	SCOLON
661	6	0	24	}	RCURL
662	1	385	3	fclose	IDENTIFIER
663	7	0	49	(	LPARA
664	1	358	16	dp	IDENTIFIER
665	8	0	49	)	RPARA
666	4	0	64	;	SCOLON
667	6	0	25	}	RCURL
668	78	0	2	int	INT
669	1	669	1	main	IDENTIFIER
670	7	0	50	(	LPARA
671	8	0	50	)	RPARA
672	5	0	26	{	LCURL
673	1	356	4	FILE	IDENTIFIER
674	14	0	6	*	MULTIPLICATION
675	1	358	17	dp	IDENTIFIER
676	4	0	65	;	SCOLON
677	68	0	7	char	CHAR
678	1	604	6	i	IDENTIFIER
679	17	0	15	,	COMMA
680	1	680	1	dizi	IDENTIFIER
681	9	0	2	[	LBRAC
682	2	682	1	20	CONSTANT
683	10	0	2	]	RBRAC
684	17	0	16	,	COMMA

685	1	685	1	dizi2	IDENTIFIER
686	9	0	3	[	LBRAC
687	2	682	2	20	CONSTANT
688	10	0	3	]	RBRAC
689	17	0	17	,	COMMA
690	1	690	1	array	IDENTIFIER
691	9	0	4	[	LBRAC
692	2	682	3	20	CONSTANT
693	10	0	4	]	RBRAC
694	4	0	66	;	SCOLON
695	78	0	3	int	INT
696	1	696	1	j	IDENTIFIER
697	17	0	18	,	COMMA
698	1	698	1	k	IDENTIFIER
699	17	0	19	,	COMMA
700	1	590	4	a	IDENTIFIER
701	11	0	38	=	ASSIGNMENT
702	2	702	1	0	CONSTANT
703	4	0	67	;	SCOLON
704	82	0	1	double	DOUBLE
705	1	705	1	z	IDENTIFIER
706	4	0	68	;	SCOLON
707	1	44	7	List	IDENTIFIER
708	1	708	1	flist	IDENTIFIER
709	17	0	20	,	COMMA
710	1	51	17	list	IDENTIFIER
711	17	0	21	,	COMMA
712	1	712	1	list2	IDENTIFIER
713	4	0	69	;	SCOLON
714	68	0	8	char	CHAR
715	1	715	1	dosyaadi1	IDENTIFIER
716	9	0	5	[	LBRAC
717	2	595	2	10	CONSTANT
718	10	0	5	]	RBRAC
719	17	0	22	,	COMMA
720	1	720	1	dosyaadi2	IDENTIFIER
721	9	0	6	[	LBRAC
722	2	595	3	10	CONSTANT
723	10	0	6	]	RBRAC
724	17	0	23	,	COMMA
725	1	725	1	dosyaadi3	IDENTIFIER
726	9	0	7	[	LBRAC
727	2	595	4	10	CONSTANT
728	10	0	7	]	RBRAC
729	4	0	70	;	SCOLON
730	75	0	4	struct	STRUCT
731	1	731	1	time	IDENTIFIER
732	1	732	1	t	IDENTIFIER
733	4	0	71	;	SCOLON
734	1	47	2	CreateList	IDENTIFIER

735	7	0	51	(	LPARA
736	21	0	1	&	BAND
737	1	708	2	flist	IDENTIFIER
738	8	0	51	)	RPARA
739	4	0	72	;	SCOLON
740	1	47	3	CreateList	IDENTIFIER
741	7	0	52	(	LPARA
742	21	0	2	&	BAND
743	1	51	18	list	IDENTIFIER
744	8	0	52	)	RPARA
745	4	0	73	;	SCOLON
746	1	358	18	dp	IDENTIFIER
747	11	0	39	=	ASSIGNMENT
748	1	362	4	fopen	IDENTIFIER
749	7	0	53	(	LPARA
750	101	0	9	string_literal	S_L
751	8	0	53	)	RPARA
752	4	0	74	;	SCOLON
753	84	0	8	if	IF
754	7	0	54	(	LPARA
755	1	358	19	dp	IDENTIFIER
756	32	0	7	==	EQ_OP
757	1	58	17	NULL	IDENTIFIER
758	8	0	54	)	RPARA
759	1	93	6	printf	IDENTIFIER
760	7	0	55	(	LPARA
761	101	0	10	string_literal	S_L
762	8	0	55	)	RPARA
763	4	0	75	;	SCOLON
764	1	93	7	printf	IDENTIFIER
765	7	0	56	(	LPARA
766	101	0	11	string_literal	S_L
767	8	0	56	)	RPARA
768	4	0	76	;	SCOLON
769	85	0	1	for	FOR
770	7	0	57	(	LPARA
771	1	604	7	i	IDENTIFIER
772	11	0	40	=	ASSIGNMENT
773	1	590	5	a	IDENTIFIER
774	4	0	77	;	SCOLON
775	1	604	8	i	IDENTIFIER
776	30	0	1	<=	LE_OP
777	1	705	2	z	IDENTIFIER
778	4	0	78	;	SCOLON
779	1	604	9	i	IDENTIFIER
780	34	0	1	++	INC_OP
781	8	0	57	)	RPARA
782	5	0	27	{	LCURL
783	1	378	3	fputc	IDENTIFIER
784	7	0	58	(	LPARA

785 1 604 10 i IDENTIFIER
786 17 0 24 , COMMA
787 1 358 20 dp IDENTIFIER
788 8 0 58) RPARA
789 4 0 79 ; SCOLON
790 6 0 26 } RCURL
791 1 385 4 fclose IDENTIFIER
792 7 0 59 (LPARA
793 1 358 21 dp IDENTIFIER
794 8 0 59) RPARA
795 4 0 80 ; SCOLON
796 1 362 5 fopen IDENTIFIER
797 7 0 60 (LPARA
798 101 0 12 string_literal S_L
799 8 0 60) RPARA
800 4 0 81 ; SCOLON
801 1 604 11 i IDENTIFIER
802 11 0 41 = ASSIGNMENT
803 1 628 3 fgetc IDENTIFIER
804 7 0 61 (LPARA
805 1 358 22 dp IDENTIFIER
806 8 0 61) RPARA
807 4 0 82 ; SCOLON
808 86 0 8 while WHILE
809 7 0 62 (LPARA
810 22 0 1 ! LOG_NEGATION
811 1 811 1 feof IDENTIFIER
812 7 0 63 (LPARA
813 1 358 23 dp IDENTIFIER
814 8 0 62) RPARA
815 8 0 63) RPARA
816 5 0 28 { LCURL
817 1 118 2 AddList IDENTIFIER
818 7 0 64 (LPARA
819 1 604 12 i IDENTIFIER
820 17 0 25 , COMMA
821 21 0 3 & BAND
822 1 708 3 flist IDENTIFIER
823 8 0 64) RPARA
824 4 0 83 ; SCOLON
825 1 118 3 AddList IDENTIFIER
826 7 0 65 (LPARA
827 1 604 13 i IDENTIFIER
828 17 0 26 , COMMA
829 21 0 4 & BAND
830 1 51 19 list IDENTIFIER
831 8 0 65) RPARA
832 4 0 84 ; SCOLON
833 1 604 14 i IDENTIFIER
834 11 0 42 = ASSIGNMENT

835	1	628	4	fgetc	IDENTIFIER
836	7	0	66	(	LPARA
837	1	358	24	dp	IDENTIFIER
838	8	0	66	)	RPARA
839	4	0	85	;	SCOLON
840	6	0	27	}	RCURL
841	1	197	2	TraverseList	IDENTIFIER
842	7	0	67	(	LPARA
843	21	0	5	&	BAND
844	1	708	4	flist	IDENTIFIER
845	17	0	27	,	COMMA
846	1	246	2	print	IDENTIFIER
847	8	0	67	)	RPARA
848	4	0	86	;	SCOLON
849	1	385	5	fclose	IDENTIFIER
850	7	0	68	(	LPARA
851	1	358	25	dp	IDENTIFIER
852	8	0	68	)	RPARA
853	4	0	87	;	SCOLON
854	1	93	8	printf	IDENTIFIER
855	7	0	69	(	LPARA
856	101	0	13	string_literal	S_L
857	8	0	69	)	RPARA
858	4	0	88	;	SCOLON
859	1	93	9	printf	IDENTIFIER
860	7	0	70	(	LPARA
861	101	0	14	string_literal	S_L
862	8	0	70	)	RPARA
863	4	0	89	;	SCOLON
864	1	864	1	scanf	IDENTIFIER
865	7	0	71	(	LPARA
866	101	0	15	string_literal	S_L
867	17	0	28	,	COMMA
868	1	680	2	dizi	IDENTIFIER
869	8	0	71	)	RPARA
870	4	0	90	;	SCOLON
871	85	0	2	for	FOR
872	7	0	72	(	LPARA
873	1	696	2	j	IDENTIFIER
874	11	0	43	=	ASSIGNMENT
875	2	702	2	0	CONSTANT
876	4	0	91	;	SCOLON
877	1	680	3	dizi	IDENTIFIER
878	9	0	8	[	LBRAC
879	1	696	3	j	IDENTIFIER
880	10	0	8	]	RBRAC
881	33	0	13	!=	NE_OP
882	2	702	3	0	CONSTANT
883	4	0	92	;	SCOLON
884	1	696	4	j	IDENTIFIER

885 34 0 2 ++ INC_OP
886 8 0 72) RPARA
887 5 0 29 { LCURL
888 1 261 2 movetofront IDENTIFIER
889 7 0 73 (LPARA
890 21 0 6 & BAND
891 1 51 20 list IDENTIFIER
892 17 0 29 , COMMA
893 1 680 4 dizi IDENTIFIER
894 9 0 9 [LBRAC
895 1 696 5 j IDENTIFIER
896 10 0 9] RBRAC
897 8 0 73) RPARA
898 4 0 93 ; SCOLON
899 6 0 28 } RCURL
900 1 93 10 printf IDENTIFIER
901 7 0 74 (LPARA
902 101 0 16 string_literal S_L
903 8 0 74) RPARA
904 4 0 94 ; SCOLON
905 1 197 3 TraverseList IDENTIFIER
906 7 0 75 (LPARA
907 21 0 7 & BAND
908 1 51 21 list IDENTIFIER
909 17 0 30 , COMMA
910 1 246 3 print IDENTIFIER
911 8 0 75) RPARA
912 4 0 95 ; SCOLON
913 1 93 11 printf IDENTIFIER
914 7 0 76 (LPARA
915 101 0 17 string_literal S_L
916 8 0 76) RPARA
917 4 0 96 ; SCOLON
918 1 197 4 TraverseList IDENTIFIER
919 7 0 77 (LPARA
920 21 0 8 & BAND
921 1 51 22 list IDENTIFIER
922 17 0 31 , COMMA
923 1 350 2 file1yaz IDENTIFIER
924 8 0 77) RPARA
925 4 0 97 ; SCOLON
926 1 47 4 CreateList IDENTIFIER
927 7 0 78 (LPARA
928 21 0 9 & BAND
929 1 712 2 list2 IDENTIFIER
930 8 0 78) RPARA
931 4 0 98 ; SCOLON
932 85 0 3 for FOR
933 7 0 79 (LPARA
934 1 604 15 i IDENTIFIER

935	11	0	44	=	ASSIGNMENT
936	1	590	6	a	IDENTIFIER
937	4	0	99	;	SCOLON
938	1	604	16	i	IDENTIFIER
939	30	0	2	<=	LE_OP
940	1	705	3	z	IDENTIFIER
941	4	0	100	;	SCOLON
942	1	604	17	i	IDENTIFIER
943	34	0	3	++	INC_OP
944	8	0	79	)	RPARA
945	1	118	4	AddList	IDENTIFIER
946	7	0	80	(	LPARA
947	1	604	18	i	IDENTIFIER
948	17	0	32	,	COMMA
949	21	0	10	&	BAND
950	1	712	3	list2	IDENTIFIER
951	8	0	80	)	RPARA
952	4	0	101	;	SCOLON
953	1	93	12	printf	IDENTIFIER
954	7	0	81	(	LPARA
955	101	0	18	string_literal	S_L
956	8	0	81	)	RPARA
957	4	0	102	;	SCOLON
958	1	93	13	printf	IDENTIFIER
959	7	0	82	(	LPARA
960	101	0	19	string_literal	S_L
961	8	0	82	)	RPARA
962	4	0	103	;	SCOLON
963	1	864	2	scanf	IDENTIFIER
964	7	0	83	(	LPARA
965	101	0	20	string_literal	S_L
966	17	0	33	,	COMMA
967	1	685	2	dizi2	IDENTIFIER
968	8	0	83	)	RPARA
969	4	0	104	;	SCOLON
970	85	0	4	for	FOR
971	7	0	84	(	LPARA
972	1	698	2	k	IDENTIFIER
973	11	0	45	=	ASSIGNMENT
974	2	702	4	0	CONSTANT
975	4	0	105	;	SCOLON
976	1	685	3	dizi2	IDENTIFIER
977	9	0	10	[	LBRAC
978	1	698	3	k	IDENTIFIER
979	10	0	10	]	RBRAC
980	33	0	14	!=	NE_OP
981	2	702	5	0	CONSTANT
982	4	0	106	;	SCOLON
983	1	698	4	k	IDENTIFIER
984	34	0	4	++	INC_OP

985 8 0 84) RPARA
986 5 0 30 { LCURL
987 1 434 2 movetorear IDENTIFIER
988 7 0 85 (LPARA
989 21 0 11 & BAND
990 1 712 4 list2 IDENTIFIER
991 17 0 34 , COMMA
992 1 685 4 dizi2 IDENTIFIER
993 9 0 11 [LBRAC
994 1 698 5 k IDENTIFIER
995 10 0 11] RBRAC
996 8 0 85) RPARA
997 4 0 107 ; SCOLON
998 6 0 29 } RCURL
999 1 93 14 printf IDENTIFIER
1000 7 0 86 (LPARA
1001 101 0 21 string_literal S_L
1002 8 0 86) RPARA
1003 4 0 108 ; SCOLON
1004 1 197 5 TraverseList IDENTIFIER
1005 7 0 87 (LPARA
1006 21 0 12 & BAND
1007 1 712 5 list2 IDENTIFIER
1008 17 0 35 , COMMA
1009 1 246 4 print IDENTIFIER
1010 8 0 87) RPARA
1011 4 0 109 ; SCOLON
1012 1 93 15 printf IDENTIFIER
1013 7 0 88 (LPARA
1014 101 0 22 string_literal S_L
1015 8 0 88) RPARA
1016 4 0 110 ; SCOLON
1017 1 197 6 TraverseList IDENTIFIER
1018 7 0 89 (LPARA
1019 21 0 13 & BAND
1020 1 712 6 list2 IDENTIFIER
1021 17 0 36 , COMMA
1022 1 392 2 filemyaz IDENTIFIER
1023 8 0 89) RPARA
1024 4 0 111 ; SCOLON
1025 1 93 16 printf IDENTIFIER
1026 7 0 90 (LPARA
1027 101 0 23 string_literal S_L
1028 8 0 90) RPARA
1029 4 0 112 ; SCOLON
1030 1 93 17 printf IDENTIFIER
1031 7 0 91 (LPARA
1032 101 0 24 string_literal S_L
1033 8 0 91) RPARA
1034 4 0 113 ; SCOLON

1035 1 93 18 printf IDENTIFIER
1036 7 0 92 (LPARA
1037 101 0 25 string_literal S_L
1038 8 0 92) RPARA
1039 4 0 114 ; SCOLON
1040 1 864 3 scanf IDENTIFIER
1041 7 0 93 (LPARA
1042 101 0 26 string_literal S_L
1043 17 0 37 , COMMA
1044 1 690 2 array IDENTIFIER
1045 8 0 93) RPARA
1046 4 0 115 ; SCOLON
1047 1 1047 1 gettime IDENTIFIER
1048 7 0 94 (LPARA
1049 21 0 14 & BAND
1050 1 732 2 t IDENTIFIER
1051 8 0 94) RPARA
1052 4 0 116 ; SCOLON
1053 1 93 19 printf IDENTIFIER
1054 7 0 95 (LPARA
1055 101 0 27 string_literal S_L
1056 17 0 38 , COMMA
1057 1 732 3 t IDENTIFIER
1058 27 0 1 . DIRECT_SEL
1059 1 1059 1 ti_hour IDENTIFIER
1060 17 0 39 , COMMA
1061 1 732 4 t IDENTIFIER
1062 27 0 2 . DIRECT_SEL
1063 1 1063 1 ti_min IDENTIFIER
1064 17 0 40 , COMMA
1065 1 732 5 t IDENTIFIER
1066 27 0 3 . DIRECT_SEL
1067 1 1067 1 ti_sec IDENTIFIER
1068 17 0 41 , COMMA
1069 1 732 6 t IDENTIFIER
1070 27 0 4 . DIRECT_SEL
1071 1 1071 1 ti_hund IDENTIFIER
1072 8 0 95) RPARA
1073 4 0 117 ; SCOLON
1074 1 590 7 a IDENTIFIER
1075 11 0 46 = ASSIGNMENT
1076 2 702 6 0 CONSTANT
1077 4 0 118 ; SCOLON
1078 1 1078 1 strcpy IDENTIFIER
1079 7 0 96 (LPARA
1080 1 715 2 dosyaadi1 IDENTIFIER
1081 17 0 42 , COMMA
1082 101 0 28 string_literal S_L
1083 8 0 96) RPARA
1084 4 0 119 ; SCOLON

1085 85 0 5 for FOR
1086 7 0 97 (LPARA
1087 1 705 4 z IDENTIFIER
1088 11 0 47 = ASSIGNMENT
1089 2 702 7 0 CONSTANT
1090 4 0 120 ; SCOLON
1091 1 705 5 z IDENTIFIER
1092 30 0 3 <= LE_OP
1093 2 1093 1 1000 CONSTANT
1094 4 0 121 ; SCOLON
1095 1 705 6 z IDENTIFIER
1096 34 0 5 ++ INC_OP
1097 8 0 97) RPARA
1098 5 0 31 { LCURL
1099 85 0 6 for FOR
1100 7 0 98 (LPARA
1101 1 696 6 j IDENTIFIER
1102 11 0 48 = ASSIGNMENT
1103 2 702 8 0 CONSTANT
1104 4 0 122 ; SCOLON
1105 1 690 3 array IDENTIFIER
1106 9 0 12 [LBRAC
1107 1 696 7 j IDENTIFIER
1108 10 0 12] RBRAC
1109 33 0 15 != NE_OP
1110 2 702 9 0 CONSTANT
1111 4 0 123 ; SCOLON
1112 1 696 8 j IDENTIFIER
1113 34 0 6 ++ INC_OP
1114 8 0 98) RPARA
1115 5 0 32 { LCURL
1116 1 583 2 naccess IDENTIFIER
1117 7 0 99 (LPARA
1118 1 690 4 array IDENTIFIER
1119 9 0 13 [LBRAC
1120 1 696 9 j IDENTIFIER
1121 10 0 13] RBRAC
1122 17 0 43 , COMMA
1123 21 0 15 & BAND
1124 1 590 8 a IDENTIFIER
1125 17 0 44 , COMMA
1126 1 715 3 dosyaadi1 IDENTIFIER
1127 8 0 99) RPARA
1128 4 0 124 ; SCOLON
1129 6 0 30 } RCURL
1130 6 0 31 } RCURL
1131 1 1047 2 gettime IDENTIFIER
1132 7 0 100 (LPARA
1133 21 0 16 & BAND
1134 1 732 7 t IDENTIFIER

```
1135 8      0      100    )          RPARA
1136 4      0      125    ;          SCOLON
1137 1      93     20     printf     IDENTIFIER
1138 7      0      101    (          LPARA
1139 101    0      29     string_literal  S_L
1140 17     0      45     ,          COMMA
1141 1      732    8      t          IDENTIFIER
1142 27     0      5      .          DIRECT_SEL
1143 1      1059   2      ti_hour    IDENTIFIER
1144 17     0      46     ,          COMMA
1145 1      732    9      t          IDENTIFIER
1146 27     0      6      .          DIRECT_SEL
1147 1      1063   2      ti_min     IDENTIFIER
1148 17     0      47     ,          COMMA
1149 1      732    10     t          IDENTIFIER
1150 27     0      7      .          DIRECT_SEL
1151 1      1067   2      ti_sec     IDENTIFIER
1152 17     0      48     ,          COMMA
1153 1      732    11     t          IDENTIFIER
1154 27     0      8      .          DIRECT_SEL
1155 1      1071   2      ti_hund    IDENTIFIER
1156 8      0      101    )          RPARA
1157 4      0      126    ;          SCOLON
1158 1      590    9      a          IDENTIFIER
1159 11     0      49     =          ASSIGNMENT
1160 2      702    10     0          CONSTANT
1161 4      0      127    ;          SCOLON
1162 85     0      7      for        FOR
1163 7      0      102    (          LPARA
1164 1      696    10     j          IDENTIFIER
1165 11     0      50     =          ASSIGNMENT
1166 2      702    11     0          CONSTANT
1167 4      0      128    ;          SCOLON
1168 1      690    5      array      IDENTIFIER
1169 9      0      14     [          LBRAC
1170 1      696    11     j          IDENTIFIER
1171 10     0      14     ]          RBRAC
1172 33     0      16     !=         NE_OP
1173 2      702    12     0          CONSTANT
1174 4      0      129    ;          SCOLON
1175 1      696    12     j          IDENTIFIER
1176 34     0      7      ++         INC_OP
1177 8      0      102    )          RPARA
1178 5      0      33     {          LCURL
1179 1      583    3      naccess    IDENTIFIER
1180 7      0      103    (          LPARA
1181 1      690    6      array      IDENTIFIER
1182 9      0      15     [          LBRAC
1183 1      696    13     j          IDENTIFIER
1184 10     0      15     ]          RBRAC
```

```
1185 17    0     49    ,          COMMA
1186 21    0     17    &          BAND
1187 1     590   10    a          IDENTIFIER
1188 17    0     50    ,          COMMA
1189 1     715   4     dosyaadi1  IDENTIFIER
1190 8     0     103   )          RPARA
1191 4     0     130   ;          SCOLON
1192 1     590   11    a          IDENTIFIER
1193 34    0     8     ++         INC_OP
1194 4     0     131   ;          SCOLON
1195 6     0     32    }          RCURL
1196 1     93    21    printf     IDENTIFIER
1197 7     0     104   (          LPARA
1198 101   0     30    string_literal  S_L
1199 17    0     51    ,          COMMA
1200 1     590   12    a          IDENTIFIER
1201 8     0     104   )          RPARA
1202 4     0     132   ;          SCOLON
1203 1     590   13    a          IDENTIFIER
1204 11    0     51    =          ASSIGNMENT
1205 2     702   13    0          CONSTANT
1206 4     0     133   ;          SCOLON
1207 1     93    22    printf     IDENTIFIER
1208 7     0     105   (          LPARA
1209 101   0     31    string_literal  S_L
1210 8     0     105   )          RPARA
1211 4     0     134   ;          SCOLON
1212 1     93    23    printf     IDENTIFIER
1213 7     0     106   (          LPARA
1214 101   0     32    string_literal  S_L
1215 8     0     106   )          RPARA
1216 4     0     135   ;          SCOLON
1217 1     864   4     scanf      IDENTIFIER
1218 7     0     107   (          LPARA
1219 101   0     33    string_literal  S_L
1220 17    0     52    ,          COMMA
1221 1     690   7     array      IDENTIFIER
1222 8     0     107   )          RPARA
1223 4     0     136   ;          SCOLON
1224 1     1047  3     gettime    IDENTIFIER
1225 7     0     108   (          LPARA
1226 21    0     18    &          BAND
1227 1     732   12    t          IDENTIFIER
1228 8     0     108   )          RPARA
1229 4     0     137   ;          SCOLON
1230 1     93    24    printf     IDENTIFIER
1231 7     0     109   (          LPARA
1232 101   0     34    string_literal  S_L
1233 17    0     53    ,          COMMA
1234 1     732   13    t          IDENTIFIER
```

```
1235 27     0       9       .       DIRECT_SEL
1236 1      1059    3       ti_hour         IDENTIFIER
1237 17     0       54      ,       COMMA
1238 1      732     14      t       IDENTIFIER
1239 27     0       10      .       DIRECT_SEL
1240 1      1063    3       ti_min IDENTIFIER
1241 17     0       55      ,       COMMA
1242 1      732     15      t       IDENTIFIER
1243 27     0       11      .       DIRECT_SEL
1244 1      1067    3       ti_sec IDENTIFIER
1245 17     0       56      ,       COMMA
1246 1      732     16      t       IDENTIFIER
1247 27     0       12      .       DIRECT_SEL
1248 1      1071    3       ti_hund         IDENTIFIER
1249 8      0       109     )       RPARA
1250 4      0       138     ;       SCOLON
1251 1      1078    2       strcpy IDENTIFIER
1252 7      0       110     (       LPARA
1253 1      720     2       dosyaadi2     IDENTIFIER
1254 17     0       57      ,       COMMA
1255 101    0       35      string_literal  S_L
1256 8      0       110     )       RPARA
1257 4      0       139     ;       SCOLON
1258 85     0       8       for     FOR
1259 7      0       111     (       LPARA
1260 1      705     7       z       IDENTIFIER
1261 11     0       52      =       ASSIGNMENT
1262 2      702     14      0       CONSTANT
1263 4      0       140     ;       SCOLON
1264 1      705     8       z       IDENTIFIER
1265 30     0       4       <=      LE_OP
1266 2      1093    2       1000    CONSTANT
1267 4      0       141     ;       SCOLON
1268 1      705     9       z       IDENTIFIER
1269 34     0       9       ++      INC_OP
1270 8      0       111     )       RPARA
1271 5      0       34      {       LCURL
1272 85     0       9       for     FOR
1273 7      0       112     (       LPARA
1274 1      696     14      j       IDENTIFIER
1275 11     0       53      =       ASSIGNMENT
1276 2      702     15      0       CONSTANT
1277 4      0       142     ;       SCOLON
1278 1      690     8       array   IDENTIFIER
1279 9      0       16      [       LBRAC
1280 1      696     15      j       IDENTIFIER
1281 10     0       16      ]       RBRAC
1282 33     0       17      !=      NE_OP
1283 2      702     16      0       CONSTANT
1284 4      0       143     ;       SCOLON
```

```
1285 1      696   16    j           IDENTIFIER
1286 34     0     10    ++          INC_OP
1287 8      0     112   )           RPARA
1288 5      0     35    {           LCURL
1289 1      583   4     naccess     IDENTIFIER
1290 7      0     113   (           LPARA
1291 1      690   9     array       IDENTIFIER
1292 9      0     17    [           LBRAC
1293 1      696   17    j           IDENTIFIER
1294 10     0     17    ]           RBRAC
1295 17     0     58    ,           COMMA
1296 21     0     19    &           BAND
1297 1      590   14    a           IDENTIFIER
1298 17     0     59    ,           COMMA
1299 1      720   3     dosyaadi2   IDENTIFIER
1300 8      0     113   )           RPARA
1301 4      0     144   ;           SCOLON
1302 6      0     33    }           RCURL
1303 6      0     34    }           RCURL
1304 1      1047  4     gettime     IDENTIFIER
1305 7      0     114   (           LPARA
1306 21     0     20    &           BAND
1307 1      732   17    t           IDENTIFIER
1308 8      0     114   )           RPARA
1309 4      0     145   ;           SCOLON
1310 1      93    25    printf      IDENTIFIER
1311 7      0     115   (           LPARA
1312 101    0     36    string_literal  S_L
1313 17     0     60    ,           COMMA
1314 1      732   18    t           IDENTIFIER
1315 27     0     13    .           DIRECT_SEL
1316 1      1059  4     ti_hour     IDENTIFIER
1317 17     0     61    ,           COMMA
1318 1      732   19    t           IDENTIFIER
1319 27     0     14    .           DIRECT_SEL
1320 1      1063  4     ti_min      IDENTIFIER
1321 17     0     62    ,           COMMA
1322 1      732   20    t           IDENTIFIER
1323 27     0     15    .           DIRECT_SEL
1324 1      1067  4     ti_sec      IDENTIFIER
1325 17     0     63    ,           COMMA
1326 1      732   21    t           IDENTIFIER
1327 27     0     16    .           DIRECT_SEL
1328 1      1071  4     ti_hund     IDENTIFIER
1329 8      0     115   )           RPARA
1330 4      0     146   ;           SCOLON
1331 1      590   15    a           IDENTIFIER
1332 11     0     54    =           ASSIGNMENT
1333 2      702   17    0           CONSTANT
1334 4      0     147   ;           SCOLON
```

1335 85 0 10 for FOR
1336 7 0 116 (LPARA
1337 1 696 18 j IDENTIFIER
1338 11 0 55 = ASSIGNMENT
1339 2 702 18 0 CONSTANT
1340 4 0 148 ; SCOLON
1341 1 690 10 array IDENTIFIER
1342 9 0 18 [LBRAC
1343 1 696 19 j IDENTIFIER
1344 10 0 18] RBRAC
1345 33 0 18 != NE_OP
1346 2 702 19 0 CONSTANT
1347 4 0 149 ; SCOLON
1348 1 696 20 j IDENTIFIER
1349 34 0 11 ++ INC_OP
1350 8 0 116) RPARA
1351 5 0 36 { LCURL
1352 1 583 5 naccess IDENTIFIER
1353 7 0 117 (LPARA
1354 1 690 11 array IDENTIFIER
1355 9 0 19 [LBRAC
1356 1 696 21 j IDENTIFIER
1357 10 0 19] RBRAC
1358 17 0 64 , COMMA
1359 21 0 21 & BAND
1360 1 590 16 a IDENTIFIER
1361 17 0 65 , COMMA
1362 1 720 4 dosyaadi2 IDENTIFIER
1363 8 0 117) RPARA
1364 4 0 150 ; SCOLON
1365 1 590 17 a IDENTIFIER
1366 34 0 12 ++ INC_OP
1367 4 0 151 ; SCOLON
1368 6 0 35 } RCURL
1369 1 93 26 printf IDENTIFIER
1370 7 0 118 (LPARA
1371 101 0 37 string_literal S_L
1372 17 0 66 , COMMA
1373 1 590 18 a IDENTIFIER
1374 8 0 118) RPARA
1375 4 0 152 ; SCOLON
1376 1 590 19 a IDENTIFIER
1377 11 0 56 = ASSIGNMENT
1378 2 702 20 0 CONSTANT
1379 4 0 153 ; SCOLON
1380 1 93 27 printf IDENTIFIER
1381 7 0 119 (LPARA
1382 101 0 38 string_literal S_L
1383 8 0 119) RPARA
1384 4 0 154 ; SCOLON

```
1385 1      93     28     printf  IDENTIFIER
1386 7      0      120    (       LPARA
1387 101    0      39     string_literal  S_L
1388 8      0      120    )       RPARA
1389 4      0      155    ;       SCOLON
1390 1      864    5      scanf   IDENTIFIER
1391 7      0      121    (       LPARA
1392 101    0      40     string_literal  S_L
1393 17     0      67     ,       COMMA
1394 1      690    12     array   IDENTIFIER
1395 8      0      121    )       RPARA
1396 4      0      156    ;       SCOLON
1397 1      1047   5      gettime         IDENTIFIER
1398 7      0      122    (       LPARA
1399 21     0      22     &       BAND
1400 1      732    22     t       IDENTIFIER
1401 8      0      122    )       RPARA
1402 4      0      157    ;       SCOLON
1403 1      93     29     printf  IDENTIFIER
1404 7      0      123    (       LPARA
1405 101    0      41     string_literal  S_L
1406 17     0      68     ,       COMMA
1407 1      732    23     t       IDENTIFIER
1408 27     0      17     .       DIRECT_SEL
1409 1      1059   5      ti_hour         IDENTIFIER
1410 17     0      69     ,       COMMA
1411 1      732    24     t       IDENTIFIER
1412 27     0      18     .       DIRECT_SEL
1413 1      1063   5      ti_min IDENTIFIER
1414 17     0      70     ,       COMMA
1415 1      732    25     t       IDENTIFIER
1416 27     0      19     .       DIRECT_SEL
1417 1      1067   5      ti_sec IDENTIFIER
1418 17     0      71     ,       COMMA
1419 1      732    26     t       IDENTIFIER
1420 27     0      20     .       DIRECT_SEL
1421 1      1071   5      ti_hund         IDENTIFIER
1422 8      0      123    )       RPARA
1423 4      0      158    ;       SCOLON
1424 1      1078   3      strcpy IDENTIFIER
1425 7      0      124    (       LPARA
1426 1      725    2      dosyaadi3    IDENTIFIER
1427 17     0      72     ,       COMMA
1428 101    0      42     string_literal  S_L
1429 8      0      124    )       RPARA
1430 4      0      159    ;       SCOLON
1431 85     0      11     for     FOR
1432 7      0      125    (       LPARA
1433 1      705    10     z       IDENTIFIER
1434 11     0      57     =       ASSIGNMENT
```

1435 2 702 21 0 CONSTANT
1436 4 0 160 ; SCOLON
1437 1 705 11 z IDENTIFIER
1438 30 0 5 <= LE_OP
1439 2 1093 3 1000 CONSTANT
1440 4 0 161 ; SCOLON
1441 1 705 12 z IDENTIFIER
1442 34 0 13 ++ INC_OP
1443 8 0 125) RPARA
1444 5 0 37 { LCURL
1445 85 0 12 for FOR
1446 7 0 126 (LPARA
1447 1 696 22 j IDENTIFIER
1448 11 0 58 = ASSIGNMENT
1449 2 702 22 0 CONSTANT
1450 4 0 162 ; SCOLON
1451 1 690 13 array IDENTIFIER
1452 9 0 20 [LBRAC
1453 1 696 23 j IDENTIFIER
1454 10 0 20] RBRAC
1455 33 0 19 != NE_OP
1456 2 702 23 0 CONSTANT
1457 4 0 163 ; SCOLON
1458 1 696 24 j IDENTIFIER
1459 34 0 14 ++ INC_OP
1460 8 0 126) RPARA
1461 5 0 38 { LCURL
1462 1 583 6 naccess IDENTIFIER
1463 7 0 127 (LPARA
1464 1 690 14 array IDENTIFIER
1465 9 0 21 [LBRAC
1466 1 696 25 j IDENTIFIER
1467 10 0 21] RBRAC
1468 17 0 73 , COMMA
1469 21 0 23 & BAND
1470 1 590 20 a IDENTIFIER
1471 17 0 74 , COMMA
1472 1 725 3 dosyaadi3 IDENTIFIER
1473 8 0 127) RPARA
1474 4 0 164 ; SCOLON
1475 6 0 36 } RCURL
1476 6 0 37 } RCURL
1477 1 1047 6 gettime IDENTIFIER
1478 7 0 128 (LPARA
1479 21 0 24 & BAND
1480 1 732 27 t IDENTIFIER
1481 8 0 128) RPARA
1482 4 0 165 ; SCOLON
1483 1 93 30 printf IDENTIFIER
1484 7 0 129 (LPARA

```
1485 101   0     43    string_literal  S_L
1486 17    0     75    ,         COMMA
1487 1     732   28    t         IDENTIFIER
1488 27    0     21    .         DIRECT_SEL
1489 1     1059  6     ti_hour         IDENTIFIER
1490 17    0     76    ,         COMMA
1491 1     732   29    t         IDENTIFIER
1492 27    0     22    .         DIRECT_SEL
1493 1     1063  6     ti_min IDENTIFIER
1494 17    0     77    ,         COMMA
1495 1     732   30    t         IDENTIFIER
1496 27    0     23    .         DIRECT_SEL
1497 1     1067  6     ti_sec IDENTIFIER
1498 17    0     78    ,         COMMA
1499 1     732   31    t         IDENTIFIER
1500 27    0     24    .         DIRECT_SEL
1501 1     1071  6     ti_hund         IDENTIFIER
1502 8     0     129   )         RPARA
1503 4     0     166   ;         SCOLON
1504 1     590   21    a         IDENTIFIER
1505 11    0     59    =         ASSIGNMENT
1506 2     702   24    0         CONSTANT
1507 4     0     167   ;         SCOLON
1508 85    0     13    for       FOR
1509 7     0     130   (         LPARA
1510 1     696   26    j         IDENTIFIER
1511 11    0     60    =         ASSIGNMENT
1512 2     702   25    0         CONSTANT
1513 4     0     168   ;         SCOLON
1514 1     690   15    array     IDENTIFIER
1515 9     0     22    [         LBRAC
1516 1     696   27    j         IDENTIFIER
1517 10    0     22    ]         RBRAC
1518 33    0     20    !=        NE_OP
1519 2     702   26    0         CONSTANT
1520 4     0     169   ;         SCOLON
1521 1     696   28    j         IDENTIFIER
1522 34    0     15    ++        INC_OP
1523 8     0     130   )         RPARA
1524 5     0     39    {         LCURL
1525 1     583   7     naccess         IDENTIFIER
1526 7     0     131   (         LPARA
1527 1     690   16    array     IDENTIFIER
1528 9     0     23    [         LBRAC
1529 1     696   29    j         IDENTIFIER
1530 10    0     23    ]         RBRAC
1531 17    0     79    ,         COMMA
1532 21    0     25    &         BAND
1533 1     590   22    a         IDENTIFIER
1534 17    0     80    ,         COMMA
```

1535	1	725	4	dosyaadi3	IDENTIFIER
1536	8	0	131	)	RPARA
1537	4	0	170	;	SCOLON
1538	1	590	23	a	IDENTIFIER
1539	34	0	16	++	INC_OP
1540	4	0	171	;	SCOLON
1541	6	0	38	}	RCURL
1542	1	93	31	printf	IDENTIFIER
1543	7	0	132	(	LPARA
1544	101	0	44	string_literal	S_L
1545	17	0	81	,	COMMA
1546	1	590	24	a	IDENTIFIER
1547	8	0	132	)	RPARA
1548	4	0	172	;	SCOLON
1549	1	1549	1	getch	IDENTIFIER
1550	7	0	133	(	LPARA
1551	8	0	133	)	RPARA
1552	4	0	173	;	SCOLON
1553	90	0	2	return	RETURN
1554	2	702	27	0	CONSTANT
1555	4	0	174	;	SCOLON
1556	6	0	39	}	RCURL

Ogr prog birimlerine ayrildi.Simdi isleyen ve islenenler vardir.

operand

1000	3
ti_hund	6
ti_sec	6
ti_min	6
ti_hour	6
t	31
time	1
dosyaadi3	4
dosyaadi2	4
dosyaadi1	4
list2	6
flist	4
z	12
0	27
k	5
j	29
array	16
dizi2	4
20	3

dizi 4
1 1
i 18
10 4
ad 2
a 24
yedek 10
dp 25
FILE 4
pre 6
temp 41
newnode 5
string_literal 44
nodeptr 6
x 19
NULL 17
list 22
List 1
head 12
L 1
Node 1
entry 7
next 24
node 1
ListEntry 1

n2=44 N2=477

operator
---

getch 1
strcpy 3
. 24
gettime 6
scanf 5
feof 1
! 1
++ 16
<= 5
for 13
& 25
double 1
main 1
+ 1
fgetc 4
[] 23
int 3

naccess 7
movetorear 2
filemyaz 2
fclose 5
fputc 3
fopen 5
* 6
file1yaz 2
&& 1
movetofront 2
print 4
visit 2
TraverseList 6
!= 20
while 8
, 81
AddList 4
return 2
if-else 3
printf 31
== 7
if 8
sizeof 1
malloc 1
MakeNode 2
= 60
-> 40
() 133
CreateList 4
void 10
List 6
L 0
Node 9
{} 39
adr_op 20
node 1
struct 4
; 174
ListEntry 7
char 8
typedef 3
include 5
5

n1=60 N1=876

EK 3 PROGRAM ÖRNEĞİ

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

#include "fibsrch.h"

/*
 * If val is found in arr, return the index of its location in arr.
 * Otherwise, return the index of the smallest element greater than val
 */
static int binsrch_geq(const int *arr, int n, int val)
{
register int low, high, mid;
int geq;

  low=0; high=n-1; geq=-1;

  /* binary search for finding the element with value val */
  while(low<=high){
    mid=(low+high)>>1; //(low+high)/2;
    if(val<arr[mid]){
        high=mid-1;
        geq=mid;
    }
    else if(val>arr[mid])
        low=mid+1;
    else
      return mid; /* found */
  }

  return geq; /* not found */
}

int fibsrch(const int *arr, int n, int val)
{
register int k, idx, offs;
static int prevn=-1, prevk=-1;

/*  Precomputed Fibonacci numbers F0 up to F46. This implementation assumes
that the size n
 *  of the input array fits in 4 bytes. Note that F46=1836311903 is the largest
Fibonacci
 *  number that is less or equal to the 4-byte INT_MAX (=2147483647). The next
Fibonacci
 *  number, i.e. F47, is 2971215073 and is larger than INT_MAX, implying that it
does not
 *  fit in a 4 byte integer. Note also that the last array element is INT_MAX rather
than
```

```
 *  F47. This ensures correct operation for n>F46.
 */
const static int Fib[47+1]={0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610,
987, 1597, 2584, 4181, 6765,
          10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229,
832040, 1346269, 2178309, 3524578,
          5702887, 9227465, 14930352, 24157817, 39088169, 63245986,
102334155, 165580141, 267914296,
          433494437, 701408733, 1134903170, 1836311903, INT_MAX};

  /* find the smallest fibonacci number that is greater or equal to n. Store this
   * number to avoid recomputing it in the case of repetitive searches with identical
n.
   */
  if(n!=prevn){

    k=(n>1)? binsrch_geq(Fib, sizeof(Fib)/sizeof(int), n) : 1;
   /* the above binary search call can be substituted by the following code
fragment: */
    {
      register int f0, f1, t;

      for(f0=0, f1=1, k=1; f1<n; t=f1, f1+=f0, f0=t, ++k)
                          ;
    }
    prevk=k;
    prevn=n;
  }
  else k=prevk;

  /* If the sought value is larger than the largest Fibonacci number less than n,
   * care must be taken top ensure that we do not attempt to read beyond the end
   * of the array. If we do need to do this, we pretend that the array is padded
   * with elements larger than the sought value.
   */
  for(offs=0; k>0;  ){
    idx=offs+Fib[--k];

    /* note that at this point k  has already been decremented once */
    if(idx>=n || val<arr[idx]) // index out of bounds or val in 1st part
      continue;
    else if (val>arr[idx]){ // val in 2nd part
      offs=idx;
      --k;
    }
    else // val==arr[idx], found
      return idx;
  }

  return -1; // not found
```

```
}

/* Sample driver program for fibsrch() */

main()
{
int data[]={1, 4, 5, 7, 9, 11, 13, 16, 18, 20, 25, 27, 30, 32, 33, 36, 39, 41, 44, 47,
51, 53, 55};
int i, x, n;

  x=30; n=sizeof(data)/sizeof(int);
  i=fibsrch(data, n, x);
  if(i>=0)
    printf("%d found at index %d\n", x, i);
  else
    printf("%d was not found\n", x);
}
```

EK 4 EK 3 TEKİ PROGRAMIN ÇIKTISI

sira	tid	index	adet	tname	token
1	50	0	1	#	HASH
2	98	0	1	include	INCLUDE
3	100	3	1	<stdio.h>	LIBRARY
4	50	0	2	#	HASH
5	98	0	2	include	INCLUDE
6	100	6	1	<stdlib.h>	LIBRARY
7	50	0	3	#	HASH
8	98	0	3	include	INCLUDE
9	100	9	1	<limits.h>	LIBRARY
10	50	0	4	#	HASH
11	98	0	4	include	INCLUDE
12	100	12	1	"fibsrch.h"	LIBRARY
13	97	0	1	static	STATIC
14	78	0	1	int	INT
15	1	15	1	binsrch_geq	IDENTIFIER
16	7	0	1	(	LPARA
17	69	0	1	const	CONST
18	78	0	2	int	INT
19	52	0	1	adr_op	ADR_OP
20	1	20	1	arr	IDENTIFIER
21	17	0	1	,	COMMA
22	78	0	3	int	INT
23	1	23	1	n	IDENTIFIER
24	17	0	2	,	COMMA
25	78	0	4	int	INT
26	1	26	1	val	IDENTIFIER
27	8	0	1	)	RPARA
28	5	0	1	{	LCURL
29	70	0	1	register	REGISTER
30	78	0	5	int	INT
31	1	31	1	low	IDENTIFIER
32	17	0	3	,	COMMA
33	1	33	1	high	IDENTIFIER
34	17	0	4	,	COMMA
35	1	35	1	mid	IDENTIFIER
36	4	0	1	;	SCOLON
37	78	0	6	int	INT
38	1	38	1	geq	IDENTIFIER
39	4	0	2	;	SCOLON
40	1	31	2	low	IDENTIFIER
41	11	0	1	=	ASSIGNMENT
42	2	42	1	0	CONSTANT
43	4	0	3	;	SCOLON
44	1	33	2	high	IDENTIFIER
45	11	0	2	=	ASSIGNMENT
46	1	23	2	n	IDENTIFIER
>>>1<<<>>>0<<<47	12	0	1	minus	MINUS

```
48    2     48    1     1     CONSTANT
49    4     0     4     ;     SCOLON
50    1     38    2     geq   IDENTIFIER
51    11    0     3     =     ASSIGNMENT
>>>0<<<>>>0<<<52    51    0     1     uminus      UMINUS
53    2     48    2     1     CONSTANT
54    4     0     5     ;     SCOLON
55    86    0     1     while WHILE
56    7     0     2     (     LPARA
57    1     31    3     low   IDENTIFIER
58    30    0     1     <=    LE_OP
59    1     33    3     high  IDENTIFIER
60    8     0     2     )     RPARA
61    5     0     2     {     LCURL
62    1     35    2     mid   IDENTIFIER
63    11    0     4     =     ASSIGNMENT
64    7     0     3     (     LPARA
65    1     31    4     low   IDENTIFIER
66    13    0     1     +     PLUS
67    1     33    4     high  IDENTIFIER
68    8     0     3     )     RPARA
69    37    0     1     >>    RIGHT_OP
70    2     48    3     1     CONSTANT
71    4     0     6     ;     SCOLON
//(low+high)/2;
72    84    0     1     if    IF
73    7     0     4     (     LPARA
74    1     26    2     val   IDENTIFIER
75    18    0     1     <     LT
76    1     20    2     arr   IDENTIFIER
77    9     0     1     [     LBRAC
78    1     35    3     mid   IDENTIFIER
79    10    0     1     ]     RBRAC
80    8     0     4     )     RPARA
81    5     0     3     {     LCURL
82    1     33    5     high  IDENTIFIER
83    11    0     5     =     ASSIGNMENT
84    1     35    4     mid   IDENTIFIER
>>>1<<<>>>0<<<85    12    0     2     minus MINUS
86    2     48    4     1     CONSTANT
87    4     0     7     ;     SCOLON
88    1     38    3     geq   IDENTIFIER
89    11    0     6     =     ASSIGNMENT
90    1     35    5     mid   IDENTIFIER
91    4     0     8     ;     SCOLON
92    6     0     1     }     RCURL
93    88    0     1     else  ELSE
94    84    0     1     if    IF
95    7     0     5     (     LPARA
96    1     26    3     val   IDENTIFIER
```

```
97    19    0     1     >          GT
98    1     20    3     arr        IDENTIFIER
99    9     0     2     [          LBRAC
100   1     35    6     mid        IDENTIFIER
101   10    0     2     ]          RBRAC
102   8     0     5     )          RPARA
103   1     31    5     low        IDENTIFIER
104   11    0     7     =          ASSIGNMENT
105   1     35    7     mid        IDENTIFIER
106   13    0     2     +          PLUS
107   2     48    5     1          CONSTANT
108   4     0     9     ;          SCOLON
109   88    0     2     else       ELSE
110   90    0     1     return     RETURN
111   1     35    8     mid        IDENTIFIER
112   4     0     10    ;          SCOLON
113   6     0     2     }          RCURL
114   90    0     2     return     RETURN
115   1     38    4     geq        IDENTIFIER
116   4     0     11    ;          SCOLON
117   6     0     3     }          RCURL
118   78    0     7     int        INT
119   1     119   1     fibsrch    IDENTIFIER
120   7     0     6     (          LPARA
121   69    0     2     const      CONST
122   78    0     8     int        INT
123   52    0     2     adr_op     ADR_OP
124   1     20    4     arr        IDENTIFIER
125   17    0     5     ,          COMMA
126   78    0     9     int        INT
127   1     23    3     n          IDENTIFIER
128   17    0     6     ,          COMMA
129   78    0     10    int        INT
130   1     26    4     val        IDENTIFIER
131   8     0     6     )          RPARA
132   5     0     4     {          LCURL
133   70    0     2     register   REGISTER
134   78    0     11    int        INT
135   1     135   1     k          IDENTIFIER
136   17    0     7     ,          COMMA
137   1     137   1     idx        IDENTIFIER
138   17    0     8     ,          COMMA
139   1     139   1     offs       IDENTIFIER
140   4     0     12    ;          SCOLON
141   97    0     2     static     STATIC
142   78    0     12    int        INT
143   1     143   1     prevn      IDENTIFIER
144   11    0     8     =          ASSIGNMENT
>>>0<<<>>>0<<<145   51    0     2     uminus     UMINUS
146   2     48    6     1          CONSTANT
```

147 17 0 9 , COMMA
148 1 148 1 prevk IDENTIFIER
149 11 0 9 = ASSIGNMENT
>>>0<<<>>>0<<<150 51 0 3 uminus UMINUS
151 2 48 7 1 CONSTANT
152 4 0 13 ; SCOLON
153 69 0 3 const CONST
154 97 0 3 static STATIC
155 78 0 13 int INT
156 1 156 1 Fib IDENTIFIER
157 9 0 3 [LBRAC
158 2 158 1 47 CONSTANT
159 13 0 3 + PLUS
160 2 48 8 1 CONSTANT
161 10 0 3] RBRAC
162 11 0 10 = ASSIGNMENT
163 5 0 5 { LCURL
164 2 42 2 0 CONSTANT
165 17 0 10 , COMMA
166 2 48 9 1 CONSTANT
167 17 0 11 , COMMA
168 2 48 10 1 CONSTANT
169 17 0 12 , COMMA
170 2 170 1 2 CONSTANT
171 17 0 13 , COMMA
172 2 172 1 3 CONSTANT
173 17 0 14 , COMMA
174 2 174 1 5 CONSTANT
175 17 0 15 , COMMA
176 2 176 1 8 CONSTANT
177 17 0 16 , COMMA
178 2 178 1 13 CONSTANT
179 17 0 17 , COMMA
180 2 180 1 21 CONSTANT
181 17 0 18 , COMMA
182 2 182 1 34 CONSTANT
183 17 0 19 , COMMA
184 2 184 1 55 CONSTANT
185 17 0 20 , COMMA
186 2 186 1 89 CONSTANT
187 17 0 21 , COMMA
188 2 188 1 144 CONSTANT
189 17 0 22 , COMMA
190 2 190 1 233 CONSTANT
191 17 0 23 , COMMA
192 2 192 1 377 CONSTANT
193 17 0 24 , COMMA
194 2 194 1 610 CONSTANT
195 17 0 25 , COMMA
196 2 196 1 987 CONSTANT

197	17	0	26	,	COMMA
198	2	198	1	1597	CONSTANT
199	17	0	27	,	COMMA
200	2	200	1	2584	CONSTANT
201	17	0	28	,	COMMA
202	2	202	1	4181	CONSTANT
203	17	0	29	,	COMMA
204	2	204	1	6765	CONSTANT
205	17	0	30	,	COMMA
206	2	206	1	10946	CONSTANT
207	17	0	31	,	COMMA
208	2	208	1	17711	CONSTANT
209	17	0	32	,	COMMA
210	2	210	1	28657	CONSTANT
211	17	0	33	,	COMMA
212	2	212	1	46368	CONSTANT
213	17	0	34	,	COMMA
214	2	214	1	75025	CONSTANT
215	17	0	35	,	COMMA
216	2	216	1	121393	CONSTANT
217	17	0	36	,	COMMA
218	2	218	1	196418	CONSTANT
219	17	0	37	,	COMMA
220	2	220	1	317811	CONSTANT
221	17	0	38	,	COMMA
222	2	222	1	514229	CONSTANT
223	17	0	39	,	COMMA
224	2	224	1	832040	CONSTANT
225	17	0	40	,	COMMA
226	2	226	1	1346269	CONSTANT
227	17	0	41	,	COMMA
228	2	228	1	2178309	CONSTANT
229	17	0	42	,	COMMA
230	2	230	1	3524578	CONSTANT
231	17	0	43	,	COMMA
232	2	232	1	5702887	CONSTANT
233	17	0	44	,	COMMA
234	2	234	1	9227465	CONSTANT
235	17	0	45	,	COMMA
236	2	236	1	14930352	CONSTANT
237	17	0	46	,	COMMA
238	2	238	1	24157817	CONSTANT
239	17	0	47	,	COMMA
240	2	240	1	39088169	CONSTANT
241	17	0	48	,	COMMA
242	2	242	1	63245986	CONSTANT
243	17	0	49	,	COMMA
244	2	244	1	102334155	CONSTANT
245	17	0	50	,	COMMA
246	2	246	1	165580141	CONSTANT

247	17	0	51	,	COMMA
248	2	248	1	267914296	CONSTANT
249	17	0	52	,	COMMA
250	2	250	1	433494437	CONSTANT
251	17	0	53	,	COMMA
252	2	252	1	701408733	CONSTANT
253	17	0	54	,	COMMA
254	2	254	1	1134903170	CONSTANT
255	17	0	55	,	COMMA
256	2	256	1	1836311903	CONSTANT
257	17	0	56	,	COMMA
258	1	258	1	INT_MAX	IDENTIFIER
259	6	0	4	}	RCURL
260	4	0	14	;	SCOLON
261	84	0	1	if	IF
262	7	0	7	(	LPARA
263	1	23	4	n	IDENTIFIER
264	33	0	1	!=	NE_OP
265	1	143	2	prevn	IDENTIFIER
266	8	0	7	)	RPARA
267	5	0	6	{	LCURL
268	1	135	2	k	IDENTIFIER
269	11	0	11	=	ASSIGNMENT
270	7	0	8	(	LPARA
271	1	23	5	n	IDENTIFIER
272	19	0	2	>	GT
273	2	48	11	1	CONSTANT
274	8	0	8	)	RPARA
275	23	0	1	?	QM
276	1	15	2	binsrch_geq	IDENTIFIER
277	7	0	9	(	LPARA
278	1	156	2	Fib	IDENTIFIER
279	17	0	57	,	COMMA
280	71	0	1	sizeof	SIZEOF
281	7	0	10	(	LPARA
282	1	156	3	Fib	IDENTIFIER
283	8	0	9	)	RPARA
284	15	0	1	/	DIVISION
285	71	0	2	sizeof	SIZEOF
286	7	0	11	(	LPARA
287	78	0	14	int	INT
288	8	0	10	)	RPARA
289	17	0	58	,	COMMA
290	1	23	6	n	IDENTIFIER
291	8	0	11	)	RPARA
292	25	0	1	:	COLON
293	2	48	12	1	CONSTANT
294	4	0	15	;	SCOLON
295	5	0	7	{	LCURL
296	70	0	3	register	REGISTER

297	78	0	15	int	INT
298	1	298	1	f0	IDENTIFIER
299	17	0	59	,	COMMA
300	1	300	1	f1	IDENTIFIER
301	17	0	60	,	COMMA
302	1	302	1	t	IDENTIFIER
303	4	0	16	;	SCOLON
304	85	0	1	for	FOR
305	7	0	12	(	LPARA
306	1	298	2	f0	IDENTIFIER
307	11	0	12	=	ASSIGNMENT
308	2	42	3	0	CONSTANT
309	17	0	61	,	COMMA
310	1	300	2	f1	IDENTIFIER
311	11	0	13	=	ASSIGNMENT
312	2	48	13	1	CONSTANT
313	17	0	62	,	COMMA
314	1	135	3	k	IDENTIFIER
315	11	0	14	=	ASSIGNMENT
316	2	48	14	1	CONSTANT
317	4	0	17	;	SCOLON
318	1	300	3	f1	IDENTIFIER
319	18	0	2	<	LT
320	1	23	7	n	IDENTIFIER
321	4	0	18	;	SCOLON
322	1	302	2	t	IDENTIFIER
323	11	0	15	=	ASSIGNMENT
324	1	300	4	f1	IDENTIFIER
325	17	0	63	,	COMMA
326	1	300	5	f1	IDENTIFIER
327	41	0	1	+=	ADD_ASSIGN
328	1	298	3	f0	IDENTIFIER
329	17	0	64	,	COMMA
330	1	298	4	f0	IDENTIFIER
331	11	0	16	=	ASSIGNMENT
332	1	302	3	t	IDENTIFIER
333	17	0	65	,	COMMA
334	34	0	1	++	INC_OP
335	1	135	4	k	IDENTIFIER
336	8	0	12	)	RPARA
337	4	0	19	;	SCOLON
338	6	0	5	}	RCURL
339	1	148	2	prevk	IDENTIFIER
340	11	0	17	=	ASSIGNMENT
341	1	135	5	k	IDENTIFIER
342	4	0	20	;	SCOLON
343	1	143	3	prevn	IDENTIFIER
344	11	0	18	=	ASSIGNMENT
345	1	23	8	n	IDENTIFIER
346	4	0	21	;	SCOLON

```
347   6     0     6     }       RCURL
348   88    0     3     else    ELSE
349   1     135   6     k       IDENTIFIER
350   11    0     19    =       ASSIGNMENT
351   1     148   3     prevk   IDENTIFIER
352   4     0     22    ;       SCOLON
353   85    0     2     for     FOR
354   7     0     13    (       LPARA
355   1     139   2     offs    IDENTIFIER
356   11    0     20    =       ASSIGNMENT
357   2     42    4     0       CONSTANT
358   4     0     23    ;       SCOLON
359   1     135   7     k       IDENTIFIER
360   19    0     3     >       GT
361   2     42    5     0       CONSTANT
362   4     0     24    ;       SCOLON
363   8     0     13    )       RPARA
364   5     0     8     {       LCURL
365   1     137   2     idx     IDENTIFIER
366   11    0     21    =       ASSIGNMENT
367   1     139   3     offs    IDENTIFIER
368   13    0     4     +       PLUS
369   1     156   4     Fib     IDENTIFIER
370   9     0     4     [       LBRAC
371   35    0     1     --      DEC_OP
372   1     135   8     k       IDENTIFIER
373   10    0     4     ]       RBRAC
374   4     0     25    ;       SCOLON
375   84    0     1     if      IF
376   7     0     14    (       LPARA
377   1     137   3     idx     IDENTIFIER
378   31    0     1     >=      GE_OP
379   1     23    9     n       IDENTIFIER
380   29    0     1     ||      OR_OP
381   1     26    5     val     IDENTIFIER
382   18    0     3     <       LT
383   1     20    5     arr     IDENTIFIER
384   9     0     5     [       LBRAC
385   1     137   4     idx     IDENTIFIER
386   10    0     5     ]       RBRAC
387   8     0     14    )       RPARA
// index out of bounds or val in 1st part
388   95    0     1     continue      CONTINUE
389   4     0     26    ;       SCOLON
390   88    0     4     else    ELSE
391   84    0     1     if      IF
392   7     0     15    (       LPARA
393   1     26    6     val     IDENTIFIER
394   19    0     4     >       GT
395   1     20    6     arr     IDENTIFIER
```

```
396   9     0     6     [     LBRAC
397   1     137   5     idx   IDENTIFIER
398   10    0     6     ]     RBRAC
399   8     0     15    )     RPARA
400   5     0     9     {     LCURL
// val in 2nd part
401   1     139   4     offs  IDENTIFIER
402   11    0     22    =     ASSIGNMENT
403   1     137   6     idx   IDENTIFIER
404   4     0     27    ;     SCOLON
405   35    0     2     --    DEC_OP
406   1     135   9     k     IDENTIFIER
407   4     0     28    ;     SCOLON
408   6     0     7     }     RCURL
409   88    0     5     else  ELSE
// val==arr[idx], found
410   90    0     3     return RETURN
411   1     137   7     idx   IDENTIFIER
412   4     0     29    ;     SCOLON
413   6     0     8     }     RCURL
414   90    0     4     return RETURN
>>>0<<<>>>0<<<415   51    0     4     uminus      UMINUS
416   2     48    15    1     CONSTANT
417   4     0     30    ;     SCOLON
// not found
418   6     0     9     }     RCURL
419   1     419   1     main  IDENTIFIER
420   7     0     16    (     LPARA
421   8     0     16    )     RPARA
422   5     0     10    {     LCURL
423   78    0     16    int   INT
424   1     424   1     data  IDENTIFIER
425   9     0     7     [     LBRAC
426   10    0     7     ]     RBRAC
427   11    0     23    =     ASSIGNMENT
428   5     0     11    {     LCURL
429   2     48    16    1     CONSTANT
430   17    0     66    ,     COMMA
431   2     431   1     4     CONSTANT
432   17    0     67    ,     COMMA
433   2     174   2     5     CONSTANT
434   17    0     68    ,     COMMA
435   2     435   1     7     CONSTANT
436   17    0     69    ,     COMMA
437   2     437   1     9     CONSTANT
438   17    0     70    ,     COMMA
439   2     439   1     11    CONSTANT
440   17    0     71    ,     COMMA
441   2     178   2     13    CONSTANT
442   17    0     72    ,     COMMA
```

443	2	443	1	16	CONSTANT
444	17	0	73	,	COMMA
445	2	445	1	18	CONSTANT
446	17	0	74	,	COMMA
447	2	447	1	20	CONSTANT
448	17	0	75	,	COMMA
449	2	449	1	25	CONSTANT
450	17	0	76	,	COMMA
451	2	451	1	27	CONSTANT
452	17	0	77	,	COMMA
453	2	453	1	30	CONSTANT
454	17	0	78	,	COMMA
455	2	455	1	32	CONSTANT
456	17	0	79	,	COMMA
457	2	457	1	33	CONSTANT
458	17	0	80	,	COMMA
459	2	459	1	36	CONSTANT
460	17	0	81	,	COMMA
461	2	461	1	39	CONSTANT
462	17	0	82	,	COMMA
463	2	463	1	41	CONSTANT
464	17	0	83	,	COMMA
465	2	465	1	44	CONSTANT
466	17	0	84	,	COMMA
467	2	158	2	47	CONSTANT
468	17	0	85	,	COMMA
469	2	469	1	51	CONSTANT
470	17	0	86	,	COMMA
471	2	471	1	53	CONSTANT
472	17	0	87	,	COMMA
473	2	184	2	55	CONSTANT
474	6	0	10	}	RCURL
475	4	0	31	;	SCOLON
476	78	0	17	int	INT
477	1	477	1	i	IDENTIFIER
478	17	0	88	,	COMMA
479	1	479	1	x	IDENTIFIER
480	17	0	89	,	COMMA
481	1	23	10	n	IDENTIFIER
482	4	0	32	;	SCOLON
483	1	479	2	x	IDENTIFIER
484	11	0	24	=	ASSIGNMENT
485	2	453	2	30	CONSTANT
486	4	0	33	;	SCOLON
487	1	23	11	n	IDENTIFIER
488	11	0	25	=	ASSIGNMENT
489	71	0	3	sizeof	SIZEOF
490	7	0	17	(	LPARA
491	1	424	2	data	IDENTIFIER
492	8	0	17	)	RPARA

493	15	0	2	/	DIVISION
494	71	0	4	sizeof	SIZEOF
495	7	0	18	(	LPARA
496	78	0	18	int	INT
497	8	0	18	)	RPARA
498	4	0	34	;	SCOLON
499	1	477	2	i	IDENTIFIER
500	11	0	26	=	ASSIGNMENT
501	1	119	2	fibsrch	IDENTIFIER
502	7	0	19	(	LPARA
503	1	424	3	data	IDENTIFIER
504	17	0	90	,	COMMA
505	1	23	12	n	IDENTIFIER
506	17	0	91	,	COMMA
507	1	479	3	x	IDENTIFIER
508	8	0	19	)	RPARA
509	4	0	35	;	SCOLON
510	84	0	1	if	IF
511	7	0	20	(	LPARA
512	1	477	3	i	IDENTIFIER
513	31	0	2	>=	GE_OP
514	2	42	6	0	CONSTANT
515	8	0	20	)	RPARA
516	1	516	1	printf	IDENTIFIER
517	7	0	21	(	LPARA
518	101	0	1	string_literal	S_L
519	17	0	92	,	COMMA
520	1	479	4	x	IDENTIFIER
521	17	0	93	,	COMMA
522	1	477	4	i	IDENTIFIER
523	8	0	21	)	RPARA
524	4	0	36	;	SCOLON
525	88	0	6	else	ELSE
526	1	516	2	printf	IDENTIFIER
527	7	0	22	(	LPARA
528	101	0	2	string_literal	S_L
529	17	0	94	,	COMMA
530	1	479	5	x	IDENTIFIER
531	8	0	22	)	RPARA
532	4	0	37	;	SCOLON
533	6	0	11	}	RCURL

Ogr prog birimlerine ayrildi.Simdi isleyen ve islenenler vardir.

operand
---

string_literal 2
x 5
i 4
53 1
51 1
44 1
41 1
39 1
36 1
33 1
32 1
30 2
27 1
25 1
20 1
18 1
16 1
11 1
9 1
7 1
4 1
data 3
t 3
f1 5
f0 4
INT_MAX 1
1836311903 1
1134903170 1
701408733 1
433494437 1
267914296 1
165580141 1
102334155 1
63245986 1
39088169 1
24157817 1
14930352 1
9227465 1
5702887 1
3524578 1
2178309 1
1346269 1
832040 1
514229 1
317811 1
196418 1
121393 1
75025 1
46368 1
28657 1

17711 1
10946 1
6765 1
4181 1
2584 1
1597 1
987 1
610 1
377 1
233 1
144 1
89 1
55 2
34 1
21 1
13 2
8 1
5 2
3 1
2 1
47 2
Fib 4
prevk 3
prevn 3
offs 4
idx 7
k 9
1 16
0 6
geq 4
mid 8
high 5
low 5
val 6
n 12
arr 6

n2=86 N2=193

operator
---

printf 2
main 1
continue 1
|| 1
>= 2
-- 2

```
++      1
+=      1
for     2
:       1
/       2
sizeof  4
?       1
!=      1
fibsrch2
return  4
>       4
if-else 6
{}              11
[]              7
<       3
if      0
>>      1
+       4
<=      1
while   1
uminus          4
minus   2
=       26
;       37
register        3
()              22
,       94
adr_op          2
const   3
binsrch_geq     2
int     18
static  3
include         4
#       4

n1=40                   N1=290
```