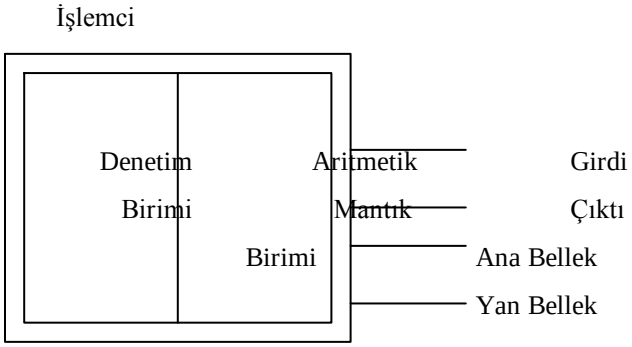
C DİLİ KULLANARAK

BİLGİSAYAR PROGRAMLAMA

ERCİYES ÜNİVERSİTESİ
KONTROL VE BİLGİSAYAR MÜHENDİSLİĞİ

GiRiŞ

Bilgisayar, verileri saklayan , bunlar üzerinde çok hızlı işlem yapan ve istenen verileri sunan bir aygıttır. Donanım (hardware) ve yazılım(software) diye iki bölüme ayrlır. Donanım bilgisayarın fiziksel bileşenleridir. Yazılım ise donanımı oluşturan bileşenlerin çalışmasını ve işlevlerini yerine getirmesini sağlayan programlardır.



Girdi birimleri : Veri ve program girilmesini sağlar. Klavye, fare, kart okuyucu ...

Çıktı birimleri : İstenen verilerin kullanıcıya sunulduğu ortam. Ekran, yazıcı...

Ana Bellek : Programların ve işlenen verilerin geçici olarak saklandığı birim.

Yan bellek : Bilgilerin (veri, program) kalıcı olarak saklandığı ortamlar. Disket, disk, manyetik şerit.

Bilgisayar broşüründe olan kavramlar, bit, byte, ... RAM, ROM...

- Bu günkü anlamda ilk bilgisayar ENIAC [Electronic Numeric integrator and computer]
  30 ton, kablolar ile programlama
- 1842, Charles Babbag , analitik makine tasarlıyor. Programlanabilir bir cihaz Öncesi Hesap Makinesi. Ada Agusta programcısı.

Bilgisayarlar :

1. Kusak 39 - 58 Vakum tüp ile çalışıyor
2. Kusak 58 - 64 Transistör
3. Kusak 64 - 75 Entegre
4. Kusak 75 - --- Yüksek ölçekli entegre

- Micro computer (PC) (bu gün 50 - MIPS)
- Workstation
- Mini Computer
- Main Frame (50 lerde 50 IPS)
- Super Computer

Bilgisayarın yapısı

- Bellek (...., birimi byte bit)
- CPU (Bilgiyi işleyen kısım bellekten okur - yazar)
- Denetim Birimi (Hangi işlem, ne zaman yapılacak, belirler (gerekli işaretleri üretir))
- Giriş/Çıkış : klavye (veya benzer)
  : ekran (veya benzer)
- İkincil (yardımcı) bellek (Kütük olarak saklı bilgiler.)
- Hardware - Software

Bilgisayarlar kendisine sorulan sorulara hemen cevap veren, bir sürü problemi çözen bir aygıt değildir. Bilgisayarda yapılan her tür iş, ilk önce insan aklının süzgecinden geçiyor, insanlar tarafından etraflıca tasarlanıyor, planlanıp programlanıyor [1].

ilk yapılan bilgisayarın karşına geçip hemen
en eski soruyu sormuşlar "Tanrı var mı".
Bilgisayar kısa bir düşünmeden sonra "Evet artık var".

Bu nedenle, önce bilgisayara problemin çözümü öğretilmelidir. Fakat bunun için bizim problemi çözmemiz gerekir. Ve daha sonra bir programlama dili ile bu çözüm bilgisayara aktarılmalıdır.

1- Problem Çözme ve Algoritmalar

1.1 Problem Çözme

Problem çözmede, soruna hemen girişmek yerine, dikkatli ve sistematik yaklaşım ilke olmalıdır. Problem iyice anlaşılmalı ve mümkün olduğu kadar küçük parçalara ayırılmaladır.
Descartes tarafından "Discourse on Method" isimli kitabında anlatılan problem çözme teknikleri;[2]

1. Doğruluğu kesin olarak kanıtlanmadıkça, hiçbir şeyi doğru olarak kabul etmeyin; tahmin ve önyargılardan kaçının.
2. Karşılaştığınız her güçlüğü mümkün olduğu kadar çok parçaya bölün.
3. Düzenli bir biçimde düşünün; anlaşılması en kolay olan şeylerle başlayıp yavaş yavaş daha zor ve karmaşık olanlara doğru ilerleyiniz.
4. Olaya bakışınız çok genel, hazırladığınız ayrıntılı liste ise hiçbir şeyi dışarıda bırakmayacak kadar kusursuz ve eksiksiz olsun.

1.2 Algoritmalar

Belirli bir görevi yerine getiren sonlu sayıdaki işlemler dizisidir.

İ.S. 9.yy da İranlı Musaoğlu Horzumlu Mehmet (Alharezmi adını araplar takmıştır) problemlerin çözümü için genel kurallar oluşturdu. Algoritma Alharezmi'nin Latince okunuşu.

Her algoritma aşağıdaki kriterleri sağlamalıdır.

1. Girdi: Sıfır veya daha fazla değer dışarıdan verilmeli.

2. Çıktı: En azından bir değer üretilmeli.

3. Açıklık: Her işlem (komut) açık olmalı ve farklı anlamlar içermemeli.

4. Sonluluk: Her türlü olasılık için algoritma sonlu adımda bitmeli.

5. Etkinlik: Her komut kişinin kalem ve kağıt ile yürütebileceği kadar basit olmalıdır.

Not: Bir program için 4. özellik geçerli değil. işletim sistemleri gibi program sonsuza dek çalışırlar .

Örnek 1.2.1 : 1'den 100'e kadar olan sayıların toplamını veren algoritma.

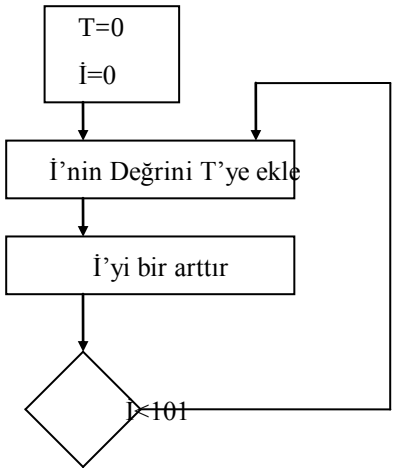
1. Toplam T, sayılar da i diye çağırılsın.
2. Başlangıçta T'nin değeri 0 ve i'nin değeri 1 olsun.
3. i'nin değerini T'ye ekle.
4. i'nin değerini 1 arttır.
5. Eğer i'nin değeri 100'den büyük değil ise 3. adıma git.
6. T'nin değerini yaz.

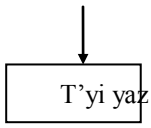
Algoritmaların yazım dili değişik olabilir. Günlük konuşma diline yakın bir dil olabileceği gibi simgelere dayalı da olabilir. Akış şeması eskiden beri kullanıla gelen bir yapıdır. Algoritmayı yazarken farklı anlamlar taşıyan değişik şekildeki kutulardan yararlanılır. Yine aynı amaç için kullanılan programlama diline yakın bir (sözde kod = pseudo code) dil , bu kendimize özgü de olabilir, kullanılabilir.

Aynı algoritmayı aşağıdaki gibi yazabiliriz.

1. T=0 ve i=0
2. i'nin değerini T'ye ekle.
3. i'yi 1 arttır.
4. i<101 ise 2.adıma git.
5. T'nin değerini yaz.

Algoritmayı bir de akış şeması ile gerçekleyelim.





Örnek 1.2.2 : $ax^2+bx+c=0$ tipi bir denklemin köklerini veren algoritma.

Girdi : a, b ve c katsayıları Çıktı : denklemin kökleri

1. a, b ve c katsayılarını al.
2. $D = b^2-4ac$ değerini hesapla.
3. D<0 ise gerçel kök yok. 7. adıma git.
4. $x1 = (-b + \sqrt{D}) / (2a)$
5. $x2 = (-b - \sqrt{D}) / (2a)$
6. değerlerini yaz.
7. Dur.

Döngü Gösterimi

Tekrarlanan adımlar

n. Koşul sağlandığı sürece

n.1 ...

n.2 ... tekrarlanan adımlar

n.3 ...

Örnek 1.2.3 : İki tamsayının çarpma işlemini sadece toplama işlemi kullanarak gerçekleyin.

Girdi : iki tamsayı

Çıktı : sayıların çarpımı

1. a ve b sayılarını oku
2. c =0
3. b>0 olduğu sürece tekrarla

.3.1. c=c + a

3.2. b = b-1

4. c değerini yaz ve dur

Örnek 1.2.4 : Bir tamsayının faktoriyelini hesaplayınız.

Girdi : Bir tamsayı

Çıktı : sayının faktoriyel

İlgili formul: Faktoriyel(n)=1*2*...*n

1. n değerini oku
2. F=1

3. n >1 olduğu sürece tekrarla

   .3.1. F=F*n

   3.2. n= n-1

4. F değerini yaz

Örnek 1.2.5 : İki tamsayının bölme işlemini sadece çıkarma işlemi kullanarak gerçekleyin. Bölüm ve kalanın ne olduğu bulunacak.

1. a ve b değerlerini oku
2. m=0
3. a>=b olduğu sürece tekrarla

   3.1 a=a-b

   3.2 m = m + 1

4. kalan a ve bölüm m 'yi yaz

Örnek 1.2.6 : 100 tane sayıyı okuyup, ortalamasını bul

1. T=0, i=0
2. i<101 olduğu sürece tekrarla

   2.1 m değerini oku

   2.2 T = T + m

   2.3 i = i + 1

3. T = T / 100
4. Ortalama T 'yi yaz
5. Dur

Örnek 1.2.7 : Bir sınava giren öğrencilerin not ortalamasının hesaplanması

1. Tüm sınav kağıtlarını inceleyip notların toplamını hesapla
2. Ortalamayı notların toplamını incelenen sınav kağıdına bölerek hesapla
3. Ortalamayı yaz.

1. Notların toplamını ve incelenen sınav kağıdı sayısını sıfır kabul et
2. Sıradaki sınav kağıdının notunu notların toplamına ekle
3. İncelenen sınav kağıdı sayısını Bir arttır
4. İncelenecek sınav kağıdı var ise 2. Adıma git
5. Ortalamayı notların toplamını incelenen sınav kağıdına bölerek hasapla
6. Ortalamayı yaz

1. Notların toplamını ve incelenen sınav kağıdı sayısını sıfır kabul et

2. Her bir sınav kağıdı için
    2.1. Sıradaki sınav kağıdının notunu notların toplamına ekle
    2.2. İncelenen sınav kağıdı sayısını bir arttır
3. Ortalamayı notların toplamını incelenen sınav kağıdına bölerek hesapla
4. Ortalamayı yaz

Koşul Gösterimi

n. Koşul doğru ise
    n.D.1
    n.D.2 doğru olduğunda işlenen adımlar
    n.D.3
aksi halde
    n.Y.1
    n.Y.2 yanlış olduğunda işlenen adımlar
    n.Y.3

Kök bulma örneğinde 3. Adımı tekrar yazarsak

3. D>=0 ise
    3.D.1 $x1 = (-b + \sqrt{D}) / (2a)$
    3.D.2 $x2 = (-b - \sqrt{D}) / (2a)$

aksi halde
    3.Y.1 Reel kök yoktur

Sorular:

* Girilen üç sayıdan en büyüğünü bulan algoritmayı yazınız.
* Tamsayılarda üs alma işlemini gerçekleştiren algoritmayı yazınız (a^b).
* 1-100 arasında tutulan bir sayıyı tahmin eden algoritmayı yazınız.

Örnek 1.2.8 : Aracın otopark ücretinin hesaplanması. Araçların en fazla 24 saat kaldığını varsayın.

0 - 2 saat	150 bin
2 - 8 saat	300 bin
8-24 saat	500 bin

1. Aracın kaç saat kaldığını öğren (t olsun).
2. t <= 2 ise
   2.D.1. ücret = 150 bin

Aksi halde
   2.Y.1. t<=8 ise
      2.Y.1.D.1. ücret = 300 bin
   Aksi halde
      2.Y.1.Y.1. ücret = 500 bin

3. ücreti yaz
4. Dur

Örnek 1.2.9: Sınavdaki en büyük notun bulan algoritma.

1. En büyük = ilk sınav kağıdındaki not (ya da olabilecek en düşük değer kabul edilebilir).
2. İncelenecek sınav kağıdı var ise
   2.1 Sınav kağıdındaki not > En büyük ise En büyük = Sınav kağıdındaki not

3. En büyük değerini yaz.
4. Dur

Algoritmanın yazımı daha simgesel olabilir. N_i i. Öğrencinin notu olsun.

1. EB = N1
2. i = 2
3. İncelenecek sınav kağıdı var ise
   3.1 Ni>EB => EB = Ni
   3.2 i = i + 1

4. EB' yi yaz.
5. Dur

Örnek 1.2.10 : Programın C dili ile yazılıp çalışır hale getirilmesi.

1. Programı bilgisayara gir
2. Kaynak dosya olarak kaydet
3. Kaynak dosyayı **derle (compile)**
4. Derleme sonucunda hata var ise
   4.1 Hataları düzelt
   4.2 3. Adıma git
5. Oluşan amaç dosyasına diğer dosyaları **bağla (link)**
6. Bağlama sonucunda hata var ise

6.1. Hataları düzelt
6.2. Hatalar kaynak dosya ile ilgili ise 2. adıma aksi halde 5. adıma git
7. Program çalıştırılmaya hazır

2- Programlamaya Giriş

Program : Belirli bir problemi çözmek için bir bilgisayar dili kullanılarak yazılmış deyimler dizisi.
Önceki bölümde bir problemin çözümü ile ilgili teknikler sunmuştuk. Bir problemi bilgisayar ile çözmek için geliştireceğimiz programın yazımında izleyeceğimiz adımlar:

i) Problemin ne olduğunu kavra. Çözüm için gereksinimleri belirle.
ii) Problemin girdilerini, çıktılarını ve diğer kısıtlama ve gereksinimleri belirle (bilgilerin giriş ve çıkış biçimlerinin nasıl olacağına kadar).
iii) Problemin çözümünü veren algoritmayı yaz.
iv) Algoritmayı bir programla dili ile yaz.
v) Programın doğru çalışıp çalışmadığını test et. Bu testi değişik veriler (girdiler) için tekrarla.

2.1 İlk Program Örneği

```
#include <stdio.h>                    Kullanılan işlevler ile ilgili başlık dosyası
main()
{
        int i ;                       Değişken tanımı
        scanf("%d",&i);          Programın gövdesi
        i:=i*i;
        printf("%d",i);
```

}

BCPL → B (1967 Ken Thompson) → C (Denis Ritchie unix i yazmak için)

az sayıda saklı sözcük
kısa ve etkin program
çok sayıda işleç
assembler e yakın kod
taşınabilir kod

kullanıcıya bırakılan kontroller (dizinin boyutu gibi)
düşük okunabilirlik

source -----> compiler -----> object -----> link
kaynak derleyeci amaç bağlama

kaynak kod : C dili ile yazılmış olan program.
derleyeci : Kaynak kodu makina koduna çevirir
amaç kodu : Kaynak kodun makina dilindeki karşılığı
bağlama : Birden fazla amaç kodu dosyasının tek dosyada birleştirilmesi

2.2 Veri Tipleri

Veri tiplerini vermeden önce yazılan bir programın geçtiği aşamalara göz atalım.

2.2.1 Int Tip

Integer = Tamsayı
Tamsayıları içerir. Bellekte 2 Byte tutar.
5 , -19 , 25000 gibi
Minimum : -2^{31} = -32768
Maksimum : $2^{31}-1$ = 32767

2.2.2 Gerçel Tipler (Float, Double)

Gerçel sayıları içerirler.
float : Bellekte 4 Byte yer tutar. 3.4E-38 ile 3.4E+38 aralığında değer alır. Hassasiyet 7-8 basamaktır.
double : Bellekte 8 Byte ter tutar. 1.7E-308 ile 1.7E308 aralığında değer alır. Hassasiyet 15-16 basamaktır.

218.1 , -5.2 , 4.0

Bilimsel gösterim biçimi $2.5*10^3$ = 2.5E3 $2.5*10^{-3}$ = 2.5E-3

2.2.3 Char Tip

Char : Karakter : Alfanumerik karakterleri içerir.

'5' , '*' , 'K'

2.3 Sabitler (CONST)

```
#include <stdio.h>
main()
{
const float PI=3.14;                                    Tanımlama kısmı
float   r, Alan;
scanf("%f",r);
alan := PI*r*r;
printf(('Alan = %f', alan);
}
```

CONST (sabit) : Değeri programın çalışması boyunca değiştirilemez. Kullanım biçimi

const tip **Belirleyici=değeri;**

const float PI=3.14;

Tip belirtilmez ise tamsayı kabul edilir.

2.4 Değişkenler

Değeri programın çalışması boyunca değiştirilekir. Kullanım biçimi:

Tip Belirleyici [,Belirleyici] ;

int i;

float alan;

2.5 Belirleyiciler (Identifier)

Bileşenlere isim verme. (Sabit, değişken, altprogram, etiket, tipler (kullanıcı tanımlı)).

2.5.1 Kısıtlamalar

. İlk karakteri rakam olamaz

. Sadece harfler, rakamlar ve _ (underscore) karakterinin birleşiminden oluşabilir.

. C diline ait saklı (reserved) sözcükler kullanılamaz.

. En fazla 31 karakter olabilir.

. Küçük - büyük harf ayırımı vardır.

2.5.2 isimlerin Seçimi

. Belirleyicinin kullanım amacına uygun anlamlı isim seçin. (Hungarian notation)

. Ne fazla kısa ne de fazla uzun olsun. 3 - 10 karakter uzunluğunda olmasını alışkanlı edinin.

2.6 Matematiksel ifadeler

Matematiksel ifadeleri günlük hayattaki biçimde bilgisayarda yazamadığımız için belli bir kurallar uymamız gerekir. Kullandığımız matematiksel işlemler ve anlamları şöyledir:

+ , - , * , / toplama, çıkarma, çarpma, bölme

% tamsayı bölme sonucundaki kalanı verir (modulus)

7. / 2 ---> 3.5 (nokta sayının gerçel yorumlanmasını sağlar)

7 / 2 ---> 3 (sayılar int kabul edilip sonuçta int tipine dönüştürülür)

7 % 2 ---> 1

2 % 7 ---> 2

2 / 7 ---> 0

Matematiksel ifadeler hesaplanırken izlenen adımlar:

1. Önce parantez içindeki ifadeler hesaplanır. İçiçe parantezler var ise hesaplamaya en içteki parantezden başlanır.

2. ilk önce * , / ve % işlemleri daha sonra + ve - işlemleri yapılır.

3. Öncelik sırası aynı olan işlemlerde hesaplama soldan sağa doğru yapılır. 5./2*3 --> 7.5

Bir C Programının Genel Yapısı

başlık dosyaları. Kullanılan deyimler ile ilgili

```
main()
{
        sabitler ;              sabitlerin tanımlama bölümü
        değişkenler;    değişkenleri tanımlama bölümü
        deyimler;       Program gövdesi
}
```

2.7 Atama Deyimi (Assignment)

Bir değişkene bir değer(sabit, değişken, ifade) vermek.

DeğişkenAdı = değer;

x = 8 ;

```
y = x + 15 ;
y = (8 - x) / (2 - x) ;
```

Örnek 2.7.1: $5x^2+2$ fonksiyonunun x=4.8 noktasındaki değerini

```
main()
{
   float x, y ;
   x = 4.8;
   y = 5 * x * x + 2;
}
```

Örnek 2.7.2: Vize ve final notlarınından geçme notu

```
main()
{
   float vize, final, ort;
   vize = 50;
   final = 60;
   ort = vize * 0.4 + final * 0.6;
}
```

Örnek 2.7.3: 1998 yılına kadar geçen gün sayısı

```
main()
{
   float gun;
   int yil = 1996;
   gun = 365 * (yil - 1);
   gun = gun + (yil -1 ) / 4;
}
```

2.8 printf işlevi

Program içinde istenen değerleri çıktı birimlerine göndermek için kullanılır. Normal kullanımında çıktı birimi olarak ekran kabul edilir. Kullanım biçimi:

```
  prinf( kontrol, <parametreler>)
```

<kontrol> ile belirtilen bilgiye göre parametreleri yazar. Kontrol metini iki kısımdan oluşur.

- sıradan karakterler: doğrudan yazılırlar.
- dönüşüm/biçim belirleyici karakterler: Değerlerin nasıl yazılacağı bildirilir.

```
  printf(" sonuç =   %d", k);
                              ---   integer değer yazılacaktır
              ---------             ekrana aynen aktarılır
```

Dönüşüm belirlemek için önce % karakteri ve ardından dönüşümün nasıl olacağını belirten karakter verilir. Bu karakterlerden bazıları:

```
    d : decimal (integer)
    u : unsigned decimal
    c : char (tek karakter)
    s : string
    e : float/double sayıyı bilimsel gösterimde yaz
    f : float/double sayıyı [-] mmm.nnnnn biçiminde yaz
   ld : long integer
   lu : unsigned long integer
   Le,Lf : long double
```

ESC dizileri : \n : satır başı, \a : zil , \t : tab, \b : bir karakter geri

program parçası	ekranda görünen / imleçin konumu
i = 5 ;	
printf("%d",i) ;	5
printf("'i=%d",i) ;	i=5
printf("i=") ;	
printf("%d",i) ;	i=5 -
printf("i=\n") ;	i=
printf("%d",i) ;	5 alt satıra geçer
printf('i=%d\n",i) ;	i=5 alt satıra geçer
printf("%d - %d",i, 5*i);	5 - 25

Biçimli yazdırma

% karakteri ile dönüşüm karakteri arasına aşağıdaki karakterler de kullanılabilir.

- : sola dayalı yaz
m : yazılacak değer için ayırılan alan
n : değerin kaç karekteri yazılacağı

```
 s = "ABCDEF"
printf("%10s   ",s);          . . . . A B C D E F

printf("%10.3s   ",s);        . . . . . . . A B C
```

```
x = 128.5 ;
printf("%7.2f",x) ;                  # 128.50

x = 85.47 ;
printf("%6.3f",x) ;                  85.470

printf("%6.1f",x) ;                  ## 85.5
```

Örnek 2.8.1 : Yarıçapı belli dairenin alanını hesaplayan programı yazınız. (ilk yazılan program)

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

```
main()
{
const float PI=3.14;
float     r, alan;
r = 7;
alan := PI*r*r;
printf(('Alan = %f', alan);
}
```

Örnek 2.8.2 : En fazla dört basamak olabilen sayının basamak değerlerini yazdır. % ve / işlemlerinin kullanımı.

```
#include <stdio.h>
main()
{
  int i,y;

        y=1985;
        i= y / 1000;
        printf("%d",i);
        y= y-i*1000;
        i= y / 100;
        printf(" %d",i);
        y = y-i*100;
        i= y / 10;
        printf(" %d",i);
        y = y-i*10;
        printf(" %d\n",y);
        i = 1985;
        printf("%d ",i / 1000);
        printf("%d ",(i / 100) % 10);
        printf("%d ",(i / 10) % 10);
        printf("%d\n",i   % 10);
}
```

Örnek 2.8.3: $ax^2+bx+c=0$ tipi bir denklemin köklerini veren programı yazınız.

Girdi : a, b ve c katsayıları

Çıktı : denklemim kökleri

Algoritma :

1. a, b ve c katsayılarını oku.

2. Delta= değerini hesapla.

3. x_1 ve x_2 değerlerini hesapla.

4. Kökleri yaz.

Programın kodlanması:

```
#include <stdio.h>              /* printf işlevi için */
#include <math.h>               /* sqrt işlevi için       */
main()
{
    float a, b, c;
    float x1, x2;
    float d;

    a = 1;
    b = -3;
    c = 2;
    d = b * b - 4 * a * c;
    x1 = (-b + sqrt(d)) / (2 * a);
    x2 = (-b - sqrt(d)) / (2 * a);

    printf("Kökler = %f ,   %f",x1, x2);
 }
```

C dilinde karekök almak için bir deyim yoktur. Örnekte bunu yerine getiren C diline eklenmiş olan sqrt() fonksiyonu kullanılmştır. Aşağıda buna benzer artık C derleyecilerinde standart olmuş bazı fonksiyonlar verilmiştir. Bu işlevler math.h başlık dosyasında tanımlıdır.

Fonksiyon	x ,y		Sonuç
abs(x)	int	int	x'in mutlak değeri
fabs(x)	double	double	x'in mutlak değeri
pow(x, y)	double	double	x^y
sqrt(x)	double	double	x'in karekökü
exp(x)	double	double	e^x değeri
log(x)	double	double	ln(x) değeri
log10(x)	double	double	$\log_{10}(x)$ değeri
ceil(x)	double	double	x ten büyük ilk tamsayı
floor(x)	double	double	x ten küçük ilk tamsayı

Örnekler:

ceil(5)	5
ceil(5.2)	6

ceil(-5.2) -5

floor(5) 5

floor(5.2) 5

floor(-5.2) -6

2.9 scanf İşlevi

Klavyeden veri okumak için kullanılır. Yapı olarak printf işlevi ile hemen hemen aynıdır. Kullanım biçimi:

```
scanf( kontrol, <değişkenler>)
```

Girilen karakterler <kontrol> metininde belirtilen biçimlere göre değişkenlere aktarılır.
Değişkenler işaretçi tipinde olmalıdır. Yani parametre olarak değişkenin adresi gönderilmelidir.
Ayırıcılar boşluk, tab, enter

```
scanf("%f %f %f ", &a, &b, &c);
```

scanf işlevinin değeri
0 ise hiçbir değişkene değer atanmamış
>0 ise başarılı bir şekilde değer atanan değişken sayısı

```
int a,b,c;
float m,n;
```

scanf("%d", &a); Klavyeden tamsayı okur. Girilen değer a değişkenine aktarılır.

scanf("%d %d",&a,&b) Klavyeden girilen ilk değer a değişkenine, ikinci değer b değişkenine aktarılır.

scanf("%f %d", &m, &a); Klavyeden ilki gerçel, ikincisi tamsayı olmak üzere iki değer okur.

İkinci dereceden denklem çözümünün yapıldığı örnekte katsayıları klavyeden okutmak istersek

```
scanf("%f %f %f ", &a, &b, &c);
```

Farklı kullanıcı arayüzünde yazarsak

```
printf("Katsayıları sırasıyla giriniz (a b c) :" );   scanf("%f %f %f ", &a, &b, &c);

printf("a katsayısını giriniz : ");   scanf("%f", &a);
printf("b katsayısını giriniz : ");   scanf("%f", &b);
printf("c katsayısını giriniz : ");   scanf("%f", &c);
```

Örnek 2.9.1: Vize ve final notlarınından ortalamayı hesaplayan programda değerlerin klavyeden okunmuş hali.

```
main()
{
  float vize, final, ort;
  printf("Vize notunu giriniz ");   scanf("%f", &vize);
```

```
  printf("Final notunu giriniz "); scanf("%f", &final);
  ort = vize * 0.4 + final * 0.6;
  printf("Ortalaması = &f\n", ort);
}
```

2.10 Mantıksal ifadeler

Sonucu Doğru veya Yanlış olan ifadelerdir. Sonuç sıfır ise yanlış aksi halde doğru kabul edilir.

İlişkisel işleçler(operatör) : iki değer arasındaki ilişkiyi test etmek için kullanılır.

işleç	anlamı
>	büyük
>=	büyük - eşit
==	eşit
<	küçük
<=	küçük - eşit
!=	eşit değil

x=8, y=5 için

x > y Doğru

x < y Yanlış

x !=y Doğru

Mantıksal işleçler : İki mantıksal ifade arasındaki ilişki üzerindeki ilişkide kullanılır.

! DEĞİL (NOT)

&& VE (AND)

|| VEYA (OR)

(X>0) && (X>Y)

(X>0) || (Y>0)

İfadelerde işleçlerin yürütülme sırası

işleç	Önceliği
( )	en yüksek (ilk yürütülür)
!	
*, /, %	
+, -	

<, <=, >=, >

==, !=

&&, ||

= en düşük (son yürütülür)

= işleci sağdan sola, diğerleri soldan sağa doğru yürütülür.

Görüldüğü gibi ifadelerde matematiksel ve mantıksal işlemler bittikten sonra ilişki test edilir.

X=50, Y=80, Z=45 için

((X / 4 + Y / 4 + Z / 2) >= 50) && (Z >= 50)

3 - Döngü ve Koşul Deyimleri

Programlar (algoritmalar) üç temel blok kullanılarak gerçekleştirilebilirler. Bunlar; ardarda, bir koşula bağlı olarak ve sonlu sayıda yineleme (döngü) dir.

3.1 Koşul Deyimleri

Birkaç seçenekten birini seçmek veya bir deyimin bir koşula bağlı olarak işlemek için kulanılır.

3.1.1 if-then-else Deyimi

if (<mantıksal ifade>**)**
    blok_doğru;
else
    blok_yanlış;

Mantıksal ifade doğru ise blok_doğru, yanlış ise else sözcüğünden sonraki blok_yanlış yürütülür. else kısmı seçimlidir, gerekmiyorsa kullanılmayabilir.

Örnek 3.1. 1.1 Girilen sayının tek/çift olduğunu yazan program

```
#include <stdio.h>
main()
{
   int i;
   scanf("%d", &i);
```

```
    if ( i % 2 == 1)
        printf("Tek");
    else
         printf("Çift");
}
```

Bileşik (Compound) Deyimler

{ ve } karakterleri ile sınırlandırılmış bir dizi deyimden oluşur.

```
{
      i = 5;
      j = i/2;
      k = i+1;
}
```

Eğer bloklarda birden fazla deyim kullanmak gerektiğinde bileşik deyim kullanılır.

```
if (yil % 4 == 0)      {
            subat =29;
            gunyil = 366;
      }
else      {
            subat =28;
            gunyil = 365;
      }
```

Örnek 3.1.1.2 : İkinci dereceden denklemin köklerinin bulunması.

```
if (delta<0)
     printf("Gerçel kök yoktur.\n");
else
      {
          x1 = (-b + sqrt(delta)) / (2 * a);
                x2 = (-b - sqrt(delta)) / (2 * a);
                printf("Birinci kök = %f\n" , x1);
                printf("ikinci kök   = %f\n" , x2);
      }
```

Örnek 3. 1.1.3 : Klavyeden girilen karakterin rakam olduğunun tesbiti.

```
char c;
c = getch();
if ((c>='0') && (c<='9')
      printf("Rakam girdiniz.");
```

Örnek 3. 1.1.4 : Girilen üç sayıdan en küçüğünün bulunması (İçiçe IF kullanımı).

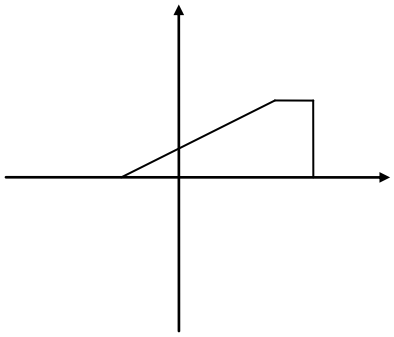
```
scanf("%d%d%d", &s1, &s2, &s3);
if ((s1<s2) && (s1<s3))
     ek =s1;
else
```

```
        if (s2<s3)
            ek =s2;
        else
            ek = s3;
    printf('En küçük olanı = %f", ek);
```

II. yol :

```
    scanf("%d%d%d", &s1, &s2, &s3);
    ek = s1;
    if (ek>s2)
        ek =s2;
    if (ek>s3)
         ek =s3;
    printf('En küçük olanı = %f", ek);
```

Örnek 3. 1.1.5: Fonksiyonun girilen t değeri için aldığı değeri hesaplayıp yazan program.



```
main()
{
  float y, x;

  printf("x değerini giriniz");
  scanf("%f", &x);
  if (x > -1   && x<2)
       y = 2/3 * (x + 1);
  else
   if (x>2 && x<=3)
       y = 2;
   else
      y = 0;

  printf("Fonksiyonun değeri = &f", y);
}
```

Örnek 3. 1.1.6: Vergi iadesini hesaplan programı yazınız. (elseif yapısı)
(0-60 bin : %10, 60-120 bin : %,120-200 bin : %12,>200 bin : %5)

```
main()
{
   float fat_top, vergi_iade;
   printf("Fatura toplamlarını giriniz ");
   scanf("%f", &fat_top);
   if (fat_top < 60000)
       vergi_iade = fat_top * 0.10;
   else if (fat_top < 120000)
       vergi_iade = 6000 + (fat_top - 60000) * 0.20;
   else if(fat_top < 200000)
       vergi_iade = 18000 + (fat_top - 120000) * 0.12;
   else
       vergi_iade = 27600 + (fat_top - 200000) * 0.05;

   printf("Ödenecek vergi iadesi = %f\n", vergi_iade);

}
```

Örnek 3. 1.1.7: Girilen tarihteki günün adını veren programı yazınız.

```
#include <studio.h>
main ( )
{
    int gun, ay, yıl ;
   long gt ;
   printf(“Tarihi gir”) ; scanf ( “%d %d %d “,&gun)

/* oncekı yıllardakı gun sayısını hesapla */

   gt=( yıl*1)*365 + yıl/4;

/* bu yildaki aylardaki gunleri ekle */

if (ay==2)
            gt = gt + 31 ;
else if (ay ==3)
            gt = gt + 31 + 28 ;
else if (ay ==4)
            gt = gt + 31 + 28 +31;
else if (ay ==5)
            gt = gt + 31 + 28 +31+ 30 ;
else if (ay ==6)
            gt = gt + 31 + 28 +31+ 30 +31;
else if (ay ==7)
            gt = gt + 31 + 28 +31+ 30 +31+ 30 ;
else if (ay ==8)
            gt = gt + 31 + 28 +31+ 30 +31+ 30 + 31 ;
else if (ay ==9)
            gt = gt + 31 + 28 +31+ 30 +31+ 30 + 31+30 ;
else if (ay ==10)
            gt = gt + 31 + 28 +31+ 30 +31+ 30 + 31+30 + 31;
 else if (ay ==11)
            gt = gt   + 31 + 28 +31+ 30 +31+ 30 + 31+30 + 31+ 30 ;
```

```
else if (ay ==12)
        gt = gt + 31 + 28 +31+ 30 +31+ 30 + 31+30 + 31+ 30 +31;

/*Bu ayı ekle */
gt = gt+ gun;
if(yıl%4==0 && ay>2),
        gt =gt+1;
gt=gt %7,
if(gt==1)
        printf("Pazar");
else if(gt==2)
        printf("Pazartesi");
else if(gt==3)
        printf("Salı");
else if(gt==4)
        printf("Carsamba");
else if(gt==5)
        printf("Persembe");
else if(gt==6)
        printf("Cuma");
else if(gt==7)
        printf("Cumartesi");
}
```

3.1.2 switch Deyimi

```
switch(<seçici>) {
        case seçenek1 : Deyim;
        case seçenek2 : Deyim;
            .
            .
            .
        default :    Deyim;
}
```

Seçicinin aldığı değere eşit seçeneğin olup olmadığına bakar. Var ise o noktadan sonraki deyimler yürütülür. **switch** deyiminin sonuna gelindiğinde veya **break** deyimi ile karşılaşıldığında yürütme işlemi durur ve programın akışı switch deyimini izleyen deyim ile devam eder.

```
switch(i) {
   case 1 : printf("Bir");
   case 2 : printf("İki");
   default : printf("Hiçbiri");
}
```

```
i=1 ise çıkış   BirİkiHiçbiri
i=2 ise çıkış   İkiHiçbiri
```

Sorunu ortadan kaldırma için her durum için break deyimi eklenmeli.

. Seçici **Ordinal** tiplerden biri olmalıdır (Ordinal tip: tüm değerleri listelenebilinen veri tipleri - integer, char).
. Seçici ile seçenekler aynı tipte olmalıdır.

. default kısmı seçimliktir. Seçeneklerin hiçbiri uygun değil ise yürütülür.

```
#include <stdio.h>
main()
{
  char islem;
  int s1, s2, s3;
  printf("Önce işlemi sonra sayıları girin ");
  scanf("%c%d%d",&islem, &s1, &s2);
  switch (islem) {
     case '+' : s3 = s1 + s2; break;
     case '-' : s3 = s1 - s2; break;
     case '*' : s3 = s1 * s2; break;
     case '/' : s3 = s1 / s2; break;
     default : printf ("Hatalı işlem");
  }
  printf("\nSonuç = %d",s3);
}
```

Örnek 3.1.2.1: Mevsimleri yaz.

```
scanf("%d", &ay);
switch (ay) {
        case 3:
        case 4:
        case 5: printf("ilkbahar"); break;
        case 6:
        case 7:
        case 8: printf("yaz"); break;
        case 9:
        case 10:
        case 11: printf("sonbahar"); break;
        case 12:
        case 1:
        case 2: printf("kış"); break;
}
```

switch deyimi yerine if deyimi kullanılabilir. Ancak switch deyimi programı daha okunabilir kıldığı için gerekli olduğu durumlarda kullanılmalıdır.

Örnek 3.1.2.2 : 16'lık sistemdeki rakamın 10'luk sistemdeki karşılığı (char tipinin sayı gibi davranışı).

```
switch(c) {
   case '0':
   case '1':
   . . .
   case '9': i = c - '0'; break;
   case 'a':
   case 'A': i = 10; break;
   . . .
   case 'f':
   case 'F': i = 15; break;
}
```

Örnek 3.1.2.3: Sınav notunu harfe dönüştüren programı yazınız.
(>=90 :AA, 85-89:BA, 80-84:BB, 75-79:CB, 70-74:CC, 60-69:D, <60 :F)

Örnek 3.1.2.4: Belirtilen tarihin hangi güne denk geldiğini bulan programı yazınız.
else if yapısı yerine switch kullanarak

3.2 Döngü Deyimleri (Yineli)

Bir ya da birden fazla deyimin tekrar edilemesini sağlarlar. C dilinde while, for ve do-while deyimleri döngü işlevini saşlar. Tekrar edilen deyimlere döngü gövdesi denir.

3.2.1 while Deyimi

while <mantıksal ifade>
Deyim

Mantıksal ifade doğru olduğu sürece Deyim yürütülür. Eğer yanlış ise kontrol bir sonraki deyime geçer.

Örnek 3.2.1.1 : 1'den 100'e kadar olan sayıların toplamı.

1. i =1
2. j = 0
3. i < 101 olduğu sürece
   3.1 j = j + i
   3.2 i = i + 1
4. Toplam j ‘ yi yaz

```
main()
{
    int i, j;
    i =1;
    j = 0;
    while (i<101) {
            j =j+i;
            i =i+1
    }
     printf("Toplam = %d",j);
}
```

Örnek 3.2.1.2: Toplama ve çarpma kullanarak çarpma işmeini gerçekleyiniz.

1. a ve b sayılarını oku
2. c =0
3. b>0 olduğu sürece tekrarla
   .3.1. c=c + a
   3.2. b = b-1
4. c değerini yaz ve dur

```
main()
{
  int a, b, c;
  printf (“iki sayıyı giriniz “);   scanf(“%d%d”, &a, &b);
  c = 0;
  while (b > 0) {
    c = c + a;
    b = b - 1;
  }
  printf(“Sonuç = &d\n”, c);
}
```

Örnek 3.2.1.3: Girilen sayının faktoriyelini hesaplayan programı yazınız.

1. n değerini oku
2. F=1
3. n >1 olduğu sürece tekrarla

   .3.1. F=F*n

   3.2. n= n-1
4. F değerini yaz

```
main()
{
  int n;
  long f;
  printf (“sayıyı giriniz “);   scanf(“%d”, &n);
  f = 1;
  while (n > 1) {
    f = f * n;
    n = n - 1;
  }
  printf(“Sonuç = &d\n”, f);
}
```

Örnek 3.2.1.4: Klavyeden girilen sayıları oku. Sayıların toplamı 21'den büyük veya eşit olduğu zaman dur.

```
main()
{
      int i, j = 0;
      while (j<21) {
          scanf("%d",&i);
          j =j+i;
      }
      printf("Toplam = %d",j);
}
```

Örnek 3.2.1.5: 1993 yılı itibarı ile ülke nüfüsu 60 milyondur. Yıllık nüfüs artış oranı %2.3 tür. Sonraki 10 yılda ülke nüfüsunu yıllara göre listeleyen program.

```
/* Nufus Tablosu   */
#include <stdio.h>
main()
{
  int          i;    /* sayac */
  int        yil;    /* yillar */
  float nufus;     /* nufus miktari */
  float artis;     /* artis orani */

  artis = 0.023;
  yil = 1993;
  nufus = 60000000;
  printf("%d - %10.0f\n",yil,nufus);
  i = 1;
  while (i < 11)
    {
      nufus = nufus * (1 + artis);
      printf("%d - %10.0f\n",yil + i,nufus);
      i = i + 1;
    }
}
```

Örnek 3.2.1.6 : Girilen tamsayının mükemmel sayı olup olmadığının söyleyen programı yazınız.
(mükemmel sayı = tam bölenlerin toplamı sayının kendisine eşit)

Örnek 3.2.1.7 : Girilen tamsayının kaç basamaktan oluıştuğunu söyleyen programı yazınız.

Örnek 3.2.1.8 : Girilen tamsayı içerisinde kaç tane 1 olduğunu söyleyen programı yazınız.

Örnek 3.2.1.9: Girilen tamsayının son üç basamağını yuvarlayan programı yazınız.
son üç basamağı >=500 > 1000 e, <500 ise 0 a yuvarlayacak
(2560 → 3000, 2490 → 2000)

Örnek 3.2.1.10: Sınavın ortalamasını hesaplayan programı yazınız.
1. Durum : sınava giren öğrenci sayısı belli

```
main()
{
  int i, ogr_say, not_top, not;
  float ort;
  not_top = 0;
  i = 0;
  printf("Öğrenci sayısını giriniz "); scanf("%d", &ogr_say);
  while (i < ogr_say) {
        printf"Sıradaki öğrencinin notu   = "); scanf("%d", &not);
        not_top = not_top + not;
        i = i + 1;
   }
   ort = float (not_top) / ogr_say;
   printf("Ortalama = &f\n", ort);
}
```

2. Durum : sınava giren öğrenci sayısı belli değil

Bu durumda girişin bittiğini gösterir bir bilgiye (işarete) ihtiyaç vardır. Sınav notu için 0 dan küçük bir değer girildiğinde girme işleminin sona erdiğinin varsayalım.

```
ogr_say = 0;
printf"Sıradaki öğrencinin notu   = "); scanf("%d", &not);
while (not >= 0) {
        not_top = not_top + not;
        ogr_say = ogr_say + 1;
        printf"Sıradaki öğrencinin notu   = "); scanf("%d", &not);
}
```

3.2.2 For Deyimi

```
for (ifade1 ; ifade2 ; ifade3 )
    ifade;
```

ifade2 doğru (veya farklı 0) olduğu sürece ifade yürütülür (bitiş koşulu).
Döngünün ilk adımından önce ifade1 yürütülür (başlangıç adımı).
Döngünün her adımında ifade3 yürütülür (artış miktarı).

```
for (i = 1; i < 5; i++)
    printf("%d ",i);
```

ifade1, ifade2 ve ifade3 seçimliktir. ifade2 belirtilmez ise herzaman doğru olduğu (== 1) kabul edilir. Yani sonsuz döngü oluşur.

```
for (i = 1; ; i++)
    printf("%d ",i);
```

Örnek 3.2.2.1: 1'den 100'e kadar olan sayıların toplamı.

```
j =0;
for (i=1; i<=100; i=i+1)
            j =j+i;
printf("Toplam %d",j);
```

Örnek 3.2.2.2: Girilen sayının faktöriyelini bulunuz.

```
    fact =1;
    for (j=1; j<=i; j++)
        fact =fact*j;
    printf("Faktöriyel   =%f",fact);
}
```

Örnek 3.2.2.3: Çarpım tablosu. (içi içe döngüler)

```
main()
{
    int i,j;
    for (i=1; i<=10; i++) {
        for (j =1; j<=10; j++)
        printf("%4.0d",i*j);
        printf("\n");
    }
}
```

Örnek 3.2.2.4: ? işlevini çiziniz(0-8 noktaları arasında).

3.2.3 do-while Deyimi

Bir koşul doğru olana kadar döngü yürütülür.

```
do
    Deyim
while (<mantıksal ifade>)
```

Mantıksal ifade doğru olduğu sürece döngü tekrar edilir. Yanlış olduğunda while sözcüğünden sonraki deyim yürütülür.

5 sayısı girilene kadar oku

```
do
     scanf("%d",&i);
while (i!=5);

i =1;
do {
     printf("%d",i*i);
     i =i+1;
} while (i<=10);
```

Örnek 3.2.3.1: Sadece +, - kullanarak * işlemini gerçekleştirme.

```
main()
{
   int a, b;
   int c;
   scanf("%d%d", &a, &b);
   c = 0;
   do {
      c = c + a;
      b = b - 1;
   } while (b>0);
   printf("%d\n", c);
}
```

Karşılaştırma

while : Koşul başlangıçta test ediliyor. Döngü sıfır veya daha fazla yürütülüyor.
do-while : Koşul sonda test ediliyor. Döngüye en az bir defa kesin giriliyor.

Örnek 3.2.3.2: 0 - 100 arasında tutulan sayının tahmini.

```
main()
{
  int         tahmin;    /*   tahminimiz   */
  int            min;    /*   Tahminin alt siniri   */
  int            max;    /*   Tahminin ust siniri   */
  char         cevap;    /*   Kullanicinin cevabi   */
  min = 0;    max = 100;
  do {
    tahmin = (max - min) / 2 + min;
    printf("Tahminim %d\n",tahmin);
    printf("Buyuk / Kucuk / Esit ");
    scanf("%c",&cevap);
    if (cevap == 'B')
       max = tahmin - 1;
     else
        if (cevap == 'K')
          min = tahmin + 1;
  } while (cevap != 'E');
}
```

Örnek 3.2.3.3: Sin(x) fonksiyonun belli bie x için değerini seri açılımı ile hesaplayınız. Serinin ilk 10 terimini kullanınız. n tek sayı.

```
main()
{
  float      x;          /* fonksiyonun hesaplanacağı değer */
  float sinx;            /*   sin(x) in değeri */
  float     xt;          /* eklenen terimin payı */
  float fakt;            /* eklenen terimin paydası, faktoriyel */
  int isaret;            /* terimin +/- olmasını sağlar */
  int         i;
  clrscr();
  printf("Hesaplanacak değer : ");
  scanf("%f",&x);
  sinx = x;
  isaret = -1;
  fakt = 1;
  xt = x;
  for (i = 2; i<=10; i++) {
       xt = xt * x * x;
       fakt = fakt * (2 * i - 2) * (2 * i - 1);
       sinx = sinx + isaret * xt / fakt;
       isaret = -isaret;
  }
  printf("Değeri = %f\n",sinx);
}
```

Örnek 3.2.3.4 : Aynı örneği belli sayıda terim için değil, sonucu 4 basamak hassasiyetle hesaplayınız.

4 - İşlevler

Problem çözmenin ilkelerinden biri problemi mümkün olduğu kadar çok parçaya bölmek idi. Daha sonra bu parçalar bağımsız olarak düşünülüp çözümleri elde edilebilir. C'de bu bölme işlemi işlev kullanarak yapılır.

Belli bir işi gerçekleştiren program deyimlerinin karmaşık programları düzenlemek ve basitleştirmek için programın bir birimi olarak gruplandırılması.

(işlev kullanmanın temel nedenleri : divide and conquer , software reusability)

Örnek 4.1 : Bir tamsayının faktöriyelini hesaplayan işlev

```
main()
{
  int i;
  prinf("Sayıyı giriniz "); scanf("%d", &i);
  printf("%d\n", fakt(i));
}
```

```
long fakt ( int n)
{
  int j;
  long f = 1;
  for ( j = 2; j < =n; j = j   + 1),
        f = f * j;
  return f;
  }
```

Örnek 4.2 : Bir tamsayının kübünü veren işlev

```
#include <stdio.h>
main()
{
 int sayi;
 int kub(int); /* işlevin prototipi */
```

```
 printf("sayıyı gir ");
 scanf("%d", &sayi);

 printf("Kübü = %d\n", kub(sayi));
}

int kub(int i)
{   return i*i*i; }
```

İşlevin tanımlanma biçimi

```
   dönüş_tipi işlev_adı(parametreler)
   {
     yerel tanımlamalar
     deyimler
   }
```

dönüş_tipi: Eğer işlev bir değer geri gönderecek ise değerin tipini belirtir. Belirtilmez ise Int kabul edilir. Eğer işlev değer göndermeyecek ise dönüş_tipi yerine **void** yazılır.

işlev_adı: İşlev çağırılırken kullanılacak ad (belirleyici).

parametreler: İşlev için gerekli değerleri içerir. Her parametre değişken tanımlar gibi tanımlanır. Herbirinin arasında ',' kullanmak gerekir.

yerel tanımlamalar: Bu işleve özgü(değişken,sabit) tanımlamalar.

Eğer işlev bir değer gönderecek ise bu return deyimi ile yapılır.

return değer;

İşlevin Prototipi

Tanımlana bir işlevin ana modül içerisinde prototipinin yazılması gerekir. Prototip ile işlevin dönüş değeri ve aldığı parametrelerin tipleri tanımlanır. Bu bilgiye göre C derleyecisi işlev çağırıldığında değerlerin uygun olduğunu sınar.

int kub(int) kub işlevi int bir değeri alır ve yine int bir değer üretir.

Örnek 4.3 : Üç sayıdan en büyüğünü veren işlev.
Girdi : üç tane tamsayı
Çıktı : girdilerin en büyüğü

```
int max(int s1, int s2, int s3)
{
 if (s1>s2 && s1>s2)
     return s1;
 else
     if (s2>s3)
         return s2;
     else
         return s3;
}
```

Bu işlevi kullanan bir program

```
#include <stdio.h>
main()
{
 int max(int, int, int);
 clrscr();
 printf("%d \n", max(1,2,3));
 printf("%d \n", max(14,8,12));
 printf("%d \n", max(1,6123,3123));
}
...
işlevin tanım bloğu
...
```

Örnek 4.4: Üs alma işlevini gerçekleyen işlev (üs bilgisi tamsayı).

```
Girdi : taban ve üs değeri
Çıktı : tabanın üs. kuvveti
float us (float a, int   b)
{
     int i;
     float r;

     r = 1;
     for (i = 1; i<=b; i++)
          r = r * a;
      return   r;
}
```

Fonksiyonun program içinden çağırılışı

```
     d = us(2,3);
     d = us(a, 4);
     d = d * us(3,4);
```

Üs alma işlemini gerçeklemenin diğer yolu

$a^b = e^{blna} = Exp(b * Ln(a))$

Exp(x : Real) : Real e^x

Ln(x : Real) : Real $Ln(x)$

```
float us (float a, float   b)
{
     if (a>0)
          return= exp( b * log (a ));
     else
          return= -1;
}
```

Genel değişken (global): Her yerde (main ve diğer işlevler) geçerlidir, değerine erişilebilir.
Programın en başında, main işlevinin dışında tanımlanır.

Yerel değişken (local): Sadece tanımlandığı modülde geçerlidir, değerine erişilebilir.

Modül içinde tanımlanır.

```
int a;     /*tüm işlevlerden değerine erişilebilir, değeri değiştirilebilir.   (global) */

main()
{
    int b;   /* sadece main işlevi içerisinden erişilebilir   (local ) */
    ...
}

islev1(...);
{
    int b;     /* sadece islev1 işlevi içerisinden erişilebilir. main işlevindeki değişkendan bağımsızdır   (local ) */
    int c;     /* sadece islev1 işlevi içerisinden erişilebilir   (local ) */
    ...
}
```

Örnek 4.5 : 8 bit olarak girilen bir sayıya eşlenik (parity) bitini ekleyen programı yazınız (çift eşlenik)
Eşlenik biti en düşük anlamlı bit olarak eklenecektir.

Algoritma

1. Sayıyı oku.
2. Sayıdaki birlerin sayısını bul.
3. Birlerin sayısı tek ise 1, çift ise 0 ekle.

Programı yazarken 2. ve 3. adımlar işlev olarak yazılacaktır.

```
/* Eşlenik biti ekleme */
int bulbir ( int );
int eklebir ( int , int );

main()
{
  int girsayi, birler;

  clrscr();
  scanf("%d", &girsayi);
  birler = bulbir(girsayi);
  girsayi = eklebir(birler, girsayi);
  printf("%d\n", girsayi);
}

int bulbir ( int sayi)
{
    int kacbir = 0;
    while (sayi>0) {
```

```
        kacbir = kacbir + (sayi % 2 );
        sayi = sayi / 2;
    }
    return kacbir;
} /* BulBir */

int eklebir ( int birler, int sayi)
{
    return (2 * sayi + (birler % 2));
} /* Eklebir */
```

* Karşı taraf için gereken algoritmayı yaz (sayı doğru mu, doğru ise değeri).

Örnek 4.6 : A işyerindeki bir personelin hangi dilimden gelir vergisi ödeyeceğini (yüzdesini) belirten fonksiyonu yazınız.

Girdi : personelin birikmiş vergi matrahı
Çıktı : Kesinti yüzdesi
İlgili bilgi : 0-20 Milyon %25, 20-40 Milyon %30, 40-80 Milyon %35, 80-160 Milyon %40
>180 Milyon %45

```
int   v_oran ( float bvd);
{
    int i;
     if (bvd < 20000000)
          i = 25;
     else if   (bvd < 40000000)
          i = 30;
     else if   (bvd < 80000000)
          i =   35;
      else   if   (bvd < 160000000)
           i =   40;
      else
           i = 45;
      return i;
}
```

Örnek 4.7 : Gelir vergisi matrahı ve önceki aylara ait birikmiş vergi matrahı belli bir personelin gelir vergisini hesaplayan fonksiyonu yazınız.

Vergi := Gelir vergisi matrahı * Vergi oranı

Not : Eğer personel bu ay dilim değiştiriyorsa gelir vergisi gerektiği şekilde eski ve yeni dilime paylaştırılarak kesinti hesaplanmalıdır.

Örnek 4.8 : Girilen harfi büyük harfe dönüştüren fonksiyon
Girdi : karakter
Çıktı : karakter (girdi harf ise büyük harfe dönüştürülmüş)

```
char   buyuk(char c);
{
     if (c >= 'a'   && c <= 'z')
          return c - 32;
     else
          return c;
}
```

Örnek 4.9:
Girdi : iki sayı ve aralarındaki işlem
Çıktı: iki sayı arasındaki işlemin sonucu

```
int calculate(char islem, int a, ibt b, int c)
{
    int s;
     switch( islem )
          case '+' : s = a + b ; break;
          case '-' : s = a - b ; break;
          case '*' : s = a * b ; break;
          case '/' : s = a / b ; break;
         default : s = 0;
    }
    return s;
}
```

Çağırma biçimi

```
      a = calculate ('+',4,8');
      a = calculate ('/', 4,2) * calculate ('*', 2,2);
```

Örnek 4.10: Bir işlevin köklerinden birini Newton yönetimi ile bulan prg. (x2-2)

```
#include <stdio.h>
#include <math.h>

double fn(double);
double fnt(double);

main()
{
 double x,x0;
 double hata;


 hata = 0.0001;            /* izin verilen hata */
 x = 8;                          /* başlangıç değeri */
 do {
    x0 = x;
    x = x0 - fn(x0)/fnt(x0);
 } while (fabs(x-x0)>hata);

 printf ("Sonuç = %f\n", x);
 printf ("f(x)   = %f\n", fn(x));
}

double fn(double x) /* işlevin değeri */
{
 return x*x-2;
}
```

```
double fnt(double x)   /* işlevin türevinin değeri */
{
  return 2*x;
}
```

Örnek :4.11: İki sayının ortak katlarının en küçüğünü veren işlevi yazınız.

```
#include <stdio.h>
long okek(long, long);
main()
{
  long i, j;
  clrscr();
  printf("İki tamsayı giriniz :");
  scanf("%ld%ld", &i, &j);
  printf("sayıların okeki = %ld \n" okek(i, j));

}
```

long okek(long p, long q)

```
/* p < q olmali. Degil ise yerlerini değiştir. */
{
  long i, k;
  if (p>q) {
       i=p;
       p=q;
       q=i;
  }

  /* p nin öyle bir katını bul ki q sayısına tam bölünsün
     2 sayısından itibaren taranmaya başlanabilir fakat
     p/q yeterli
  */

  i = q / p;
  do {
    k = p*i;
    i = i+1;
  } while ( k % q != 0);
  return k;
```

Örnek :4.12: İstenen sayıdaki asal sayıyı listeleyen programı yazınız.

```
#include <stdio.h>

int asal(int);
main()
{
  int i ;      /* asal olup olmadığı sınanacak sayılar */
  int kac; /* ilk kaç asal sayının listeleneceği bilgisi */
  int j;         /* listelenen asal sayıların adetini tutar */
```

```
  clrscr();
  printf("ilk kaç asal sayı listelenecek : ");
  scanf("%d", &kac);


  /* for deyiminde ifade3 kısmında döngü değişkeni yerine
     farklı bir değişkenin değeri değiştiriliyor. Döngü değişkeni
     ide incelenen sayı asal ise arttırılıyor
  */

  i = 2;
  for (j = 1; j<=kac;i++)
       if (asal(i)) {
           printf("%d \n", i);
           j++;
       }
}
int asal(int n)
/* sayı asal ise 1 değilk ise 0 değerini alır */
{
          int i;
          if (n % 2 == 0)
                    return (n==2);
          if (n % 3 == 0)
                    return (n==3);
          if (n % 5 == 0)
                    return (n==5);
          for (i=7; i*i <= n; i+=2)
                    if (n % i == 0)
                              return 0;
          return 1;
}
```

Örnek :4.13: Belirtilen tarihteki ayın kaç gün sürdüğünü veren işlevi yazınız.
Girdi : Ay ve yıl
Çıktı : Aydaki gün sayısı

```
int aydakigun(int ay, int yil)
{
  int i;
  switch (ay) {
       case 1:
       case 3:
       case 5:
       case 7:
       case 8:
       case 10:
       case 12 : i = 31;break;
       case 2 :
                 if (yil % 4 == 0)
                    i = 29;
                 else
```

```
                    i = 28;
                break;
        default : i = 30;
   }
   return i;
}
```

Örnek :4.14: Önümüzdeki yılda Ramazan Bayramı nın hangi tarihte kutlanacağını hesaplayan programı yazınız. Bu yılki tarih klavyeden gün ay ve yıl olarak okunacakır.
Girdi: Bu yıl kutlanılan tarih
Çıktı : Sonraki tarih

```
#include <stdio.h>

int aydakigun(int, int);

main()
{
   int gun,ay, yil;
   int i, j ;

   clrscr();
   printf("Bu yılki tarihi giriniz (G A Y) : ");
   scanf("%d %d %d",&gun, &ay, &yil);
   yil = yil + 1;
   gun = gun - 11;
   if (gun < 1 ) {
       ay = ay - 1;
       if (ay < 1 ) {
           ay = 12;
           yil = yil - 1 ;
       }
       gun = gun + aydakigun(ay, yil);
   }
   printf("\nGelecek bayram = %d/%d/%d\n",gun,ay,yil);
}
```

Örnek :4.15: Girilen bir tamsayının basamaklarının sırasının tersinin oluşturduğu sayıyı veren işlevi yazınız.

```
#include <stdio.h>
long tersi(long);
main()
{
   long s;
   clrscr();
   printf("Sayıyı giriniz : ");
   scanf("%ld", &s);
   printf("Basamaklarının ters çevirilmişi   = %ld \n", tersi(s));
}

long tersi(long k)
{
   long a, b;
```

```
  a = 0;
  while (k>0) {
      b = k % 10;
      a = a * 10 + b;
      k = k / 10;
  }
  return a;
}
```

4.2 Özyineleme (Recursive) İşlevler

Kendini çağıran fonksiyonlardır.

Örnek 4.2.1: Faktoriyel hesabı

```
long fakt(long i)
/* fakt = i! */
{
  if (i<2)
      return 1;
  else
      return i*fakt(i-1);
}
```

Örneğin n=4 için fonksiyonun kendisini çağırdığı satır şöyledir:

```
         4 * fakt(3),   3 * fakt(2),   2 * Fakt(1)
24 = 4 *     6   <--   3 *     2   <--     2 *     1
```

Örnek 4.2.2: Ortak bölenlerin en büyüğünü bulan program (Euklid yöntemi).
gcd = greatest common divisor

```
long gcd (long m, long n )
/* m ve n sayılarının ortak bölenlerinin en büyüğü*/
{
  if (n == 0)
      return m;
  else
      return gcd( n, m % n);
}
```

Örnek 4.2.3: İki sayı göreceli asal olup olmadığını veren fonksiyon (relativeli prime = gcd() = 1)

Örnek 4.2.4: Sayının 9 un katı olup olmadığının basamak toplamlarının 9 olması ile bulunması (Özyinelemeye örnek olsun)

```
int kati9(long s)
{
  int t;
  if ( s < 10)
      return ( s == 9);
  else {
        t = 0;
```

```
        while ( s > 0) {
                t = t + s % 10;
                s = s   / 10;
        }
        s = kati9(s);
   }
}
```

5- Diziler

Şu ana kadar basit değişkenler kullandık (her değişkinin bir değeri var).

Yapısal değişken: benzer verilerin tek bir değişken altında gruplandırılması.

Dizi veri yapısı: Aynı tip verilerin toplanıp tek isim verilmesi.

5.1 Dizi Tanımı ve Kullanımı

```
Tip_Adı   değişken[boyut];
```

Örneğin

```
float  a[100];
```

Bu tanımlama ile a isimli değişkeni 100 gerçel değerin saklandığı bir diziyi gösterir. Bu 100 veriye a değişkeni ile erişilir.

Dizinin herhangi bir elemanına erişmek veya değiştirmek için kaçıncı eleman olduğunu gösterir indis bilgisini vermek gerekir. İlk elemanın indisi 0 dır.

```
A[4]    dizinin 5. elemanı
A[0]    dizinin ilk elemanı
A[1] := 45;    dizinin 2. elemanına 45 atanır
A[7] := A[7] + A[1];    dizinin 8. elemanına kendisi ile 2. elemanın toplamı atanır
```

Dizinin eleman değerler tanımlama anında yapılabilir.
int a[10] = {25, 18, 20, 0, 29, 5, 4, 8,19,13}

0	1	2	3	4	5	6	7	8	9
25	18	20	0	29	5	4	8	19	13

Dizi Kullanıma Örnekler

Örnek: Dizi elemanlarına değer ata, yazdır.

```
/* 1-10 arsındaki sayıların karesini dizi elemanlarına yükle yazdır */
main()
{
  int a[10];
  int i;

  for (i=0; i<=9; i++)
       a[i] = (i+1)*(i+1);

  for (i=0; i<=9; i++)
       printf("%d . elemanın değeri = %d\n", i, a[i]);
}
```

Ekranda çıkacak görüntü

```
0 . elemanın değeri   = 1
1 . elemanın değeri   = 4
...
9 . elemanın değeri = 100
```

Örnek : Dizide ilk 20 Fibonacci sayıyı oluştur

```
/* ilk 20 Fibonacci
    f[0] = 0, f[1] = 1, f[i+1] = f[i] + h[i-1] */
#include <stdio.h>
main()
{
  int fib[20];
  int i;


  fib[0] = 0;
  fib[1] = 1;
  for (i=2; i<=19; i++)
       fib[i] = fib[i-1] + fib[i-2];


  clrscr();
  for (i=0; i<=19; i++)
       printf("%d . Fibonacci sayısı = %d\n", i, fib[i]);
}
```

Ekranda çıkacak görüntü

```
0. Fibonacci sayısı = 0
1. Fibonacci sayısı = 1
2. Fibonacci sayısı = 1
...
19. Fibonacci sayısı = 4181
```

Örnek : Klavyeden 10 sayı oku. Tersten yazdır.

```
#include <stdio.h>
main()
{
  int a[10];
  int i;


  for (i=0; i<=9; i++) {
       printf("%d. sayıyı gir ",i);
       scanf("%d", &a[i]);
  }

  printf("\n----------------------\n");

  for (i=9; i>=0; i--)
```

```
        printf("%d . sırada girilen sayı = %d\n", 9-i, a[i]);
}
```

Diziyi Bir İşleve Gönderme

Şu ana kadar öğrenilen bilgi çerçevesinde işlevlere gönderilen parametrelerin içeriklerinin işlev içerisinde değiştirilmesi mümkün değildir. İşleve değişkenin değeri gönderilir. İşlev içerisindeki işlemlerden parametre etkilenmez.

Dizilerde durum böyle değildir. Normal kullanımda (şu ana kadar görülen) dizi işleve gönderildiğinde elemanlar değiştirilebilir (referans). Dizinin sadece herhangi bir elemanı gönderildiğinde ise değeri değiştirilemez (değer)

Diziyi işleve gönderirken sadece adını parametre olarak yazmak yeterlidir.

Örnek: işlev içerisnde dizi eleman değerlerinin değiştirilmesi

```
#include <stdio.h>
void kareleri(int []);
main()
{
  int a[10];
  int i;

  for (i=0; i<=9; i++)
       a[i] = i + 1 ;


  clrscr();
  printf("Dizinin elemanlarının değerleri\n");
  for (i=0; i<=9; i++)
       printf("%d   ",a[i]);

  kareleri(a);

  printf("\n\nKare alma işlemi sonrası dizinin elemanlarının değerleri\n");

  for (i=0; i<=9; i++)
       printf("%d   ",a[i]);
}

void kareleri(int a[])
{
  int i;

  for (i=0; i<=9; i++)
       a[i] = a[i] * a[i];
}
```

Ekranda çıkacak görüntü

```
Dizinin elemanlarının değerleri
1   2   3   ..                                    10

Kare alma işlemi sonrası dizinin elemanlarının değerleri
1   4   9   ..                                        100
```

Notlar

* Tanımlanan dizi boyutunun dışında bir eleman kullanımı durumunda C dili hata vermez (uyarmaz).
* İlk indisin değerinin sıfır olması dolayısıyla dizinin n. elemanın indisi n değil n-1 dir.
* Gerektiği durumlarda dizi elemanlarının ilklendirilmesi (sıfırlamak) unutulmamalı
  - tanımlama anında int a[10] = {13, 45 ..}; /* fazla eleman yazılırsa hata oluşur */
    int a[] = { 13, 45, 56}; /* boyut belirtilen değer kadar olur */
  - giriş yaptırarak scanf("%d", &a[i]);
  - doğrudan değer atayarak a[i] = 4;

Dizinin Boyutunu Değişken Kılma

#define önişlemcisi direktifi (preprocessor directive)

Simgesel sabit tanımlanmasını sağlar. C dili deyimlerinden biri değildir (#include gibi). Kullanım biçimi

#define SabitAdı Değeri

Program çalıştırıldığında derleme işlemi öncesinde program içerisinde SabitAdı simgesinim geçtiği yerlere Değeri konur.

Örnek : 25 kişilik bir sınıftaki öğrencilerin sınav sonuçlarını okuyup ortalamasını bulan program.
* not bilgilerin saklanacağı veri yapısını belirle ve tipi tanımla
* notları girdiren yöntemi yaz
* Ortalamayı bulan fonksiyonu yaz

```
#include <stdio.h>
#define OGR_SAY 25

void giris(int []);
float ortalama(int []);
main()
{
  int ogr[OGR_SAY];

  clrscr();

  giris(ogr);

  printf("Ortalama = %f\n", ortalama(ogr));
}

void giris(int ogr[])
{
  int i;

  for (i=0; i<OGR_SAY; i++)   /* 0 dan başladığı için < kullanıldı */
  {
      printf("%d. öğrencinin notunu gir : ",i+1);
      scanf("%d", &ogr[i]);
  }
}
```

```
float ortalama(int ogr[])
{
  float x;
  int i;

  x = 0;
  for (i=0; i<OGR_SAY; i++)
       x = x + ogr[i];
  return x / OGR_SAY;
}
```

```
 * 50'den küçük değerleri yazan işlev
 * 50'den küçük değerlerin sayısını veren işlev
 * en yüksek notu veren işlev
 * Standart sapmayı veren işlen ( (abs(Xi - Xort) / OGR_SAY))
```

Örnek : İki dizinin toplamını üçüncü bir diziye yazan programı yazınız (A, B aynı tip dizi, C = A + B). Doğrudan C := A + B yazılamayacağı için dizinin ilgili elemanlarını tek tek toplamalıyız.

```
#include <stdio.h>
#define MAX 20
void dizitopla(int [], int [], int []);
main()
{

  int a[MAX], b[MAX], c[MAX];
  int i;

  for (i=0; i<MAX; i++) {
       a[i] = i;
       b[i] = 2 * i;
  }

  dizitopla(a, b, c);

  clrscr();
  for (i=0; i<MAX; i++)
       printf(" %4d + %4d = %4d \n", a[i], b[i], c[i]);
}

void dizitopla(int a[], int b[], int c[])
/* c = a +b */
{
  int i;

  for (i=0; i<MAX; i++)
       c[i] = a[i] + b[i];
}
```

5.3 Sıralama

Dizi elemanlarının küçükten büyüğe veya büyükten küçüğe doğru dizilmesi işlemi. Farklı performansa sahip birçok yöntem vardır. Şimdilik en basit olanlardan biri incelenecektir.

Selection Sort (seçim)

Dizinin en küçük elemanı bulunur ve birinci eleman ile yer değiştirilir. Bu işlem (n-1). elemana kadar tekrar edilir. Her adımda en küçük eleman dizinin kalan elemanlarından bulunur. Örneğin aşağıda 4 elemanlı bir diziye yöntemin uygulanması gösterilmiştir.

Buble Sort

Bu yöntemde ardışıl iki eleman karşılaştırılır. Eğer önceki eleman sonrakinden büyük ise elemanlar yer değiştirilir. Bu şekilde dizi taranır. Eğer herhangi bir değişiklik yapılmış ise işlem tekrar edilir.

Başlangıç	1. adım	2. adım	3. adım
34	21	18	18
21	18	21	21
18	25	25	25
25	34	34	34

Görüldüğü gibi, bu yöntemde dizi sıralandıktan sonra bir defa daha taranıyor.

Selection Sort Kodlaması

1. Dizinin en küçük elemanının indisini veren işlev
2. Bu işlevi kullanarak diziyi sıralama

Bubble Sort Kodlaması

5.4 Çok Boyutlu Diziler

Çok boyutlu bilgileri veya veri tablolarını saklamak için kullanılır. İki boyutlu diziler daha sık kullanılır. Örneğin; yıllara ve aylara enflasyon rakamının takibi, matematikteki matriş işlemlerinin gerçeklenmesi, öğrenciler ve aldıkları derslerin takibi.

Tanımlama biçimi : İki türlü yapılabilir.

```
Tip   Değişken_Adı[ indis1 ][ indis2 ][ indisn ];
```

Örneğin ;

```
float Enf[5][12];
```

Enf tipi iki boyutlu bir dizidir. Birinci boyut (satır) yılları, ikinci boyut (sütun) ayları gösterir.

	1.Ay	2.Ay	3.Ay	4.Ay	5.Ay	6.Ay	7.Ay	8.Ay	9.Ay	10.Ay	11.Ay	12.Ay
1												
2												
3												
4												
5												

C dilinde boyut sayısında bir sınır yoktur. Biligisayarın belleği boyutu sınırlar.

Dizilerin bellekte saklanma biçimi :
Erişimin kolay olması için dizinin tüm elemanları peşpeşe saklanır. Burada önce satır sonra sütunlar (tanımlanış sırasına göre) saklanır .

!!! Çok boyutlu dizileri bir işleve gönderirken ilk boyut dışındaki boyutların büyüklüklerini vermek zorunludur.

Örnek : İki boyutlu dizi üzerine.

```
#include <stdlib.h>

#define SATIR 5
#define SUTUN 5

double enbuyuk(double [][SUTUN], int );
void   matris_oku(double [][SUTUN]);

main()
{
  double a[SATIR][SUTUN];
  double b[SATIR];   /* satirlardaki en buyuk elemanlar */
  double c[SATIR];   /* satirlardaki sutunlarin toplami */
  int    i, j;

  matris_oku(a);

  for (i = 0; i < SATIR; i++)
       b[i] = enbuyuk(a,i);

  /* Satirdaki elemanlarin toplamlarinin olusturdugu matris */
  for (i = 0; i < SATIR; i++)
      {
        c[i] = 0;
        for (j = 0; j < SUTUN; j++)
             c[i] = c[i] + a[i][j];
      }

  clrscr();

  for (i = 0; i < SATIR; i++) {
       for (j=0; j<SUTUN; j++)
```

```
            printf("%3.0f ", a[i][j]);                          /* Biçimli yazdırma */

        printf("   %4.0f   %4.0f\n",b[i],c[i]);
   }

}


double enbuyuk(double a[][SUTUN], int sat)
{
   double   r;
   int      i;

   r = a[sat][0];    /*   ilk eleman en buyuk   */
   for (i = 1; i < SUTUN; i++)
        if (a[sat][i] > r)
            r = a[sat][i];
   return r;
}    /* Function EnBuyuk */

void   matris_oku(double a[][SUTUN])
{
   int i, j;

   for (i = 0; i < SATIR; i++)
        for (j = 0; j < SUTUN; j++)
             a[i][j] = random(10);
}   /* matris_oku */
```

Örnek : Bir kare matrisin

- o. Matrisin simetrik olup olmamasını sınayan program
- o. Matrisin transpozesini bir diğer matrise kopyalayan program
- o. İki matrisinin çarpımını gerçekleştiren program
- o. köşegeninin alt kısmındaki elemanları sıfırlayan yöntemi,
- o. Determinantını hesaplayan fonksiyonu,
- o. M<=P ve N<=Q olmak üzere A matrisi MxN ve B matrisi PxQ boyutlarındadır. B matrisi içerisinde A matrisininin olup olmadığı belirlenecektir. A matrisinin B matrisi içerisinde kaç kez bulunduğunu ve bunların B matrisi içerisinde başlangıç konumlarını (satır ve sütun olarak) veren programı yazınız.
- o. MxN elemanlı bir matrisi tek boyutlu bir diziye dönüştüren program
- o. P elemanlı bir diziyi MxN boyutlu matrise dönüştüren program (P sayısı M ve N e tam bölünür).

```
 /* Program Matris-2   */

#include <stdlib.h>

#define SATIR 3
#define SUTUN 3
```

```
void   matris_oku(int [][SUTUN]);
void matris_yaz(int [][SUTUN]);
int simetrik(int [][SUTUN]);          /* matrisin simetrikliğini sınar */
void transpoze(int [][SUTUN], int [][SUTUN]); /* 2. = 1.nin transpozesi */
void matris_carp(int [][SUTUN], int [][SUTUN], int [][SUTUN]); /* 3 = 1*2 */
/*
  alt_ucgen_sifirla(a);

  matris_yaz(a);
  printf("------------------------\n");

  transpoze(a, b);


  matris_yaz(b);

  matris_carp(a, b, c);

  printf("------------------------\n");

  matris_yaz(c);

  if (simetrik(a))
      printf("Simetriktir\n");
  else
      printf("Simetrik değildir\n");


*/

  getch();

}


void   matris_oku(int a[][SUTUN])
{
  int i, j;

  for (i = 0; i < SATIR; i++)
       for (j = 0; j < SUTUN; j++)
            a[i][j] = 1 + random(5);
}   /* matris_oku */

void matris_yaz(int mat[][SUTUN])
{
  int i, j;

  for (i = 0; i<SATIR; i++) {
       for (j = 0; j<SUTUN; j++)
            printf("%3d ",mat[i][j]);
       printf("\n");
  }
}
```

```
int simetrik(int a[][SUTUN])
{
  int i,j;
  int durum;

  durum = 1; /*simetrik olduğunu varsay */


  for (i = 0; i<SATIR; i++)
       for (j = 0; j<SUTUN; j++)
            if (a[i][j] != a[j][i]) {
                durum = 0;
                break;

            }

  return durum;
}    /* simetrik */

void transpoze(int a[][SUTUN], int b[][SUTUN])
{
  int i, j;

  for (i = 0; i<SATIR; i++)
       for (j = 0; j<SUTUN; j++)
            b[i][j] = a[j][i];
}

void matris_carp(int a[][SUTUN], int b[][SUTUN], int c[][SUTUN])
/* c = a * b */
{
  int i, j, k;

  for (i = 0; i<SATIR; i++)
       for (j = 0; j<SUTUN; j++) {
            c[i][j] = 0;
            for (k=0; k<SUTUN; k++)
                 c[i][j] = c[i][j] + a[i][k]*b[k][j];
       }

}


void alt_ucgen_sifirla(int a[][SUTUN])
{
  int i,j;

  for (i = 0; i<SATIR; i++)
       for (j = 0; j<i; j++)
            a[i][j] = 0;

}

int det(int a[][SUTUN])
```

```
{
  int i, j, k;
  int s, sx1, sx2;

  s = 0;

/*
  for (j = 0; j<SUTUN; j++) {
       sx1 = 1;
       sx2 = 1;
       for (k = 0; k<SATIR; k++) {
            i = (j + k) % SUTUN;
            sx1 = sx1 * a[k][i];
            sx2 = sx2 * a[k][SATIR-k];
       }
       s = s + sx1 - sx2;
  }

*/
  return s;
}
```

6. Karakter işleme

Birçok program sayıların yanı sıra metinler üzerine işlem yapar. İlk akla gelen bir kelime işlem programıdır. Bu tür bir programda metinlerle ilgili olarak, araya karakter girme, silme, değiştirme ve arama gibi işlemler tanımlıdır. C dilindeki char tipi bu tür verilerin saklanması için kullanılır.

6.1 Tanımlamalar ve Kullanım

karakter: Bilgisayarda tanımlı herhangi bir karakteri gösterir. Karakter sabitler tek tırnak içinde belirtilir.

'A', '0'

'\65' : \ işaretinin ardından belirtilen ASCII kodlu karakter

C dili karakterleri ayrıca int değer gibi de görür. 'A' karakteri sayısal olarak A harfinin ASCII tablosundaki karşılığı olan 65 olarakta yorumlanabilir.

dizgi (string): Bir dizi karakterden oluşur. '\0' karakteri dizginin sonunu gösterir.Dizgi sabitler çift tırnak içinde gösterilir.

"ALİ", "Bir sayı girin"

Bu idafede dil \0 karakterini otomatik koyar.

Karakter ile dizgi birbirinden farklı şeylerdir. Dizginin sonunda herzaman \0 karakteri vardır.
'A' ile "A" birbirinin aynısı değildir. 'A' karakterini gösteren 65 değerinde bellekte int olarak salanırken "A" ise bir dizgi olup A ve \0 karakterlerini içerir.

Tanımlama biçimi:

1. Dizi şeklinde tanımlama

```
char a[7] = {'K','O',,'C','A','E','L','İ'};
char a[8] = "KOCAELİ";              /* Son eleman olrak \0 karakteri eklendiği için   eleman sayısını bir fazla
belirtmeli */                                          2. İşaretçi olarak tanımlama
```

**char ***değişken;

```
char *a;
```

Doğrudan Değer Atama

```
char a[20];
char *b;

a[0] = 'A';      /* dizinin bir elemanına değer verir gibi ve tek tırnak içinde ( karakter sabiti) */
                   \0 karakteri ayrıca belirtilmelidir.
```

Dizi biçiminde tanımlı dizgiye sabir bir dizgi atarma için strcpy işlevi kullanılır.

strcpy(char *hedef, const char *kaynak); kaynak bilgisini hedef değişkenine kopyalar (string.h)
Hedef dizgisinde kaynak dizgisini içerecek kadar yer olmalıdır.

strcpy(a, "ALİ");

b = "ALİ"; /* çift tırnak içinde atanacak değer . \0 karakteri otomatik eklenir */

Değerini Yazdırma

printf("%s", a);
puts(a); sonu \0 ile biten karakter dizisini ekrana yazar ve imleçi alt satıra geçirir (stdio.h)

Klavyeden Değer Atama

scanf("%s", a); /* Boşluk karakterine kadar okuma yapacağı için içerisinde bu karakterin geçmesi olası girişlerde doğru çalışmaz. Adres operatörü kullanılmıyor */

gets(a) ; Satır sonu karakterine kadar (ENTER tuşu) basılan karakterleri okur (stdio.h)

Karakter Dizisinin Herhangi Bir Karakterine Erişme

Dizilerde olduğu gibi erişilmek istenen karakterin indisi bildirilir.

a[2] = 'G'; b[2] = 'G';
a[4] = '\0';
printf("%c ", b[3]); /* b'nin 4. elemanını yazar */

Örnek 6.1.1 : Girilen bir metnin uzunluğunu veren programı yazınız.

```
#include <stdio.h>
main()
{
  char *s;
  int i;
  gets(s);
  i = 0;                                                    for (i=0; s[i]!='\0'; i++)
  while (s[i]!='\0')                                          {}                          /*
boş işlem */
    i++;

  printf("Uzunluk %d \n", i);
}
```

strlen (char *) : Gönderilen değerin uzunluğunu verir. (string.h)

Örnek 6.1.2: Girilen bir metni tersten yazdıran programı yazınız.

```
#include <string.h>
#include <stdio.h>
main()
{
  char *s;
  int i;
  gets(s);
  for (i=strlen(s)-1; i>=0; i--)
        printf("%c", s[i]);
```

```
  printf("\n");
}
```

Örnek 6.1.3 : Girilen cümleyi oluşturan kelimelerin sadece ilk harflerini yazdıran program.
1. c = sıradaki karakteri oku
2. Eğer c boşluk ise 4. adıma git
3. Hiç karakter yazılmaış ise c'yi yaz.
4. Eğer son karakter ise dur aksi halde bir sonraki karakteri göster ve 1.adıma git.

Yukarıdaki algoritmada çözümün en önemli noktası herhangi bir kelimenin ilk karakterinin yazılıp yazılmadığıdır. Bunun izlenmesi için bir değişken kullanalım.

```
/* Girilen cümlenin ilk harflerini yaz */
main()
{
  int i;
  int yaz; /* 1/0 -> karakteri yaz/yazma */
  char *s;

  clrscr();
  gets(s);
  yaz = 1;
  for (i = 0; i<strlen(s); i++)
      if (s[i] == ' ')
          yaz = 1;
      else
          if (yaz) {
              printf("%c", s[i]);
              yaz = 0;
          }
}
```

Örnek 6.1.4: Girilen cümleyi, kelimeleri bozmadan tersten yazdıran program.
Ali zil çaldı --> çaldı zil Ali

İki Dizginin Karşılaştırılması

C dilinde diziler doğrudan karşılaştırılamazlar. Her bir elemanını ayrı ayrı karşılaştırılmalıdır. Küçüklük-büyüklük bilgisi ASCII tabloda önce-sonra bulunma bilgisine özdeştir. Karşılaştırma amacıyla **strcmp(dizgi1, dizgi2)** işlevi kullanılır. İşlev üç değer alır.
<0 ise dizgi1 < dizgi2
=0 ise dizgi1 = dizgi2
>0 ise dizgi1 > dizgi2

Örnek 6.1.5:

```
int main()
 { char *s1 = "aaa",   *s2 = "aba;
     int i;
     i = strcmp(s1, s2);
     if (i > 0)
         printf("s1 büyük \n");
     else if (i<0)
         printf("s2 büyük\n");
     else
```

```
        printf("eşitler\n");
 }
```

İki Dizgiyi Toplama (ekleme)

Dizgiler ile ilgili diğer sık kullanılan işlev **strcat(hedef, kaynak)** tir. kaynak dizgisini hedeh dizgisine ekler. Hedef dizgisinde kaynak dizgisini içerecek kadar yer olmalıdır.

```
int main()
 {
    char *s1 = "aaa     ",   *s2 = "bbb";
    int i;
    strcat(s1, s2);                  /* (string.h) */
    printf("%s\n", s1);
 }
```

6.2 atoi , atof, atol .. İşlevlerini Gerçekleme

Genellikle bir programın akışı boyunca bazı verilerin tiplerinin değiştirilmesi gerekir. Bu değişim string tipindeki bir değişkenin sayısal tipe dönüştürülmesi veya tersi biçiminde olabilir. Sayısal string : içeriği C'nin sayı gösterimine uygun olan metin. '5.87' , '0.3E5' , '3423'

Örnek 6.2.1: String tipi bir değişkenin değerini tamsayısal tipe dönüştüren işlevi yazınız.(atoi (stdlib.h))
Girdi : Tamsayısal string
Çıktı : Karşılık gelen sayı

```
#include <stdlib.h>
#include <string.h>
int deger (char *);
main()
{
  char *c;
  int i;

  c="93t54";
  i=deger(c);
  printf("%s dizi katarının sayısal karşılığı %d \n",c, i);
}
int deger( char *s)
{
  int i,j;
  int x;    /*sayı */
  x = 0;
  for (i = 0; i<strlen(s); i++) {
       j = s[i] - '0';
       x = x * 10 + j ;
  }
  return x; }
```

Örnek 6.2.2 : Önceki örnekteki değer işlevini ondalık sayıları da (nokta dikkate alınacak) işleyecek şekilde geliştiriniz.

Yukarıda yazmaya çalıştığımız işlevin C dilinde karşılığı mevcuttur (stdlib.h).
int atoi(char *s) : tamsayıya dönüştürür.
long int atol(char *s) : uzun tamsayıya dönüştürür.

double atof(char *s) : gerçel sayıya dönüştürür.
Bu işlevler dönüştürme işlemine s içerisinde geçersiz bir karakter ile karşılaşıldığında son verir. Ve bu noktaya kadar karakterlerin oluşturduğu sayıyı verir. İlk karakter geçersiz ise işlevin değeri tanımsızdır. Aynı işlevlerin biraz daha gelişmişleri ise;

double strtod(const char *s, char **ss) :
long int strtol(const char *s, char **ss, int base) :
unsigned int strtol(const char *s, char **ss, int base) :

s : string bilgi
ss : s değişkeninde ilk geçersiz karakter ve sonrasında katar dizini gösterir
base : dönüşümün hangi tabana göre yapılacağını belirtir.

Örnek 6.2.3: atoi, atol, atof işlevlerinin davranışları.

```
main()
{
  char *s;
  int i;
  long x;
  double f;

  s="5";
  printf("%d    %ld    %f\n",atoi(s),atol(s),atof(s));
  s="5.4";
  printf("%d    %ld    %f\n",atoi(s),atol(s),atof(s));
  s="5.2e4=KL";
  printf("%d    %ld    %f\n",atoi(s),atol(s),atof(s));
}
```

s değeri	çıktı		
5	5	5	5.000000
5	5	5	5.400000
5	5	5	52000.000000

Giriş İşlevleri

```
main()
{
  char c1, c2;
  char s[10];
  clrscr();
  c1 = getchar(); /* klavyeden ENTER tuşuna basılana kadar tuş bekler */
                  /* cursor on, karakter yazılır */
  c2 = getchar(); /* klavyenin tamponundaki sıradakini okur */
  printf("\n%c   ,   %c\n",c1,c2);
  c1 = getche(); /* tuş basılmasını bekler,   ENTER beklenmez*/
                  /* cursor on, karakter yazılır */
  c2 = getche(); /* tuş basıldıktan sonra kontrol bu satıra geçer */
  printf("\n%c   ,   %c\n",c1,c2);
  c1 = getch(); /* tuş basılmasını bekler,   ENTER beklenmez */
                  /* cursor on, karakter yazılmaz      x   int putch(int c): c ekrana yazılır*/
  c2 = getch(); /* tuş basıldıktan sonra kontrol bu satıra geçer */
```

```
  printf("\n%c   ,   %c\n",c1,c2);
}
```

Örnek 6.2.4: Tamsayı bir değeri dizgiye dönüştüren programı yazınız. (ecvt, fcvt, gcvt var ancak ANSI C değil)

```
#include <stdio.h>

int main()
 {
     char *s;
     int i;
     i = 2137;
     itostr(s, i);
     printf("%d     %s\n",i, s);
}
itostr(char *hedef, int sayi)
{
  int i, j, kb;
  j = sayi;
  for (kb = 0; j>0; kb++)
       j = j / 10;
  i = sayi;
  for (j=kb-1; j>=0; j--) {
       hedef[j] = '0' + i % 10;
       i = i / 10;
  }
  hedef[kb] = '\0';
}
```

sprintf işlevi : sayısal değeri dizgiye dönüştürmek için kullanılabilir. Yapısı;

sprintf(dizgi, kontrol, değişkenler)

Kullanım şekli printf işlevine yakındır. Kontrol ve değişken bilgisine göre ekrana yazılması gereken bilgi parametre olarak gönderilen dizgiye kopyalanır.

```
char *s;
int i = 57;
sprintf(s, "%d", i);
```

ifadesi sonunda s dizgisinde "57" değeri oluşur.

```
float f = 2.1;
sprintf(s, "%f", f);
```

ifadesi sonunda s dizgisinde "2.100000" değeri oluşur.

Örnek 6.2.5: Bir dizginin belirli bir parçasını veren işlev. (strncpy(s2, s1, n))

```
#include <stdio.h>
int main()
 {
     char *s1;
     char *s2;
     s1 = "01234567890";
     altdizgi(s2, s1, 0, 5);
```

```
    printf("%s\n",s2);
}
void altdizgi(char *hedef, char *kaynak, int bas, int uzunluk)
{
  int i;
  for(i = 0; i < uzunluk && kaynak[bas + i] != '\0'; i++)
        hedef[i] = kaynak[i + bas];
  hedef[i] = '\0';
}
```

Örnek 6.2.6: Bir metin içerisinde bir metin parçasın bulma (konumunu belirleme).
Girdi : . aranacak metin parçası (String) , aramanın yapılacağı metin (String)
Çıktı : aranan bilgi var ise başlangıç konumu (Integer) =-1 ise yok demektir

```
main()
{
  char *a="12345";
  char *b="34";
  printf("%s    %s\n",a, b);
  printf("%d\n",kon(a, b));
}

int   kon(char *s1, char *s2)
{
  int i = 0, j, konum = -1;
  while (i < strlen(s1) - strlen(s2))
    {
      konum = i;
      for (j = 0; j < strlen(s2); j++)
          if (s1[i + j] != s2[j])
              konum = -1;
      if (konum >= 0)
          break;
      else i = i + 1;
    }
  return konum;
}
```

C dilinde benzer bir işlevi sağlayan deyim

```
char *strstr(const char *dizgi1, const char *dizgi2)
```

dizgi1 içerisinde dizgi2 yi arar. Var ise işlev dizgi1 içerisinde dizgi2 nin başladığı konumdan sonraki dizgiyi işaret eder. Yok ise işlevin değeri NULL dır.

NULL : işaretçinin herhangi bir yeri göstermemesi. Değerinin boş olması.

Örnek 6.2.7: Bir dizgi içerisindeki bir karakterin bulunması. i harfini İ ile değiştiren program.

```
#include <string.h>
#include <stdio.h>
int main()
 {
    char *s1, *s2, *s3;
    int i;
    clrscr();
    s1="ali zil çaldi";
```

```
    strcpy(s2, s1);
    for (i=0; i<strlen(s1); i++)
        if (s1[i] == 'i')
            s1[i] = 'İ';
    printf("%s\n",s1);
```

char *strchr(char *s, char c) : s dizgisi içerisinde c karakterinin konumunu gösteren dizgiyi gönderir.

```
    while (strchr(s2, 'i') !=NULL) {
        s3 = strchr(s2, 'i');
        s3[0] = 'İ';
    }
    printf("%s\n",s2);
 }
```

Örnek 6.2.8: Bir cümledeki tüm küçük harfleri büyüye çeviren program.
(int tolower(int c) , int toupper(int c)) (ctype.h)

```
#include <string.h>
#include <stdio.h>

void buyuk(char *);
int main()
 {
    char *s1, *s2, *s3;
    int i;
    clrscr();
    s1="ali zil çaldi";
    strcpy(s2, s1);
    for (i=0; i<strlen(s1); i++)
        printf("%c",toupper(s1[i]));

    buyuk(s2);
    printf("\n%s",s2);
    getch();
 }
void buyuk(char *s)
{
  int i;
  for (i=0; i<strlen(s); i++)
      s[i] = toupper(s[i]);
}
```

Örnek 6.2.9: Girilen metindeki harflerin sıklığını (frekansını) hesaplar.

```
#include <strinh.h>
main()
{
  char *s;
  char *harfler = "abcdefghijklmnopqrstuvwxyz";
  char c;
  int fr[26] = {0,};
  int i;
  clrscr();
  gets(s);
  for (i = 0 ; i <strlen(s); i++) {
      c = tolower(s[i]);
      if (strchr(harfler, c) != NULL)
          fr[c - 'a']++;
  }
  for (i = 0; i < 26; i++)
      printf("%c   -   %d\t\t", 'a'+i, fr[i]);
}
```

7. Diğer Veri Tipleri

Standart C dilinde şu ana kadar öğrendiğimiz veri tiplerine ek olarak aşağıdaki tipler mevcuttur.

7.1 C de Tanımlı Veri Tipleri

Veri Tipleri

Tip	Aralık	Uzunluk(Byte)
unsigned char	0 .. 255	1
char	-128 .. 127	1
enum	-32,768 .. 32,767	2
unsigned int	0 .. 65,535	2
short int	-32,768 .. 32,767	2
int	-32,768 .. 32,767	2
unsigned long	0 .. 4,294,967,295	4
long	-2,147,483,648 .. 2,147,483,647	4
float	3.4 * (10**-38) .. 3.4 * (10**+38)	4
double	1.7 * (10**-308) .. 1.7 * (10**+308)	8
long double	3.4 * (10**-4932) .. 1.1 * (10**+4932)	10

printf işlevinde dönüşüm karakteriden önce

h short
u unsigned
l long integer
L long double

u dönüşüm karakteri unsigned

7.2 Kullanıcı Tanımlı Veri Tipleri

Enumerated (Sıralı, Numaralandırılmış) Tipler

Programın okunabilirliğini arttırmak için değerlerin sıralı bir küme olarak tanımlanması.

enum [tip_adi] { sabit adı [= değer],...} [değişken listesi];

değer : sabite otomatik atanan değer. Belirtilmez ise 0'dan başlar. Sıradaki sabit öncesinin bir fazlasına eşittir.

Belirtilen tipte tanımlı bir değişken ancak listedeki değerleri alır.

enum renk {BEYAZ, SARI, SIYAH};

tanımlaması ile oluşan enum renk tipinde BEYAZ 0, SARI 1 ve SIYAH 2 değerlerine eşit sabitlerdir. Listedeki sabitler belirkeyici olmalıdır. Sayısal, karakter ve string bilgi yazılamaz.
enum renk = {"Beyaz", "Sari"}; X
enum renk = {0, 1, 2}; X

. İlişkisel işleçler tanımlıdır. Beyaz < Siyah, Sari > Beyaz

Bu tip değişkenler switch ve for deyimlerinde kontrol değişkeni olarak kullanılabilir.

Bu tipte bir değişken bilinen yapıda tanımlanır.

```
enum   renk {BEYAZ, SARI, SIYAH};
enum renk a;
```

a değişkeni BEYAZ, SARI ve SIYAH değerlerinden birini alabilir.

Örnek 7.2.1: enum kullanımı.

```
main()
{

  enum gunler {PAZAR,PAZARTESI,SALI,CARSAMBA,PERSEMBE,CUMA,CUMARTESI };
  enum gunler gun;

  clrscr();

  for (gun = PAZAR; gun <= CUMARTESI; gun++)
       printf("%d ", gun);
  printf("\n");


  for (gun = PAZAR; gun <= CUMARTESI; gun++)
       switch (gun) {
            case PAZAR:          printf("Pazar"); break;
            case PAZARTESI:    printf("Pazartesi"); break;
            case SALI:           printf("Sali"); break;
            case CARSAMBA:     printf("Carsamba"); break;
            case PERSEMBE:     printf("Persembe"); break;
            case CUMA:           printf("Cuma"); break;
            case CUMARTESI:    printf("Cumartesi"); break;
        }    /* Case */
}
```

Örnek 7.2.2: enum kullanımı.

```
main()
{
  enum gunler {PAZAR,PAZARTESI,SALI,CARSAMBA,PERSEMBE,CUMA,CUMARTESI };
  enum gunler gun;
  char *gunad[]={"Pazar","Pazartesi","Sali","Carsamba",
                 "Persembe","Cuma","Cumartesi"};
  clrscr();
  for (gun = PAZAR; gun <= CUMARTESI; gun++)
       printf("%s\n", gunad[gun]);
```

}

Örnek 7.2.3: Sabitlere farkli değerler atama

```
enum renkler { sari = 4, mavi, kirmizi };
enum renkler i;
main()
{
   i = mavi;
   printf("%d   \n", i);
}
```

Ekrana 5 yazar.

Problem : Bu tip değişkenler doğrudan okunup yazılamazlar.

* Okumak ise yazmaya oranla biraz daha zordur. Okuma işlemi diğer standart tipler kullanılarak dolaylı bir şekilde gerçekleştirilir.

8- Yapılar (struct)

Farklı tipte birden fazla verinin oluşturduğu bütün.

Veri tipi olarak daha önce diziyi tanımlamıştık. Dizide birbirinin aynı tipi olan veriler peşpeşe saklanıyordu. Yapılar da aynı dizi gibi birbirleriyle ilgili olan verileri içerir. Fakat diziden farkı, tek bir yapı elemanı birbirnden farklı tipte birden fazla veriyi içerir. Bu sayede bilgiler daha düzenli şekilde gösterilir.
Yapı yeni bir tip olmayıp var olan standart tiplerden oluşur.

8.1 Yapı Tanımı

Bir öğrencinin numarasını, adını ve sınavdan aldığı notu içeren bir yapı tanımlaması aşağıdaki gibidir.

```
struct ogryapi {
        char numarasi[10];
        char adi[20];
        int notu ;
}

struct ogryapi ogr;
```

Artık ogryapi isimli bir veri tipi vardır ve üç üyeden (alandan) oluşmaktadır (member =üye : kayıdı oluşturan farklı veri tiplerinin her biri).

* Tanımlamada her alanın adı ve tipi belirtilir.
* Yapı tipi olduğunu gösterir struct saklı sözcüğü kullanılır.
* Aynı alan adı farklı yapılarda (bağımsız olarak) kullanılabilir.

Tanımlama genel olarak

```
struct Yapı_Adı {
        Alan1;
        Alan2;
              . . .
        Alan_n;
};
```

Bu tipte bir değişken ise

```
        struct Yapı_Adı Değı_Adı;
```

şeklinde yapılır.

Tanımlanan tipte bir değişken tanımlandığınıda, değişken için bellekte tüm alanları içerecek şekilde yer ayırılır. İlgili değişkendeki herhangi bir alan üzerinde işlem yapmak için aralarında nokta karakteri olmak koşuluyla sırasıyla değişkenin adı ve ilgili alanın adı verilir. Yukarıdaki öğrenci örneğinde öğrecinin numarasına ogr.numarasi şeklinde erişilir.

Örnek 8.1.1: Karmaşık sayılarda toplama işlemi gerçekleştirme.

```
main()
{
```

```
  struct complex {
    float real, imag;
  };
  struct complex z1, z2, z3;

  clrscr();
  printf("1. sayının reel ve imajiner kısımları gir ");
  scanf("%f %f", &z1.real, &z1.imag);
  printf("2. sayının reel ve imajiner kısımları gir ");
  scanf("%f %f", &z2.real, &z2.imag);

  z3.real = z1.real + z2.real;
  z3.imag = z1.imag + z2.imag;
  printf("%2.0f + %2.0fj\n",z3.real, z3.imag);
}
```

Örnek 8.1.2: Yapı içerisinde başka bir yapı kullanımı.

```
  struct tarih {
      int    gun;
      int    ay;
      int    yil;
  };
  struct kisiler {
      char adi[21];
      struct tarih   dogum;
  };
main()
{
   struct kisiler   a;
   strcpy(a.adi,"veli");
   a.dogum.gun = 15;
   a.dogum.ay = 8;
   a.dogum.yil = 1905;

   printf("%s %d\n",a.adi,a.dogum.yil);
   getch();
}
```

Diğer bir tanımlama şekli

```
struct   {
           Alan1;
           Alan2;
                  . . .
           Alan_n;
} değişkenler;
```

```
  struct {
    float real, imag;
  }z1, z2, z3;
```

Örnek 8.1.3: Öğrenci ile ilgili olarak numara ve sınav notu oku. Sınavın ortalamasını ve ortalamayı geçenleri listele

```
#define OGRSAY   3
main()
{
    struct ogrenci{
      char no[10];
      int   notu;
    };

    struct ogrenci ogr[OGRSAY];
    int i, j;
    float t, ort;

    clrscr();

    for (i=0; i<OGRSAY; i++) {
         printf("%d. öğrencinin numarası ve notu : ", i+1);
         scanf("%s%d",ogr[i].no, &ogr[i].notu);
    }

    t = 0;
    for (i=0; i<OGRSAY; i++)
         t = t + ogr[i].notu;

    ort = t / OGRSAY;
    printf("-----------------------------------------\n");
    printf("Ortalama = %f\n", ort);
    printf("-----------------------------------------\n");
    printf("Ortalamayı geçen öğrenciler\n");
    for (i=0; i<OGRSAY; i++)
         if (ogr[i].notu > ort)
             printf("%s\t\t%d\n", ogr[i].no, ogr[i].notu);

    getch();
}
```

Tanımlama Anında Yapı Değişkenine Değer Atama

```
struct ogryapi     a = {"95003", 56};
```

Veri Tiplerine Yeni İsim Verme

C dilinde var olan veri tiplerini yeni bir isim vermekmümkündür. Tanımlama biçimi:

typedef tanımlı_tip yeni_ismi;

```
typedef int tamsayi;
```

Artık tamsayı isminde bir veri tipi vardır. Değişken tanımlamada kullanılabilir.

```
tamsayi i, j;


typedef unsigned char byte;
byte x;
```

Yapılar ile kullanımı

```
struct complexyapi {
        float re, im;
};
tydef struct complexyapi complex;
complex z1, z2;
```

veya

```
typedef struct { float re, im } complex;
complex z;
```

Örnek 8.1.4: İşlevin değerinin yapı olması durumu

```
typedef struct {
  float re, im;
} complex;

main()
{
  complex topla(complex, complex);
  complex cikart(complex, complex);
  complex carp(complex, complex);

  complex z1={1,1}, z2={2,2};
  complex z3;

  clrscr();
  printf("%3.0f %3.0fj     %3.0f %3.0fj\n",z1.re,z1.im,z2.re,z2.im);

  z3 = topla(z1, z2);
  printf("%3.0f   %3.0f\n",z3.re,z3.im);

  z3 = cikart(z1, z2);
  printf("%3.0f   %3.0f\n",z3.re,z3.im);

  z3 = carp(z1, z2);
  printf("%3.0f   %3.0f\n",z3.re,z3.im);
  getch();
}
```

```
complex topla(complex a, complex b)
{
 complex z;
 z.re = a.re + b.re;
 z.im = a.im + b.im;
 return z;
}
complex cikart(complex a, complex b)
{
 complex z;
 z.re = a.re - b.re;
 z.im = a.im - b.im;
 return z;
}
complex carp(complex a, complex b)
{
 complex z;
 z.re = a.re * b.re - a.im * b.im;
 z.im = a.re * b.im + a.im * b.re;
 return z;
}
```

Union: Değişken Yapısı

Bir yapıdaki üyelerin hepsi aynı durumda kullanılmayabilir. Üyelerin bir bölümü belirli bir durum için geçerli iken bir kısmı farklı bir durum için geçerlidir. Gereksiz yer harcamalarına karşı union tipi kullanılabilir.
Tanımlama biçimi:

```
union ad {
      tip değişken_1;
      tip değişken_2;
      ...
      tip değişken_n
 };
```

Aynı yerde belirtilen alanlardan herhangi biri saklanır. En uzun alan ne ise o kadar yer ayrılır.

```
union ikisi {
     int kisa;
     long uzun;
};
union ikisi i;
main()
{
  i.uzun = 32768;
  printf("%d   %ld \n",i.kisa, i.uzun);   {-32768 32768 }
}
```

Union Değişkenin Tanımlama Anında Değer Alması

Değişkene tanımlama anında değer atandığında alanlardan ilki kullanılıyor gibi davranır dil. Bu nedenle atanacak değerin tipine dikkat edilmelidir.

```
union sayilar {
    int i;
    float f;
};
union sayilar a = {2};      /* a.i = 2 anlamındadır */

union sayilar a = {2.7};      /* a.i = 2 değeri atanır. */
```

Bitler Üzerine Çalışmalar

VE , VEYA ve DEĞİL işleçleri mantıksal ifadeleri bağlamakta kullanıldığı gibi doğrudan tamsayılar üzerinde de kullanılabilir. Bu kullanımda işleçler doğrudan tamsayıyı oluşturan bitleri etkiler.

```
 x &  y   : x ve y tamsayılarının bitlerinin sırayla VE işlemine tutar.   (AND)
 x | y    : x ve y tamsayılarının bitlerinin sırayla VEYA işlemine tutar.   (OR)
~ x       : x   tamsayının her bir bitinin tersini alır (NOT)
x ^ y     : x ve y tamsayılarının bitlerinin sırayla dışlayan VEYA işlemine tutar.   (exclusive OR)
x <<  n   : x sayısını n bit sola kaydırır. Boşalan yerlere 0 gelir.
x >> n    : x sayısını n bit sağa kaydırır. Boşalan yerlere 0 gelir.
```

x	y	&	|	^
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

```
    5   &    3   = 1
(101) &  (011) = (001)
         |          = (111) yani 7
5 << 1 = 10 --> (101) shl 1 = (1010)
```

Hex-Octal gösterim

```
0xn : n 16 lıdır (HEXADECIMAL)        0x11 : 17
0n   : n 8 lidir    (OCTAL)                  011    : 9
n     : n 10 ludur (DECIMAL)             11 :   11
```

Klavyedeki tuşların durumunu gösterir bilgi bellekten okunduğunda her bitin anlamı şöyledir.

Bit No	
0	sağ shift basılı/basılı değil
1	sol shift basılı/basıldı değil
2	kontrol tuşu basılı/basılı değil
3	alt tuşu basılı/basılı değil
4	scroll tuşu açık/kapalı
5	num lock tuşu açık/kapalı

6 caps lock tuşu açık/kapalı

Bu durumda num lock tuşunun açık olup olmadığını anlamak için okunan bilginin 5. bitinin değerini sınamak gerekecektir. Okunan bilginin x olduğunu varsayr ise;

```
if (x & 32)
    printf("num lock açık");
else
    printf("num lock kapalı");
```

x & 32 işlemine x değerinin 32 sayısı ile maskelenmesi denir. 32 değeri maske diye çağırılır. 32 sayısının 8 bit olduğunu düşünürsel sayının sadece 5. biti bir olup diğerleri sıfırdır.

32 = (0010 0000)

x & 32 işlemi ile x değerinin 5. bitinin bir olup olmaması sınanır. Bir ise sonuç 32 (farklı sıfir), sıfır ise sonuç 0 olur.

Örnek 8.1.5: Klavyedeki Num Lock tuşunun durumunu söyleyen programı yazınız.

```
#include <dos.h>
main()
{
 unsigned int b1;
 clrscr();
 b1 = peek(0x0040, 0x0017);   /* dos.h> */
 if (b1 & 32)
   printf("Num lock açık\n");
 else
   printf("Num lock kapalı\n");
 getch();
```

İlgili işlevler

poke (segment, offset, int value)	X	peek (segment, offset)
pokeb (segment, offset, char value)	X	peekb(segment, offset)

Örnek 8.1.6: Öğrencinin doğum tarihi 2 Byte tamsayı olarak saklanmaktadır.
Bu bilginin kodlaması şöyledir.

bit	anlamı
0 - 4	gün
5 - 8	ay
9 -15	yıl (+1970)

Böyle bir bilgiyi çözen program.

```
#include <dos.h>
main()
{
 unsigned int i;
 clrscr();
 scanf("%d", &i);
```

```
 printf("%2d", i&0x1f);
 i = i >> 5;
 printf("/%2d", i&0x1f);
 i = i >> 4;
 printf("/%4d", i+1970);

}
```

Örnek 8.1.7
unsigned swap(unsigned) : gönderilen tamsayının düşük ile yüksek sekizlisinin yerini değiştirilmişini verir
unsigned max(unsigned) : gönderilen tamsayının düşük sekizlisi ile yüksek sekizlisinden büyüğünü verir

```
main()
{
  unsigned i, j;

  i = 0xAA11;
  clrscr();
  printf("%x    %x   %x\n", i, swap(i), max(i));

  getch();
}

unsigned swap(unsigned a)
{
 return (a>>8) | (a<<8);
}
unsigned max(unsigned a)
{
  unsigned lo, hi, m;

  lo = a & 0x00FF;
  hi = a >> 8;
  m = (lo>hi) ? lo:hi;
  return m;
}
```

Örnek 8.1.8: Aşağıdaki işlevlerin gerçekleştirilmesi.

```
void binary_yaz(unsigned x); { x tamsayısının 2li düzendeki karşılığını yazar }
unsigned copybits(x, b, n)    { x sayısının sağdan b. bitinden itibaren n bitini verir}
unsigned ters(x, b, n):          { x sayısının sağdan b. bitinden itibaren n bitini tersini alır}
unsigned rdon(x,   n):           { x sayısını n bit sağa döndürür}
```

/* Programın başlangıcı */

```
typedef unsigned int word; /* kolaylık i.in */

void binary_yaz(word);
word copybits(word x, word b, word   n);
word ters(word x, word b, word n);
word rdon(word x, word n);
```

```
word sdon(word x, word n);

main()
{
  word   i, j;

  clrscr();

  for (i=16; i>0; i--)
       printf("%x", i-1);
  printf("\n");

  i = 0xee;
  binary_yaz(i); printf("\n");
  j = copybits(i, 5, 4);   binary_yaz(j); printf(" copybits(i, 5, 4) \n");

  j = ters(i, 5, 4);   binary_yaz(j); printf(" ters(i, 5, 4)\n");

  j = rdon(i, 4);   binary_yaz(j); printf(" rdon(i, 4)\n");

  getch();

}
word copybits(word x, word b, word   n)
/*   x sayısının sağdan b. bitinden itibaren n bitini verir.   */
/*   ilk bitin numarası 0   */
{
  word i;

  i = x >> (b + 1 - n);    /*   ilgili bit bloğunun sağa dayalı olacak   */
                           /*   biçimde kaydır.                          */

  i = i & ~(~(0) << n);    /*   sağdaki n bit için maske oluştur   */
  return i;
}    /*   End Of copybits   */

word ters(word x, word b, word n)
/*   x'in b. bitinden itibaren n bitin tersini alır   */
{
  word p, r;

  p = ~(x);
  r = ~(~(0) << n) << (b - n + 1);    /*   seçilen bitler için maske */
  p = p & r;

  x = x & ~(r);
  return x | p;
/*
8 bit için ters(. ,5,4)
       76543210
x = 0110 1000
p = 1001 0111
r =   00111100      ~(~(0) << n) << (b - n + 1)
p = 00010100          p = p & r
```

```
x = 0100 0000          x & ~(r)
    = 01010100           x | p   dönen
}     /*   End of test   */

word rdon(word x, word n)
/*   x'i n bit sağa döndürür   */
{
  word i;
  for (i = 1; i <= n; i++)
     {
        if (x % 2 == 1)
           {
             x = x / 2;
             x = x | 0x8000;
           }
        else
           x = x / 2;
     }
  return x;
}     /*   End of rdon   */

word sdon(word x, word n)
/*   x'i n bit sağa kaydırır. RDon fonksiyonun başka biçimi */
{
  word i;

  i = copybits(x,n - 1,n);
  x = x >> n;
  i = i << (16 - n + 1);
  return x | i;
}     /*   End of RDon   */

void binary_yaz(word x)
{
  int i;
  word m;

  m = 0x8000;
  for (i = 0; i<16; i++) {
       if (x & m)
           printf("1");
       else
           printf("0");
       m = m >> 1;
  }
}
```

9. Pointer (gösterici, işaretçi)

9.1 Tanımlanması ve Kullanımı

Bir veri bloğunun bellekte bulunduğu adresi içeren (gösteren) veri tipidir. Tanımlama biçimi:

```
veri tipi   *p;
```

p değişkeni <veri tipi> ile belirtilen tipte bir verinin bellekte saklandığı adresi içerir.

```
int *iptr;
float *fptr;
```

Bu kadar tanımla sonucunda bellekte p değişkeni mevcuttur. Ancak işaret ettiği veri bloğu yoktur. Bunun için iki yol vardır. Birincisi kullanılan herhangi bir değişkeni işaret etmek, ikincisi ise veri bloğunu boş belleği kullanarak oluşturmak.

1. İşaretçi değişkenin var olan bir değişkenin bulunduğu adresi göstermesi.

Bu işlemi yapabilmek için var olan değişkenin adresinin bilinmesi gerekmektedir.

& işleci : Bir değişkenin adresinin belirlenmesi için kullanılır. Kullanım biçimi:

```
&değişken
```

&i : i değişkenin adresini verir.

```
main()
{
  int i;
  int *iptr;
  i = 5;
  iptr = &i;
  clrscr();
  printf("i değişkeninin adresi      %p\n", &i);
  printf("iptr değişkeninin değeri %p\n", iptr);
}
```

Bellek modeline göre SSSS:OOOO veya OOOO biçiminde adres yazar.

8FF8:1000

2. Veri bloğunu boş belleği kullanarak oluşturmak.

Bu yolla veriler için dinamik yer ayırılır. Bunun için malloc işlevi kullanılır
void *malloc(n) : Boş bellekten n byte yer ayırıp başlangıç adresini döndürür.
iptr = (*int) malloc(2);
!!!!!!!!! Daha sonra dönüş yapılacak. sizeof, cast işleci (*tip) ...

Veriye işaretçi değişken yoluyla erişim

Bir işaretçinin gösterdiği adresteki veriye erişmek için işaretçi değişkeninin önüne * karakteri konur.

```
main()
{
  int i;
  int *iptr;
  iptr = &i;
  *iptr = 8;
  printf("i değişkeninin değeri %d\n", i);
  printf("iptr adresinin içeriği %d\n", *iptr);

}
```

Ekranda çıktı :
i değişkeninin değeri 8
iptr adresinin içeriği 8

!!! İşaretçi değişkenin gösterdiği adresin içeriği değişken ilklendirmeden kullanılmamalıdır

9.2 İşaretçi Aritmetiği

İşaretçi değişkenler üzerinde toplama ve çıkartma işlemleri (++, --) geçerlidir. Ancak eklenecek değer tamsayı olmalıdır.
İşaretçi değişkenin değeri 1 arttırıldığı zaman değişken bir sonraki veri bloğunu işsaret eder. Değişkenin alacağı yeni değer işaretçi değişkenin ne tip bir veri bloğunu işaret ettiğine bağlıdır.

```
int *iptr, i;
...
iptr = &i;          i değişkenin adresinin 1000 olduğunu varsayalım. iptr nin değeri 1000 dir.
iptr++;               iptr nin değeri 1002 olur. (  int değeri işaret ettiği için)
```

aynı örneği double için yaparsak

```
double *iptr, i;
...
iptr = &i;          i değişkenin adresinin 1000 olduğunu varsayalım. iptr nin değeri 1000 dir.
iptr++;               iptr nin değeri 1008 olur. (  double değeri işaret ettiği için)
```

```
int *iptr, i, j;
...
iptr = &i;          i değişkenin adresinin 1000 olduğunu varsayalım. iptr nin değeri 1000 dir.
*(iptr+4)=2;        1008 adresinin içeriğini 2 yapar.
```

!!! Arttırma işaret edilen veri bloğuna göre yapılır Yani bir sonraki veri bloğunun gösterilmesi sağlanır.

iptr++ ; bir sonraki veri bloğunu göster
(*iptr)++; iptr değişkeninin gösterdiği adresteki değeri 1 arttır

9.3 İşaretçiler ve Diziler

İşarteçiler üzerinde geçerli aritmetik yardımıyla dizilere işaretçi değişkenler ile erişmek mümkündür.

```
#include <stdio.h>
main()
{
  int i[10], j;
  int *iptr;

  for (j=0; j<10; j++)
       i[j]=j;

  /* Dizinin başlangıç adresine erişmek için ilk elemanın adresi kullanılabilir  &i[0]  veya doğrudan */

  iptr = i;

  clrscr();

  for (j=0; j<10; j++) {
       printf("%d ", *iptr);
       iptr++;
  }
  printf("\n");
/* iptr artık dizinin başını göstermez */

  iptr = i;
  for (j=0; j<10; j++)
       printf("%d ", *(iptr+j));

  printf("\n");
/* iptr hala dizinin başını gösterir */
  getch();
}
```

Örnek 9.3.1: İşaretçi ve dizgi kullanımı.

```
#include <stdio.h>
main()
{
  char *a="1234567890";
  char b[11];
  char *p1, *p2;
```

```
  printf("%s\n", a);
  p1 = a;
  p2 = b;
  while (*p1 != '\0') {
       *p2 = *p1;
       p1++;
       p2++;
  }
  printf("%s\n", b);
}
```

9.4 İşlevleri Referans Yoluyla Çağırma

Şu ana yazdığımız işlevlerde gönderilen parametrelerin (diziler hariç) değerlerinin değiştirilmesi mümkün değil idi. İşlev çağırıldığı zaman parametrelerin bir kopyası çıkartılıp işleve gönderiliyordu. Bir işlevin birden fazla değer gönderebilmesi için işaretçilere gereksinimiz vardır.

```
void arttir(int);
main()
{
  int i;
  i = 5;
  printf("öncesi %d\n", i);
  arttir(i);
  printf("sonrası %d\n", i);
  getch();
}
void arttir(int k)
{
  k++;
}
```

Çıktı :
öncesi 5
sonrası 5

Gönderilen parametrenin kopyası işleve gönderildiği için işlev içerisinde yapılan değişiklikler işlevin çağırıldığı yeri etkilemez. Eğer parametredeki değişikliklerin işlevin çağırıldığı yerde de geçerli olmasını istiyorsak işleve parametrenin adresini göndermek gerekir.

```
void arttir(int*);
main()
{
  int i;
  i = 5;
  printf("öncesi %d\n", i);
  arttir(&i);
  printf("sonrası %d\n", i);
  getch();
```

```
}
void arttir(int *k)
{
  (*k)++;
}
```

öncesi 5
sonrası 6

Örnek 9.4.1: Sayısal dizgiyi tamsayıya dönüştüren işlevde iyileştirme. Geçersiz karakterin konumu da verilsin.
int deger(char *s, int *konum)
konum = -1 ise tüm karakterler rakam
>=0 ise geçersiz karakterin konumu

Örnek 9.4.2 : Sıraya dizme. Yer değişikliği işlevde ve parametrelere referans yolu ile erişim.

```
#include <stdio.h>
#include <conio.h>

#define N 20
void degistir (int *, int *);

main()
{
   int s[N];
   int i, k;

   clrscr();

   for (i=0; i<N; i++) {
        s[i] = rand() % 100;
        printf("%4d",s[i]);
   }
   printf("\n");
   k=1;
   do {
       k=0;
       for (i=0; i<N-1; i++)
            if (s[i] > s[i+1]) {
                degistir (&s[i], &s[i+1]);
                k = 1;
            }
   } while (k);

   for (i=0; i<N; i++)
        printf("%4d",s[i]);
   printf("\n");
   getch();
```

```
}

void degistir (int *a, int *b)
{
  int gec;
  gec = *a;
  *a = *b;
  *b = gec;
}
```

!!! Dizilerde işaretçi olduğu için a değişkeni bir dizi(veya işaretçi ise)
a[i] ile *(a+i) ifadeleri aynı anlamı taşır.

Örnek 9.4.3: İşleve gönderilen dizinin işlev içerisinde işaretçi olarak kullanımı.

```
#include <stdio.h>
#include <conio.h>

#define N 5

float ort (int *);

main()
{
    int s[N];
    int i, k;

    clrscr();
    for (i=0; i<N; i++) {
          s[i] = rand() % 100;
          printf("%4d",s[i]);
    }
    printf("\n");
    getch();
}

float ort (int *a)
{
  int i;
  float t = 0;

  for (i=0; i<N; i++)
        t = t + *(a+i);

  return t/N;
}
```

Örnek 9.4.5: işleve gönderilen işaretçinin işlev içerisinde dizi olarak kullanımı .

void malloc(n): En az n byte uzunluğunda bellekten yer ayırır. İşlevin değeri
>0 ise bloğun bellekteki yeri, NULL yer yok demektir.

```
int *i;
i = (int *) malloc(2000) ;      2000 byte yer ayırıp bloğun başlangıç adresini i 'ye atar
                                                    ( 1000 elemanlı int dizisi )
double *x;
x = (double *) malloc(8*2000);   2000 elemanlı double dizi
```

sizeof(n) : n ifadesinin/tipinin byte olarak uzunluğunu verir.

```
i = (int *) malloc(1000*sizeof(int)) ;      1000 tane int değer içerecek bellek uzunluğu

x = (double *) malloc(2000*sizeof(double));   2000 elemanlı double dizi
```

void free (void *block) : mallock işlevivi tersi. Block değişkenin tuttuğu yeri boş belleğe gönderir

```
#include <stdio.h>
#include <conio.h>
#define N 8
float ort (int []);

main()
{
    int *s;
    int i, k;

    s = (int *) malloc(2*N);
    clrscr();
    for (i=0; i<N; i++) {
         s[i] = rand() % 10;
         printf("%4d",*(s+i));
    }
    printf("\n");

    printf("Ortamala = %.2f\n",ort(s));

    getch();
}

float ort (int a[])
{
  int i;
  float t = 0;

  for (i=0; i<N; i++)
       t = t + a[i];
  return t/N;
}
```

Örnek 9.4.6 : Bir dizinin elemanlarının işaretçi olması.

Daha önce yapılan bir örnekte ay isimleri bir dizide saklanmıştı.

```
main()
{
  char *aylar[] = {"", "Ocak", "ªubat", "Mart", "Nisan",
                   "Mayıs", "Haziran", "Temmuz", "Ağustos",
                   "Eylül", "Ekim", "Kasım", "Aralık"};
  int i;
  printf("Ayın sırasını gir "); scanf("%d", &i);
  if (i>0 && i<13)
     printf("%s\n", aylar[i]);
  getch();
}
```

Benzer şekilde

```
        float *a[100];
```

tanımlaması her bir elemanı bellekte bir 'float' sayıyı gösteren işaretçi olan 100 elemanlı bir dizidir.

Örnek 9.4.7 : Bir işaretçinin adresini içeren işaretçiler.

```
main()
{
  int i;
  int *iptr;
  int **iptrptr;

  i = 5;
  iptr = &i;
  iptrptr = &iptr;

  clrscr();
  printf("      i      ve      &i      :   %d     %p\n", i, &i);
  printf("   *iptr   ve     iptr     :   %d     %p\n", *iptr, iptr);
  printf("*iptrptr ve   iptrptr :   %p     %p\n", *iptrptr, iptrptr);

  getch();
}
```

Ekrana çıktı:

```
        i      ve      &i       :   5     8FDD:1000
```

```
    *iptr   ve    iptr     :  5    8FDD:1000
*iptrptr ve   iptrptr :   8FDD:1000    8FDD:0FFC
```

9.5 İşaretçiler ve Yapılar

Bir işaretçi işleve parametre olarak gönderildiğinde basit değişken gibi değişkenin kopyası alınıp gönderiliyordu. Yapının büyük olduğu durumlarda bu da sorun çıkartır. İşaretçinin bir yapı verisini göstermesi.

```
struct ogrenci{
   char no[10];
   int   notu;
};

struct ogrenci *a
```

Tanımlamasında a değişkenini oluşturan alanlara erişmek için, bilinen yol:

```
        *a.notu=56;
        strcpy((*a).no, "95001");
```

Bunun farklı kullanımı:

```
        a->notu=56;
        strcpy(a->no, "95001");
```

Örnek 9.5.1 : Yapının adresinin işleve gönderilmesi.

```
#include <stdio.h>
typedef struct {
     char adi[35];
     char adres1[40];
     char adres2[40];
     char tel[15];
     float borc;
} kisiler;
void yaz(kisiler *z);
main()
{
   kisiler   a;
   clrscr();
   printf("Adını gir   : "); gets(a.adi);
   printf("Adres-1      : "); gets(a.adres1);
   printf("Adres-2      : "); gets(a.adres2);
   printf("Telefonu     : "); gets(a.tel);
   printf("Borcu        : "); scanf("%f", &(a.borc));

   yaz(&a);
}
void yaz(kisiler *z)
```

```
{
  clrscr();
  printf("Adı           : "); puts(z->adi);
  printf("Adresi        : "); puts(z->adres1);
  printf("              : "); puts(z->adres2);
  printf("Telefonu      : "); puts(z->tel);
  printf("Borcu         : "); printf("%.0f\n", z->borc);
}
```

9.6 Dinamik bellek kullanımı

Üç boyutlu dizi tanımı ve kullanımı.

Üç boyut --> Gerilim - Akım - Zaman

Örnek 9.6.1: Tek boyutlu diziyi üç boyutlu gibi kullanma.

```
/* Üç boyutlu dinamik dizi kullanımı. Her boyut farklı uzunlukta
    Dizi tek boyutlu gözüküyor. Ancak indis, hesaplanarak bulunuyor

    Yani

        *(a + i*y*z + j*z +k)
                  ^         ^
                  ^         ^ 3. boyutun uzunluğu
                  ^2. boyutun uzunluğu

    şeklinde kullanılabilir.
*/

#define x      4
#define y      5
#define z      9
void matris_yaz(int *);
main()
{
  int *a;
  int i, j, k;

  a=(int *) malloc(x * y * z * sizeof(int)); /* eleman sayisi kadar yer ac */

  clrscr();

  for (i=0; i<x; i++) {
       printf("i = %d \n", i);
       for (j=0; j<y; j++) {
            for (k=0; k<z; k++) {
                 *(a + i*y*z + j*z +k) = i*j*k;
                 printf("%5d ",*(a + i*y*z + j*z +k));
            }
            printf("\n");
```

```
        }
    }

    matris_yaz(a);

}

void matris_yaz(int *m)
{
    int i, j, k;

    clrscr();
    for (i=0; i<x; i++) {
        printf("i = %d \n", i);
        for (j=0; j<y; j++) {
            for (k=0; k<z; k++) {
                printf("%5d ",*m);
                *m++;
            }
            printf("\n");
        }
        getch();
    }
}
```

Örnek 9.6.2:

```
/* Uc boyutlu dinamik dizi kullanimi. Her boyut farkli uzunlukta
    Bu yontem ile diziye normal dizi gibi erismek mumkun
    Yani
        a[i][j][k]
    seklinde kullanilabilir.
*/

#define x      8
#define y      4
#define z     10
main()
{
/*
    typedef int *boyut1;
    typedef boyut1 *boyut2;
    typedef boyut2 *boyut3;

    boyut3 a;
*/
    double ***a;

    int i,j,k;

    a=(double *) malloc(x*sizeof(double*));                 /* 1. boyut icin yer ayir   (işaretçiler)    */
    for (i=0; i<x; i++)                                                 /* 2. boyut icin yer ayir.   (işaretçiler)   */
        *(a+i)=(double *) malloc(y*sizeof(double*)); /* 1. boyutun her elemani n    */
                                                                        /* elemanli diziyi gosterir    */
```

```
  for (i=0; i<x; i++)                          /* 3. boyut icin yer ayir (matrisin elemanlar)      */
       for (j=0; j<y; j++)
            *(*(a+i) + j) = (double *) malloc(z*sizeof(double));

  clrscr();

  for (i=0; i<x; i++)
       for (j=0; j<y; j++)
            for (k=0; k<z; k++)
                 *(*(*(a + i) + j) + k) = i*j*k;
  for (i=0; i<x; i++) {
       printf("i = %d \n", i);
       for (j=0; j<y; j++) {
            for (k=0; k<z; k++)
                 printf("%4.1f ",a[i][j][k]);
            printf("\n");
       }
       getch();
  }
}
```

9.7 İşaretçilerle İlgili Diğer Konular

İşaretçinin Belirli Bir Adresi Göstermesi

```
#include <dos.h>
#include <stdio.h>
char far *ekran;
void kaydir_Y(void);
void kaydir_A(void);
void main()
{
    int i, j;
    char c;

    ekran = MK_FP(0xB800, 0);
    clrscr();
    for (i=0; i<25; i++)
         for (j=0; j<80; j++)
              ekran[160*i+2*j] = 65+i;

    while (1) {
        c = toupper(getch());
        switch(c) {
            case 'A' :   /* yukari */
                         kaydir_Y();
                         break;
            case 'Z' :   /* asagi */
                         kaydir_A();
```

```
                        break;
            case 'Q' : exit(0);
        }
    }
}
void kaydir_Y(void)
{
  int i, j;
  for (i=8; i<=12; i++)
        for (j=30; j<60; j++)
            ekran[160*(i-1) + 2*j] = ekran[160*i + 2*j];
}
void kaydir_A(void)
{
  int i, j;
  for (i=12; i>=8; i--)
        for (j=30; j<60; j++)
            ekran[160*(i+1) + 2*j] = ekran[160*i + 2*j];
}
```

İşlev İşaretçileri

İşaretçinin bir işlevin bulunduğu adresi içermesi durumudur. Normal işaretçi gibi işlevin adresini içeren değişken tanımı yapılmalıdır. Örneğin;

```
int (*fnptr) (int, int)
```

fnptr değişkeni iki tane int parametresi olup bir int değer geri gönderen bir işlevin adresini içerebilir.

(int *fnptr (int, int) : iki int parametresi olup int işaretçi geri gönderir)

Örnek 9.7.1: Aynı isim ile farklı iki işlevi çağırma.

```
int kare(int);
int kub(int);
main()
{
  int (*islem)(int);                /* bir int değer alıp geriye int değer gönderen     bir işlevin adresi   */
  int i;
  char c;

  clrscr();

  printf("1 / 2 : kare / küb hesabı : ");

  c = getch();
  printf("\nSayıyı gir : ");
  scanf("%d", &i);

  if (c == '1')
     islem = kare;          /* kare işlevinin adresi islem değişkenine kopyalanır */
  else
```

```
        islem = kub;

   printf("Sonuç = %d\n", islem(i));

   getch();
}

int kare(int s)
{
 return s*s;
}

int kub(int s)
{
 return s*s*s;
}
```

Void İşaretçiler

İşaretçiler void olarak tanımlanabilir. Bu biçimde tanımlanan işaretçilerin gösterdiği adresteki değere erişmek için veri tipi belirtilmelidir.

```
main()
{
   void *a;

   a = (char*) malloc(10);
   strcpy(a,"12345");

   printf("%s\n",    a);
   free(a);

   a = (double*) malloc(sizeof(double));
   *(double*)a = 3.123;                    /* değere erişirken veri tipi belirt */
   printf("%f\n", *(double *)a);

   getch();
}
```

10. Daha Önce Geçmeyen Konular

Deyimler

continue: Döngünün herhangi bir satırında iken, bir sonraki döngü admını geçilmesini sağlar.

Örnek 10.1:

```
#include <stdio.h>
int main()
{
int i;

    for(i = 1 ; i < 10 ; i++)
    {
        if (i == 5)
            break;
        printf("%d\n", i);
    }

    for(i = 1 ; i < 10 ; i++)
    {
        if (i == 5)
            continue;
        printf("%d\n", i);
    }
}
```

gotoxy(x, y) : İmleci metin ekranında istenilen noktaya taşır.

void gotoxy(int x, int y)

gotoxy(25, 23);

printf("Bir tuşa basınız.");

wherex(), wherey() : Bulunulan yer

exit: Herhangi bir yerde programın çalışmasını sonlandırır.

void exit(int durum); : Dosyalar kapatılır, durum parametresinin değeri programın çağırıldığı yere geri gönderilir.

goto : Programın herhangi bir yerinden başka bir yere atlanmasını sağlar (aynı işlev içerisinde).

Gidilecek yerin tanımlanması gerekir. Kulanım şekli;

```
goto son            programın akışı son: etiketinin alduğu yerden devam eder
...
son :
return;
```

Örnek: Matriste sıfırdan küçük değerli ilk elemanın yerini bulan program.

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

```
main()
{
  int a[10][10];
  int i, j;

  randomize();

  for (i = 0; i<10; i++)
       for (j = 0; j<10; j++)
            a[i][j] = (rand() % 100) - 20;

  for (i = 0; i<10; i++)
       for (j = 0; j<10; j++)
            if (a[i][j] < 0)
                goto bulundu;

  printf("Sıfırdan küçük sayı yok\n");
  goto son;

bulundu:
  printf("Sayının konumu ve değeri (%d, %d) =   %d   \n",i, j, a[i][j]);

son:
  return 0;
}
```

? : İşleci

değişken = mantıksal ifade ? ifade1 : ifade2

mantıksal ifade doğru is değişkene ifade1, yanlış ise ifade2 değeri atanır.

```
a = (i == 5) ? 1 : 0;
```

Eğer i'nin değeri 5 ise a'ya 1, aksi halde 0 atanır. Aynı ifade açık olarak yazılırsa;

```
if (i ==5)
    a = 1;
else
    a = 0;
```

Örnek 10.2:

```
main()
{
  int a;
  int i, j;

  scanf("%d%d", &i, &j);

  a = (i > j) ? i : j;
```

```
  printf("%d\n", a);
}
```

Bileşik Atama İşleci

a = a + b işlemi a += b şeklinde de yazılabilir. = işleci ile birlikte bu şekilde kullanılabilen diğer işleçler;

* / % + - << >> & ^ |

Bu işleçler için
a = a işleç b ile
a işleç= b
aynı anlamı taşır.

Virgül İşleci

Şu ana kadar kullanılan biçim dışında,

Birden fazla deyimin aynı ifadede yürütülmesini sağlar. İfade tek bir sonuç verecek ise parantez içerisine alınmalıdır. İfadeler soldan sağa yürütülür ve son ifade parantez içi ifadenin değeri olur.

i = (5, 3, 8); i = 8

i = (j = 2; j = j + 5); i = 7

for döngüsünün bölümlerinde birden fazla deyim yürütmede kullanılır.

```
for (t = 0, i = 0; i<100; i++)
        t = t + i;
```

Geçici Veri Tipi Değiştirme

(type cast)

Bir değerin tipinin geçici olarak değiştirmek mümkündür. Kullanım şekli,.

(veri tipi) ifade;

ifade (değişken, sabit) tipi belirtilen tipe dönüştürülür.

Örnek 10.3:

```
#include <stdlib.h>
main()
```

```
{
  float a, b;
  int i, j;

  i = 500;
  j = 500;

  a = i * j;
  b = (float) i*j;

  printf("%f    %f\n", a, b);

}
```

-12144.000000   250000.000000

İfade İçerisinde Atama Deyimi Kullanma

```
#include <stdlib.h>
main()
{
  char c;
  while ((c=getch()) != 'S')
        printf("%c", c);
  printf("\n%c tuşuna bastınız\n", c);
getch();
}
```

((c=getch()) != 'S') ifadesinde ilk önce en içteki parantez c = getch() yürütülür. Daha sonra karşılaştırma işlemi yapılır.

++ / -- İşleçleri

İfade içerisinde değişkenin

ardında ise değişkenin değeri ifadede kullanılır ve değeri bir arttırılır

önünde ise değişkenin değeri değeri bir arttırılır ve yeni değeri ifadede kullanılır.

i = 2;

k = 5*i++; --> k = 10

k = 5*++i; --> k = 15

#define Önişlemcisi

define önişlemcisi ile işlev benzeri makrolar yazmalar mümkündür.

```
#define kub(x)   ((x)*(x)*(x))
```

```
main()
{
  int i;

  i = 5;
  printf("%d \n", kub(i));
}
```

Makro tanım satırında parametrelerin parantez içerisine alınması unutulmalı.

Makro kub(i+1) biçiminde çağırıldığında, i=2 için

((x)*(x)*(x)) --> (i+1)*(i+1)*(i+1) --> 27

(x * x * x) --> (i+1* i+ 1* i+1) --> 6

```
#define ustal(a, b)   (pow((a), (b)))
```

11. Grafik

Basit grafik işlemleri nasıl gerçekleştirilir (sorumlu değiller).

Normal çalışma anında text ekranda (25, 80) boyutlarında.
* void textbackground(int newcolor); : alt zemin rengi
* void textcolor(int newcolor); : yazı rengi
* void textattr(int newattr); 8 bitlik sayı. hem alt zemin hem de yazı rengi
verilir Bbbbffff (B = blink)

* void highvideo(void); Yazı parlaklığı
* void lowvideo(void);
* void normvideo(void);

Bu işlevler doğrudan ekrana yazan deyimlere yöneliktir (cprintf, cput)

```
#include <conio.h>
main()
{
  int i;
  textbackground(BLACK);

  clrscr();

  for (i=0; i<=15;i++) {
       textcolor(i);
       cputs("HÜSEYİN PEKTAŞ\r\n");
  }

  getch();
}
```

Grafik çizimi için bilgisayarın grafik modunda olması gerekir.

/*
* void far initgraph(int far *graphdriver, int far *graphmode, char far *pathtodriver);
Grafik sistemini verilen degerlere gore ayarlar. Grafik komutlarini kullanmadan once calistirilmalidir.

* closegraph; Grafik sistemini kapatir.
* graphresult: int; Grafik islemi ile ilgili bilgi verir.
* char *far grapherrormsg(int errorcode); Belirtilen grafik hatasi icin mesaji verir.

* line(x1, y1, x2, y2);
* lineto(x, y); Bulunulan yerden (x,y) ye kadar çizer
* linerel(x, y); Bulunulan yerin (x,y) ötesine kadar çizer
* moveto(x, y);

* void far cleardevice(void); Grafik ekranını siler

* circle(x, y , r);

* putpixel(x, y, renk);
* getpixel(x, y); Belirtilen noktanin piksel degerini verir

* getx; Bulunulan
* gety; noktalar

* getmaxx; Maksimum olabilecek
* getmaxy; noktalar

* outtext(char far *) : Bulunulan yere metni yazar
* outtextxy(x, y, char far *) : Belirtilen yere metni yazar

* int far getcolor(void); : çizim rengi öğren
* void far setcolor(int color); : çizim rengini değiştir

* int far getbkcolor(void); : alt zemin rengini öğren
- void far setbkcolor(int color);: alt zemin rengini değiştir

EGA_BLACK	0	EGA_DARKGRAY	56
	1	EGA_LIGHTBLUE	57
EGA_BLUE	2	EGA_LIGHTGREEN	58
EGA_GREEN	3	EGA_LIGHTCYAN	59
EGA_CYAN	4	EGA_LIGHTRED	60
EGA_RED	5	EGA_LIGHTMAGENTA	61
EGA_MAGENTA	7	EGA_YELLOW	62
EGA_LIGHTGRAY	20	EGA_WHITE	63
EGA_BROWN			

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

main()
{
   int gd, gm, hata;
   int x, y;

   gd = DETECT;   /* ekran sürücüsünü otomatik tanı */
                  /* CGA, HERC, VGA..*/

   initgraph(&gd, &gm, "c:\\diller\\tc3\\bgi\\");
   /* gd            gm
      VGA     ¦ VGALO     ¦  0  ¦  640 x 200 ¦ 16 color¦  2
         9       ¦ VGAMED ¦  1  ¦  640 x 350 ¦ 16 color¦  2
                 ¦ VGAHI     ¦  2  ¦  640 x 480 ¦ 16 color¦  1
   */
```

```
    hata = graphresult();

   if (hata != grOk)   /* grOk = 0 tanımlı */
   {
      printf("Grafik hatası: %s\n", grapherrormsg(hata));
      getch();
      exit(1);
   }

   x = getmaxx();
   y = getmaxy();

   setbkcolor(EGA_RED);
   setcolor(EGA_YELLOW);
   cleardevice();

   line(0, 0, 0, y);
   line(0, y, x, y);
   line(x, y, x, 0);
   line(x, 0, 0, 0);

   getch();

   cleardevice();

   moveto(50, 50);
   lineto(50, 100);
   lineto(100,100);
   lineto(100,50);
   lineto(50, 50);

   getch();

   closegraph();
}
```

Örnek 11.1: İşlev çizimi

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

double fonk(double x);
double fonk1(double x);

main()
{
   int gd, gm, hata;
   double x, y;
   int orgx, orgy;
```

```
    gd = DETECT;
    initgraph(&gd, &gm, "c:\\diller\\tc3\\bgi\\");
    hata = graphresult();
    if (hata != grOk)
    {
        printf("Grafik hatası: %s\n", grapherrormsg(hata));
        getch();
        exit(1);
    }

    orgx = 20;
    orgy = getmaxy() / 2;

    line(0, orgy, getmaxx(), orgy);     /* x ekseni */
    line(20, 0, 20, orgy+50);                /* y ekseni */

    outtextxy(getmaxx()-50, orgy-10, "Zaman");
    outtextxy(22, 0, "Volt");

    for (x = 0; x< 6.28; x = x + 0.005)   {
          y = fonk1(x);
          putpixel(orgx + 80*x, orgy - y, 2);   /* renk yeşil     EGA_GREEN*/
                                        /* - çünkü eksenin üzerinde çizsin */

    }

    getch();

    closegraph();
}

double fonk(double x)
{
  double y;
  y = 100 * sin(10 * x);
  return y;
}
double fonk1(double x)
{
  double y;
  y = 1 - exp(-x);
  return 200*y;
}
```

12. KÜTÜKLER

12.1 TEXT Kütükler

(sıralı erişim, ascii, metin)

Kütük(file): Bilgilerin diskte saklanış biçimi.

Text kütüğü :
- karakterlerden oluşur.
- her satırın sonunda satır sonunu gösterir özel karakter '\n' vardır.
- kütüğün sonunda kütük sonunu gösterir özel karakter 'eof' vardır.

Örnek:
Birinci satır<\n>
Bu da ikincisi<\n><eof>
Diskte satırlar peşpeşedir. İlk satırdaki <\n> karakterinden hemen sonra ikinci satırın ilk karakteri gelir.

Kütüklerin Kullanımı

Bir kütüğü kullanmadan önce iki işlemim yapılması germektedir. Kütük ile ilgili işaretçinin tanımlanması ve kütüğün açılması.

Tanımlı FILE tipinde işaretçi değişken tanımlayarak ilk işlem gerçekleştirilir.
```
FILE *kp;              /*FILE yapısı stdio.h içerisinde tanımlı */
```

İkinci aşamada kütük açma deyimi kullanılır.
```
kp = fopen(kütük adı, mod)
```

kütük adı : açmak istediğimiz kütüğün adı (string). Büyük - küçük harf ayrımı yok.
mod : kullanım amacı (string). Okuma, yazma değişiklik ...

r	: okumak için
w	: yazmak için. Kütük yok ise oluşturulur, var ise içeriği silinir.
a	: eklemek için. Kütük yok ise oluşturulur , var ise sondan itibaren yazılır
r+	: Var olan bir kütüğü okumak ve yazmak için
w+	: Okumak ve yazmak için yeni bir kütük oluştur. Dosya var ise içeriği silinir.
a+	: Eklemek için. Okuma ve yazma yapılabilir. . Dosya yok ise oluşturulur..

Örnek 12.1.1 : Kütüğe yazma
```
#include <stdio.h>
main()
{
  FILE *kp;
  char *s;
  int i;

  kp = fopen("dene", "w"); /* NULL ise işlem başarısız */
  if (kp == NULL)
     exit(1);
```

```
  for (i=0; i<5; i++) {
        gets(s);
        fprintf(kp, "%s\n", s);
  }
  getch();
  fclose(kp);   /* kütüğü kapatır. Program sona erdiğinde otomatik kapatılır */
    /* fcloseall() hepsini kapatır. Dönen , kapanan kütük sayısı.   Hata var ise EOF döner
*/
}
```

Yazarken kullanılan fprintf deyiminin çalışması printf gibidir. Sadece ilk parametre olarak kütük değişkenini vermek gerekir.

Benzer kütük işlevleri

```
int fgetc(FILE *stream);
int fputc(int c, FILE *stream);
char *fgets(char *s, int n, FILE *stream);      n-1 karakter veya satır sonuna kadar okur
int fputs(const char *s, FILE *stream);             satır sonu bilgisini ayrıca yazmak gerekir
```

Örnek 12.1.2: Kütükten okuma

```
#include <stdio.h>
main()
{
  FILE *kp;
  char *s;
  int i;

  kp = fopen("dene", "r");
  if (kp == NULL)
     exit(1);

  clrscr();
  for (i=0; i<5; i++) {
        fscanf(kp, "%s", s);
        printf("%s\n", s);
  }

  /* Kütükteki kayıt sayısı bilimiyor ise */
  rewind(kp);
  /*     fseek(stream, 0L, SEEK_SET)
                              SEEK_SET 0 baştan
                              SEEK_CUR 1 bulunulan yerden
                              SEEK_END 2 sondan
  */
  fscanf(kp, "%s", s);
  while (!feof(kp)) {
        printf("%s\n", s);
        fscanf(kp, "%s", s);
  }
  getch();
```

```
}
```

int feof(FILE *) : kütüğün sonuna erişilmiş ise sıfırdan farklı bir değer alır. Aksi halde sıfırdır.
rewind(FILE *) : kütük işaretçisinin (kütükteki hangı kayıdın okunacağı bilgisi) kütüğün başını göstermesini sağlar.

Örnek 12.1.3: Ekleme ve okuma

a+ modunda kütük aç. + eklenmez ise kütük işaretçisi kütük başına taşınmıyor

```
#include <stdio.h>
main()
{
  FILE *kp;
  char *s;
  int i;

  kp = fopen("dene", "a+");
  if (kp == NULL)
     exit(1);

  clrscr();
  for (i=0; i<2; i++) {
       gets(s);
       fprintf(kp, "%s\n", s);
  }
  rewind(kp);
  fscanf(kp, "%s", s);
  while (!feof(kp)) {
       printf("%s\n", s);
       fscanf(kp, "%s", s);
       getch();
  }
  getch();
}
```

Örnek 12.1.4: Matrisin elemanlarının değerlerinin kütükten öğrenilmesi.
kütüğün ismi matrisa.dat
matris elemanları double
boyut 4x3

```
#define SAT 4
#define SUT 3
#include <stdio.h>
main()
{
  FILE *kp;
  double a[SAT][SUT];
  int i, j;

  kp = fopen("matrisa.dat", "r");
  if (kp == NULL)
```

```
      exit(1);

  for (i=0; i<SAT; i++)
       for (j=0; j<SUT; j++)
            fscanf(kp, "%lf", &a[i][j]);

  clrscr();

  for (i=0; i<SAT; i++) {
       for (j=0; j<SUT; j++)
            printf("%8.1f", a[i][j]);
       printf("\n");
  }

  getch();
}
```

Örnek 12.1.5: Hesaplanan değerlerin kütüğe yazdırılması.
1-exp(-x) fonksiyonu 0-10 aralığında 0.02 adımlarla hesaplanıp kütüğe yazılıyor

```
#include <stdio.h>
#include <math.h>

#define ADIM 0.02
#define T0    0
#define TS    10

main()
{
  FILE *kp;
  double f, t;

  kp = fopen("sonuc.dat", "w");
  if (kp == NULL)
     exit(1);
  for (t=T0; t<TS; t = t + ADIM) {
       f = 1 - exp(-t);
       fprintf(kp, "%6.3f       %12.8f\n", t, f);
       printf("%6.3f       %12.8f\n",t, f);
  }
  getch();
}
```

12.1.1 Text Kütükte Değişiklik

Kütük işaretçisi değişiklik yapılacak kaydı göstermelidir. Ayrıca yeni bilgi eski bilgi ile aynı olmalıdır. Bu nedenle text kütüklerde değişiklik başvural bir şey değil. Gerektiği durumlarda kayıtlar geçici olarak başka bir kütüğe yazılır.

int remove(const char *k_adi) : kütüğü siler
int rename(const char *eski, const char *yeni) : kütüğün ismi değiştirilir

Örnek 12.1.6: Girilen dosyada herbir harfin kaçdefa geçtiğini bulup, çoktan aza doğru yazar.

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

void degistir(int *, int *);

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

  char dosyaadi[64];
  FILE *fp;
  char c;

  long toplam = 0;
  int frek[2][26] = {0,};
  int i, j, k;

  for (i=0; i<26; i++)
       frek[0][i] = 65 + i;

  if (argc<2) {
      printf("Dosya adını belitmediniz\n");
      exit(1);
  }
  else
      strcpy(dosyaadi, argv[1]);

  fp = fopen(dosyaadi, "r");

  while (!feof(fp)) {
      c = toupper(fgetc(fp));
      if (c >= 'A' & c <='Z' )
            frek[1][c - 'A']++;
  }

  for (i = 0; i < 26; i++)
        toplam = toplam + frek[1][i];

  clrscr();
  printf("İncelenen Dosya : %s\n", dosyaadi);
  printf("Toplam harf        : %ld", toplam);

  /* Çoktan aza doğru sırala */
  do {
      k = 0; /* bayrak */
      for (i=0; i < 25; i++)
            if (frek[1][i] < frek[1][i+1] ) {
                k = 1;
                degistir(&frek[0][i], &frek[0][i+1]);
```

```
                degistir(&frek[1][i], &frek[1][i+1]);
            }
    } while (k == 1);

    for (i = 0; i < 26; i++) {
        if (i<13)
            gotoxy(1, 5+ i);
        else
            gotoxy(40, 5 + i - 13);

        printf("%c:   %6d   %5.3f",frek[0][i] , frek[1][i], (float)frek[1][i]/toplam);
   }

    getch();
}

void degistir(int *a, int *b)
{
    int c;
    c   = *a;
    *a = *b;
    *b = c;
}
```

Örnek çıktı**:** (frekans classics.txt)

İncelenen Dosya : classics.txt
Toplam harf : 122850

E:	14657	0.119	M:	2870	0.023
T:	10449	0.085	F:	2630	0.021
A:	9902	0.081	P:	2440	0.020
I:	9889	0.080	Y:	2292	0.019
S:	9504	0.077	W:	2257	0.018
O:	9492	0.077	G:	2172	0.018
N:	8521	0.069	B:	1779	0.014
R:	7706	0.063	V:	1279	0.010
H:	6215	0.051	K:	1260	0.010
L:	5333	0.043	Z:	288	0.002
D:	4463	0.036	J:	236	0.002
C:	3760	0.031	X:	218	0.002
U:	3121	0.025	Q:	117	0.001

12.2 Rastgele Erişimli Kütükler

- Veriler sabit uzunluktaki bloklar (kayıt) şeklinde saklanır.
- Veriler karekterler dizisi biçiminde saklı.
- Değişkenlerin içeriği bellekte saklanış biçimiyle kütüğe yazıldığı için okuma/yazma işlemleri text kütüklere göre daha hızlı.

- Aynı sebepten dolayı veriler kütükte daha az yer kaplar.
- Gerçel sayılarda hassasiyet kayıbı yok.
- İstenilen kayıda doğrudan erişmek mümkün.

Bilgiler sabit uzunlukta saklandığı için değişiklik anında kayıt bozulması yok
İstenilen bilgiye erişmesi kolay.

Kütüklerin Bu Biçimde Kullanılması İçin

fwrite ve fread deyimleri kullanılır

fwrite(veri_ptr, uzunluk, kac_tane, kp) : Belirtilen sayıda sabit uzunluktaki veriyi kütüğe yazar

veri_ptr : Yazılacak veriye işaretçi
uzunluk : Yazılacak verinin uzunluğu
kac_tane : Verinin kaç defa ard arda yazılacağı
kp : Kütük işaretçisi

Kütüğe uzunluk * kac_tane kadar byte yazılır. Dönüş değeri
yazılan veri sayısı (byte değil)

fead(veri_ptr, uzunluk, kac_tane, kp) : Belirtilen sayıda sabit uzunluktaki veri kütükten okunur
Örnek 12.2.1: Rastgele erişimli kütük

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

typedef struct {
     char no[11];
     int notu;
} ogryapi;

FILE *kp;
ogryapi ogr;

void giris(void);
void liste(void);

main()
{
  int i;
  char c;

  kp = fopen("ogr.dat", "r+");

  if (kp == NULL) {
      printf("%d\n", errno);   /* 2 ise file not found */
      kp = fopen("ogr.dat", "w+");
  }
  do {
```

```
        clrscr();
        fseek(kp, 0, 2);
        printf("Dosya uzunluğu   %ld\n", ftell(kp));
        printf("Kayıt sayısı      %d\n\n", ftell(kp)   / sizeof(ogryapi));

        printf("1-Giriş\n2-Liste\n3-Çıkış\n");
        c = getch();
        switch (c) {
             case '1': giris(); break;
             case '2': liste(); break;
             case '3': exit(0);
        };
    } while (1);

void giris(void)
{
    int i;

    clrscr();

    printf("Öğrenci no : "); scanf("%s", ogr.no);
    printf("          Notu : "); scanf("%d", &ogr.notu);

    i = atoi(ogr.no);

    fseek(kp, (i-1)*sizeof(ogryapi), 0);
    fwrite(&ogr, sizeof(ogryapi), 1, kp);

    fflush(kp);
    /*int fflush(FILE *) tampon bllektekiler diske yazılır
      int flushall(void)
    */

}

void liste(void)
{
    clrscr();
    fseek(kp, 0, 0);
    fread(&ogr, sizeof(ogryapi), 1, kp);
    while (!feof(kp)) {
         printf("%s      %d\n", ogr.no,ogr.notu);
         fread(&ogr, sizeof(ogryapi), 1, kp);
    }
    getch();
}
```

Örnek 12.2.2 : Dizinin kütüğe yazılması, okunması

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

#define MAX 10
```

```
FILE *kp;
int a[MAX];

void giris(void);
void liste(void);

main()
{
Önceki örnek gibi
}


void giris(void)
{
  int i;

  clrscr();
  for (i=0; i<MAX; i++)
       a[i] = rand() % 100;

  fseek(kp, 0, 0);
  fwrite(a, sizeof(int), MAX, kp);
}


void liste(void)
{
  int i;
  clrscr();

  fseek(kp, 0, 0);
  fread(a, sizeof(int), MAX, kp);

  for (i=0; i<MAX; i++)
       printf("%d   ", a[i]);

  printf("\n");
  getch();
}
```

EK A. Örnekler

Örnek A.1: Paralel portun kullanımı

```
/* PC deki paralel porta erişim
oku       : paralel portu okur. Eski tip portlar çift yönlü olmadığı için
            durum uçları giriş amaçlı kullanıldı
            ( 4-bit , bi-directiona, EPP, ECP )

yaz       : paralel port data uçlarına bilgi gönderir

oku_yaz : durum uçlarından okuduğunu veri uçlarına gönderir

*/

#include <stdio.h>
#include <dos.h>
#include <conio.h>

typedef unsigned word;

void binary_yaz(word);
void oku(void);
void yaz(void);
void oku_yaz(void);

word pdata=0x378;
word pstat=0x379;
word bout, bin;
char c;

main()
{

  do {
      clrscr();
      printf("1 - Oku\n");
      printf("2 - Yaz\n");
      printf("3 - Okuduğunu Yaz\n");
      printf("4 - Çık\n");

      c = getch();

      switch(c) {
            case '1' : oku(); break;
            case '2' : yaz(); break;
            case '3' : oku_yaz(); break;
            case '4' : exit(0);
      }
  } while (1);
```

```
}

void yaz(void)
{
  clrscr();
  printf("Gönderilecek veri :");
  scanf("%d", &bout);
  binary_yaz(bout); printf("\n");
  outport(pdata, bout);
  c = getch();
}
void oku(void)
{
  clrscr();
  do {
      bin = inportb(pstat);
      binary_yaz(bin);
      printf("     %d\n", bin);
      c = getch();
  } while (c != 'q');
}

void oku_yaz(void)
{
  clrscr();
  do {
      bin = inportb(pstat);
      gotoxy(1,8) ; printf("Okunan    = ");
      binary_yaz(bin);

      outport(pdata, bin);
      gotoxy(1,9) ; printf("Yazılan = ");
      binary_yaz(bin);

      printf("     %d\n", bin);
      c = getch();
  } while (c != 'q');
}

void binary_yaz(word x)
{
  word i;
  word m;


  m = 0x8000;
  for (i = 0; i<16; i++) {
       if (x & m)
           printf("1");
       else
           printf("0");
       m = m >> 1;
  }
  printf(" ");
```

}

Örnek A.2. Kesme (interrupt) kullanımı

Intel 8086 Registers

General Purpose Registers	Segment Registers
AH/AL AX (EAX) Accumulator	CS Code Segment
BH/BL BX (EBX) Base	DS Data Segment
CH/CL CX (ECX) Counter	SS Stack Segment
DH/DL DX (EDX) Data	ES Extra Segment
	(FS) 386 and newer
(Exx) 386+ 32 bit register	(GS) 386 and newer

Pointer Registers	Stack Registers
SI (ESI) Source Index	SP (ESP) Stack Pointer
DI (EDI) Destination Index	BP (EBP) Base Pointer
IP Instruction Pointer	

FLAGS - Intel 8086 Family Flags Register

0	CF Carry Flag	9	IF Interrupt Flag
1	1	A	DF Direction Flag
2	PF Parity Flag	B	OF Overflow flag
3	0	C,D	IOPL I/O Privilege Level (286+ only)
4	AF Auxiliary Flag	E	NT Nested Task Flag (286+ only)
5	0	F	0
6	ZF Zero Flag	10	RF Resume Flag (386+ only)
7	SF Sign Flag	11	VM Virtual Mode Flag (386+ only)
8	TF Trap Flag (Single Step)		

```
#include <dos.h>
#include <stdio.h>
main()
 {
    union REGS a;

    clrscr();
    a.h.ah = 0x2A;          /* Dos servislerinden 2Ah = get system date */
    int86(0x21,&a,&a); /* int 21h dos servisleri */

    printf("makinanin tarihi = %d/%d/%d\n", a.h.dl, a.h.dh, a.x.cx);

    a.h.ah = 0x36;          /* Diskteki bos alan */
```

```
    a.h.dh = 0;             /* bulunulan, A=1 */
    int86(0x21,&a,&a);

    printf("makinanadaki yer = %ld\n", (long)a.x.ax*a.x.bx*a.x.cx);

    getch();
}
```

ÖRNEKLER

Örnek : 1996-97 2.dönem vize

```
/*Soru 1 */
#include <stdio.h>
#include <string.h>

char *kelime(char *);

main()
{
  char *cumle = "ali zil caldi mi";
  char *s, *kelimeler[20];
  int i, kk = 0;
  clrscr();

  /* yol 1:
  s = kelime(cumle);
  while (strlen(s)) {
          strcpy(kelimeler[kk++], s);
          s = kelime(cumle);
   }
  */

  /* yol 2: strtok islevini kullanarak
  s = strtok(cumle, " ");
  while (s !=NULL) {
          strcpy(kelimeler[kk++], s);
          s = strtok(NULL, " ");
  }
  */

  for (i=kk-1; i>=0; i--)
       printf("%s ", kelimeler[i]);

  getch();
}

char *kelime(char *s)
{
 char a[20], *b, d[20];
```

```
 int i;
 while (s[0] == ' ')
     s++;


 strcpy(d, s);

 b = strchr(s, ' ');
 if (b == NULL)
     strcpy(a, s);
 else
     {
       i = abs(b-s);
       strncpy(a, d, i);
       a[i] = '\0';
     }
 strcpy(s, b);
 return a;
}


/* Soru 3
struct veriler {
   double ka, ya, yg;
};
struct veriler a[60000];


for (i=0; i<60000; i++)
      if (a[i].ka == a[i].ya)
          printf("%f        %f\n",i*20, a[i].ka);
Soru 3*/
```

EK B. DİNAMİK VERİ YAPILARI

Self-referential structure : üye, tanımlı olduğu yapıya bir işaretçi

```
struct yapi {
     char    adi[21];
     struct yapi *ptr;
};
```

B.1 Sıralı Bağlı Liste

.....

Örnek B.1: Sıralı bağlı listenin gerçekleştirilmesi.

```
#include <stdlib.h>
#include <string.h>
struct listyapi {
     char    adi[21];
     struct listyapi *sonraki;
};
typedef struct listyapi listnode;
typedef listnode *listptr;
listptr headnode;    /* Herzaman listenin baţýný gösterir */

void seeklist(char *searchthis, listptr *prevnode)
{
  listptr c;

  c = headnode;
  *prevnode = c;
  while ((c->sonraki != NULL)) {
        c = c->sonraki;
        if (strcmp(c->adi, searchthis) >= 0) break;
        *prevnode = c;
  }
}

void kayit(char *s)
/* prevnode kayıdı newnode kayıdını, newnode kayıdı prevnode'nin daha önce gösterdiği kayıdını gösterir. */
{
    listptr newnode, prevnode;

    newnode = malloc(sizeof(listnode)); /* yeni kayıda yer a‡ */

    strcpy(newnode->adi, s);                /*bilgiyi yeni kayıda yaz */

    seeklist(newnode->adi, &prevnode);   /* listedeki yerini bul */
```

```
    newnode->sonraki = prevnode->sonraki;   /* listeye ekle */
    prevnode->sonraki = newnode;
}

void iptal(char *s)
/* newnode kayıdı silinir. prevnode kayıdı newnode kayıdının gösterdiği kayıdını gösterir. */
{
    listptr newnode, prevnode;

    seeklist(s, &prevnode);
    newnode = prevnode->sonraki;
    prevnode->sonraki = newnode->sonraki;
    free(newnode);
}

void listlist(void)
{
    listptr currentnode;

    currentnode = headnode;
    if (currentnode != NULL) currentnode = currentnode->sonraki;
    while (currentnode != NULL)
        {
            printf("%s   ",currentnode->adi);
            currentnode = currentnode->sonraki;
        }
    printf("\n");

}    /* Procedure listlist; */

main()
{
  char        sec;
  char        *s;

  headnode = malloc(sizeof(*headnode));
  strcpy(headnode->adi," listenin baţý");
  headnode->sonraki = NULL;

  do {
    clrscr();
    printf("Boş yer : %ld\n\n",coreleft());

    listlist();

    printf("\n\n1 - Giriş\n2 - iptal\n3 - Son\n\nSeç \n");

    sec = getch();
    switch (sec) {
      case '1':
                  printf("Adı "); gets(s);
                  kayit(s);
                  break;
```

```
        case '2': printf("Adı \n"); gets(s);
                  iptal(s);
                  break;

        case '3': exit(0);break;
      }
   }  while (1);
}
```

Problem : Polinomlar sıralı bağlı listede saklı. İki polinomun toplamını üçücnü polinomda oluşturan program

B.2 Listelerin Özel Biçimleri (Uygulamaları)

B.2.1 Stack (yığın) : Son giren ilk çıkar (LIFO)
(Fonksiyon çağırıldığında geri dönüş adresi, Fonksiyona gönderilecek parametreler)

Örnek B.2.1: Stack kullanarak girilen cümleyi kelimeleri bozmadan tersten yazdır

```
#include <stdio.h>
#include <stdlib.h>
struct stackyapi {
     char deger[50];
     struct stackyapi *sonraki;
};
typedef struct stackyapi stacknode;
typedef stacknode *stackptr;
stackptr top;

void push(char *gelen)
{
  stackptr newnode;
  newnode = malloc(sizeof(stacknode));
  strcpy(newnode->deger, gelen);
  newnode->sonraki = top;
  top = newnode;
}

int pop(char *giden)
{
   stackptr oldnode;

   if (top != NULL) {
       oldnode = top;
       strcpy(giden, top->deger);
       top = top->sonraki;
       free(oldnode);
       return 0;
   }
   else
       return 1; /* yığın boş */
}
```

```
main()
{
  char *kelime, *cumle;
  stackptr current;

  top = NULL;
  clrscr();
  printf("Cumleyi girin : "); gets(cumle);
  kelime = strtok(cumle, " ");
  while (kelime) {
      push (kelime);
      kelime = strtok(NULL, " ");
  }
  kelime = (char *) malloc(20); /* kelime NULL idi, yer açalım*/
  while (!pop(kelime))
       printf("%s ", kelime);

  printf("\n");
  getch();
}
```

Problem :

infix : operand operator operand 2 + 7
postfix : operand operand operator 2 7 +
Soldan sağa işlem yapılır. Öncelik (parantez) yok.

postfix biçimindeki ifadeyi hesaplama ?
infix → postfix dönüşümü ?

B.2.1: Kuyruk (queue) ilk giren ilk çıkar (FIFO)

Bilgisayarda çoklu işlem
Yazıcıyı ortak kullanım

B.2.3: Ağaç (tree) Doğrusal olmayan yapı
İkili ağaç (kök, çocuk, yaprak, derece)

MEMORY MODELLERİ

Tiny : Code + data + stack aynı 64 K lık segment te
Small : Code için 64 K, data + stack ayrı 64 K lık segment
Medium : Code için far pointer (yani 1 MB code) , data 64 K (data kullanmayan uzun programlar)
Compact: Medium un tersi. (çok data kullanan kısa programlar)
Large : Code için 1 MB, data için 1 MB
Huge : Large ile aynı fakat static data 64 K dan fazla olabilir.