Ders 3: *Operatörler*

Giriş

Operatörler, değişkenler veya sabitler üzerinde matematiksel ve karşılaştırma işlemlerini yapan simgelerdir. Yani bir operatör bir veya daha fazla nesne (değişken) üzerinde işlem yapan sembollerdir. Bu kısımdam aritmetik operatörler, atama operatörleri ve `sizeof` operatörü anlatıcaktır. Karşılaştırma Operatörleri, Mantıksal Operatörler ve Bit Düzeyinde işlem yapan operatörler daha sonraki bölümlerde incelenektir.

3.1 Aritmetik Operatörler

Değişken veya sabitler üzerinde temel aritmetik işlemleri gerçekleyen operatörlerdir. Bunlar Tablo 3.1'de listelenmiştir.

Tablo 3.1: *Aritmetik Operatörler*

Operatör	Açıklama	Örnek	Anlamı
+	toplama	`x + y`	x ve y nin toplamı
-	çıkarma	`x - y`	x ve y nin farkı
*	carpma	`x * y`	x ve y nin çarpımı
/	bölme	`x / y`	x ve y nin oranı
%	artık bölme	`x % y`	`x / y` den kalan sayı

3.2 Atama Operatörleri

Bu operatörler bir değişkene, bir sabit vaya bir aritmetik ifade atamak (eşitlemek) için kullanılır.
Birleşik atama: bazı ifadelerde işlem operatörü ile atama operatörü birlikte kullanılarak, ifadeler daha kısa yazılabilir. Eğer ifade

```
değişken = değişken [operatör] aritmetik ifade;
```

şeklinde ise, daha kısa bir biçimde

```
değişken [operatör]= aritmetik ifade;
```

olarak yazılabilir. Bu operatörler Tablo 3.2'de listelenmiştir.

Tablo 3.2: *Atama Operatörleri*

Operatör	Açıklama	Örnek	Anlamı
=	atama	x = 7;	x = 7;
+=	ekleyerek atama	x += 3	x = x + 3
-=	eksilterek atama	x -= 5	x = x - 5
*=	çarparak atama	x *= 4	x = x * 4
/=	bölerek atama	x /= 2	x = x / 2
%=	bölüp, kalanını atama	x %= 9	x = x % 9
++	bir arttırma	x++ veya ++x	x = x + 1
--	bir azaltma	x-- veya --x	x = x - 1

Bu tanımlamalara göre, aşağıdaki atamaları inceleyiniz:

```
/* bir arttırma işlemleri */
i++;
++i;
i += 1;
i = i + 1;


/* karmaşık atamalar */
f *= i;       // f = f * i; anlamında
f *= i+1;     // f = f * (i+1); anlamında
z /= 1 + x;   // z = z / (1+x); anlamında
```

Bir arttırma veya eksiltme operatörlerini kullanırken dikkatli olunmalıdır. Çünkü aşağıdaki türden atamalar bazen karışıklığa neden olur.

```
a = 5;     // a = 5
b = a++;   // a = 6 ve b = 5
c = ++a;   // a = 7 ve c = 7
```

Program 3.1: *Aritmetik ve atama operatörlerinin kullanımı*

```
01: /* 03prg01.c: Aritmetik ve atama
02: operatorlerinin kullanimi */
03:
04: #include <stdio.h>
05:
06:
07: main()
08: {
09:     int x, y;  /* yerel degikenlerin
10: bildirimi */
11:
12:     x = 1;    /*  x in baslangic degeri
13: */
14:     y = 3;    /*  y nin baslangic degeri
15: */
16:
```

```
17:     printf(" x = %d ve y = %d, olarak
18: veriliyor.\n", x, y);
19:
20:     x = x + y;
21:     printf("x <- x + y  atamsinin
22: sonucunda x=%d dir\n", x);

        x = 1;  /* x e tekrar 1 degeri
    ataniyor */
        x += y;
        printf("x += y  atamasinin sonucunda
    x=%d dir\n", x);

     return 0;
    }
```

ÇIKTI

```
x = 1 ve y = 3, olarak veriliyor.
x <- x + y  atamasinin sonucunda x=4 dir
x += y  atamasinin sonucunda x=4 dir
```

3.3 `sizeof` Operatörü

Veri tiplerinin, değişkenlerin ve dizilerin bellekte kapladığı alan `sizeof` operatörü ile öğrenilebilir. Genel kullanımı:

```
sizeof(nesne)
```

şeklindedir. Program 3.2'de bu operatörün nasıl kullanıldığı gösterilmiştir. Ayrıca bkz: Program 2.1 ve Bölüm 10, Bölüm 12.

Program 3.2: *sizeof operatörün kullanımı*

```
01: /* 03prg02.c
02:    sizeof operatörünün değişik
03: nesnelerle kullanımı */
04:
05: #include <stdio.h>
06:
07: int main(){
08:
09:    int    i;              /* bir tamsayı
10: */
11:    int    dizi[5];        /* 5 elemanlı
12: bir tamsayı dizi */
13:
14:    double d;              /* bir gercel
15: sayı */
```

```
16:    double mizan[6];        /* 6 elemanlı
17: bir gercel dizi */
18:
19:    char   c;               /* tek bir
20: karakter */
21:    char   str[] = "masa"; /* bir
22: karakter topluluğu */
23:
24:
25:    printf("sizeof(int)    =
26: %d\n",sizeof(int));
27:    printf("sizeof(i)      =
28: %d\n",sizeof(i));
29:    printf("sizeof(dizi)   =
30: %d\n\n",sizeof(dizi));
31:
32:    printf("sizeof(double)=
    %d\n",sizeof(double));
       printf("sizeof(d)      =
    %d\n",sizeof(d));
       printf("sizeof(mizan)  =
    %d\n\n",sizeof(mizan));

       printf("sizeof(char)   =
    %d\n",sizeof(char));
       printf("sizeof(c)      =
    %d\n",sizeof(c));
       printf("sizeof(str)    =
    %d\n",sizeof(str));

    return 0;
    }
```

ÇIKTI

```
sizeof(int)    = 4
sizeof(i)      = 4
sizeof(dizi)   = 20

sizeof(double)= 8
sizeof(d)      = 8
sizeof(mizan)  = 48

sizeof(char)   = 1
sizeof(c)      = 1
sizeof(str)    = 5
```

Programda `sizeof(int)` değeri ile `sizeof(i)` değerinin aynı olduğu görülür. `dizi`nin boyutu 5 olduğu için, `sizeof(dizi) = sizeof(int)*5 = 20` şeklinde

hesaplanmaktadır. Diğerleri için benzer durum söz konusu. Ancak, `str` 4 elemanlı bir dizi olduğu halde `sizeof(str) = 5` dir. Neden? Bunu ilerideki bölümlerde öğreneceğiz.

Ders 4: *Temel Giriş/Çıkış Fonksiyonları*

Giriş

Temel giriş/çıkış fonksiyonları, bütün programla dillerinde mevcuttur. Bu tür fonksiyonlar, kullanıcıya ekrana veya yazıcıya bilgi yazdırmasına, ve bilgisayara klavyeden veri girişi yapmasına izin verir. Temel giriş/çıkış fonksiyonları kullanılırken `stdio.h` başlık dosyası programın başına eklenmelidir. Bu kısımda, en çok kullanılan giriş/çıkış fonksiyonları anlatılacaktır.

4.1 `printf()` Fonksiyonu

Standart C kütüphanesinde bulunan `printf()` fonksiyonu, değişkenlerin tuttuğu değerleri, onların adreslerini veya bir mesajı ekrana belli bir düzenle (format) standart çıkışa (stdout), yani ekrana, yazdırmak için kullanılan fonksiyondur. Daha önce yazılan örnek programlarda `printf()` fonksiyonundan yararlanmıştık. Şimdi bu fonksiyonun nasıl kullanıldığına bakalım.

Genel yazım biçimi:

```
int printf(const char *format, ...);
```

Basit olarak ekrana `Hata oluştu!..` şeklinde bir mesaj yazırma işlemi:

```
printf("Hata Oluştu!..");
```

şeklindedir. Çoğu zaman ekrana, programda kullanılan bir değişkenin değeri yazdırılmak istenebilir. Örneğin ekrana bir tamsayı değişkeninin içeriğini basırımak için, `printf()`

```
....
int x = 12;
printf("x in değeri %d dir", x);
....
```

gibi kullanılır. Bu program parçasının ekran çıktısı şöyle olacaktır:

```
x in değeri 12 dir
```

Bu örnekte `printf` fonksiyonuna iki parametre aktarılmıştır. Birincisi ekranda gösterilecek ve çift tırnaklar arasına yazılan ifadeler, ikincisi ise ekranda sayısal değeri gösterilmek istenen değişken (`x`).

*`format` üç kısımdan oluşmaktadır:

I. **Düz metin (literal string)**: yazdırılmak istenen ileti.
Örneğin: `printf("Ben gelmedim kavga için...");` gibi.

II. **Konrol karakterleri (escape squence)**: değişkenlerin ve sabitlerin nasıl yazılacağını belirtmek veya imlecin alt satıra geçirilmesi gibi bazı işlemlerin gerçekleştirilmesi için kullanılır. Bu karakterler Tablo 4.1'de listelenmiştir.
Örneğin: `printf("\tDostun evi gönlüdür...\n");` gibi.

Tablo 4.1: *Kontrol karakterleri*

Karakter	Anlamı
`\a`	Ses üretir (alert)
`\b`	imleci bir sola kaydır (backspace)
`\f`	Sayfa atla. Bir sonraki sayfanın başına geç (formfeed)
`\n`	Bir alt satıra geç (newline)
`\r`	Satır başı yap (carriage return)
`\t`	Yatay TAB (horizontal TAB)
`\v`	Dikey TAB (vertical TAB)
`\"`	Çift tırnak karakterini ekrana yaz
`\'`	Tek tırnak karakterini ekrana yaz
`\\`	\ karakterini ekrana yaz
`%%`	% karakterini ekrana yaz

III. **Tip belirleyici (conversion specifier)**: % işareti ile başlar ve bir veya iki karakterden oluşur (%d gibi). Ekrana yazdırılmak istenen değişkenin tipi, % işaretinden sonra belirtilir (Bkz. Tablo 4.2) Örneğin: `printf("x in değeri %d dir");` gibi.

Tablo 4.2: *Tip karakterleri*

Tip Karakteri	Anlamı	Yazdırılacak veri tipi
`%c`	tek bir karakter	`char`
`%s`	karakter dizisi (string)	`char`
`%d`	işaretli ondalık tamsayı	`int, short`
`%ld`	uzun işaretli ondalık tamsayı	`long`
`%u`	işaretsiz ondalık tamsayı	`unsigned int, unsigned sho`
`%lu`	işaretsiz uzun tamsayı	`unsigned long`
`%f`	Gerçel sayı	`float`
`%lf`	Çift duayarlı gerçel sayı	`double`

Tip karakterlerini kullanarak, birden çok veri tipi yazdırılabilir. Örneğin:

```
        ...
        int     not= 12;
        float   pi = 3.14;
        char    kr = 'A';

        printf(" not = %d , pi = %f ve kr = %c dir", not, pi,
kr);
        ...
```

gibi.

`printf()` fonksiyonu esnektir. Parametreler herhangi bir C deyimi olabilir. Örneğin `x` ve `y` nin toplamı şöyle yazılabilir:

```
        printf("%d", x+y);
```

`printf` fonksiyonu kullanımı Program 4.1'de verilmiştir.

Program 4.1: *printf() fonksiyonunun kullanımı*

```
01: /* 04prg01.c
02:    Sayısal değerleri ekrana yazdırmak
03: için printf fonksiyonunun kullanımı */
04:
05: #include <stdio.h>
06:
07: main()
08: {
09:
10:    int   a = 2,    b = 10,   c = 50;
11:    float f = 1.05, g = 25.5, h = -0.1,
12: yuzde;
13:
14:     printf("3 tamsayi       : %d %d
15: %d\n", a, b, c);
16:     printf("3 tamsayi [TAB] : %d \t%d
17: \t%d\n", a, b, c);
18:
19:     printf("\n");
20:
21:     printf("3 reel sayi (yanyana) : %f
22: %f %f\n", f, g, h);
23:     printf("3 reel sayi (altalta) :
24: \n%f\n%f\n%f\n\n", f, g, h);
25:
26:     yuzde = 220 * 25/100.0;
27:     printf("220 nin %%25 i %f dir\n",
    yuzde);
        printf("%f/%f isleminin sonucu =
    %f\n", g, f, g / f);
```

```
        printf("\nprogram sonunda beep sesi
    cikar...\a");

        return 0;
    }
```

ÇIKTI

```
3 tamsayi       : 2 10 50
3 tamsayi [TAB] : 2     10      50

3 reel sayi (yanyana) : 1.050000 25.500000
-0.100000
3 reel sayi (altalta) :
1.050000
25.500000
-0.100000

220 nin %25 i 55.000000 dir
25.500000/1.050000 isleminin sonucu =
24.285715

program sonunda beep sesi cikar...
```

`printf` fonksiyonunun geri dönüş değeri `int` tipindedir. Bu geri dönüş değeri çıktının kaç karakter olduğunu gösterir. Yani, `printf` fonksiyonu, *`*format`* ile tanımlanmış karakter topluluğunun kaç bayt olduğu hesaplar[6]. Program 4.2, `printf`'in bu yönünüde ortaya çıkaran bir programdır.

Program 4.2: *printf() fonksiyonunun kullanımı*

```
01: /* 04prg02.c
02:    printf fonksiyonunun geri dönüş
03: değerini gösterir */
04:
05: #include <stdio.h>
06:
07: int main()
08: {
09:    int karSay;
10:    int sayi = 1234;
11:
12:    karSay = printf("Ugurlu sayim =
13: %d\n",sayi);
14:
15:    printf("Ust satirda karakter sayisi:
16: %d dir\n", karSay);

       return 0;
    }
```

ÇIKTI

```
Ugurlu sayim = 1234
Ust satirda karakter sayisi: 20 dir
```

11. satırdaki işlemle, hem ekrana `Ugurlu sayim = 1234` iletisi bastırılmakta, hem de `karSay` değişkenine bu iletinin uzunluğu atanmaktadır. Ekrana basılan karakterlerin sayısı (`\n`karakteri dahil) 20 dir.

4.2 `scanf()` Fonksiyonu

Birçok programda ekrana verilerin bastırılmasının yanısıra klavyeden veri okunması gerekebilir. `scanf()` fonksiyonu klavyeden veri okumak için kullanılan fonksiyondur. `printf()` gibi`scanf()` fonksiyonuda Tablo 4.1 ve Tablo 4.2'de verilen karakterleri kullanır. Örneğin klaveden bir `x` tamsayısı okumak için:

```
scanf("%d",&x);
```

satırını yazmak yeterli olacaktır. Burada `&` işareti *adres operatörü* olarak adlandırılır ve Böüm 11'de ayıntılı olarak açıklanacaktır. Klavyeden iki farklı sayı okunmak istendiğnde `scanf()` fonksiyonu şöyle kullanılabilir:

```
scanf("%d %f",&x,&y);
```

veriler klavyeden

```
16 1.56
```

yada

```
16    1.56
```

veya

```
16
1.56
```

şekilinde girilebilir. Program 4.3'de `scanf()` fonsiyonunun kullanımı gösterilmiştir.

Program 4.3: *scanf() fonksiyonun kullanımı*

```
01: /* 04prg03.c
02:     scanf() fonksiyonu ile int ve float
03: tipindeki verilerin okunması */
04:
05: #include <stdio.h>
06:
07: main()
08: {
09:     int     t;
10:     float   g;
```

```
11:
12:     printf("Bir gercel sayi girin: ");
13: scanf("%f",&g);
14:     printf("Bir tamsayi girin    : ");
15: scanf("%d",&t);
16:
17:     printf("\n");
18:
19:     printf("\t %f * %f = %f\n",g,g,g*g);
20:     printf("\t %d * %d = %d\n",t,t,t*t);

    return 0;
   }
```

ÇIKTI

```
Bir gercel sayi girin: 1.34
Bir tamsayi girin    : 12

        1.340000 * 1.340000 = 1.795600
        12 * 12 = 144
```

4.3 puts() Fonksiyonu

Ekrana yazdırılacak ifade bir karakter topluluğu ise, `printf()`'e alternatif olarak `puts()` fonksiyonu kullanılabilir. Ancak `puts()`, ekrana bu karakter topluluğu yazdıktan sonra, imleci alt satıra geçirir. Buna göre:

```
printf("Sevgi varlığın mayasıdır.\n");
```

ile

```
puts("Sevgi varlığın mayasıdır.");
```

kullanımları eşdeğerdir.

`puts()` fonksiyonu Tablo 4.1 de verilen kontrol karakterleri ile kullanılabilir.

```
puts("Bu birinci satır...\nBu ikinci satır.");
Bu birinci satır...
Bu ikinci satır.
```

4.4 gets() Fonksiyonu

Klavyeden bir karakter topluluğu okumak için kullanılır. Okuma işlemi yeni satır karakteriyle(`\n`) karşılasılıncaya kadar sürer. `puts()` - `gets()` arsındaki ilişki, `printf()` - `scanf()` arasındaki gibidir. Yani,

```
scanf("%s",str);
```

ile

```
gets(str);
```

aynı anlamdadır. `puts()` - `gets()` fonksiyonlarının kullanımı daha sonra ayrıntılı işlenecektir.

4.5 `getchar()` Fonksiyonu

Bu fonksiyon ile standart girişten bir karakter okunur. Programı istenen bir yerde dudurup, bir karakater girinceye kadar bekletir. Örneğin:

```
...
for(i=0; i<10; i++)
{
   getchar();
   printf("%d\n",i);
}
...
```

Yukarıdaki program parçası 0-9 arası sayıları sırasıyla ekranda göstermek için kullanılır. Fakat her rakamı yazdırılmadan önce klavyeden herhangi bir karakter girip [Enter] tuşuna basılması beklenir. Bu bekleme `getchar()` fonksiyonu ile gerçekleştirilir.

4.6 Formatlı Çıktı

Bundan önceki programlardaki değişkenler serbest biçimde (free format), yani derleyicinin belirlediği biçimde ekrana yazdırılmıştı. Bazen giriş ve çıkışın biçimi kullanıcı tarafından belirlenmesi gerekebilir. Bu işlem:

Tamsayılarda `%d` yerine `%wd`
Gerçel sayılarda `%f` yerine `%w.kf`
Stringlerde `%s` yerine `%ws`

biçimindeki kullanım ile sağlanır. Burada `w` yazılacak olan sayının *alan genişliği* olarak adlandırılır. Gerçel bir değişken ekrana yazılacaksa, değişkenin virgülden sonra kaç basamağının yazdırılacağı `k` sayısı ile belirlenir. Ancak `w > k + 2` olmalıdır.

```
int i=583,j=1453;

printf("%d %d\n",i,j);     /* serbest biçim */
printf("%5d %8d\n",i,j);   /* formatlı */
```

program parçasının ekran çıktısı şöyledir:

ÇIKTI

```
583 1453
  583     1453
```

Birinci satır serbest formatta ikinci satır ise formatlı yazılmıştır. `i` değişkeninin tuttuğu 583 sayısı `%5d` formatıyla yazdırılınca, bu sayı için 5 alan genişliği tanımlanır arakasından sağdan başlayarak sayı bu alana yazılır. Benzer olarak `j` değişkeni, 8 alan genişlikli bir bölgeye yazılır.

Gerçel sayılarda iş biraz daha karışık. Örneğin:

```
int x=123.456;

printf("%f\n",x);      /* serbest biçim */
printf("%8.2f\n",x);   /* formatlı */
```

program parçası çalıştırıldığında aşağıdaki sonuç gözlenir:

ÇIKTI

```
123.456001
  123.46
```

Birinci satır serbest formatta ikinci satır ise formatlı yazılmıştır. İkinci satırda `x` değişkeni için ayrılan alan genişliği 8 ve noktadan sonra 2 basamağa kadar hassasiyet önemsenmiştir. Dikkat edilirse noktadan sonra sayı uygun bir şekilde yuvarlanmış ve sayı sağa dayalı olarak yazılmıştır.

Son olarak formatlı çıktı ile ilgili bir örnek Program 4.4'de verilmiştir. İnceleyiniz.

Program 4.4: *printf() in formatlı kullanımı*

```
01: /* 04prg04.c: Formatlı çıktı */
02:
03: #include <stdio.h>
04:
05: main()
06: {
07:    float  x = 7324.25, y = 244.531;
08:    int    i = 1299;
09:    char  *c = "Merhaba C";
10:
11:    printf("%10d\n"  ,i);
12:    printf("%10s\n"  ,c);
13:
14:    printf("%10.5f\n",x);
15:    printf("%10.1f\n",y);
16:
17:   return 0;
18: }
19:
```

ÇIKTI

```
     1299
 Merhaba C
7324.25000
     244.5
```

Ders 6: *Karşılaştırma Deyimleri*

Giriş

Program içerisinde bazen iki veya daha fazla değerin karşılaştırılması gerekebilir. Bunun için, bütün programlama dillerinde karşılaştırma deyimleri mevcuttur. C dili, `if`, `switch` ve `?`olmak üzere üç tip karşılaştırma işlemi yapmaya izin verir. Ancak `?` bir operatördür. `if` karşılaştırma deyimi ile, diğer programlama dilinde olduğu gibi `if-else` yapısı da kurulabilir.`switch` deyimi, bir değişkenin içeriğine göre program akışını yönlendirir.

6.1 Karşılaştırma Operatörleri ve Mantıksal Operatörler

Tablo 6.1'de listelenen Karşılaştırma Operatörleri, sayısal değerleri veya karakterleri mukayese etmek için kullanılır.

Tablo 6.1: *Karşılaştırma Operatörleri*

Operatör	Açıklama	Örnek	Anlamı
>	büyüktür	`x > y`	x, y den büyük mü?
<	küçüktür	`x < y`	x, y den küçük mü?
==	eşittir	`x == y`	x, y ye eşit mi?
>=	büyük-eşittir	`x >= y`	x, y den büyük yada eşit mi?
<=	küçük-eşittir	`x <= y`	x, y den küçük yada eşit mi?
!=	eşit değil	`x != y`	x, y den farklı mı?

Birden çok karşılaştırma işlemi, Tablo 6.2'deki Mantıksal Operatörler'le birleştirilebilir.

Tablo 6.2: *Mantıksal Operatörler*

Operatör	Açıklama	Örnek	Anlamı
`&&`	mantıksal VE	`x>2 && x<y`	x, 2 den büyük VE y den küçük mü?
`||`	mantıksal VEYA	`x>2 || x<y`	x, 2 den büyük VEYA y den küçük mü?
!	mantıksal DEGIL	`!(x>2)`	x, 2 den büyük değilse

C dilinde, bir mantıksal işlemin sonucu tamsayı `0` (sıfır) veya başka bir değer olur. `0` *olumsuz* `0`'dan farklı değerler *olumlu* olarak yorumlanır. Buna göre, aşağıdaki program parçasının

```
...
int x = 1, y = 2, s, u, z;

s = 2 > 1;
u = x > 3;
z = x <= y && y >0;

printf("%d\t%d\t%d", s, u, z);
...
```

çıktısı:

```
1       0       1
```

şeklinde olur. Bunun nedeni:

- 2 her zaman 1 den büyük olduğu için `s` değişkenine 1,
- `x = 1 < 3` olduğu için `x` değişkenine 0,
- `z = x <= y && y >0;` eşitliğin sağtarafının sonucu olumlu olduğu için `z` değişkenine 1 atanır.

6.2 `if, if-else` Yapısı

Bu deyimler, koşullu işlem yapan deyimlerdir. `if` ve `else` tek bir karşılaştırma deyimi olup `else` kullanımı isteğe bağlıdır. Eğer bu koşul olumlu ise `if` den sonraki bölüm yürütülür ve`else` den sonraki bölüm atlanır. Koşul olumsuz ise `if` den sonraki küme atlanır ve eğer varsa, `else` den sonraki kümedeki işlemler gerçekleştirilir.

`if` deyiminin yapının genel biçimi şöyledir:

```
if(koşul)
{
  ...
   deyimler; (küme)
  ...
}
```

`if` deyimi kullanılırken kümenin başlangıcı ve bitişini gösteren, küme parantezleri kullanılmasında kullanıcıya bir esneklik sunulmuştur. Eğer `if` deyiminden sonra icra edilecek deyimler tek satırdan oluşuyorsa, bu işaretlerin kullanılması zorunlu değildir.

Yani, `if` deyimden sonra `{` ve `}` işaretleri kullanılmamışsa, bu deyimi takip eden sadece ilk satır işleme konur. Bu durum, `else if`, `else` deyimlerinde ve daha sonra işlenecek `for` ve `while` gibi döngü deyimlerinde de geçerlidir.

Buna göre aşağıdaki kullanım

```
if(x == y){
  puts("x ve y esit");
}
```

ile

```
if(x == y)
  puts("x ve y esit");
```

eşdeğerdir.

`if` deyiminin `else` ile birlikte kullanımı şu şekildedir:

```
if(koşul){
  ...
  deyimler; (küme1)
  ...
}

else{
  ...
  deyimler; (küme2)
  ...
}
```

Bu yapının kullanılmasına dair bir örnek Program 6.1'de gösterilmiştir. Program, klavyeden girilen bir tamsayının çift olup olmadığını sınar. Bilindiği gibi, çift sayılar, 2 ile kalansız bölünebilen sayılardır.

Program 6.1: *if-else deyiminin kullanımı*

```
01: /* 06prg01.c
02:    Klavyeden girilen bir sayının çift
03: olup olmadığını sınar. */
04:
05: #include <stdio.h>
06:
07: int main()
08: {
09:       int sayi;
10:
11:       printf("Bir sayi girin: ");
12:       scanf("%d",&sayi);
13:
14:
15:       if (sayi % 2 == 0)
16:           printf("sayi cifttir.\n");
```

```
17:         else
18:             printf("sayi tektir.\n");
19:
20:    return 0;
    }
```

ÇIKTI

```
Bir sayi girin: 3
sayi tektir.
```

Mantıksal Operatörler kullanarak birden çok karşılaştırma birleştirilebilir. Buna iyi bir örnek Program 6.2'de gösterilmiştir. Program, bir yılın artık yıl olup olmadığını sınar. Bir yıl içinde, Şubat ayı 29 gün olursa o yıl artık yıl olarak adlandırılır. Artık yıl peryodik olarak 4 yılda bir gelir. Genel kanı "*bir yıl 4 ile tam bölünebirse o yıl artık yıldır*" şeklindedir. Fakat 1996 artık yıl iken 1800 artık yıl değildir. Genel sorgulama söyle olmalıdır: Eğer bir yıl

- 4 ile tam bölünüyorsa VE 100'e tam bölünmüyorsa VEYA
- 400 'e tam bölünüryorsa

o yıl artık yıldır.

Program 6.2: *if-else deyiminin kullanımı*

```
01: /* 06prg02.c: Bir yılın artık yil olup
02: olmadığını sınar. */
03:
04: #include <stdio.h>
05:
06: void main()
07: {
08:    int yil;
09:
10:    printf("Bir yil girin: ");
11:    scanf("%d",&yil);
12:
13:    if( yil % 4 == 0 && yil % 100 != 0 ||
14: yil % 400 == 0 )
15:          printf("%d artik yil\n",yil);
16:
17:    else
18:         printf("%d artik yil
    degil\n",yil);

    }
```

ÇIKTI

```
Bir yil girin: 1996
1996 artik yil
```

Eğer program içinde kullanılacak koşulların sayısı ikiden çok ise aşağıdaki yapı kullanılır:

```
if(koşul_1)
{
  ...
  deyimler; (küme_1)
  ...
}
else if(koşul_2)
{
  ...
  deyimler; (küme_2)
  ...
}
.
.
.
else if(koşul_n-1)
{
  ...
  deyimler; (küme_n-1)
  ...
}
else
{
  ...
  deyimler; (küme_n)
  ...
}
```

Program 6.3, $ax^2 + bx + c = 0$ formundaki ikinci dereceden bir polinomun köklerini hesaplamaktadır. Programda `delta` değerinin sıfırdan küçük olması durumda köklerin karmaşık sayıya dönüşeceğide göz önüne alınmıştır. Bu program `if`, `else if` ve `else` yapısı göstermek için klasik bir örnektir.

Program 6.3: *if, else if, else yapısı*

```
01:  /* 06prg03.c:
02:     ax*x + bx + c = 0 denkleminin
03: (karmaşık sayılı kökler dahil) çözümü */
04:
05: #include <stdio.h>
06: #include <math.h>
07:
08: int main()
09: {
10:     float a, b, c, delta, x1, x2, x,
```

```
11: kok_delta;
12:
13:     printf("a, b, c degerlerini
14: girin:\n");
15:     scanf("%f %f %f",&a,&b,&c);
16:
17:     delta = b*b - 4.0*a*c;
18:
19:     if( delta > 0.0 ){
20:        x1 = ( -b + sqrt(delta) )/( 2.0*a
21: );
22:        x2 = ( -b - sqrt(delta) )/( 2.0*a
23: );
24:
25:        printf("\nReel kokler:");
26:        printf("\nx1 = %f",x1);
27:        printf("\nx2 = %f",x2);
28:     }
29:     else if( delta < 0.0 ){
30:        kok_delta = ( sqrt(-delta) ) /
31: (2.0*a);
32:        x = -0.5*b/a;
33:
34:        printf("\nKarmasik kokler:");
35:        printf("\nx1 = %f + (%f)i", x,
36: kok_delta);
37:        printf("\nx2 = %f - (%f)i", x,
38: kok_delta);
39:     }
40:     else{
          x = -0.5*b/a;

          printf("\nKokler eşit:");
          printf("\nx1 = x2 = %f",x);
       }

     return 0;
   }
```

ÇIKTI

```
a, b, c degerlerini girin:
2 4 -8

Reel kokler:
x1 = 1.236068
x2 = -3.236068
```

ÇIKTI

```
a, b, c degerlerini girin:
1 1 1

Karmasik kokler:
x1 = -0.500000 + (0.866025)i
x2 = -0.500000 - (0.866025)i
```

6.3 `switch - case` Yapısı

Bu deyim bir *değişken*in içeriğine bakarak, programın akışını bir çok seçenekten birine yönlendirir. `case` (durum) deyiminden sonra değişkenin durumu belirlenir ve takip eden gelen satırlar (deyimler) işleme konur. Bütün durumların aksi söz konu olduğunda gerçekleştirilmesi istenen deyimler `default` deyiminden sonraki kısımda bildirilir. Genel yazım biçimi:

```
switch(değişken)
{
     case sabit1:
        ...
        deyimler;
         ...
     case sabit2:
         ...
         deyimler;
         ...
     .
     .
     .
     case sabitn:
         ...
         deyimler;
         ...
     default:
         ...
         hata deyimleri veya varsayılan deyimler;
         ...
}
```

Program Program 6.4'te `switch` deyiminin basit bir kullanımı gösterilmiştir.

Program 6.4: *switch-case yapısının kullanımı*

```
01: /* 06prg04.c: switch - case yapısının
02: kullanımı */
03:
04: #include <stdio.h>
05:
```

```
06: int main(void)
07: {
08:     char kr;
09:
10:     printf("Lutfen bir karakter
11: girin\n");
12:
13:     kr = getchar(); /* tek bir
14: karakterin okunması */
15:
16:     switch (kr)
17:     {
18:        case 'a':
19:           printf("a harfine
20: bastiniz\n");
21:        case 'b':
22:           printf("b harfine
23: bastiniz\n");
24:        default:
              printf("a veya b ye
    basmadiniz\n");
        }

    return 0;
    }
```

ÇIKTI

```
Lutfen bir karakter girin
a
a harfine bastiniz
b harfine bastiniz
a veya b ye basmadiniz
```

ÇIKTI

```
Lutfen bir karakter girin
b
b harfine bastiniz
a veya b ye basmadiniz
```

ÇIKTI

```
Lutfen bir karakter girin
k
a veya b ye basmadiniz
```

ÇIKTI

```
Lütfen bir karakter girin
c
a veya b ye basmadiniz
```

Programda, klavyeden okunan tek bir karakter değişkenin içeriğine bakılıp uygun dallanmalar yaptırılmıştır. 11. satırda değişken `getchar()` fonksiyonu ile okutulmuştur.
Eğer `'a'` veya`'b'` karakterlerinden biri girilirse, ekrana bu harflerin girildiğine dair mesaj yazılacak, aksi takdirde bu karakterin dışında bir karakterin giriş olarak kullanıldığı gösteren bir mesaj yazılacaktır. Örneğin `'c'` karakteri klavyeden girilmiş ise `a veya b ye basmadiniz` gibi. Fakat `'a'` karakterleri girildiğinde ekrana her üç durumda yazdırılmaktadır. Bunun sebebi, `case 'a':` durumunda sırasıyla 16, 18 ve 20. satırların işleme konmasıdır. Bunu engellemek için 16. satırdan sonra programın başka bir yere yönlendirilmesi gerekir. Bu yönlendirme, Bölüm 7'de anlatılacak olan `break` deyimi ile yapılır. Derleyici bu deyim ile karşılaştığında, bulunduğu yapının içinden koşulsuz olarak ayrılır ve takip eden işleme başlar[1].

Program 6.4'te `case 'a':` durumu için 16, 18 ve 20. satırlar da işleme konumuştu. Eğer klavyeden `'a'` karakterini girip ekrana sadece `a harfine bastiniz` iletisi yazdırılmak isteniyorsa, 20. satıra `break` deyimi ilave edilmelidir. `break` deyiminin kullanımı Program 6.5'te gösterilmiştir.

Program 6.5: *switch-case yapısı ve break kullanımı*

```
01: /* 06prg05.c: switch - case yapısı ve
02: break kullanımı */
03:
04: #include <stdio.h>
05:
06: int main(void)
07: {
08:       char kr;
09:
10:       printf("Lutfen bir karakter
11: girin\n");
12:
13:       kr = getchar(); /* tek bir
14: karakterin okunması */
15:
16:       switch (kr)
17:       {
18:         case 'a':
19:            printf("a harfine
20: bastiniz\n");
21:            break;
22:         case 'b':
23:            printf("b harfine
24: bastiniz\n");
```

```
25:             break;
26:          default:
27:             printf("a veya b ye
    basmadiniz\n");
                break;
          }

    return 0;
    }
```

ÇIKTI

```
Lutfen bir karakter girin
a
a harfine bastiniz
```

ÇIKTI

```
Lutfen bir karakter girin
k
a veya b ye basmadiniz
```

Program 6.6 `switch-case` yapısın farklı bir kullanımı ile ilgili bir örnektir. Programda, önce iki sayı isteniyor ardından yapılan seçimle bu sayıların toplamı, farkı, çarpımı veya oranı ekrana yazdırılıyor.

Program 6.6: *switch-case yapısı ve break kullanımı*

```
01: /* 06prg06.c: switch-case yapısı */
02:
03: #include <stdio.h>
04: #include <stdlib.h>
05:
06: int main(void)
07: {
08:       int   secim;
09:       float x,y, sonuc;
10:
11:       printf("Iki sayi girin: ");
12:       scanf("%f %f",&x,&y);
13:
14:       puts("*** Menu ***");
15:       puts("[1] Toplama");
16:       puts("[2] Cikarma");
17:       puts("[3] Carpma");
18:       puts("[4] Bolme");
19:
20:       printf("Seciminiz: ");
21:       scanf("%d",&secim);
```

```
22:
23:         switch( secim )
24:         {
25:            case 1:
26:                    sonuc = x + y;
27:                    printf("Toplam =
28: %f\n",sonuc);
29:                    break;
30:            case 2:
31:                    sonuc = x-y;
32:                    printf("Fark =
33: %f\n",sonuc);
34:                    break;
35:            case 3:
36:                    sonuc = x * y;
37:                    printf("Carpim =
38: %f\n",sonuc);
39:                    break;
40:            case 4:
41:                    sonuc = x/y;
42:                    printf("Oran =
43: %f\n",sonuc);
44:                    break;
45:          default:
46:                    puts("Yanlis secim
    !\a");
          }

      return 0;
    }
```

ÇIKTI

```
Iki sayi girin: 3 8
*** Menu ***
[1] Toplama
[2] Cikarma
[3] Carpma
[4] Bolme
Seciminiz: 1
Toplam = 11.000000
```

ÇIKTI

```
Iki sayi girin: 3 8
*** Menu ***
[1] Toplama
[2] Cikarma
[3] Carpma
[4] Bolme
```

```
Seciminiz: 7
Yanlis secim !
```

`switch-case` yapısı `if-else` yapısının bir alternatifidir. Yani, Program 6.6'daki `switch-case` kısmı, `if-else` yapısı ile de aşağıdaki gibi yazılabilirdi. İnceleyiniz.

```
switch( secim )
{
        case 1:
                sonuc = x + y;
                printf("Toplam
= %f\n",sonuc);
                break;
        case 2:
                sonuc = x-y;
                printf("Fark =
%f\n",sonuc);
                break;
        case 3:
                sonuc = x * y;
                printf("Carpim
= %f\n",sonuc);
                break;
        case 4:
                sonuc = x/y;
                printf("Oran =
%f\n",sonuc);
                break;
        default:
                puts("Yanlis
secim !\a");
}
```

```
 if(secim == 1){
                sonuc = x + y;
                printf("Toplam
= %f\n",sonuc);
 }
 else if(secim == 2){
                sonuc = x-y;
                printf("Fark =
%f\n",sonuc);
 }
 else if(secim == 3 ){
                sonuc = x * y;
                printf("Carpim
= %f\n",sonuc);
 }
 else if(secim == 4){
                sonuc = x/y;
                printf("Oran =
%f\n",sonuc);
 }
 else{
                puts("Yanlis
secim !\a");
 }
```

6.4 ? Karşılaştırma Operatörü

Bu operatör, `if-else` karşılaştırma deyiminin yaptığı işi sınırlı olarak yapan bir operatördür. Genel yazım biçimi:

```
(koşul) ? deyim1 : deyim2;
```

İlk önce koşul sınanır. Eğer koşul olumluysa `deyim1` aksi takdirde `deyim2` değerlendirilir. `deyim1` ve `deyim2` de atama işlemi yapılamaz. Ancak `koşul` deyiminde atama işlemi yapılabilir.`deyim1` ve `deyim2` yerine fonksiyon da kullanılabilir. Aşağıda bu deyimin kullanımına ait örnekler verilmiştir.

```
x = ( a > b ) ? a : b;
```

Yukarıdaki ifadede koşul a'nın b'den büyük olmasıdır. Eğer olumluysa x adlı değişkene a, değilse b değeri atanır. Bu şekilde kullanım if-else yapısı ile kurulmak istenirse:

```
if( a > b )   x = a;
else          x = b;
```

şeklinde olacaktır. Program 6.7 ? karşılaştırma operatörünün basit bir kullanımını göstermektedir.

Program 6.7: *? ve if kullanımı*

```
/* 06prg07.c: ? ve if-else yapısının
kullanımı */

#include <stdio.h>

int main(void)
{
    float x, y, z;

    printf("x : "); scanf("%f",&x);
    printf("y : "); scanf("%f",&y);

      if(y)                          /* y,
0'dan farklı mı? */
        z = ( y > x ) ? x/y : x*y; /*
y>x ise z = x/y, değilse z = x*y  */
      else
        z = 0.0;

    printf("z = %f\n",z);

 return 0;
}
```

ÇIKTI

```
x : 3
y : 5
z = 0.600000
```

ÇIKTI

```
x : 11
y : 0
z = 0.000000
```

12. satırdaki `if` deyimindeki koşul biraz farklıdır. Genel olarak koşul bu şekilde bildirilirse, koşulun 0 dan farklı olup olmadığı sınanır. Yani:

```
if(y)
```

ile

```
if( y != 0 )
```

aynı anlamdadır.

Bu kullanım çok yagındır. Eğer `y`, 0 dan farklı ise koşul olumlu olarak değerlendirilecektir. 13. satırda ? ile bir sınama yapılmaktadır. Eğer `y`, `x` den büyük ise `z` değişkenine `x/y`, aksi takdirde `x*y` değeri atanmaktadır. Eğer `y = 0` ise `z` değişkenine 0 değeri atanmaktadır.

Ders 7: *Döngüler*

Giriş

Döngü (loop) deyimleri, bir kümenin belli bir koşul altında tekrar edilmesi için kullanılır. C programlama dilinde, `while`, `do...while` ve `for` olmak üzere üç tip döngü deyimi vardır. Diğer programlama dillerinde olduğu gibi, bu deyimlerle istenildiği kadar iç-içe döngü yapısı kullanılabilir.

7.1 `while` Döngüsü

Tekrarlama deyimidir. Bir küme ya da deyim `while` kullanılarak bir çok kez yinelenebilir. Yinelenmesi için koşul sınaması döngüye girilmeden yapılır. Koşul olumlu olduğu sürece çevrim yinelenir. Bu deyimin kullanımı Program 7.1 de gösterilmiştir. Genel yazım biçimi:

```
while(koşul)
```

```
{
 ...
  döngüdeki deyimler; [küme]
 ...
}
```

Program 7.1: *while döngüsü*

```
01: /* 07prg01.c: while döngüsü */
02:
03: #include <stdio.h>
04:
05: main()
06: {
07:    int x=0;
08:
09:    while(x <= 10)
10:       printf("%d\n",x++);
11:
12:  return 0;
13: }
```

ÇIKTI

```
0
1
2
3
4
5
6
7
8
9
10
```

Program 7.1, 0-10 arasındaki sayıları ekrana yazdırmaktır. 9. satırdaki `while` deyiminden sonra `{` işareti kullanılmamıştır. Bu durumda, sadece takip eden satır (10. satır) döngünün içine dahil edilir.

7.2 `do ... while` Döngüsü

Bu deyimin `while` deyiminden farkı, koşulun döngü sonunda sınanmasıdır. Yani koşul sınanmadan döngüye girilir ve döngü kümesi en az bir kez yürütülür. Koşul olumsuz ise döngüden sonraki satıra geçilir. Bu deyimin kullanımı Program 7.2 de gösterilmiştir. Genel yazım biçimi:

```
do{
   ...
   döngüdeki deyimler;
   ...
}while(koşul);
```

Program 7.2: *do-while döngüsü*

```
01: /* 07prg02.c: do-while yapısı */
02:
03: #include <stdio.h>
04:
05: main()
06: {
07:    int sayi;
08:
09:    do
10:    {
11:       printf("Bir sayi girin : ");
12:       scanf("%d",&sayi);
13:       printf("iki kati       : %d\n",2*sayi);
14:
15:    }while( sayi>0 );   /* koşul */
16:
17:    puts("Döngü sona erdi.");
18:
19:  return 0;
20: }
```

ÇIKTI

```
Bir sayi girin : 1
iki kati       : 2
Bir sayi girin : 3
iki kati       : 6
Bir sayi girin : 4
iki kati       : 8
Bir sayi girin : -3
iki kati       : -6
Cevrim sona erdi.
```

15. satırdaki koşul olumlu olduğu sürece (`sayi>0` olduğu sürece), klavyeden yeni bir değer 12. satırda okunur. Aksi takdirde (`sayi<=0` ise) çevrimin sona erdiğine dair mesajla program sonlanır.

7.3 `for` Döngüsü

Bu deyim, diğer döngü deyimleri gibi bir kümeyi bir çok kez tekrarlamak için kullanılır. Koşul sınaması `while` da olduğu gibi döngüye girmeden yapılır. Bu döngü deyimin içinde diğerlerinden farklı olarak başlangıç değeri ve döngü sayacına sahip olmasıdır. Bu deyimin kullanımı Program 7.3 de gösterilmiştir Genel yazım biçimi:

```
for( başlangıç ; koşul ; artım )
{
 ...
  döngüdeki deyimler;
 ...
}
```

Program 7.3: *for döngüsü*

```
01: /* 07prg03.c: for döngüsü ile faktoriyel hesabı.
02: */
03:
04: #include <stdio.h>
05:
06: int main()
07: {
08:    long  i, n, faktor;
09:
10:    printf("Faktoriyeli hesaplanacak sayi girin :
11: ");
12:    scanf("%ld",&n);
13:
14:    faktor=1;
15:    for(i=1; i<=n; i++){
16:          faktor *= i;      /* n! = 1 x 2 x 3 x
17: ... x n */
18:    }
19:
20:    printf("%ld! = %ld\n", n, faktor);

     return 0;
    }
```

ÇIKTI

```
Faktoriyeli hesaplanacak sayi girin : 4
4! = 24
```

ÇIKTI

```
Faktoriyeli hesaplanacak sayi girin : 15
15! = 2004310016
```

Program da faktoriyel hesabı 16. satırda gerçekleştirilmiştir. Faktöriyel, bilindiği gibi *n*! = 1x2x3x ... x*n* tanımlanır. Gerçekte 15! = 1307674368000 olmasına rağmen, program 15! = 2004310016 olarak hesaplamıştır. Sizce bunun sebebi nedir? Cevap için bkz: Bölüm 2.

Program 7.3'de döngüye girilmeden, `faktor = 1` atması yapılmıştır.

```
faktor = 1;
for(i=1; i<=n; i++)
    faktor *= i;
```

Bu döngü öncesi ilk değer ataması, döngünün başlangıç kısmında şu şekilde de yapılabilir:

```
for(faktor=1, i=1; i<=n; i++)
    faktor *= i;
```

`printf` fonksiyonu ile desimal (taban-10) sayılarıların nasıl yazdırılacağı bundan önceki kısımlarda gösterilmişti. Program 7.4'te 0-15 arası desimal sayıların Oktal (taban-8) ve Heksadesimal (taban-16) karşılıkları ile `printf` kullanılarak yazdırılması gösterilmiştir.

Program 7.4: *Sayı sistemi*

```
01: /* 07prg04.c: Sayı sistemi
02:    %d : desimal     10 tabanındaki sayı
03:    %o : oktal        8 tabanındaki sayı
04:    %x : hexadesimal 16 tabanındaki sayı (küçük
05: harf)
06:    %X : hexadesimal 16 tabanındaki sayı (büyük
07: harf) */
08:
09: #include <stdio.h>
10:
11: int main()
12: {
13:     int i;
14:
15:     for (i=0; i<16; i++)
16:       printf("%2d  %2o  %x  %X\n", i,i,i,i);
17:
     return 0;
    }
```

ÇIKTI

```
 0   0  0  0
 1   1  1  1
 2   2  2  2
 3   3  3  3
 4   4  4  4
 5   5  5  5
 6   6  6  6
 7   7  7  7
 8  10  8  8
 9  11  9  9
10  12  a  A
11  13  b  B
12  14  c  C
13  15  d  D
14  16  e  E
15  17  f  F
```

7.4 İç içe Geçmiş Döngüler

Bir program içinde birbiri içine geçmiş birden çok döngü de kullanılabilir. Bu durumda (bütün programlama dillerinde olduğu gibi) önce içteki döngü, daha sonra dıştaki döngü icra edilir.

Üç basamaklı, basamaklarının küpleri toplamı kendisine eşit olan tam sayılara Armstrong sayı denir. Örneğin: 371 bir Armstrong sayıdır çünkü 3^3 + 7^3 + 1^3 = 371. Program 7.5'de iç içe geçmiş üç `for` döngüsü ile bütün Armstrong sayıları bulup ekrana yazar. İnceleyiniz.

Program 7.5: *iç-içe for döngüleri*

```
01: /* 07prg05.c:
02:    Üç basamaklı, basamaklarının küpleri toplamı
03: kendisine eşit olan tam
04:    sayılara  Armstrong sayı denir. Örneğin: 371
05: = 3^3 + 7^3 + 1^3.
06:    Bu program İç-içe geçmiş 3 döngü ile bütün
07: Aramstrong sayıları bulur. */
08:
09: #include <stdio.h>
10:
11: int main()
12: {
13:    int a,b,c, kup, sayi, k=1;
14:
15:    for(a=1; a<=9; a++)
16:    for(b=0; b<=9; b++)
17:    for(c=0; c<=9; c++)
18:    {
19:        sayi = 100*a + 10*b + c;       /* sayi =
20: abc (üç basamaklı) */
21:        kup  = a*a*a + b*b*b + c*c*c;  /* kup  =
22: a^3+b^3+c^3        */
23:
           if( sayi==kup ) printf("%d.
    %d\n",k++,sayi);
       }

    return 0;
    }
```

ÇIKTI

```
1. 153
2. 370
3. 371
4. 407
```

7.5 Sonsuz Döngü

Bir döngü işlemini sonsuz kere tekrarlarsa bu döngü sonzuz döngü olarak adlandırılır. Böyle bir döngü için, koşul çok önemlidir. Örneğin `while` döngüsü için:

```
...
while(1)
{
   printf("Sonsuz döngü içindeyim...\n");
}
...
```

yada

```
...
while(7>3)
{
```

```
        printf("Sonsuz döngü içindeyim...\n");
    }
    ...
```

Her iki durumda da çevrimler, sonsuz döngü durumundadır.
Çünkü `while(1)` ve `while(7>3)` ifadelerdeki koşullar daima olumludur. Bu durumda çevrim sonsuz döngüye girer.

`for` döngüsünde, başlangıç, koşul ve artım parametrelerinden herhangi birini kullanmak isteğe bağlıdır. Her hangi biri verilmediğinde döngünün nasıl davranacağı iyi yorumlanmalıdır. Örneğin `for` döngüsünün hiçbir parametresi verilmezse, döngü sonsuz çevrime girer. Yani:

```
for(;;)
    printf("Sonsuz döngü içindeyim...\n");
```

gibi.

7.6 `break` Deyimi

Bir C programında, bir işlem gerçekleştirilirken, işlemin sona erdirilmesi bu deyim ile yapılır. Örneğin, döngü deyimleri içindekiler yürütülürken, çevrimin, koşuldan bağımsız kesin olarak sonlanması gerektiğinde bu deyim kullanılır. Mesela:

```
...
do{
     scanf("%d",&x);

     if(x==0) break;

     printf("%f",1.0/x);

}while(1);
...
```

Yukarıdaki program parçasında, `do ... while` döngüsündeki koşul daima olumludur. Bu durumda döngü sonsuzdur. Fakat döngü içinde `if` deyimindeki koşul gerçekleşirse, döngü koşuluna bakılmaksızın terkedilir. Bu işlemi sağlayan `break` deyimidir.

Program 7.6 klavyeden girilen sayı pozitif olduğu sürece sayının faktoriyelini hesaplar. Sayı negatif olduğunda döngü `break` ile sonlandırılır. Inceleyiniz.

Program 7.6: *break deyiminin kullanımı*

```
01: /* 07prg06.c: n>=0 olduğu sürece n! değerini
02: hesaplar */
03:
04: #include <stdio.h>
05:
06: int main()
07: {
```

```
08:     long int i,n,faktor;
09:
10:     while(1) /* sonsuz döngü */
11:     {
12:         printf("Faktoriyeli hesaplanacak sayi
13: girin : ");
14:         scanf("%ld",&n);
15:
16:         if(n<0) break; /* döngüyü sonlandır */
17:
18:         for(faktor=1, i=1; i<=n; i++)
19:            faktor *= i;
20:
21:         printf("%ld! = %ld\n",n,faktor);
22:     }
23:
      return 0;
    }
```

ÇIKTI

```
Faktoriyeli hesaplanacak sayi girin : 2
2! = 2
Faktoriyeli hesaplanacak sayi girin : 3
3! = 6
Faktoriyeli hesaplanacak sayi girin : 5
5! = 120
Faktoriyeli hesaplanacak sayi girin : 9
9! = 362880
Faktoriyeli hesaplanacak sayi girin : 0
0! = 1
Faktoriyeli hesaplanacak sayi girin : -4
```

7.7 `continue` Deyimi

Bir döngü içerisinde `continue` deyimi ile karşılaşılırsa, ondan sonra gelen deyimler atlanır ve döngü bir sonraki çevrime girer. Örneğin:

```
        ...
        for(x=-50;i<=50;x++)
        {
             if(x<0) continue;      /* x<0 ise alttaki satırı
atla */
             printf("%d\t%f",x,sqrt(x));
        }
        ...
```

Program parçasının çıktısı:

```
        0      0.000000
        1      1.000000
        2      1.414213
        3      1.732050
        .      .
        .      .
        .      .
        50     7.071067
```

Program 7.7, x, y'den farklı olmak üzere `|x|+|y|<=3` eşitsizliğini sağlayan tamsayı çiftlerini bulup ekrana yazar. Bu eşitsizliği sağlayan toplam 22 çift vardır. Programda, her bir çift parantez içinde yazdırılmıştır. İnceleyiniz.

Program 7.7: *continue deyiminin kullanımı*

```
01: /* 07prg07.c:
02:    x, y'den farklı olmak üzere |x|+|y|<=3
03: eşitsizliğini sağlayan
04:    tamsayı çiftlerini ekrana yazar */
05:
06: #include <stdio.h>
07:
08: int main()
09: {
10:    int x,y,k=1;
11:
12:    for (x=-3;x<=3;x++)
13:    for (y=-3;y<=3;y++)
14:    {
15:       /* x=y ise yeni çevrime gir, alt satırları
16: atla */
17:       if(x==y) continue;
18:
19:       if( abs(x)+abs(y)<=3 )
20:         printf("%2d. (%2d,%2d)\n",k++,x,y);
21:    }
22:
     return 0;
    }
```

ÇIKTI

```
 1. (-3, 0)
 2. (-2,-1)
 3. (-2, 0)
 4. (-2, 1)
 5. (-1,-2)
 6. (-1, 0)
 7. (-1, 1)
 8. (-1, 2)
 9. ( 0,-3)
10. ( 0,-2)
11. ( 0,-1)
12. ( 0, 1)
13. ( 0, 2)
14. ( 0, 3)
15. ( 1,-2)
16. ( 1,-1)
17. ( 1, 0)
18. ( 1, 2)
19. ( 2,-1)
20. ( 2, 0)
21. ( 2, 1)
22. ( 3, 0)
```