

C PROGRAMLAMA



Program Akış Kontrol Yapıları



- Normal şartlarda C dilinde bir programın çalışması, komutların yukarıdan aşağıya doğru ve sırasıyla işletilmesiyle gerçekleştirilir.
- Ancak bazen problemin çözümü, bir yada birden fazla koşula bağlı olarak, yapılacak iş yada işlem adımlarına karar vermek suretiyle gerçekleştirilebilir.
- C dilinde bu amaçla kullanılan üç farklı koşul (ŞART) komutu bulunmaktadır.
- *if* deyimi
- *?:* üçlü şart operatörü (ternary conditional)
- *switch* çoklu seçim deyimi (multiply selection statement)

if Deyimi



- Verilen durum yada koşula göre istenilen işlem yada işlemleri gerçekleştirmek için kullanılır.
- Değişik kullanım biçimleri vardır.
- **Tek alternatifli if;**

```
1  #include<stdio.h>
2
3  int main(){
4
5      if ( koşul )
6          komut1;
7
8      komut2;
9  }
10
```

```
1  #include<stdio.h>
2  int main(){
3      if ( koşul ) {
4          komut1_1;
5          komut1_2;
6          komut1_3;
7      }
8      komut2;
9  }
10
```

- Bu durumda koşulun sonucu DOĞRU (true), başka bir deyişle EVET ise; koşuldan hemen sonra gelen *komut1* satırı veya satırları çalışır.
- Koşulun sonucu YANLIŞ (false), yani HAYIR ise; IF komutundan sonra gelen *komut2* satırı çalıştırılır.
- Koşul doğru olduğunda birden fazla komut satırı çalışacaksa bu komutlar blok { - } içinde yazılmalıdır.

if Deyimi



- İki alternatifli if;

```
1  #include<stdio.h>
2
3  int main(){
4
5      if ( koşul )
6          komut1;
7      else
8          komut2;
9  }
10
```

```
1  #include<stdio.h>
2  int main(){
3      if ( koşul ) {
4          komut1_1;
5          komut1_2;
6          komut1_3;
7      } else {
8          komut2_1;
9          komut2_2;
10         komut2_3;
11     }
12 }
13
```

- Bu durumda koşul doğru ise *komut1* satırı veya satırları yanlış ise *komut2* satırı veya satırları çalışacaktır.
- Birden fazla komut satırları için blok işareti { - } kullanılmalıdır.

if Deyimi - Örnek



```
1  /* Klavyeden girilen iki adet sayıdan
2   * büyük alanı bulup gösteren program
3   */
4
5  #include <stdio.h>
6
7  int main()
8  {
9      int sayi1;
10     int sayi2;
11
12     printf("1.sayi :");
13     scanf("%d",&sayi1);
14
15     printf("2.sayi :");
16     scanf("%d",&sayi2);
17
18     if ( sayi1 > sayi2 )
19         printf("\n Buyuk Olan = %d \n", sayi1);
20     else
21         printf("\n Buyuk Olan = %d \n", sayi2);
22
23     system("PAUSE");
24     return 0;
25 }
26
```

```
C:\Dev-Cpp\Proje2.exe
1.sayi :5
2.sayi :78

Buyuk Olan = 78
Devam etmek için bir tuşa basın . . .
```

?: Üçlü şart operatörü



- Bu operatör üç adet operand alır.
- C diline özgü bir operatördür.
- Genel kullanım biçimi şu şekildedir:
 - `operand1 ? operand2 : operand3`
- **operand1** ifadesinin sonucu doğru (true) ise **operand2**, yanlış (false) ise **operand3** çalıştırılır.
- `ortalama >= 60 ? printf("Geçti") : printf("Kaldı");`

Önemli bir nokta



- mantıksal işlemleri gerçekleştirirken dikkat edilemesi gereken önemli bir nokta vardır.
- Buda, mantıksal işlemlerde 0 (sıfır) yanlışı (false) ifade eder, sıfırdan farklı herhangi bir değer doğruyu (true) ifade etmesidir.
- Bu durumda aşağıdaki kodun çıktısı ne olur ?

```
1  #include<stdio.h>
2
3  int main(){
4
5      if ( 5 >= 5 && 6 )
6          printf(" Deneme 1 ");
7      else
8          printf(" Deneme 2 ");
9  }
10
```

switch çoklu seçim deyimi



- Bu deyim verilen ifadenin değerine göre verilen komut yada komut satırlarını çalıştırır.
- Verilen ifadenin değerinin sıralı giden (ordinal) olması gerekir.
- Bu ifade için kullanılacak değişken tipleri **int** ve **char** olabilir, **float** yada **double** gibi ondalık değerler olamaz.

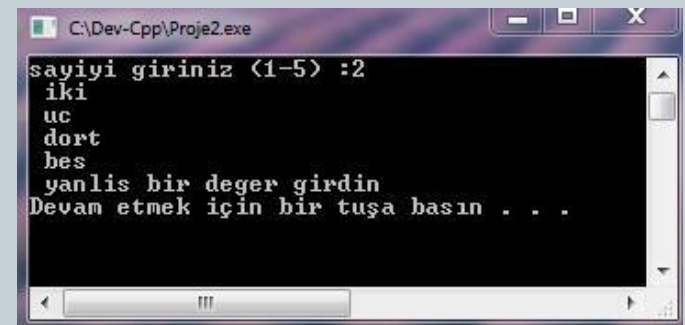
switch sözcüğünün yanındaki ifadenin değeri, blok içindeki case sözcüklerinin yanında verilen değerlerden hangisine uyuyorsa o satırdaki komut yada komutlar çalıştırılır.

```
switch (ifade) {  
  
    case değer1:  
        komut1;  
  
    case değer2:  
        komut2;  
  
    case değerN:  
        komutN;  
  
    default: defaoultkomut;  
  
}
```


Switch – örnek program



```
1  /* 1 - 5 arası rakamların okunuşlarını ekrana yazan program */
2
3  #include <stdio.h>
4
5  int main()
6  {
7      int a;
8
9      printf("sayiyi giriniz (1-5) :");
10     scanf("%d",&a);
11
12     switch (a) {
13         case 1:
14             printf(" bir \n");
15         case 2:
16             printf(" iki \n");
17         case 3:
18             printf(" uc \n");
19         case 4:
20             printf(" dort \n");
21         case 5:
22             printf(" bes \n");
23
24         default: printf(" yanlis bir deger girdiniz \n");
25     }
26
27     system("PAUSE");
28     return 0;
29 }
```

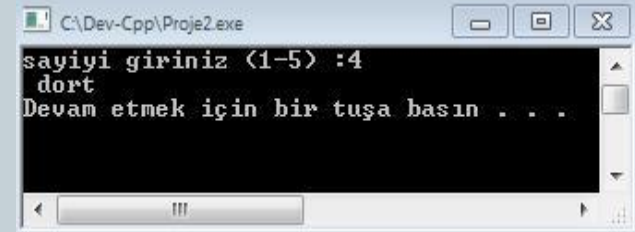


```
C:\Dev-Cpp\Proje2.exe
sayiyi giriniz (1-5) :2
iki
uc
dort
bes
yanlis bir deger girdiniz
Devam etmek için bir tuşa basın . . .
```

Görüldüğü üzere
bu kodda bir
problem var ?

Switch – örnek program

```
1  /* 1 - 5 arası rakamların okunuşlarını ekrana yazan program */
2
3  #include <stdio.h>
4
5  int main()
6  {
7      int a;
8      printf("sayiyi giriniz (1-5) :");
9      scanf("%d",&a);
10
11     switch (a) {
12         case 1:
13             printf(" bir \n");
14             break;
15         case 2:
16             printf(" iki \n");
17             break;
18         case 3:
19             printf(" uc \n");
20             break;
21         case 4:
22             printf(" dort \n");
23             break;
24         case 5:
25             printf(" bes \n");
26             break;
27         default: printf(" yanlis bir deger girdiniz \n");
28     }
29
30     system("PAUSE");
31     return 0;
32 }
33
```



Bir önceki kodda **break** sözcüğü yazılmadığı için program, uygun case sözcüğündeki komutu çalıştırdıktan sonra, sırayla geri kalan tüm komutları da çalıştırmıştı.

Burda ise her bir case sözcüğüne **break** ifadesini ekledik.

Bu sayede program uygun case sözcüğündeki komutu çalıştırdı ve **switc** bloğu sonlandı.

Döngüler (Loops)



- Algoritma içerisinde gerçekleştirilen belli adımların tekrarlanması sonucu problem çözümüne gidilebilir.
- Döngüler sayesinde geliştirilen program içerisinde gereksiz kod satırlarının giderilmesi sağlanır.
- İyi bir algoritma problemi olabilecek en kısa adımda ve en etkili biçimde çözebilendir.
- Örneğin programcının adını ekrana 10 kere yazan bir program düşünelim.

Şimdiye kadar öğrendiğimiz verilerle bu işlemi yandaki gibi gerçekleştirebiliriz.

Fakat böyle bir işlemin 100 kere yada daha fazla tekrarlanması istendiğinde ne olacak ?

Bu durumda döngüler bize yardımcı olacak 😊

```
int main()  
{  
    printf("kirami \n");  
    printf("kirami \n");  
    printf("kirami \n");  
    printf("kirami \n");  
    printf("kirami \n");  
    printf("kirami \n");  
    printf("kirami \n");  
    printf("kirami \n");  
    printf("kirami \n");  
    printf("kirami \n");  
  
    system("PAUSE");  
    return 0;  
}
```

Sayaç kontrollü döngüler



- Yapılacak tekrar miktarının bilindiği durumlarda döngü bir sayaç kullanılarak tasarlanabilir.
- Sayaç aslında tekrar edilme işleminin ne kadar yapıldığını tutan bir değişkendir.
- Böyle sayaç kontrollü bir döngüde olması gereken temel öğeler şunlardır.
 - Döngüde bir sayaç değişkeni olmalıdır. Bu değişken döngü sayacı olarak bilinir.
 - Döngü sayacına bir atama ifadesiyle ilk değer verilir.
 - Döngü sayacının değerinin sınır değere gelip gelmediği kontrol edilir
 - Döngünün gövdesinin her çalışmasından sonra, sayacın değeri bir artırma yada eksiltme işlemi ile değiştirilir.

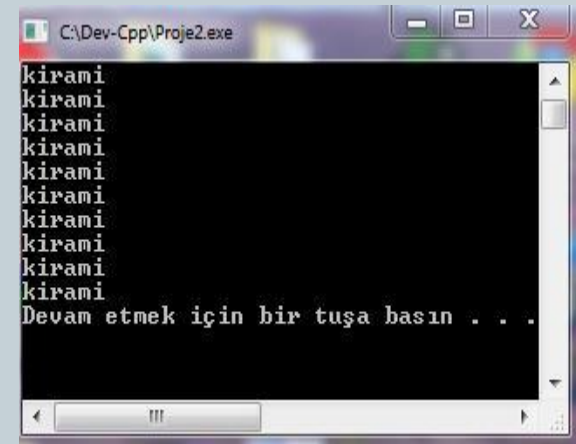
Sayaç kontrollü döngüler



- Şimdi programcının adını ekrana 10 kere yazdıran programı döngülerle tekrar gerçekleştirelim.

```
int main()
{
    int sayac;
    sayac = 1; /* sayac değişkenine ilk değer atanıyor */
    BASLA:
        printf("kirami \n");
        sayac = sayac + 1; /* sayac 1 artırılıyor */
        if(sayac<=10) /* sayac sınır değeriyle karşılaştırılıyor */
            goto BASLA;

    system("PAUSE");
    return 0;
}
```



- Görüldüğü gibi döngüler sayesinde printf() fonksiyonunu bir sefer kullanarak programcının ismini ekrana 10 kere yazdırabildik.
- Artık bu işlemi 100 kerede tekrarlasak 1000 kerede tekrarlasak mevcut kod satırı sayısında bir değişiklik olmayacak.
- **goto** komutu programın yapısıllığını ve okunabilirliğini azalttığı için programlamada pek tavsiye edilmez.

While döngüsü



- Döngü oluşturma'nın C dilinde bir çok yolu vardır.
- Bunlardan en basiti **while** döngüsüdür.

```
1  /* Klavyeden girilen iki adet sayıdan
2   * büyük alanı bulup gösteren program
3   */
4
5  #include <stdio.h>
6
7  int main()
8  {
9      int sayac = 1;
10
11     while ( sayac <= 10 ) {
12         printf("kirami \n");
13         sayac = sayac + 1;
14     }
15
16     system("PAUSE");
17     return 0;
18 }
19
```

Gözcü kontrollü döngüler (sentinel controlled loops)



- Çalıştırılması gereken adımların tekrar sayısının bilinmediği problemlerde program, kullanıcının dışarıdan belli bir değer veya değer gurubundan birini girdi olarak vermesiyle ya da program içerisinde üretilen belli bir değere göre sonlandırılır.
- Bu değer yada değerlere gözcü değeri (sentinel value) adı verilir.

Gözcü kontrollü döngüler



```
1  /* Not olarak -1 girilene kadarki notların ortalamasını bulur*/
2  #include <stdio.h>
3  int main()
4  {
5      long toplam = 0; /* ilk değer mutlaka sıfırlanmalı*/
6      int ogranciSayisi = 0; /* İlk değer sıfırlanmalı */
7      int notu;
8      float ortalama;
9      printf("Notu giriniz (sonlandırmak için -1) :");
10     scanf("%d",&notu); /* ilk notu oku */
11     while ( notu != -1 ) { /* -1'den farklı olduğu sürece*/
12         toplam = toplam + notu; /* Notu toplama ekle */
13         ogranciSayisi = ogranciSayisi + 1;
14         /* öğrenci sayısını bir artır */
15         /* Diğer notu oku. Bu ifade tekrar
16         yazılmazsa başka not okunamaz */
17         printf("Notu giriniz (sonlandırmak için -1) :");
18         scanf("%d",&notu);
19     }
20     if(ogranciSayisi != 0){
21         /* İlk çalıştığı anda -1 girilirse 0'a bölme
22         hatası olmaması için kontrol */
23
24         ortalama = toplam / ogranciSayisi;
25         printf("\n\n Ortalama : %5.2f \n", ortalama);
26     }
27     system("PAUSE");
28     return 0;
29 }
30
```

```
C:\Dev-Cpp\Proje2.exe
Notu giriniz (sonlandırmak için -1) :40
Notu giriniz (sonlandırmak için -1) :20
Notu giriniz (sonlandırmak için -1) :15
Notu giriniz (sonlandırmak için -1) :78
Notu giriniz (sonlandırmak için -1) :31
Notu giriniz (sonlandırmak için -1) :-1

Ortalama : 36.00
Devam etmek için bir tuşa basın . . .
```

Burada kaç adet not girileceği belirtilmemiştir. Kullanıcı isterse 4 nottan sonra -1 , isterse de 100 nottan sonra -1 girebilir.

Bu amaçla girilen not değeri, gözcü değeri ile kontrol edilir.

Bu çözümde sınav notu olamayacak bir değer olan -1, kontrol değeri olarak seçilmiştir.

for döngüsü



- **for** döngüsü özellikle tekrar edilen işlemlerin sayısı belli olduğunda kullanılan başka bir döngü yapısıdır.
- Bu tür durumlarda **while** deyiminden daha uygundur. Genel kullanım yapısı aşağıdaki gibidir.

```
for ( DöngüDeğişkeni = ilkDeğer ; DöngüKoşulu ; AdımMiktari )  
    komut;
```

- Döngü içerisinde birden fazla komut tekrar edilecekse komutlar blok içerisine alınmalıdır.

```
for ( DöngüDeğişkeni = ilkDeğer ; DöngüKoşulu ; AdımMiktari ) {  
    komut1;  
    .....  
    komutN;  
}
```

```
int main()  
{  
    int i;  
  
    for (i=1;i<=10;i++) {  
        printf("kirami \n");  
    }  
  
    system("PAUSE");  
    return 0;  
}
```

do – while döngüsü



- **while** döngüsü yapısına benzer.
- **while** döngüsünde koşul en başta kontrol edilir ve koşul sağlanmazsa döngü içine hiç girilmez.
- **do-while** döngüsünde ise koşul en sonda kontrol edilir ve koşul sağlanmazsa bile en az bir kez döngü içine girilir.
- Genel kullanım şekli aşağıdaki gibidir.

```
do{  
    komut1;  
    .....  
    komutN;  
} while ( koşul );
```

İç-içe Döngüler (Nested Loops)



- Bir döngü içerisinde başka bir döngü bulunuyorsa, bu tür yapılara iç-içe döngüler denir.
- Bu durumda içteki döngü dıştaki döngünün her adımında yeniden çalıştırılacaktır.

```
int n, i, j;  
n=0;  
for (i=1;i<=4;i++)  
    for (j=1;j<=3;j++)  
        n = n + 1;
```

Dıştaki döngü 4 kere tekrarlanacaktır.

İçteki döngü de dıştaki döngünün her tekrar edilişinde 3 kere tekrar edilecektir.

Böylece içteki döngünün gövdesini oluşturan atama ifadesi 12 kere tekrar edilmiş olacaktır.

Dıştaki for döngüsünden çıkıldığında, n değişkeni içerisindeki değer 12 olacaktır.

break



- **break** Deyimi

- Bu komut daha önce switch – case yapısında kullanılmıştı.
- Döngü içinde kullanılan **break** komutunun amacı, çalıştığı anda döngüyü sonlandırarak döngü dışına çıkılmasını sağlamaktır.
- Programın işleyişi döngünün dışındaki ilk komuttan devam eder.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int i;
6
7      for (i=1;i<=5;i++) {
8          if(i==3) /* i'nin değeri 3 ise*/
9              break; /* döngü dışına çık */
10         printf("i = %d \n", i);
11     }
12
13     system("PAUSE");
14     return 0;
15 }
```

```
C:\Dev-Cpp\Proje2.exe
i = 1
i = 2
Devam etmek için bir tuşa basın . . .
```

continue



- **continue** Deyimi

- Döngü içinde kullanılan bu komutun amacı çalıştığı anda döngü içindeki geri kalan komutları işletmeden döngü başı yapılmasını sağlamaktır.
- Programın işleyişi döngü içindeki bu komuttan sonra verilen tüm ifadeler atlanarak döngü başına yönlendirilir.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int i;
6
7      for (i=0;i<5;i++) {
8          if(i==3) /* i'nin değeri 3 ise*/
9              continue; /* döngü başı yap */
10         printf("i = %d \n", i);
11     }
12
13     system("PAUSE");
14     return 0;
15 }
```

```
C:\Dev-Cpp\Proje2.exe
i = 0
i = 1
i = 2
i = 4
Devam etmek için bir tuşa basın . . .
```