# BMM 111
# Bilgisayar Programlama-I

## 6. Ders

*Dr. Öğr. Üyesi Mustafa İSTANBULLU*

Çukurova Üniversitesi
Mühendislik Fakültesi
Biyomedikal Müh. Böl.

E-mail:mm.istanbullu@gmail.com

Not: Slaytlar, kaynakça bölümünde verilen listeden faydalanılarak hazırlanmıştır.

# 5. Bölüm Döngü Komutları

## 5.1. Giriş

- Önceki bölümlerde C programlama dilinin girdi, çıktı, atama ve seçme komutlarını görmüştük.
- Bu öğrendiklerimizle ancak belirli tür problemlerin programlarını yazabiliriz.
- Daha önce, bir sınıftaki 50 öğrencinin notunun alınarak not ortalamasının hesaplanması problemini ele almıştık. Bu durumda programın, 50 kez çalıştırılmasının gerek olduğunu gördük. Bu işlemi daha kolay bir hale getirmek için döngü komutlarını kullanmamız gerekir.
- Döngü komutları (repetition/loop statements), komutların birçok kez yeniden yürütülmesini sağlayan, programlamada kullandığımız önemli yapılardan biridir.

- C dilinde bize bu özelliği sağlayan komutlar **`while`**, **`for`** ve **`do-while`** komutlarıdır.

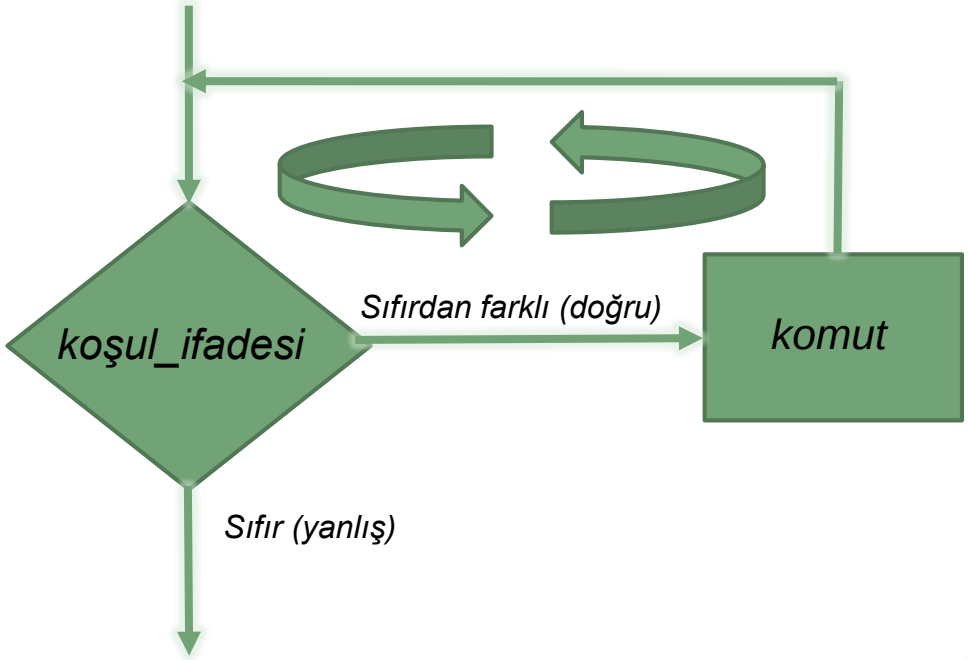
## 5.2. `while` Komutu

- **`while`** komutu genel bir döngü komutu olup bir çok programlama dilinde de benzeri bulunmaktadır.
- **`while`** komutunun C programlama dilindeki genel yapısı aşağıdaki gibidir:

```
while ( koşul ifadesi )
    komut;
```

- Burada *`koşul ifadesi`* doğru ise *`komut`* yürütülür.
- *`komut`*'un yürütümü `if` deyimindeki gibi bir kez olmayıp, koşul ifadesi doğru olduğu sürece devam eder.
- *`koşul ifadesi`* yanlış olduğu durumda yürütüm, **`while`** komutunu takip eden komutla devam eder.

- **`while`** komutunun akış şeması aşağıda verilmiştir:



- Örneğin, kullanıcının pozitif bir sayı girmesini sağlayan program parçasının,

```
printf("Bir pozitif sayi giriniz:");
scanf("%d",&n);
```

- şeklinde yazılması yeterli değildir. Çünkü kullanıcı negatif bir sayı girdiğinde `n` değeri negatif olur ve program bu değerle yürütülür.
- Kullanıcının pozitif bir sayıyı doğru bir şekilde girmesini sağlayan bir program, ancak döngü komutlarıyla yazılabilir. Bu problemin `while` komutu ile çözümü aşağıda verilmiştir.

```
printf("Bir pozitif sayi giriniz:");
scanf("%d",&n);
while(n<0)
        scanf("%d",&n);
printf("En son n degeri: %d",n);
```

**ÖNEMLİ UYARI !**

- Az önce verdiğimiz `while` genel yapısında yer alan komut herhangi bir C komutu olabilir. Bu örnekte komut olarak `scanf ()` fonksiyonu yer almaktadır.
- Birden fazla komutun döngü içinde yürütülmesi istendiğinde bileşik komut kullanılmalıdır. Örneğin;

```
printf("Bir pozitif sayi giriniz:");
scanf("%d",&n);
while(n<0)
{
        printf("Bir pozitif sayi giriniz:");
        scanf("%d",&n);
}
printf("En son n degeri: %d",n);
```

Bu örnekte kutu içindeki alan döngünün gövdesi olup bir birleşik komut içerir. `n<0` koşulu doğru olduğu sürece, döngü gövdesindeki komutlar sırasıyla tekrarlanacaktır.

- Daha öncede bahsedildiği gibi **while** komutu önemli bir döngü komutudur.
- Bu durumu göstermek amacıyla aşağıdaki program parçası ele alınsın.

```
sayac=1;
while(sayac<=10)
{
        printf("%3d",sayac);
        sayac=sayac+1;
}
```

Bu örnekte kutu içindeki alan döngünün gövdesi olup bir birleşik komut içerir. **n<0** koşulu doğru olduğu sürece, döngü gövdesindeki komutlar sırasıyla tekrarlanacaktır.

## Örnek

10 reel sayıyı girdi olarak alan ve pozitif sayıların toplamını bulan C kodunu yazınız.

Örnek

```
#include <stdio.h>
int main(void)
{
        int i;
        float sayi,toplam;
        i=1;
        toplam=0.0;
        while(i<=10)
        {
           /*reel sayinin girilmesi*/
           printf("%d.sayiyi giriniz:",i);
           scanf("%f",&sayi);
           /*reel sayinin pozitif olup olmadiginin kontrolu*/
                if(sayi>0)
                        toplam=toplam+sayi;
                i=i+1;
        } /*while komutunun sonu*/
        printf("Pozitif sayilarin toplami:%5.2f",toplam);
        system("pause");
        return(0);
}
```

## 5.2.1 `while` Komutunun Aşamaları

- **`while`** komutunu anlatırken vermiş olduğumuz 1'den 10'a kadar tamsayıları görüntüleyen örneği tekrar hatırlayalım.

```
sayac=1;
while(sayac<=10)
{
        printf("%3d",sayac);
        sayac=sayac+1;
}
```

- Bu örnekte döngümüzün kaç kere tekrar edileceğini, **`sayac`** değişkeni kontrol etmektedir. Bu bağlamda **`sayac`** değişkeni döngü değişkenidir.

- Döngü değişkeni bulunan her `while` komutunun kullanımında, aşağıda verilen aşamalar belirtilen sırayla yer almalıdır.

  - İlk değer atama aşaması (initialization step)
  - Kontrol aşaması (control step)
  - Güncelleme aşaması (updating step)

- Bu aşamalar yukarıda gördüğümüz örnek üzerinde aşağıdaki gibi gösterilebilir.

```
sayac=1;                          → ilk değer atama aşaması
while(sayac<=10)                  → Kontrol aşaması
{
    printf("%3d",sayac);
    sayac=sayac+1;                → Güncelleme aşaması
}
```

## İlk Değer Atama Aşaması

- İlk değer atama aşaması her zaman **while** döngüsüne girmeden önce yapılır.
- Bu örnekte ilk değer atama aşamamız yukarıda da görüleceği gibi **sayac=1**; dir. **sayac** değişkeninin farklı ilk değer atamaları için, farklı çıktılar elde edilecektir. Bu farklılıklar aşağıda özetlenmiştir:

```
sayac=6;
while(sayac<=10)
{
        printf("%3d",sayac);
        sayac=sayac+1;
}
```

ÇIKTI:

```
 6  7  8  9 10
```

```
sayac=10;
while(sayac<=10)
{
        printf("%3d",sayac);
        sayac=sayac+1;
}
```

**ÇIKTI:** 10

```
sayac=15;
while(sayac<=10)
{
        printf("%3d",sayac);
        sayac=sayac+1;
}
```

**ÇIKTI:** Çıktıda hiçbir şey görülmez…

```
scanf("%d",&sayac);
while(sayac<=10)
{
        printf("%3d",sayac);
        sayac=sayac+1;
}
```

**ÇIKTI:**

**Bu program parçasında ilk değer atama aşaması scanf("%d",&sayac); dır.**

**Dolayısıyla çıktı, klavyeden girilecek sayac değerine göre değişir.**

```
while(sayac<=10)
{
        printf("%3d",sayac);
        sayac=sayac+1;
}
```

**ÇIKTI:**

**Burada sayac değişkenine atama yapılmadığından bu değişkenin değeri belli değildir. Dolayısıyla, derleyiciye bağlı olarak değişik ve beklenmeyen sonuçlar elde edilebilir.**

**İstenmeyen sonuçlar elde etmemek için mutlaka ilk değer ataması bulunmalıdır.**

## Kontrol Aşaması

- Kontrol aşaması ise, **while**'ı takip eden parantez içinde belirlenir ve döngünün tekrarlanmasını kontrol eder.
- Yani, **'sayac<=10'** ifadesi ile **sayac** değişkeninin değeri kontrol edilmektedir.
- Bu koşul, her döngünün başında kontrol edilir; eğer doğru ise döngü komutları yürütülür, değilse döngüye son verilip sonraki komutla yürütüme devam edilir. Farklı örneklere göz atılırsa:

```
sayac=1;
while(sayac<10)
{
        printf("%3d",sayac);
        sayac=sayac+1;
}
```

**ÇIKTI:** 1 2 3 4 5 6 7 8 9

```
sayac=1;
while(sayac<=3)
{
        printf("%3d",sayac);
        sayac=sayac+1;
}
```

**ÇIKTI:** 1 2 3

```
sayac=1;
while(sayac>=10)
{
        printf("%3d",sayac);
        sayac=sayac+1;
}
```

**ÇIKTI:** Çıktıda hiçbir şey görülmez…

## Güncelleme Aşaması

- Son aşama olan güncelleme aşaması ise döngü değişkeninin değerinin değişmesini; dolayısıyla koşul ifadesinin değerinin değişmesini sağlar ve her zaman döngü gövdesi içinde yer alır.
- `'sayac=sayac+1;'` komutu yukarıda görüldüğü gibi döngü gövdesinde bulunmaktadır. Farklı örneklere göz atılırsa:

```
sayac=1;
while(sayac<=10)
{
        printf("%3d",sayac);
        sayac=sayac+3;
}
```

**ÇIKTI:** **1 4 7 10**

```
sayac=1;
while(sayac<=10)
{
        printf("%3d",sayac);
        sayac=sayac-1;
}
```

**ÇIKTI**

**Yürütüm sonsuz bir döngüye girer. Çıktı ise aşağıdaki gibidir.**

```
1  0  -1  -2  -3  -4  -5  .  .  .
```

```
sayac=1;
while(sayac<=10)
      printf("%3d",sayac);
```

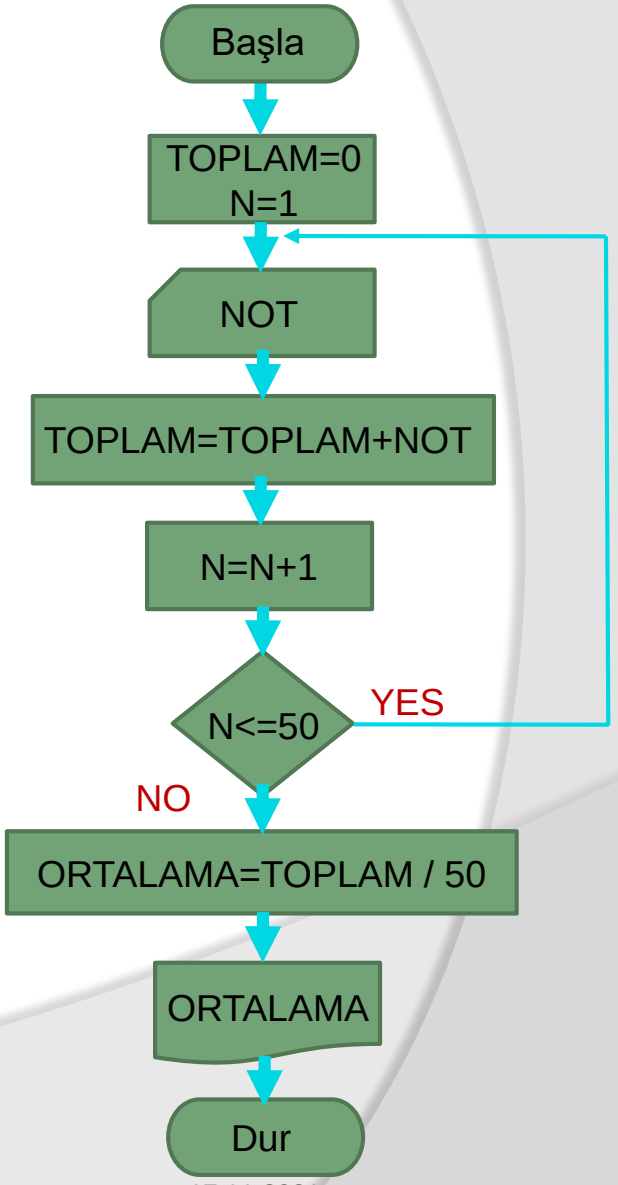
**ÇIKTI**

**(sayac<=10) koşul ifadesi hep doğru olacağından sonsuz bir döngü içine girilir. Çıktı ise aşağıdaki gibidir.**

```
1  1  1  1  1  1  1  .  .  .
```

## 5.2.1 Sonlandırma Değer Kullanımı

- Bir sınıfta 50 öğrencisi bulunan ve sınavı 100 puan üzerinden notlayan bir öğretmen, bu sınavın ortalamasını hesaplamak istemektedir. Bu sınavın ortalamasını hesaplayacak olan programa ait akış şemasını daha önce öğrenmiştik:



- Bu problem için gerekli C kodunu yazacak olursak:

```
# include <stdio.h>
int main(void)
{   int sayac,not;
    double toplam;
           sayac=1;
           toplam=0.0;
           while(sayac<=50)
    {
           printf("Ogrenci notu giriniz:");
           scanf("%d",&not);
           toplam=toplam+not;
           sayac=sayac+1;
    }
    printf("Sinav Ortalamasi:%5.2lf",toplam/50);
system("PAUSE");
return(0);
}
```

## Örnek

Pozitif sayıların girildiği ve girilen bu pozitif sayıların en büyüğünün bulunduğu C programını yazınız.
Programda sayı girme işlemi negatif bir sayı girilinceye kadar devam etmelidir.

Çözüm

```
#include <stdio.h>
int main(void)
{
    int sayi,max=0;
    /*En büyük sayının sıfır olduğunu kabul edelim..*/
    printf("Sayi giriniz: ");
    scanf("%d",&sayi);
    while(sayi>0)
    {
       /*Eğer girilen sayı max da bulunan büyük sayıdan daha
       büyükse yeni girilen sayı max değişkenine atanır..*/
       if(sayi>max)
           max=sayi;
       printf("Sayi giriniz:");
       scanf("%d",&sayi);
}
    printf("En buyuk pozitif sayi:  %d\n\n",max);
    system("pause");
    return(0);
}
```

## 5.3. Diğer Operatörler

- Döngü işlemlerini gerçekleştirebilmek amacıyla, **`while`** dışında diğer döngü komutlarını da kullanabiliriz.
- Bu komutlara geçmeden önce C programlarında sıklıkla kullanılan diğer operatörlerden bileşik atama ile artırma/azaltma operatörlerini inceleyelim.

### 5.3.1 Bileşik Atama Operatörü

- Programcıların en çok kullandığı işlemlerden biri de atama işlemidir. Birçok programı incelediğimizde atama komutlarının bir kısmının aşağıdaki gibi olduğunu görürüz.

```
toplam=toplam+sayi;
carp=carp*k;
i=i+1;
j=j-1;
```

- Bu atama komutlarının ortak özelliği yapılarının aşağıdaki gibi olmasıdır:

```
değişken = değişken operatör ifade;
```

- Bu yapıda olan atama komutlarını

```
değişken operatör = ifade;
```

- şeklinde de yazabiliriz. Bu iki atama şekli de aynı işlevi görmektedir. Örneğin;

```
toplam=toplam + sayi;
toplam += sayi;
```

- Komutları aynı işlevi görür. Burada bulunan `'+='` ifadesi bir bileşik atama operatörüdür.

- Atama komutlarından bileşik operatörlerin kullanılmasıyla ilgili diğer örnekler aşağıda verilmiştir:

| Atama Komutu | Bileşik Operatör ile Yazımı |
|---|---|
| `carp=carp*k;` | `carp *=k;` |
| `i=i+1;` | `i+=1;` |
| `j=j-1;` | `j-=1;` |
| `h=h-4/st;` | `h-=4/st;` |
| `alfa=4*(k-5)/(k+1)*alfa` | `alfa*=4*(k-5)/(k+1)` |

## 5.4. `for` Komutu

- **`for`** komutu, diğer bir döngü komutudur ve **`while`** komutu ile aynı işlevi görür.
- Ancak **`for`** komutunun genel yapısı **`while`** komutundan farklıdır.
- **`for`** komutu, döngünün kaç kere yürütüleceğinin bilindiği durumlarda tercih edilir.
- Bir önceki bölümde, **`while`** komutunu kullanırken ilk değer atama, kontrol ve güncelleme aşamalarının yer aldığından bahsetmiştik.
- Bütün bu aşamalar **`while`** komutunda aşağıdaki gibi yer almaktaydı.

```
ilk değer atama komutu;
while (kontrol ifadesi)
{
        döngü komutları;
        güncelleme komutu;
}
```

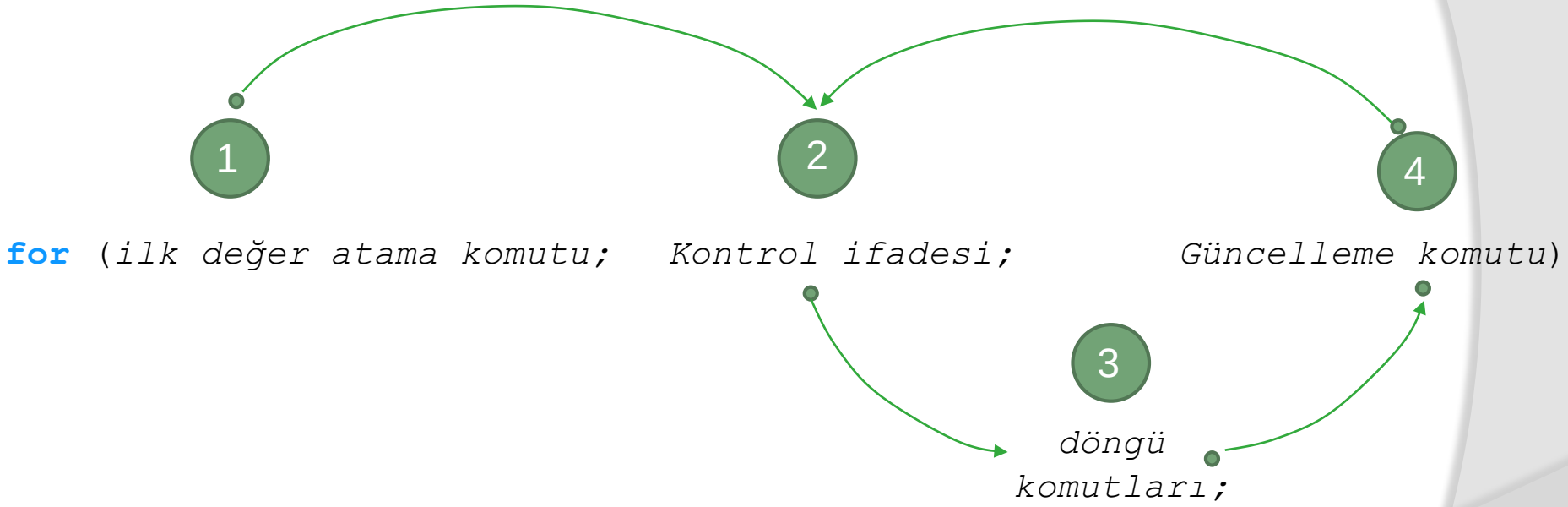
# 5. Bölüm Döngü Komutları

- Az önce belirtilen aşamalar **for** komutunda da yer almakta olup aynı zamanda **for** komutunun genel yapısı aşağıda verilmiştir.

```
for (ilk değer atama komutu; kontrol ifadesi; güncelleme komutu)
    döngü komutları;
```

- Bu yapıda özel amaçlı sözcük olan **for**'u parantez içinde '**;**' ile ayrılmış üç ayrı bölüm takip etmektedir. Bu üç bölüm **for** döngüsü içinde mutlaka bulunmalıdır.
- ***döngü komutları*** ise döngünün gövdesini oluşturmaktadır. Bu döngü gövdesinde, birden fazla komutun yer alması gerektiğinde, bileşik komut kullanılır.
- **for** komutunda ilk önce ***ilk değer atama komutu*** yürütülür. Bu komuttan sonra ikinci bölümde yer alan ***kontrol ifadesi***' ne geçilir.
- Bu ifade doğru ise döngü gövdesindeki ***döngü komutları*** yürütülür. Böylece döngünün birinci adımı tamamlanır.
- Döngünün ikinci adımına geçilebilmesi için ilk önce üçüncü bölümdeki ***güncelleme komutu*** ve arkasından ikinci bölümde yer alan ***kontrol ifadesi*** tekrar yürütülmelidir. İfademiz doğru ise ***döngü komutları*** yine yürütülür.
- Bu sıralı yürütüm işlemi ***kontrol ifadesi***'nin değeri yanlış (yani sıfır) oluncaya kadar devam eder.

- **for** komutunda yer alan komutların işleniş sırası aşağıdaki verilmiştir:




- Burada 1, 2, 3 ve 4 numaralı komutlar sırasıyla yürütüldükten sonra döngü tekrar 2, 3 ve 4 olarak devam eder. Döngü ***kontrol ifadesi*** yanlış olunca sonlanır.

- Daha önce gördüğümüz, 1'den 10'a kadar olan sayıları bir **while** döngüsü ile ekranda görüntüleyen örneğin aşamalarını tekrar hatırlayalım:

```
sayac=1;
while(sayac<=10)
{
        printf("%3d",sayac);
        sayac=sayac+1;
}
```

- Bu örneğin **for** komutu ile yeniden hazırlanmış hali aşağıda verilmiştir.

```
for (sayac=1; sayac<=10; sayac++)
        printf("%3d",sayac);
```

## Örnek

- Daha önce **while** komutunu kullanarak yazdığımız 50 kişilik bir sınıfın sınav not ortalamasını hesaplayan program parçasını, **for** komutunu kullanarak tekrar yazalım.

```
toplam=0.0;
for (sayac=1; sayac<=50; sayac++)
{
      printf("Ogrenci notu giriniz:");
      scanf("%d",&not);
      toplam+=not;
}
printf("Sinav ortalamasi:%5.2",toplam/50);
```

## Örnek

- Pozitif bir `n` sayısını ve bu `n` sayısı kadar da tamsayıyı girdi olarak alan programı yazınız. Ayrıca, bu programda girilmiş olan `n` tane tamsayının kaçının negatif, kaçının pozitif ve kaçının sıfır olduğunu hesaplayıp ekranda gösteriniz.

```
#include <stdio.h>
int main(void)
{   int pos=0,neg=0,sifir=0,n,sayi,i;
    printf("Kac tane sayi gireceksiniz:");
    scanf("%d",&n);
    /*Döngü içinde n tane sayının girilmesi ve işlenmesi*/
    for(i=1;i<=n;i=i+1)
{    printf("%d. sayi: ",i);
     scanf("%d",&sayi);
     /* Girilen sayının pozitif, negatif veya sıfır
     olup olmadığının belirlenmesi*/
     if(sayi>0)
             pos++;
     else if(sayi<0)
             neg++;
     else sifir++;              }
/*Sonuçların gösterilmesi*/
printf("\n%d adet pozitif sayi.\n",pos);
printf("%d adet negatif sayi.\n",neg);
printf("%d adet sifir sayisi.\n",sifir);
system("pause");
return(0);
}
```

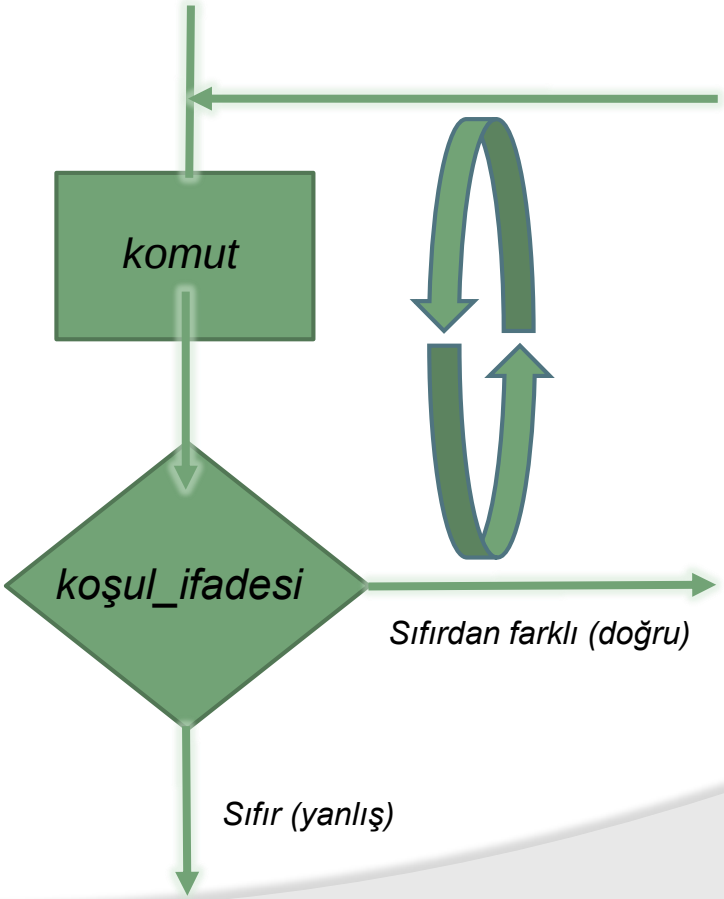
## 5.5. `do-while` Komutu

- Şimdiye kadar döngü komutlarından **`while`** ve **`for`** komutlarını inceledik.
- Bu bölümde ise bir başka döngü komutu olan **`do-while`** komutu incelenecektir.
- **`while`** ve **`for`** komutlarında koşul ifadesi döngünün başında kontrol edilirken, **`do-while`** komutunda kontrol işlemi döngünün sonunda yapılır.
- **`do-while`** komutunun genel yapısı aşağıdaki gibidir:

```
do
        komut;

while(koşul ifadesi);
```

- Diğer komutlarda olduğu gibi döngü gövdesinde yer alan komut tek bir komut olabileceği gibi, birçok komutu içeren bileşik bir komut da olabilir.



- Örneğin, aşağıdaki program parçası yine **1**'den **10**'a kadar olan sayıları yan yana ekranda görüntüleyecektir.

```
sayac=1;
do
{
        printf("%3d",sayac);
        ++sayac;
}
while(sayac<=10);
```

Yukarıdan görüleceği gibi koşul ifadesi yanlış olsa da döngü en az bir kez yürütülecektir.

Yukarıdaki döngüdeki yürütüm işlemi, sayac değişken değerinin 11 olması ile sonlanır.

## Örnek

- Aşağıdaki seçenekleri ekranda görüntüleyecek ve seçeneklere göre aşağıda tanımlanan işleri yapacak bir C programı yazınız.

```
1. Kare
2. Daire
3. Bitti
Seceneğinizi belirtiniz (1,2,3):
```

- `1.` seçenek seçildiğinde karenin alanı; `2.` seçenek seçildiğinde dairenin alanı hesaplanacaktır.
- Karenin kenar uzunluğu ve dairenin yarı çapı kullanıcı tarafından girilecektir.
- İlgili şekillerin alan hesaplama işlemleri kullanıcının `3.` seçeneği seçmesiyle sonlanacaktır.

```
#include <stdio.h>
#define PI 3.141592654
int main(void)
{    int cevap;
     double a,r,alan;
     do
     {        /*MENÜNÜN GÖRÜNTÜLENMESİ*/
              printf("\n 1. Kare");
              printf("\n 2. Daire");
              printf("\n 3. Bitti");
              printf("\n\nSeceneginizi belirtiniz:");
              scanf("%d",&cevap);
              switch(cevap)
              {case 1:/*Karenin alanının hesaplanması*/
                       printf("\nKarenin kenar uzunlugunu giriniz:");
                       scanf("%lf",&a);
                       alan=a*a;
                       printf("Karenin alani:%.2f\n",alan);
                       break;
              case 2:/*Dairenin alanının hesaplanması*/
                       printf("\nDairenin yaricap uzunlugunu giriniz:");
              scanf("%lf",&r);
              alan=PI*r*r;
              printf("Dairenin alani:%.2f\n",alan);      }
     }
     while(cevap!=3);
     return(0);}
```

## 5.6. İçiçe Döngüler

- `If` komutları nasıl içiçe olabiliyorsa, döngü komutları da içiçe olabilirler.
- İçiçe döngü (nested loop) olabilmesi için döngü komutunun gövdesinde bir veya daha fazla döngü komutu bulunmalıdır.
- İçiçe döngülerde, dışta yer alan döngünün her tekrarında içteki döngünün tamamen yürütülmesi söz konusudur. Örneğin;

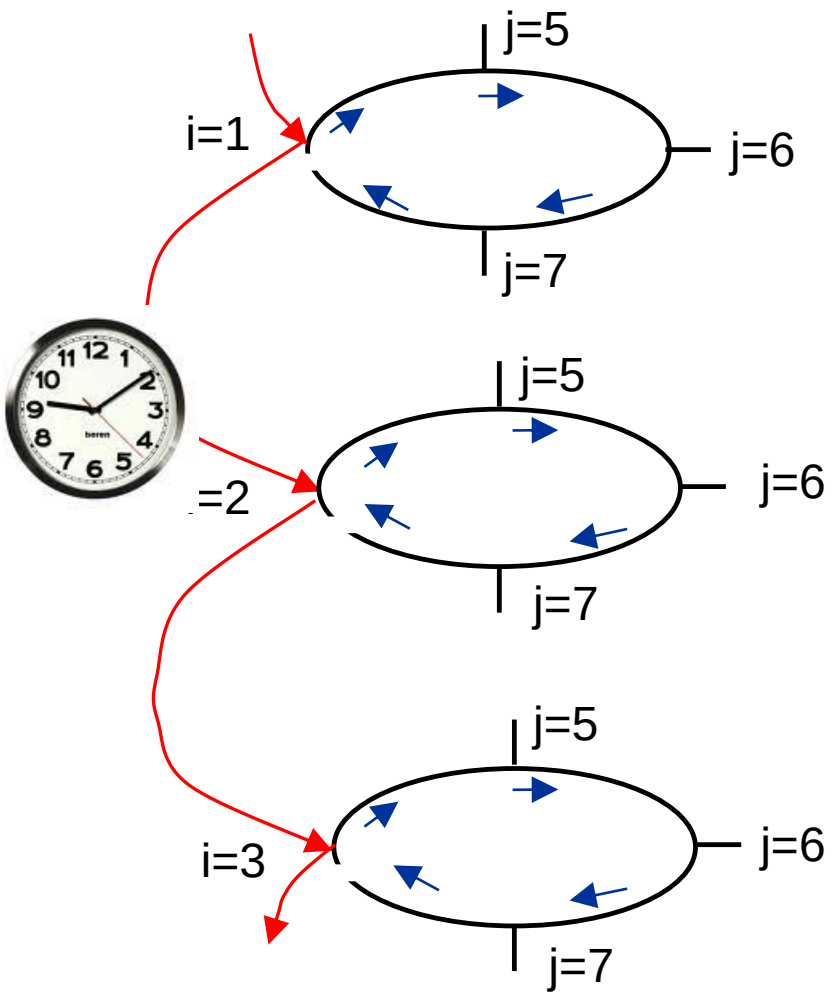
```
for (i=1; i<=3; ++i)
        for (j=5;j<=7;++j)
                printf("%3d%3d\n",i,j);
```

- Dış kutudaki `for` döngüsünün döngü değişkeni `i` iken, iç kutudaki `for` döngüsünün döngü değişkeni `j`'dir.

```
for (i=1; i<=3; ++i)
      for (j=5;j<=7;++j)
            printf("%3d%3d\n",i,j);
```

Program Çıktısı:

```
  1    5
  1    6
  1    7
  2    5
  2    6
  2    7
  3    5
  3    6
  3    7
```




İçiçe **for** döngüsünün grafiksel gösterimi

- Görüldüğü gibi, içiçe döngülerin işleyişi tıpkı bir saatteki akrep ile yelkovanın işleyişine benzemektedir.
- Akrep bir saat ilerlediğinde, yelkovan bir turunu tamamlamaktadır.
- Bu durumda içteki döngünün işleyişini yelkovana; dıştaki döngünün işleyişini ise akrepe benzetebiliriz.
- Dıştaki döngünün her tekrarında içteki döngü tümüyle döndürülür.

## 5.7. `break` ve `continue` Komutları

- Bir önceki bölümde **`break`** komutunun **`switch`** komutunda kullanımını öğrendik.
- **`switch`** içinde **`break`** komutuna gelindiğinde yürütüm, **`switch`** komutunu takip eden komutla devam ediyordu. Yani, bulunulan **`switch`** komutundan çıkılıyordu.
- Benzer durum döngü komutları için de geçerlidir.
- **`break`** komutu herhangi bir döngü komutu içinde kullanıldığında döngüden çıkılmasını sağlayacaktır.
- Özetle **`break`** komutu, "*bulunduğun bloktan çık ve bir sonraki komutla yürütmeye devam et*" anlamını taşımaktadır.

- **break** komutunun **for** döngüsünde kullanımıyla ilgili aşağıdaki örneği inceleyelim:

```
for(i=1; i<=10; i++)
{
        printf("Sayi giriniz:");
        scanf("%d",&sayi);
        if(sayi<=0)
           break;
}
printf("%d pozitif sayi girildi.", i-1);
```

- Bu program, en fazla 10 pozitif sayının girilmesine izin vermektedir.

- Döngünün herhangi bir turunda, kullanıcı tarafından girilen sayı sıfır veya sıfırdan küçük ise döngüden çıkılır ve bir sonraki komut ile yürütüm devam eder.

- **continue** komutu ise sadece döngü komutlarında kullanılmaktadır.
- Döngünün herhangi bir aşamasında **continue** komutuna gelindiğinde, o turun yürütümü o noktada kesilir ve yürütüm bir sonraki tur ile devam eder. Yani, **for** komutunda yürütüm güncelleme bölümüyle; **while** ve **do-while** komutunda ise kontrol bölümüyle devam eder.
- **continue** komutunun **for** döngüsünde kullanımıyla ilgili örnek aşağıda verilmektedir:

```
toplam=0;
for(i=1; i<=5; i++)
{
        printf("Sayi giriniz:");
        scanf("%d",&sayi);
        if(sayi<=0)
                {i--;
                 continue;}
        toplam+=sayi;
}
printf("Toplam: %d",toplam);
```

# KAYNAKÇA


- Prof.Dr. İbrahim DEVELİ, Bilgisayar Programlama Ders Notları, Erciyes Üniv. Elektrik-Elektronik Müh. Böl.
- H.Turgut UYAR, Programlamaya Giriş Ders Notları,İTÜ, 2004.
- Fedon KADİFELİ,Standart C Programlama Dili, (Tercüme),2000.
- Doç. Dr. Soner ÇELİKKOL, Programlamaya Giriş ve Algoritmalar, Murathan Yayınevi, TRABZON; 2009
- N. Ercil Çağıltay ve ark., C DERSİ PROGRAMLAMAYA GİRİŞ, Ada Matbaacılık, ANKARA; 2009.
- Çeşitli kişilerin internette paylaşıma açtığı notlardan faydalanılmıştır.