



# **YAPISAL PROGRAMLAMA**



# YAPISAL PROGRAMLAMA KAVRAMI

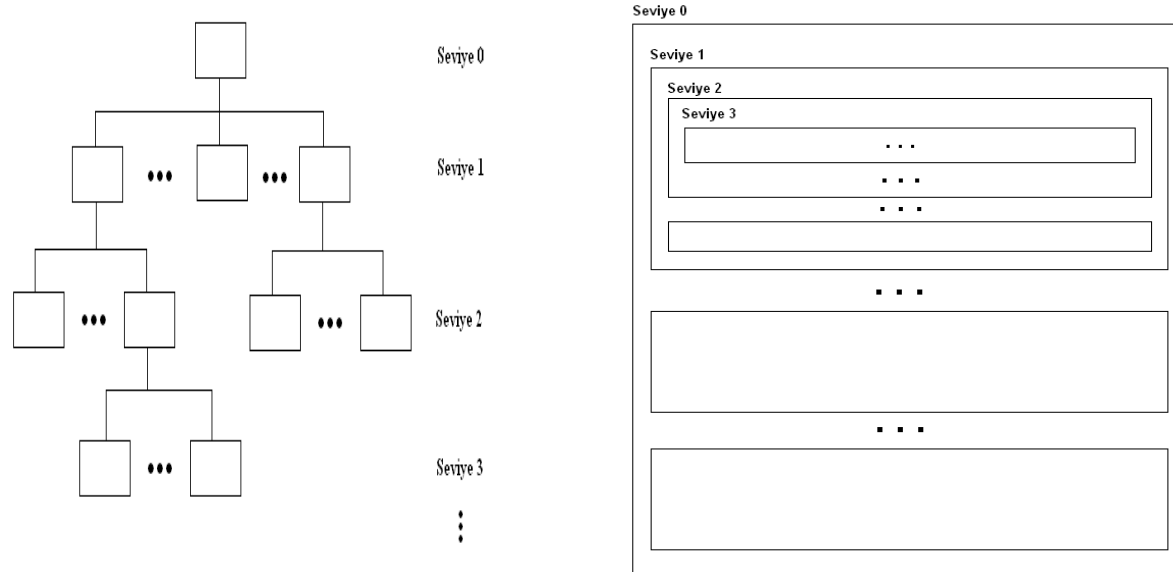
---

- Yapısal programlama, program tasarımı ve yazılmasını kurallara bağlayan ve disiplin altına alan bir yaklaşımdır.
- Yapısal programlamada problem çözümü daha kolay alt problemlere (modül) bölünür. Her bir alt problem (modül) daha düşük seviyedeki alt seviyelere bölünür.



# YAPISAL PROGRAMLAMA KAVRAMI

- Bu işlem, aşağıdaki şekilde de görülebileceği gibi her bir modülün kolaylıkla çözülebileceği seviyeye kadar devam eder.



- En üst seviyede çözümün ana mantığının sergilendiği ana modül yer alır. Alt seviyelere indikçe izlenecek adımlar ile ilgili ayrıntılar artar.



# YAPISAL PROGRAMLAMA KAVRAMI

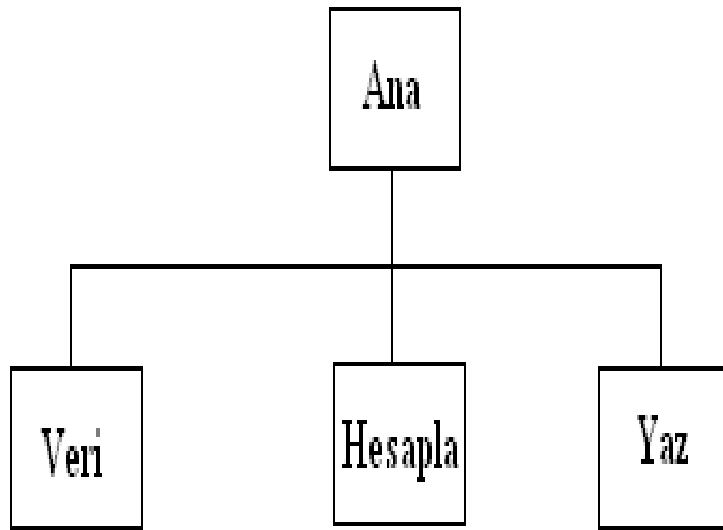
---

- Modüler program tasarımında her modül diğerlerinden bağımsız olmalıdır. Kontrol her modüle bir üst seviyedeki modülden geçmeli ve modül işlendikten sonra tekrar aynı modüle iletilmelidir.
- Modüllerin tanımlanmasında, (algoritma parçası olduğu için) sözde kod (pseudo-code) veya akış diyagramı kullanılır.
- Bir modülün çözümünde kullanılacak algoritma, sözde kod ile veya kullanılacak programlama dilinin yapısına uygun bir şekilde akış diyagramı ile ifade edilirse programa geçiş büyük ölçüde kolaylaşır.



# YAPISAL PROGRAMLAMA KAVRAMI

- Örnek 1;
- *Problem:* 1'den n' ye kadar olan tam sayıların toplamını bulmak.
- Problem çok kolay olmasına rağmen modüler programlamaya bir örnek olması açısından aşağıdaki şekilde bir tasarım düşünelim;



Ana

Veri

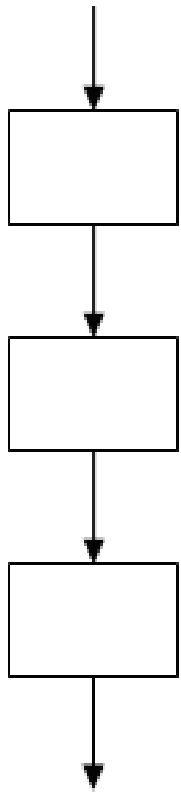
Hesapla

Yaz

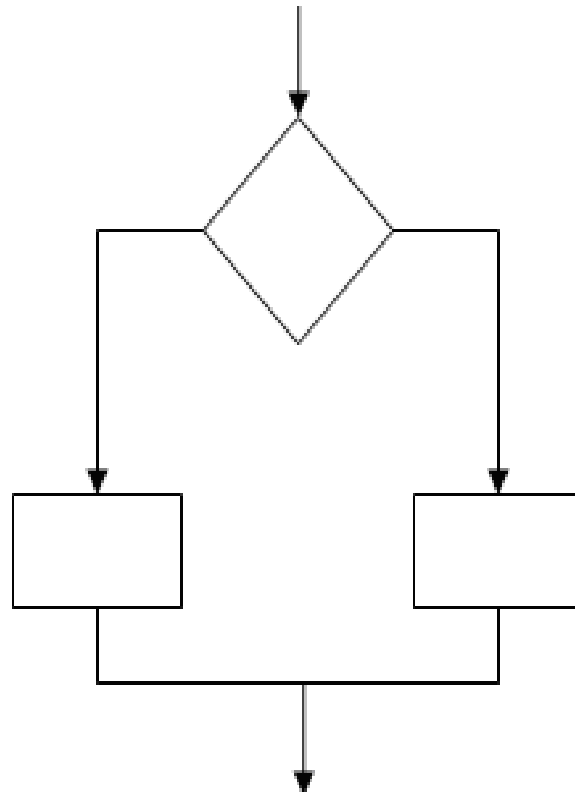


# Yapısal Programlama (Devam)

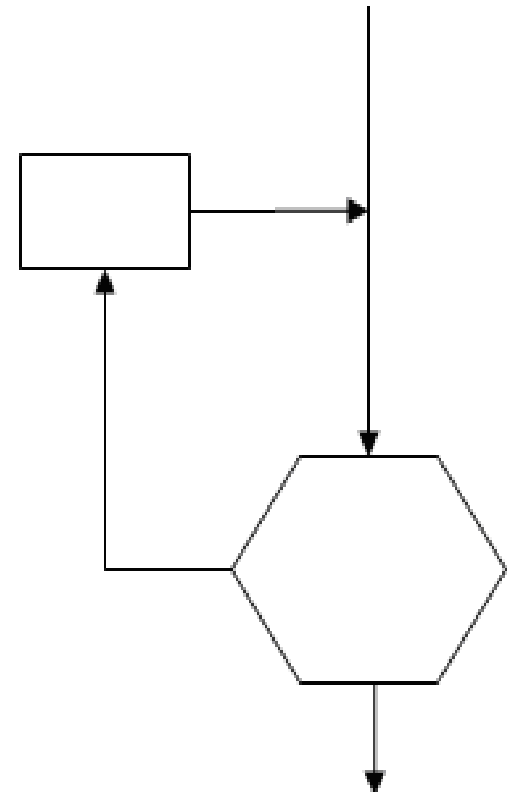
- Yapısal programlama, programlardaki akış denetimini aşağıdaki üç temel yapı ile sağlamaktadır.
  - Sıralı yapı
    - Sorgu (Seçimli) yapı
    - Tekrar (Yinelemeli) yapı



Sıra



Sorgu



Tekrar



# Sıralı Yapı-Sorgu-Yinelemeli

- Bir programda yer alan iki veya daha fazla program deyimi göründükleri sırada çalıştırılır.
- Programdaki **iki veya daha fazla yol arasından biri seçilir.**
- Programdaki herhangi bir komutun **istenilen sayıda** çalıştırılmasını gerçekleştirmek mümkündür.



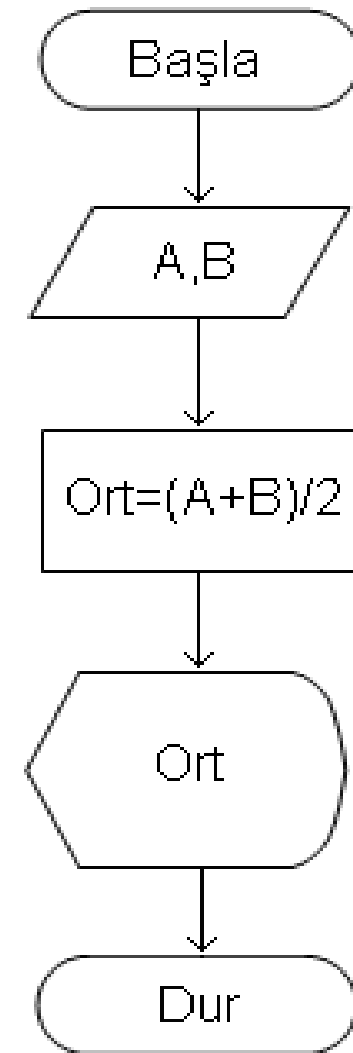


- Yapısal programlama, programlarda koşulsuz olarak akışı değiştiren *goto* gibi deyimlere yer vermez.
- Yapısal programlama, günümüzde yararlarını kanıtlamış bir programlama tekniğidir.



## Sıra yapısı

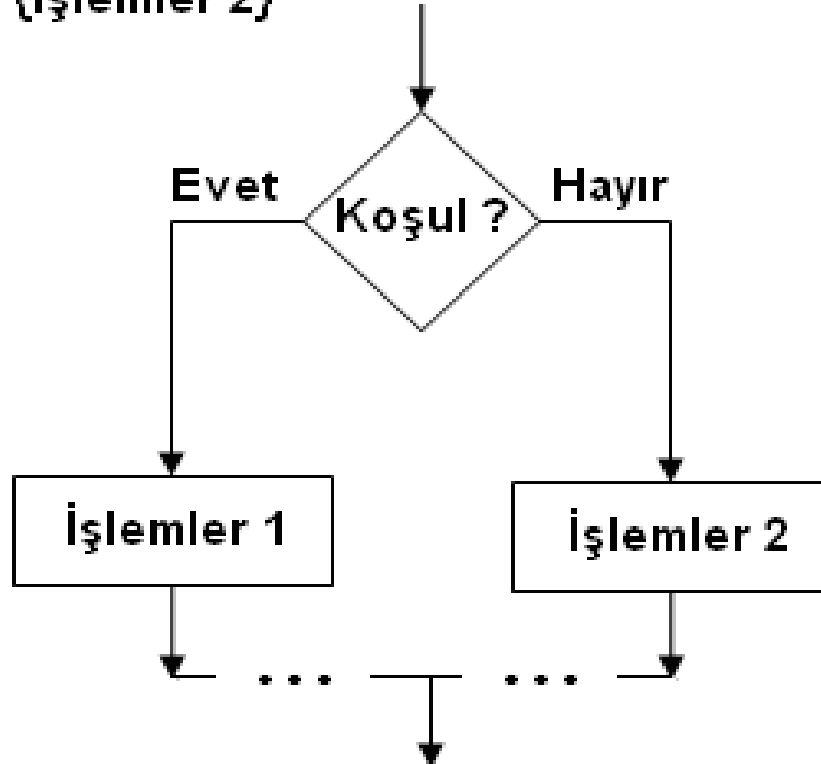
- Örneğin;
- Klavyeden girilen iki sayıyı okuyup aritmetik ortalamasını hesaplayan ve sonucu ekrana yazan bir programın akışı yandaki şekilde ifade edilebilir;



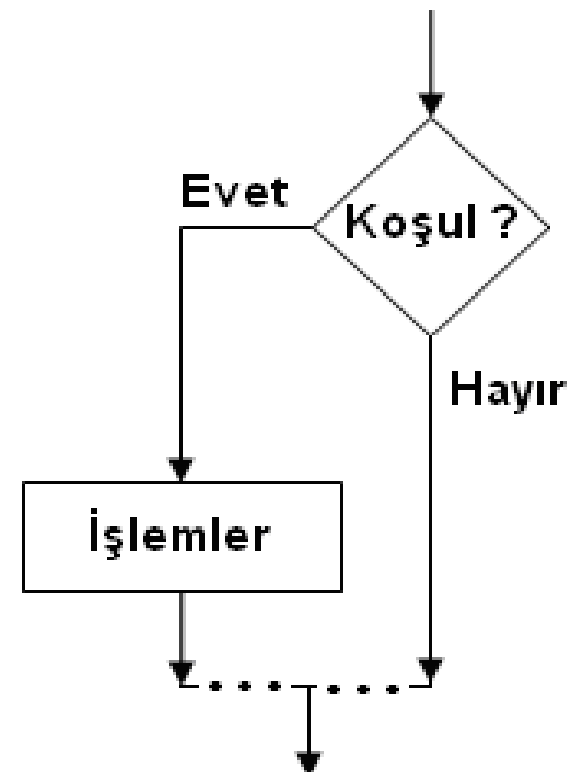


# Seçim Yapıları

```
if(koşul)  
  {işlemler 1}  
else  
  {İşlemler 2}
```



```
if(koşul)  
  {işlemler}
```





# İççe *if* ve *sallanan-else* (*dangling else*) problemi

```
if (a>b) then  
    if ( b>c)  
        then sonuc := 0  
else sonuc:=1
```



# Birleşik Deyimler:

- Her *else* deyimi kendisine en yakın eşleşmemiş *then* deyimi ile eşleştirilmelidir.
- *İçiçe if* deyimlerinde sallanan *else* probleminin çözümü için ikinci *if-then* yapısı birleşik deyim yapılmalıdır.
- Birleşik deyimler, bir dizi deyimin tek bir deyime soyutlanmasını sağlarlar.
- Pascal'da *begin ... end* yapısı, C'de ise *{ ... }* yapısı kullanılmaktadır.



```
→ if (a>b){  
    if (c>d)  
        printf(...);  
}  
→ else  
    scanf(...);
```



# ***If* Deyimlerinin Sonlandırılması**

- C ve Pascal'da *if* deyimi sözdizimi, *then* veya *else*'den sonra tek bir deyim yer almasını, daha çok deyim bulunması durumunda ise birleşik deyim oluşturulmasını gerektirir.
- Bu sözdizimdeki eksiklik, örneğin C'de birleşik deyimlerin kapanışı için kullanılan `"}`" unutulsa bile, derleme sırasında sadece eksik `"}`" uyarısı verilmesidir.



*if- then- else* yapısının sonunu  
belirten özel bir kelime

**if** .....then

deyim1

**else**

deyim2

**end if**





# Kısa devre ve tam değerlendirme

- If  $(x > 5)$  AND  $(y > 20)$  then .....
- If  $(x > 5)$  OR  $(y > 20)$  then .....

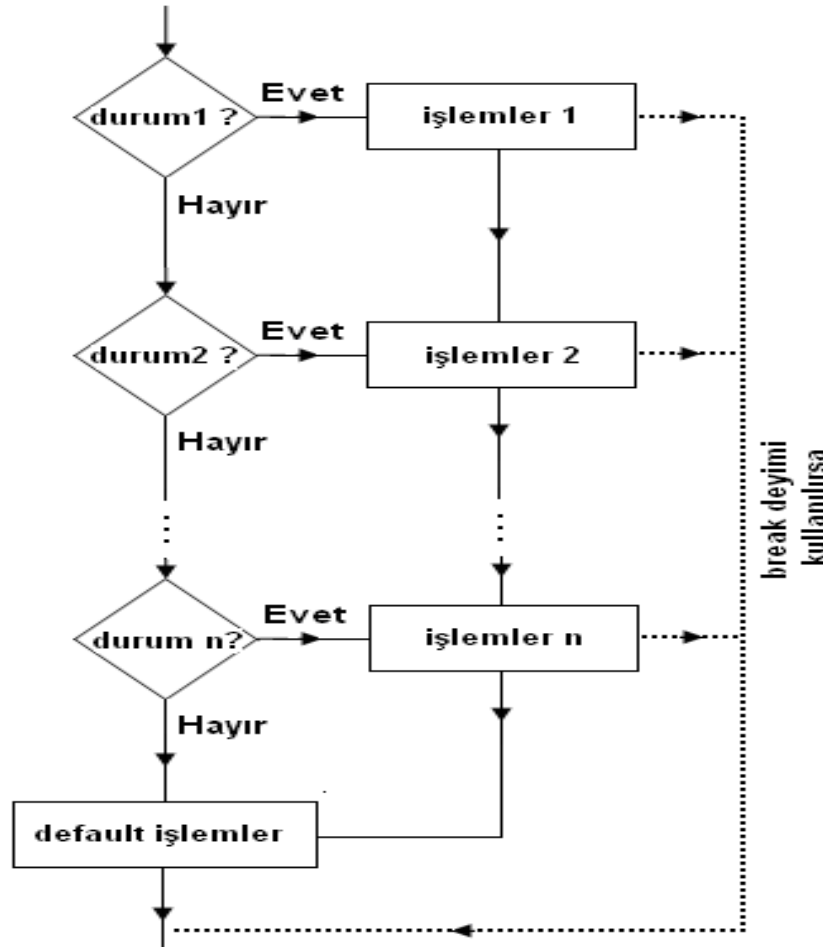


- Pascal ve C'nin çoğu gerçekleştirimi, kısa devre değerlendirmeyi uygulamaktadır.
- Ada'da ise ***and then*** ve ***or else*** olmak üzere kısa devre değerlendirme için ayrı işlemciler tanımlıdır.



# Çoklu Seçim Deyimi

- Programdaki akışı belirleyen ikiden fazla yol varsa **çoklu seçim deyimi** kullanılır.



```
switch (değişken)
{
    case sabit 1:
        işlemler 1;
    case sabit 2:
        işlemler 2;
    ...
    ...
    case sabit n:
        işlemler n;
    default:
        default işlemler;
}
```



## “switch - case” seçme yapısı

- Örnek: “switch”– “case” seçme yapısı için örnek bir C/C++ programı;

```
switch (a) {  
    case '1': printf("cok zayif \n"); break;  
    case '2': printf("zayif \n"); break;  
    case '3': printf("orta \n"); break;  
    case '4': printf("iyi \n"); break;  
    case '5': printf("pekiyi \n"); break;  
    default: printf("yanlis secim \n");  
}  
getch(); }
```



### 3.Yineleme YapılarıDöngü (*loop*)

- Bir programda yer alan bir komut yada komut gurubunun istenilen sayıda çalıştırılmasını sağlayan yapılara yineleme yapıları denir.

- Döngü Çeşitleri
  - **sayaç denetimli döngüler**
  - **mantıksal denetimli döngüler**



# a) Sayaç Denetimli Döngüler

- Bir sayaç denetimli döngüde, **döngü değişkeni** adı verilen bir değişken sayaç değerini gösterir.
- Bu sayaç değerinin **başlangıç değeri**, **bitiş değeri** ve ardışık iki değeri arasındaki farkı gösteren **adım büyüklüğü**, bir sayaç denetimli döngü gövdesinin kaç kez yineleneneceğini belirler.
- Başlangıç değerinden başlayan döngü değişkeni, her yineme için, adım büyüklüğü değerine göre artırılacak veya azaltılacaktır.



# C

- C'de sayaç denetimli döngü tasarımının genel şekli aşağıdaki gibidir.

```
for (ifade_1; ifade_2; ifade_3)
```

```
for (sayac=0; sayac>25;sayac++)  
    T=T+sayac;
```

- Birinci parametre döngü sayacına başlangıç değeri verilmesi,
- ikinci parametre koşulu ve
- üçüncü parametre her çevrimde sayacın nasıl artacağını/eksileceğini ifade eder.



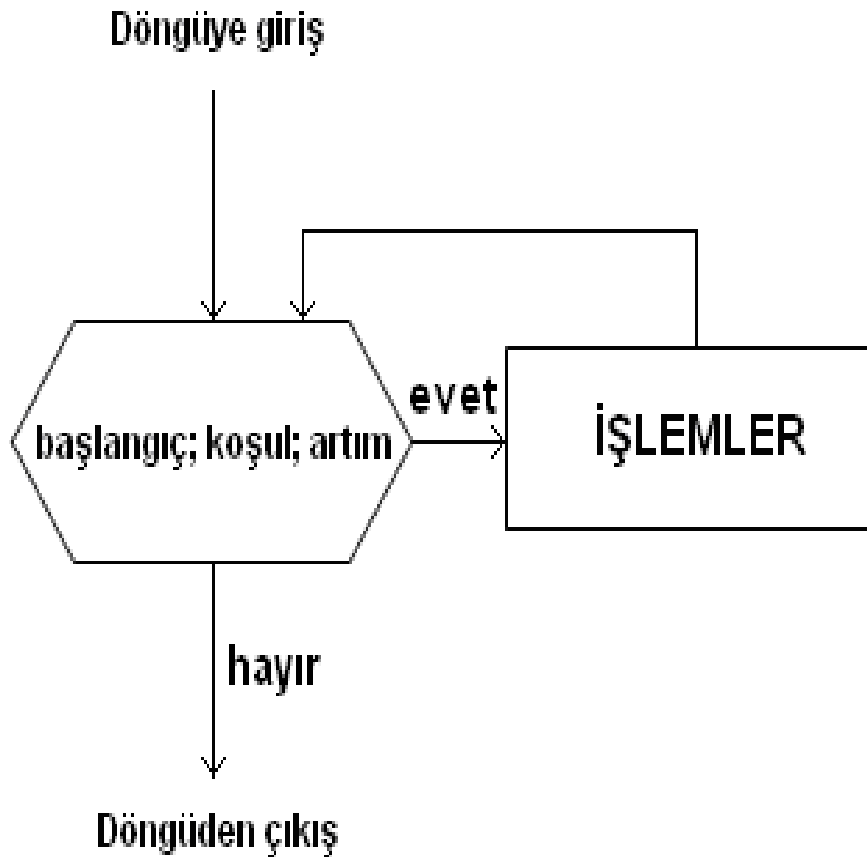
- 
- *Örnek yazılım formatları:*
  - for (k=1;k<50; k+=2)
  - for (k=5;k<=n; k++)
  - for (x=50;x>10;x--)
  - for ( ;x<10;x++)                   /\* başlangıç değeri daha önce atanmış olmalı \*/
  - for (x=2;x<n; )   /\* x döngü sayacı döngü içinde değiştirilmeli \*/





## FOR Döngüsü

- Aşağıda verilen şekilde “for” döngü yapısı akış diyagramı olarak gösterilmekte ve genel yazılım formatı verilmektedir;



```
for(sayaç başlangıcı; koşul; artım)
{
    İŞLEMLER;
}
```



## b) Mantıksal denetimli döngüler

- **önce sinanan** (*pretest*) döngü,
- **sonra sinanan** (*posttest*) döngü
- Döngü başı ve sonu arasındaki deyimlere döngü gövdesi denir.



C'de mantıksal denetimli döngüler için kullanılan yapı aşağıdaki şekilde görülmektedir.

```
while (mantıksal_ifade) } önce sınanan  
deyim
```

```
do } sonra sınanan  
deyim  
while (mantıksal_ifade)
```



## WHILE Döngüsü

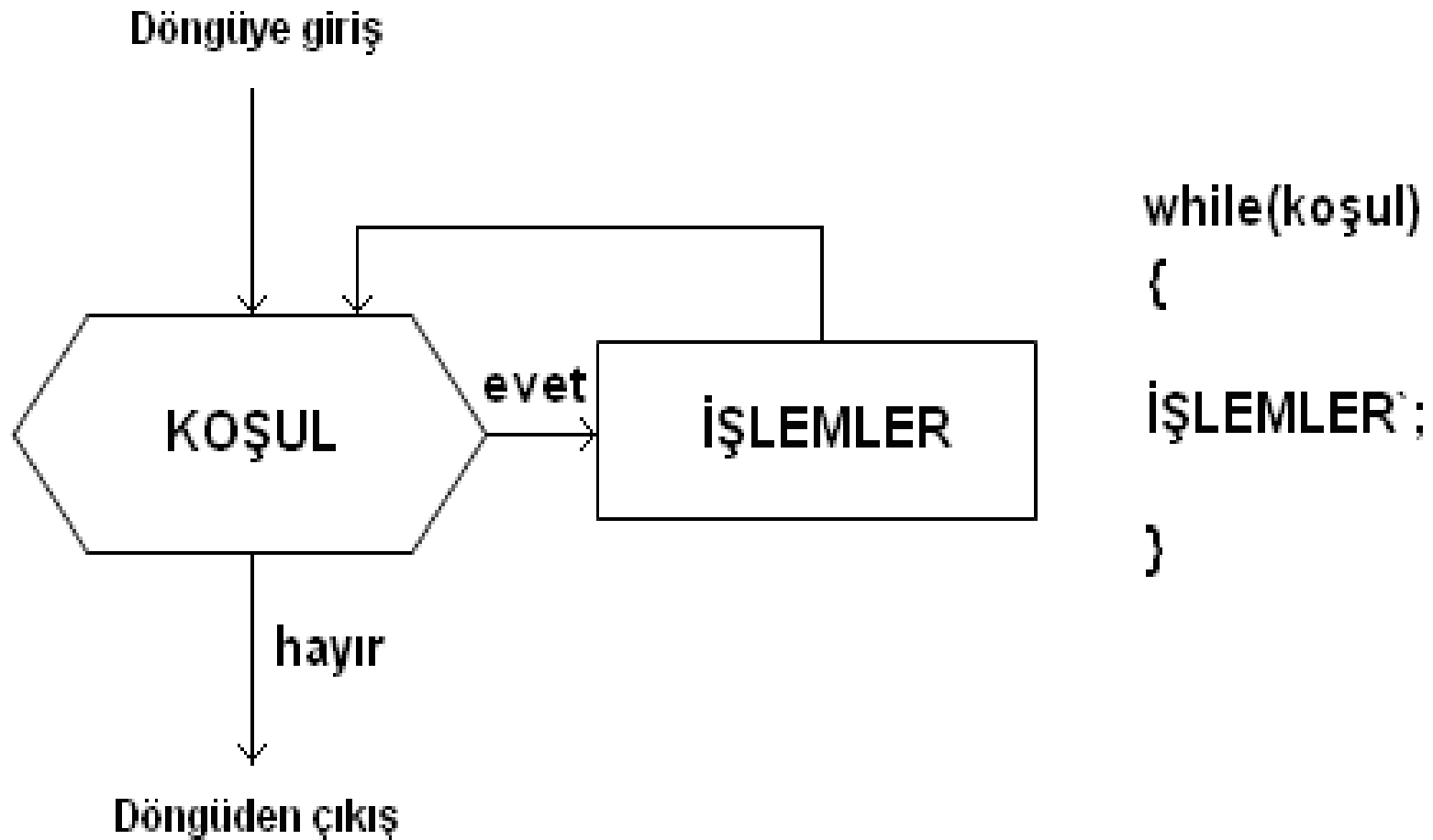
---

- “*while*” döngüsü “*for*” döngüsü gibi aynı işlemleri birçok kez tekrarlamak için kullanılır. Bu döngüde de koşul sınaması çevrime girmeden yapılır.
- Koşul tek bir karşılaştırmadan oluşabileceği gibi birden çok koşulun mantıksal operatörler ile birleştirilmesi ile de oluşturulabilir.



## WHILE Döngüsü

- Aşağıda verilen şekilde “*while*” döngü yapısı akış diyagramı olarak gösterilmekte ve genel yazılım formatı verilmektedir;





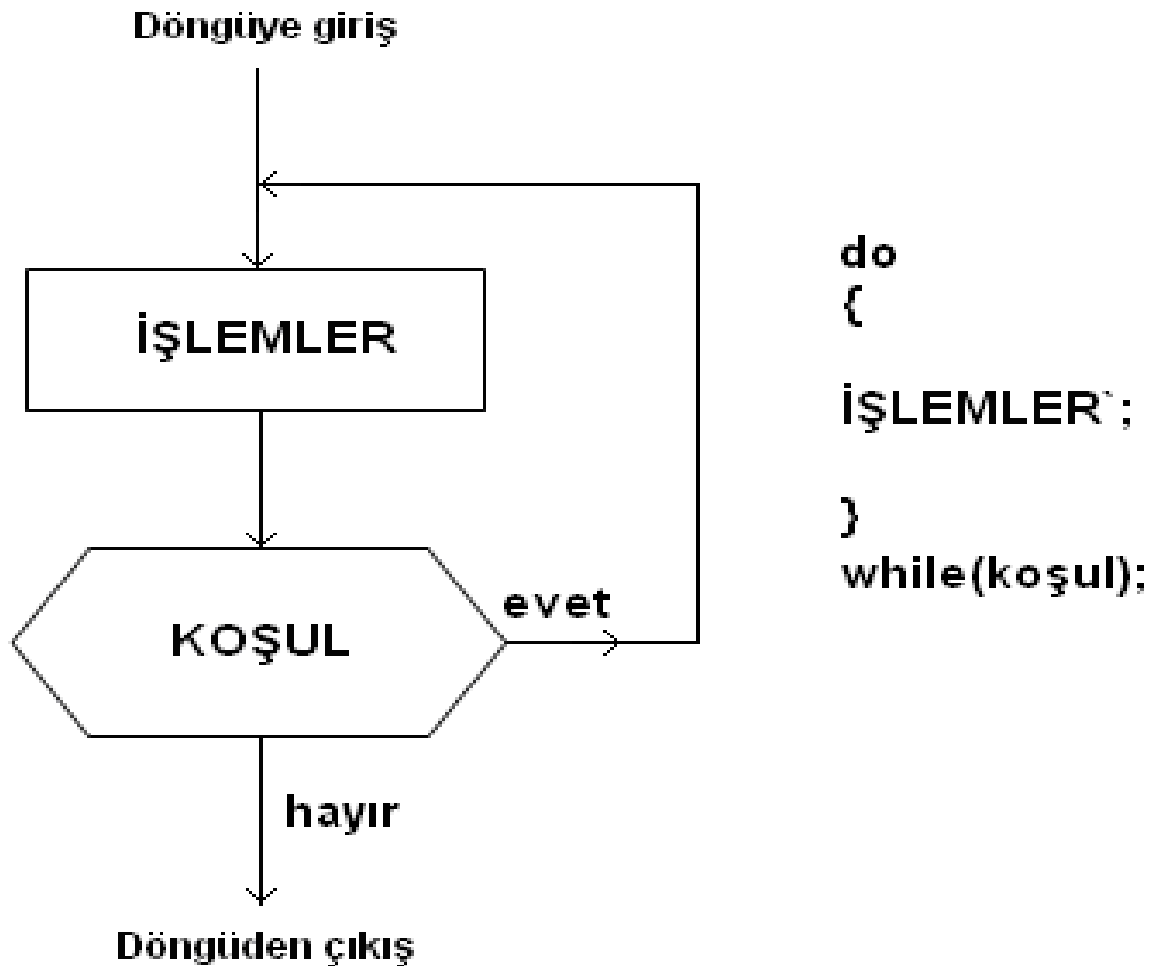
## DO ... WHILE Döngüsü

- *do ... while*” döngüsü diğer döngüler gibi aynı işlemleri birçok kez tekrarlamak için kullanılır.
- Farklı olarak, bu döngüde koşul sınaması yapılmadan çevrime girilir ve işlem kümesi en az bir kere işletilir. Bu deyim yapısında da koşul sağlandığı sürece çevrim tekrarlanır.
- Koşul tek bir karşılaştırmadan oluşabileceği gibi birden çok koşulun mantıksal operatörler ile birleştirilmesi ile de oluşturulabilir.



## DO ... WHILE Döngüsü

- Yanda verilen şekilde “do ... while” döngü yapısı akış diyagramı olarak gösterilmekte ve genel yazılım formatı verilmektedir;





# PASCAL

**Repeat**

.....

**Until mantiksal\_ifade**

# Ada

**loop**

.....

**end loop**





# Akış Denetimini Değiştirme (Koşulsuz)

- Yapısal programlama için, bir program içinde deyimlerin akışı denetlenmelidir. Ancak akış denetiminin değiştirilmesini sağlayan deyimler, yapısal programlama ilkelerine uymayan yapılar içerebilirler.
- Programlama dillerinde yer alan *go to* deyimi ve döngülerden erken çıkış için veya döngünün bir geçişinin normalden önce tamamlanması için kullanılan deyimler, bu deyimlere örnek oluşturmaktadır.



**Begin**

.....

*program kodu*

.....



*deyim*

*program kodu*

.....

**end.**



# Goto Deyimi

- **Goto** deyimi, bir programda akış denetimini koşulsuz olarak değiştirmeyi sağlayan deyimdir. Akışı yönetmek için güçlü bir deyim olmakla birlikte, akışı koşulsuz olarak değiştirme, bir programdaki deyimlerin sırasını rasgele olarak belirleyebildiği için, sorunlara yol açmaktadır.
- *goto* deyiminin programların okunabilirliğini ve güvenilirliğini azaltarak, bakım aşamasını ve programların etkin çalışmasını güçleştirdiğini göstermiştir. (Bunu ifade eden bir benzeştirmede *goto* deyiminin yer verildiği programlar, *sphagetti kodu* olarak nitelendirilmiştir.)



- Popüler diller *goto* deyimine yer vermeyi ancak kullanımını kısıtlamayı tercih etmektedir. (Pascal ve C gibi)



**Begin**

.....

*program kodu*

.....



**GOTO**

*program kodu*

.....

**end.**



## Exit, Break

Normalden önce çıkışı sağlar.


## Continue, Cycle

Döngü içinde bir bölümün atlanmasını sağlar.



# QuickBASIC'te *exit* deyimi

```
for j=1 to 5  
  input miktar  
  if miktar < 0 then exit for  
  toplam = toplam + miktar  
end
```





# Ada (exit deyimi)

```
loop
```

```
....
```

```
exit when koşul;
```

```
....
```

```
end loop;
```







# C' de *break* Deyimi:

- C'de sayaç denetimli veya mantıksal denetimli döngülerden normalden önce çıkış, *break* deyimi kullanılarak sağlanabilir. *break* deyimi çalıştırılır çalıştırılmaz, döngü dışına çıkışı sağlar.



# C' de *continue* Deyimi:

- C' de döngülerde akışı değiştirmek için *break* deyimine ek olarak denetimi en içteki döngünün sınama deyimine aktaran *continue* deyimi tanımlıdır. *continue* deyimi, döngüyü sona erdirmeden döngü gövdesinde bulunan noktadan döngü kapanış deyimine kadar olan deyimlerin atlanmasını ve döngünün bir sonraki yinelemeye devam etmesini sağlar.



# Özet

- Yapısal programlama, program tasarımı ve yazılmasını kurallara bağlayan ve disiplin altına alan bir yaklaşımdır.

## Yapısal programlama:

1. Programların anlaşılabilirliğini artırır.
2. Bir programın hatalarının ayıklanmasını kolaylaştırır.
3. Programın sınanmasını ve düzeltilme zamanını kısaltır.
4. Bir programın niteliği, güvenilirliği ve etkinliği artar.