Bölüm 4 – C'de Program Kontrolü

Outline

Amaçlar

- Bu bölümde öğreneceksiniz:
  - `for` ve `do…while` döngü yapılarını kullanmak.
  - `switch` seçim yapısını kullanarak çoklu seçimler yapmak.
  - Program kontrolünde `break` ve `continue` kullanmak.
  - Mantık operatörlerini kullanmak.

4.1 Giriş

- Bu bölümde öğreneceksiniz
  - Yeni döngü kontrol yapıları
    - `for`
    - `do...while`
  - `switch` çoklu seçim yapısı
  - `break` ifadesi
    - Belli kontrol yapılarından derhal ve hızlı bir biçimde çıkmak için kullanılır
  - `continue` ifadesi
    - Bir döngü gövdesinin geri kalan kısmını atlayarak döngünün diğer kısımlarınının çalışmasını sağlar

4.2 Döngülerin Temelleri

- Döngü
  - Döngü devam koşulları doğru (`true`) kaldığı sürece bilgisayar bir grup emri defalarca çalıştırır
- Sayıcı Kontrollü Döngü
  - Belirli döngü: döngünün kaç defa çalıştırılacağı önceden bilinir
  - Tekrarların sayısını saymak için kontrol değişkeni kullanılır
- Nöbetçi Kontrollü Döngü
  - Belirsiz döngü
  - Tekrar sayısının belli olmadığı durumlarda kullanılır
  - Nöbetçi değer veri girişinin sonlandığını belirtir

4.3 Sayıcı Kontrollü Döngüler

- Sayıcı Kontrollü Döngü gereksinimleri
  - Kontrol değişkeninin ismine (veya döngü sayıcısının)
  - Kontol değişkeninin ilk değerine
  - Kontrol değişkeninin döngü içinde artırılarak ya da azaltılarak değiştirilmesine
  - Kontrol değişkeninin son değerini kontrol edecek bir koşula (döngünün devam edip etmeyeceğini belirtmek için)

4.3 Sayıcı Kontrollü Döngüler

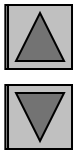
- Örnek:

```
int sayici = 1;           // ilk değerin atanması
while ( sayici <= 10 ) { // döngü koşulu
   printf( "%d\n", sayici );
   ++sayici;              // artırma
}
```

  - Alttaki ifade

```
int sayici = 1;
```

    - Değişkenin isminin `sayici` olduğunu
    - Onun bir tamsayı türünde olduğunu
    - Hafızada onun için yer ayrıldığını
    - Başlangıç değeri olarak `1` atandığını belirtir



Outline

fig04_01.c

```c
/* Fig. 4.1: fig04_01.c
   Sayıcı kontrollü döngü */
#include <stdio.h>

int main()
{
   int sayici = 1;              /* ilk değerin atanması*/

   while ( sayici <= 10 ) {     /* döngü koşulu */
      printf ( "%d\n", sayici );
      ++sayici;                 /* artırma */
   }

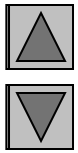
   return 0;
}
```

Program Çıktısı

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

4.3 Sayıcı Kontrollü Döngüler

- Condensed code
  - C programcıları `while` yapısını daha kısa yazabilir
  - Sayacın başlangıç değerini `0` eşitleyip
    - ```
      while ( ++sayici <= 10 )
            printf( “%d\n, sayici );
      ```
      ```

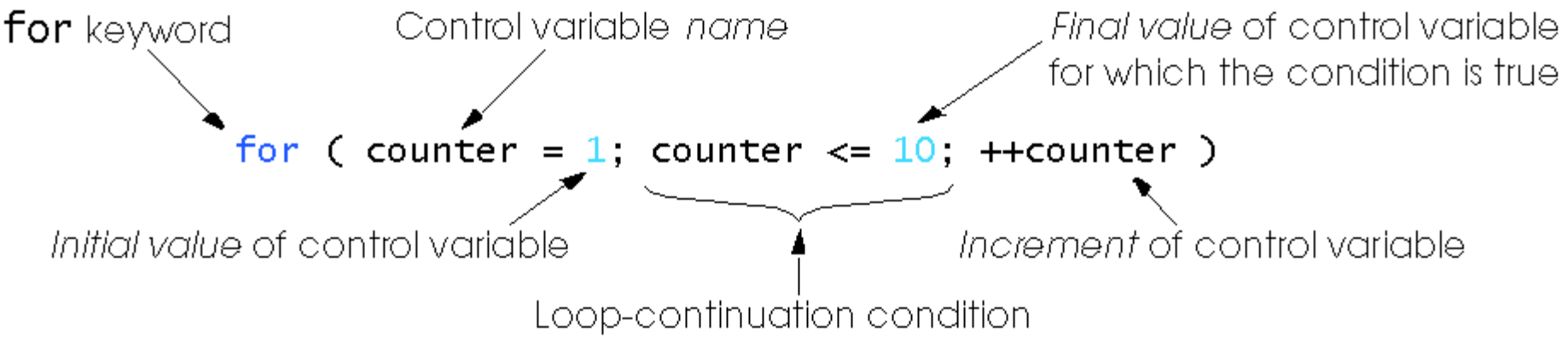


Outline

fig04_02.c

```
1  /* Fig. 4.2: fig04_02.c
2     for yapısı ile sayıcı kontrollü döngü */
3  #include <stdio.h>
4
5  int main()
6  {
7     int sayici;
8
9     /* ilk değer ataması, döngü koşulu, ve artırmanın
10       hepsi birden for yapısının başlığı içindedir */
11    for ( sayici = 1; sayici <= 10; sayici++ )
12       printf( "%d\n", sayici );
13
14    return 0;
15 }
```

4.4 for Döngü Yapısı



4.4 for Döngü Yapısı

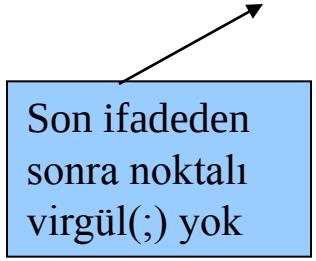
- `for` döngüsünün biçimi

  `for` (*kontol değişkenine ilk değeri atama*; *döngü devam koşulu*; *artırım*)
  ifade

- Örnek:

```
for( int sayici = 1; sayici <= 10; sayici++ )
   printf( "%d\n", sayici );
```

  – 1 den 10 kadar olan tamsayıları ekrana basar



4.4 for Döngü Yapısı

- `for` yapısı aşağıdaki gibi `while` yapısı biçimine çevirilebilir:

```
kontol değişkenine ilk değeri atama;
while (döngü devam koşulu ) {
  ifade;
  artırım;
}
```

- Kontrol değişkenlerine ilk değeri atama ve arttırım
  - Virgülle ayrılmış listeler şeklinde olabilir
  - Örnek:

```
for (int i = 0, j = 0;  j + i <= 10; j++, i++)
   printf( "%d\n", j + i );
```

4.5 for Döngü Yapısıyla İlgili Notlar ve Gözlemler

- Aritmetik ifadeler
  - İlk değeri verme, döngü devam koşulu ve artırma deyimleri aritmetik operatörler içerebilir. Örneğin, `x 2` ve `y 10` olsun

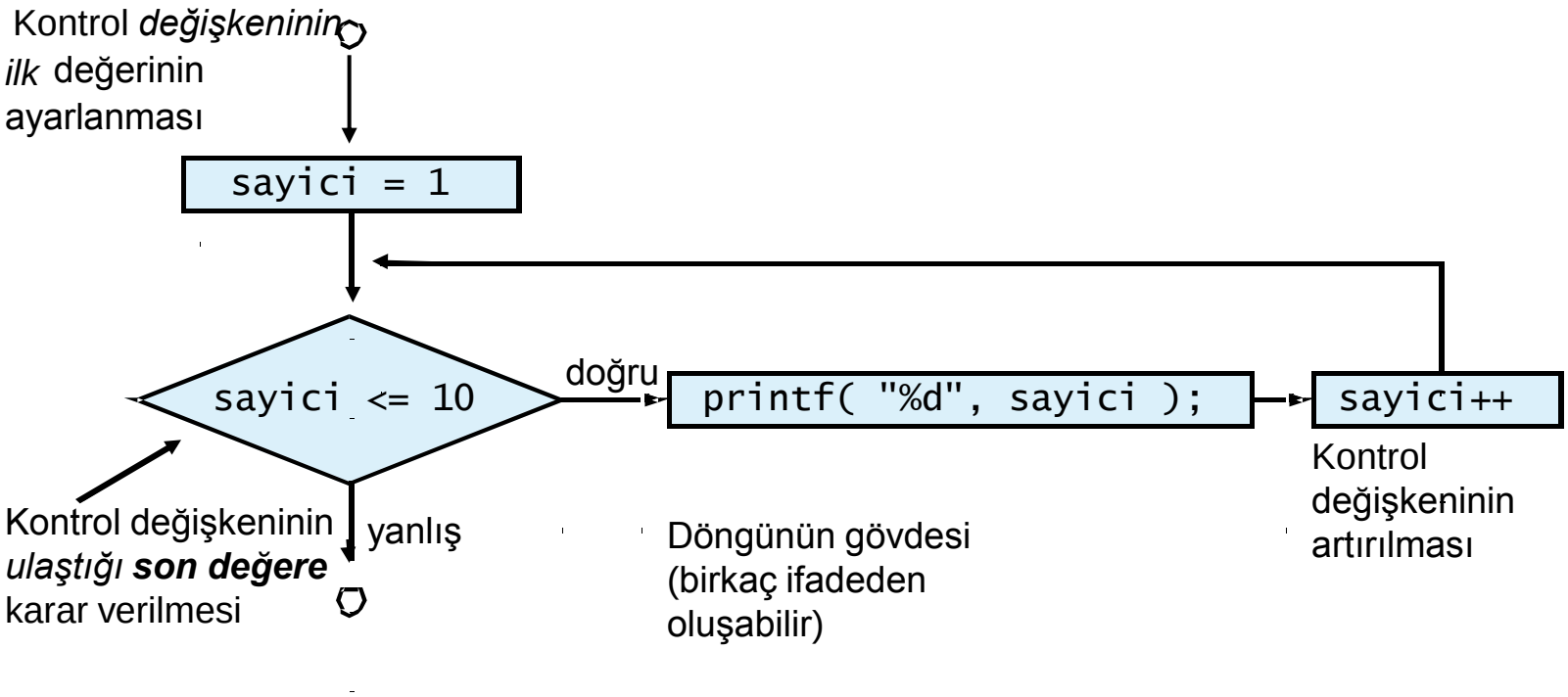
```
for ( j = x; j <= 4 * x * y; j += y / x )
```

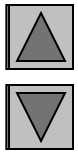
yukarıdaki for döngüsü ile aşağıdaki for döngüsü eşdeğerdir.

```
for ( j = 2; j <= 80; j += 5 )
```

- `for` ifadesi hakkında notlar:
  - Arttırma negatif olabilir (azaltma)
  - Eğer döngü devam koşulu en baştan yanlışsa (`false`)
    - `for` yapısının gövdesi tümden atlanır
    - `for` yapısından sonraki ilk satır çalıştırılır
  - Kontrol değişkeni
    - Döngü gövdesinde sıklıkla yazdırılır ya da işlemlere sokulur

4.5 for Döngü Yapısıyla İlgili Notlar ve Gözlemler





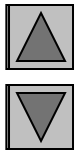
Outline

fig04_05.c

```
1  /* Fig. 4.5: fig04_05.c
2     for ile toplama */
3  #include <stdio.h>
4
5  int main()
6  {
7     int toplam = 0, sayi;
8
9     for ( sayi = 2; sayi <= 100; sayi += 2 )
10       toplam += sayi;
11
12    printf( "Toplam %d\n", toplam );
13
14    return 0;
15 }
```

Program Çıktısı

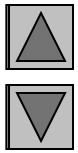
```
Sum is 2550
```



Outline

fig04_06.c (Part 1 of 2)

```
1  /* Fig. 4.6: fig04_06.c
2     Birleşik faizin hesaplanması */
3  #include <stdio.h>
4  #include <math.h>
5
6  int main()
7  {
8     int yil;
9     double miktar, anapara = 1000.0, oran = .05;
10
11    printf( "%4s%21s\n", "Yıl", "Depozito Miktarı" );
12
13    for ( yil = 1; yil <= 10; yil++ ) {
14       miktar = anapara * pow( 1.0 + oran, yil );
15       printf( "%3d%21.2f\n", yil, miktar );
16    }
17
```



Outline

```
18    return 0;
19 }
```

```
 Yil       Depozito Miktarı
   1              1050.00
   2              1102.50
   3              1157.63
   4              1215.51
   5              1276.28
   6              1340.10
   7              1407.10
   8              1477.46
   9              1551.33
  10              1628.89
```

fig04_06.c (Part 2 of 2)

Program Çıktısı

4.7 switch Çoklu Seçim Yapısı

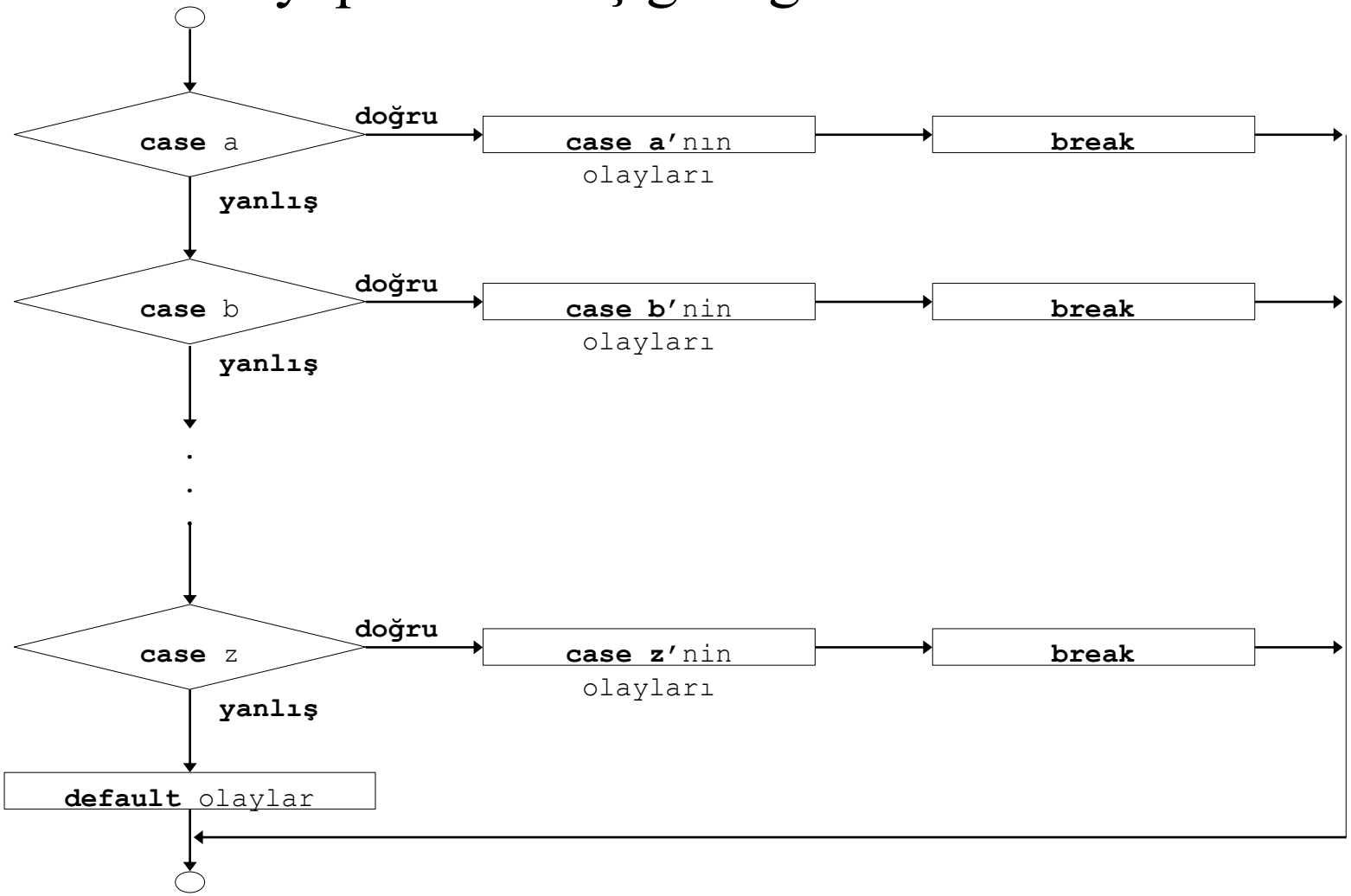
- `switch`
  - Bir değişken yada ifadenin ayrı ayrı sabitlerle karşılaştırılması ve buna bağlı olarak farklı işlemlerin yapılması gerektiren durumlarda kullanılır
- Biçim
  - Bir seri `case` yapısı ve isteğe bağlı `default` yapısı kısımlarından oluşur

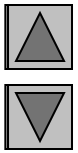
```
switch ( değer ){
    case '1':
        işlemler
    case '2':
        işlemler
    default:
        işlemler
    }
```

  - `break;` ifadeden çıkar

4.7 switch Çoklu Seçim Yapısı

- `Switch` yapısının akış grafiği

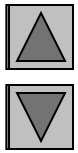




Outline

fig04_07.c (Part 1 of 3)

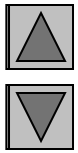
```
1  /* Fig. 4.7: fig04_07.c
2     Harf notlarının sayılması */
3  #include <stdio.h>
4
5  int main()
6  {
7     int not;
8     int aSay = 0, bSay = 0, cSay = 0,
9         dSay = 0, fSay = 0;
10
11    printf(  "Harf notlarını girin.\n"  );
12    printf(  "Çıkış için EOF karakteri girin.\n"  );
13
14    while ( ( not = getchar() ) != EOF ) {
15
16       switch ( not ) { /* while içine yuvarlanmış switch */
17
18          case 'A': case 'a':    /* not büyük A */
19             ++aSay;             /* yada küçük a iken */
20             break;
21
```



Outline

fig04_07.c (Part 2 of 3)

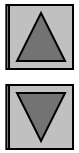
```
22          case 'B': case 'b':   /* not büyük B */
23             ++bSay;            /* yada küçük b iken */
24             break;
25
26          case 'C': case 'c':    /* not büyük C */
27             ++cSay;             /* yada küçük c iken */
28             break;
29
30          case 'D': case 'd':   /* not büyük D */
31             ++dSay;            /* yada küçük d iken*/
32             break;
33
34          case 'F': case 'f':   /* not büyük F */
35             ++fSay;            /* yada küçük f iken */
36             break;
37
38          case '\n': case ' ':  /* bunları veri olarak kabul etme*/
39             break;
40
```



Outline

fig04_07.c (Part 3 of 3)

```
41          default:     /* diğer tüm karakterleri yakala */
42             printf( "Yanlış harf notu girildi." );
43             printf( " Yeni bir not girin.\n" );
44             break;
45       }
46    }
47
48    printf( "\nHer harf notu için toplam:\n" );
49    printf( "A: %d\n", aSay );
50    printf( "B: %d\n", bSay );
51    printf( "C: %d\n", cSay );
52    printf( "D: %d\n", dSay );
53    printf( "F: %d\n", fSay );
54
55    return 0;
56 }
```



Outline

Program Çıktısı

```
Harf notlarını girin.
Çıkış için EOF karakteri girin.
a
b
c
C
A
d
f
C
E
Yanlış harf notu girildi. Yeni bir not girin.
D
A
b
^Z

Her harf notu için toplam:
A: 3
B: 2
C: 3
D: 2
F: 1
```

4.8 do/while Döngü Yapısı

- `do...while` döngüsü
  - `while` yapısına oldukça benzer
  - Döngü devam koşulu döngü gövdesi çalıştırıldıktan sonra kontrol edilir
    - Döngünün gövdesi en az bir kere çalıştırılır
  - Biçim:

```
do {
   ifade;
} while ( döngü devam koşulu );
```

4.8 do/while Döngü Yapısı

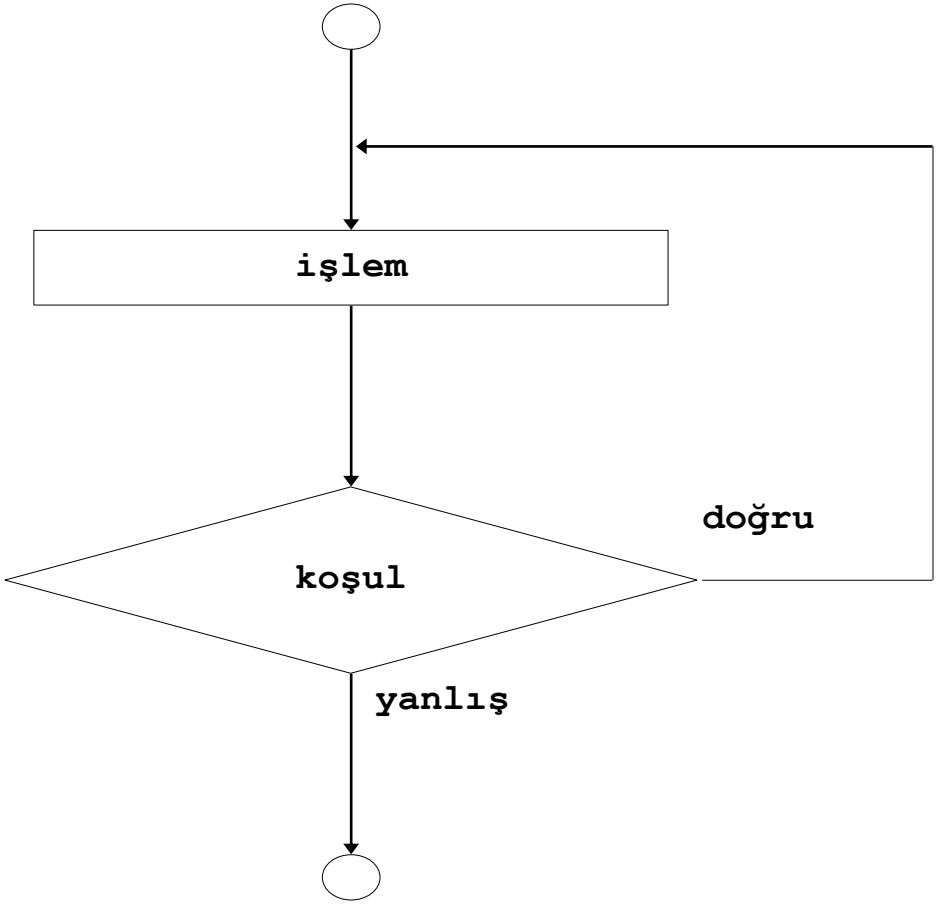
- Örnek (`sayici` değişkeninin değeri 1 olsun):

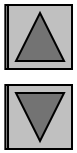
```
do {
   printf( "%d  ", sayici );
} while (++sayici <= 10);
```

  - 1 den 10 a kadar olan tamsayıları sayıları ekrana basar

4.8 do/while Döngü Yapısı

- do/while döngü yapısının akış grafiği





Outline

fig04_09.c

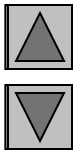
```c
/* Fig. 4.9: fig04_09.c
   do/while döngü yapısını kullanma */
#include <stdio.h>

int main()
{
   int sayici = 1;

   do {
      printf( "%d  ", sayici );
   } while ( ++sayici <= 10 );

   return 0;
}
```

Program Çıktısı

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

4.9 break ve continue İfadeleri

- `break`
  - `while`, `for`, `do…while` veya `switch` ile kullanıldığında o yapıdan çıkışı sağlar
  - Program yapıdan sonraki ilk ifadeyi çalıştırarak devam eder
  - `break` ifadesinin genel kullanımları
    - Döngüden istenen anda çıkmak
    - `switch` yapısında olduğu gibi belli bir kısmından kurtulmaktır



Outline

fig04_11.c

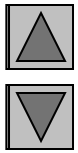
```c
/* Fig. 4.11: fig04_11.c
   for yapısı içerisinden break ifadesinin kullanılması */
#include <stdio.h>

int main()
{
   int x;

   for ( x = 1; x <= 10; x++ ) {

      if ( x == 5 )
         break; /* döngüden sadece x == 5 olduğunda çık */

      printf( "%d ", x );
   }

   printf( "\nDöngüden x == %d olduğunda çıkıldı.\n", x );
   return 0;
}
```

Program Çıktısı

```
1 2 3 4
Döngüden x == 5 olduğunda çıkıldı.
```

4.9 break ve continue İfadeleri

- `continue`
  - `while`, `for` veya `do…while` yapıları içinde çalıştığında döngü gövdesinin kalan kısmını atlar
    - Döngü sıradaki tekrar ile devam eder
  - `while` ve `do…while`
    - Döngü devam koşulu `continue` ifadesinden hemen sonra değerlendirilir
  - `for`
    - Arttırma deyimi çalıştırılır, sonra da döngü devam koşulu kontrol edilir



Outline

fig04_12.c

```c
/* Fig. 4.12: fig04_12.c
   for yapısı içerisinde continue ifadesinin kullanılması */
#include <stdio.h>

int main()
{
   int x;

   for ( x = 1; x <= 10; x++ ) {

      if ( x == 5 )
         continue; /* sadece x == 5 olduğunda döngüyü
                      atla */

      printf( "%d ", x );
   }

   printf( "\ncontinue, 5 değerinin atlanması için kullanıldı\n" );
   return 0;
}
```

Program Çıktısı

```
1 2 3 4 6 7 8 9 10
continue, 5 değerinin atlanması için kullanıldı
```

4.10 Mantık Operatörleri

- && (mantıksal ve)
  - İki koşul da aynı anda doğru (`true`) ise doğru dönürür
- || (mantıksal veya)
  - İki koşuldan en az biri doğru (`true`) ise doğru döndürür
- ! (mantıksal DEĞİL)
  - Doğruyu yanlışa, yanlışı doğruya çevirir
  - Tekli operatördür, sadece tek bir işleneni vardır
- Useful as conditions in loops

İfade	Sonuç
`true && false`	`false`
`true || false`	`true`
`!false`	`true`

4.10 Mantık Operatörleri

deyim1	deyim2	deyim1 && deyim2
0	0	0
0	sıfırdan farklı değer	0
sıfırdan farklı değer	0	0
sıfırdan farklı değer	sıfırdan farklı değer	1

Fig. 4.13 && (mantıksal ve) operatörü için doğruluk tablosu.

deyim1	deyim2	deyim1 | | deyim2
0	0	0
0	sıfırdan farklı değer	1
sıfırdan farklı değer	0	1
sıfırdan farklı değer	sıfırdan farklı değer	1

Fig. 4.14 Mantıksal veya (||) operatörü için doğruluk tablosu.

deyim	! deyim
0	1
sıfırdan farklı değer	0

Fig. 4.15 ! (mantıksal değil) operatörü için doğruluk tablosu.

4.10 Mantık Operatörleri

Operatörler						İşleyiş Biçimleri	Tipleri
++	--	+	-	!	(tip)	Sağdan sola	tekli
*	/	%				Soldan sağa	çarpım
+	-					Soldan sağa	toplam
<	<=	>	>=			Soldan sağa	karşılaştırma
==	!=					Soldan sağa	eşitlik
&&						Soldan sağa	mantıksal ve
||						Soldan sağa	mantıksal veya
?:						Sağdan sola	koşullu
=	+=	-=	*=	/=	%=	Sağdan sola	atama
,						Soldan sağa	virgül

Fig. 4.16 Operatör işlem öncelikleri ve işleyiş biçimleri.

4.11 Eşitlik (==) ve Atama (=) Operatörleri

- Tehlikeli hata
  - Genellikle yazım hatası oluşturmaz
  - Değer üretilebilen herhangi bir deyim kontrol yapılarının karar kısımlarında kullanılabilir
  - Sıfır haricindeki değerler `true`, sıfır değeri ise `false` dır
  - `==` kullanan örnek :

```
if ( puan == 4 )
   printf( “Bonus kazandınız!\n" );
```

    - `puan` değişkenini karşılaştırır, eğer 4 ise bonus ile ödüllendirilir

4.11 Eşitlik (==) ve Atama (=) Operatörleri

- == ile = in yer değiştirdiği örnek :

```
if ( puan = 4 )
    printf( "Bonus kazandınız!\n" );
```

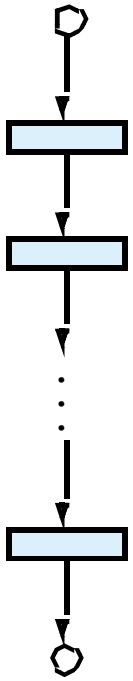
  - Burada `puan` değişkenine `4` değeri atanır
  - `4` sıfırdan farklı bir değer olduğu için ifade doğru (`true`) dur ve `puan` değerinin ne olduğuna bakılmazsızın ödüllendirilir
- Mantık hatası, yazım hatası değil

4.11 Eşitlik (==) ve Atama (=) Operatörleri

- Sol taraf değeri
  - Atama operatörünün sol tarafda bulunabilen ifadelerdir
  - Değerleri değişebilir, örnek olarak değişken isimleri
    - `x = 4;`
- Sağ taraf değeri
  - Atama operatörünün sadece sağ tarafında bulunabilen ifadelerdir
  - Sabitler, örnek olarak sayılar
    - `4 = x` şeklinde yazılamaz;
    - `x = 4` şeklinde yazılmalıdır;
  - Sol taraf değerleri sağ taraf değerleri olarak kullanılabilir, ama tersi mümkün değildir
    - `y = x;`

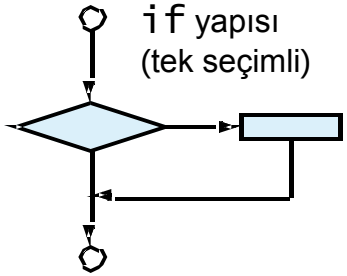
4.12 Yapısal Programlama Özeti

Sequence

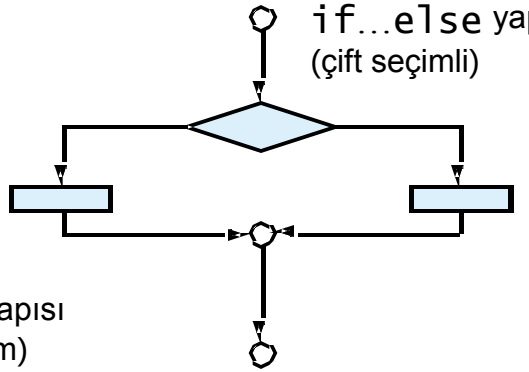


Selection

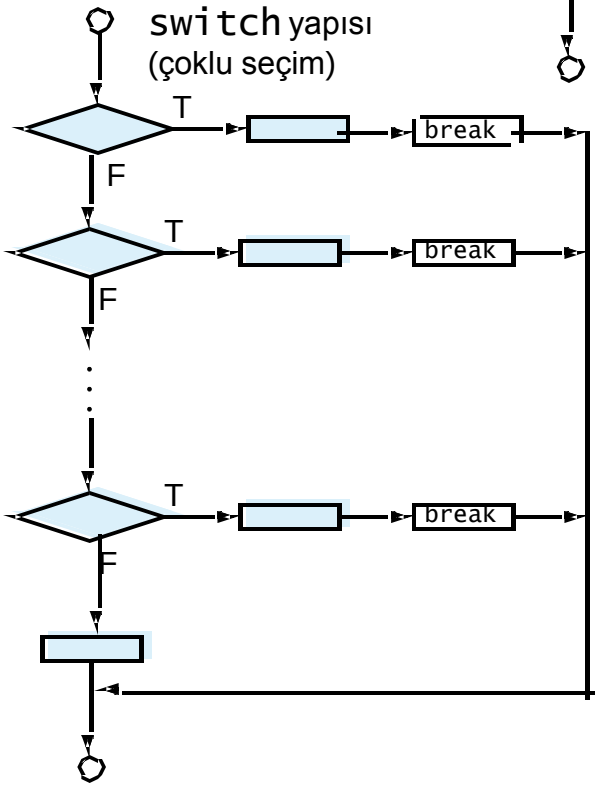
`if` yapısı
(tek seçimli)



`if...else` yapısı
(çift seçimli)



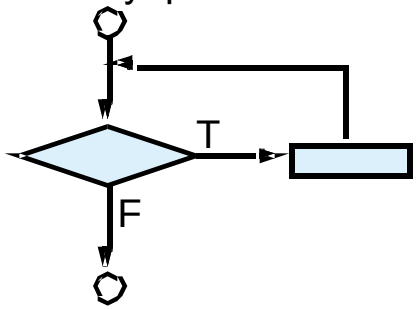
`switch` yapısı
(çoklu seçim)



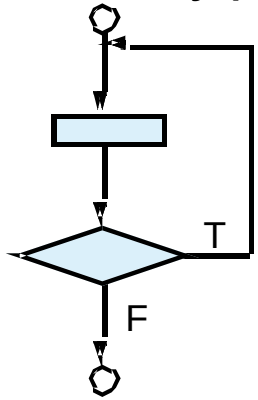
4.12 Yapısal Programlama Özeti

Döngü

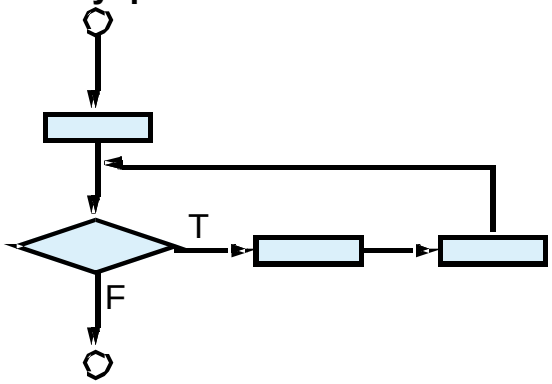
`while` yapısı



`do..while` **yapısı**



`for` **yapısı**



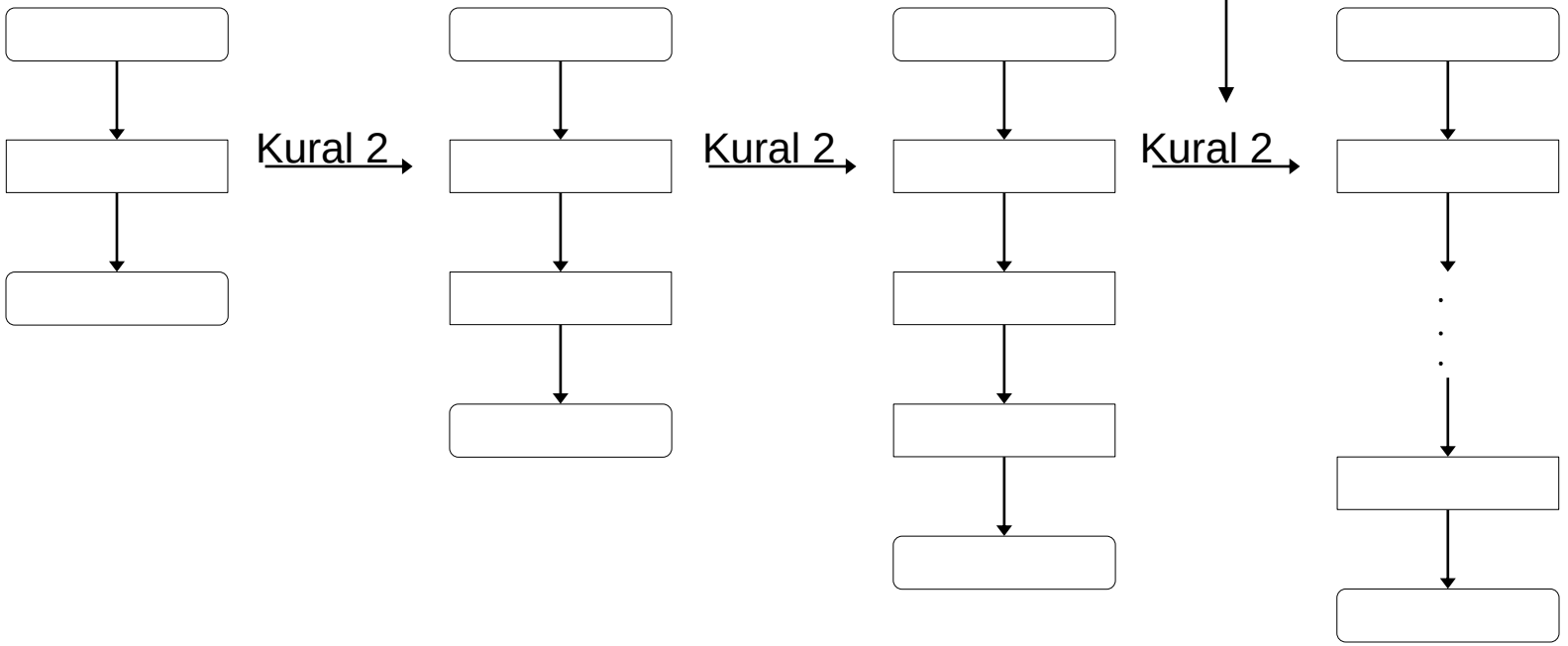
4.12 Yapısal Programlama Özeti

- Yapısal programlama
  - Programları anlama, test, hata ayıklama ve değiştirmekte yapısal programlama yapısal olmayan programlamaya göre daha kolaydır
- Yapısal programlama için kurallar
  - Kurallar programlama toplulukları tarafında geliştirilirler
  - Sadece tekli-giriş/tekli-çıkış kontrol yapıları kullanılır
  - Kurallar:
    1. "en basit akış grafiği" ile başlanır
    2. Yığma kuralı: Her dikdörtgen (işlem) dizi halinde iki dikdörtgen ile değiştirilebilir
    3. Yuvalama kuralı: Her dikdörtgen (işlem) herhangi bir kontrol yapısı ile değiştirilebilir (sıra, `if`, `if...else`, `switch`, `while`, `do...while` or `for`)
    4. 2. ve 3. cü kurallar istenen sıklıkta ve sırada uygulanabilir

4.12 Yapısal Programlama Özeti

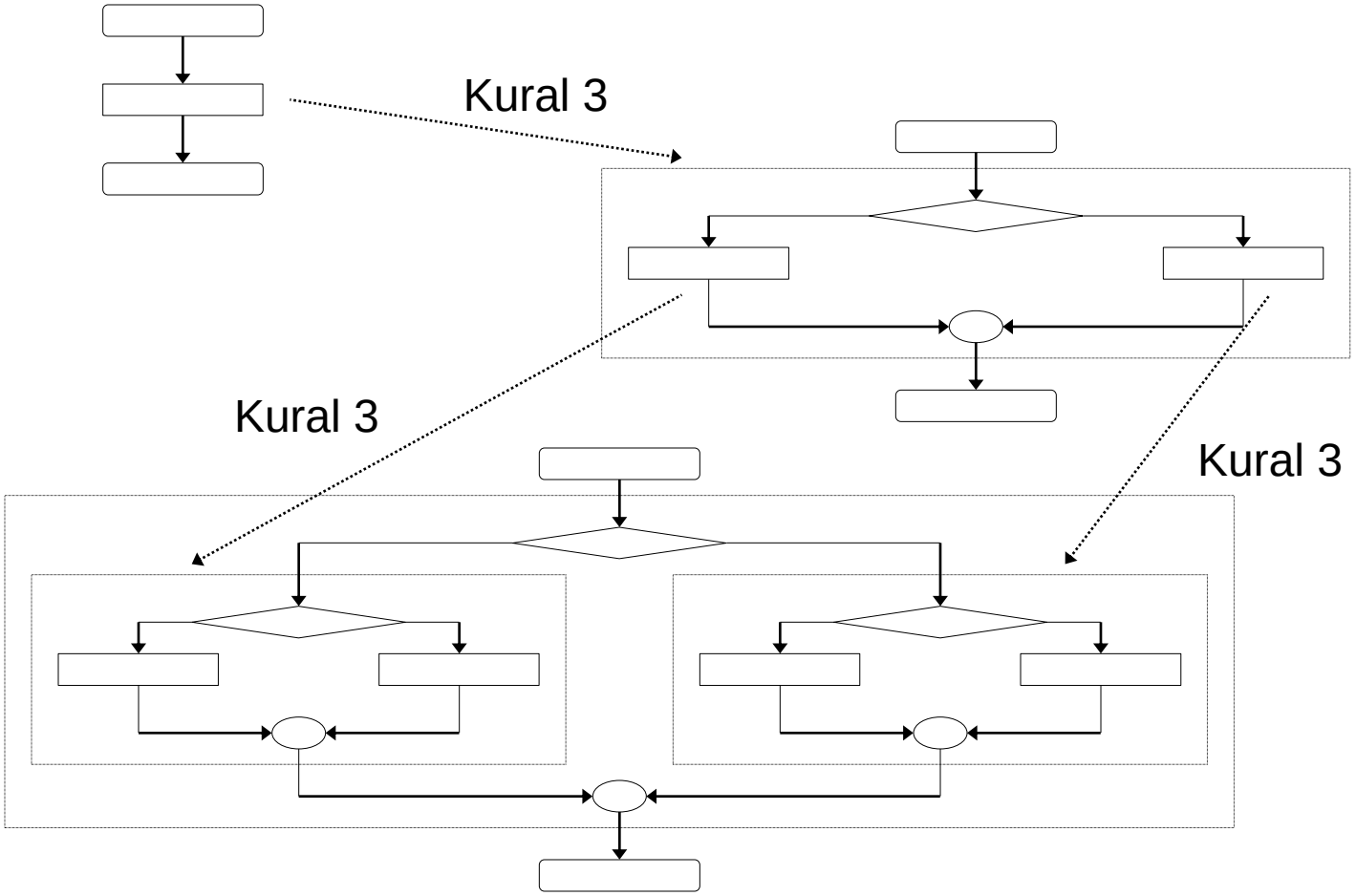
Kural 1 – En basit akış grafiği ile başla

Kural 2 – Her dikdörtgen dizi halinde iki dikdörtgen ile değiştirilebilir



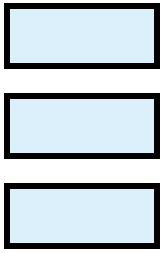
4.12 Yapısal Programlama Özeti

Kural 3 – Her dikdörtgen herhangi bir kontrol yapısı ile değiştirilebilir

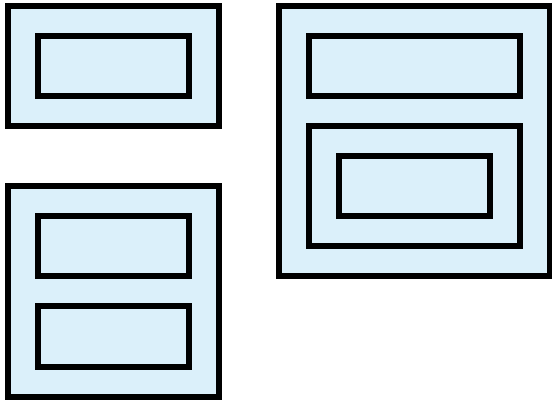


4.12 Yapısal Programlama Özeti

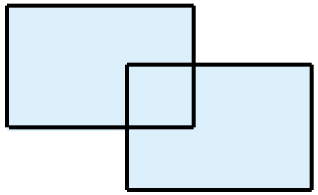
Stacked building blocks



Nested building blocks

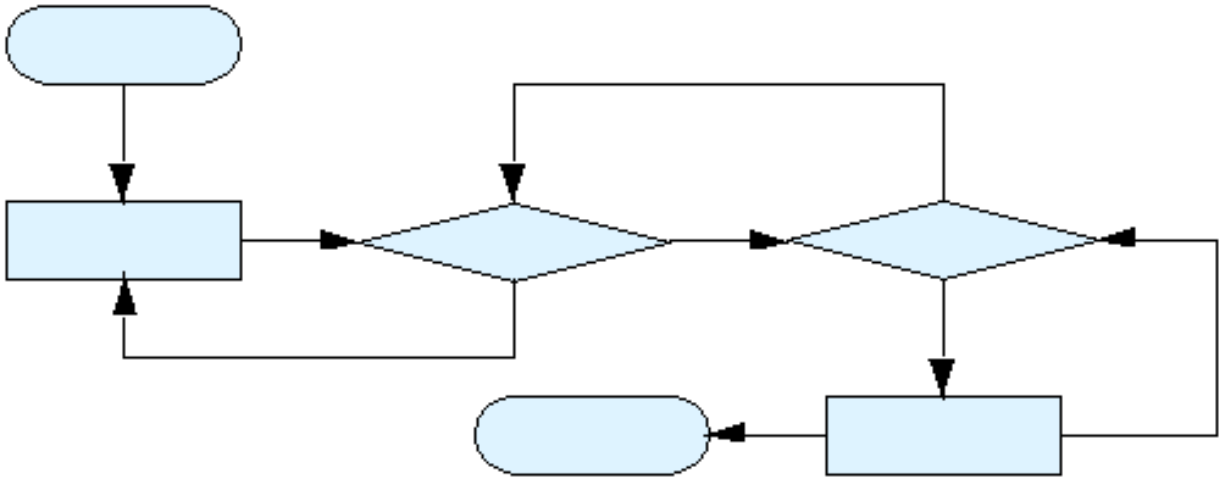


Overlapping building blocks
(Illegal in structured programs)



4.12 Yapısal Programlama Özeti

Figure 4.23 Yapısal olmayan bir akış grafiği.



4.12 Yapısal Programlama Özeti

- Tüm programlar 3 kontole parçalanabilir
  - Sıra – otomatik olarak derleyici tarafından yürütülür
  - Seçim – `if`, `if...else` veya `switch`
  - Döngü – `while`, `do...while` veya `for`
    - Sadece iki şekilde birleştirilebilir
      - Yuvalama (kural 3)
      - Yığma (kural 2)
  - Herhangi bir seçim bir `if` ifadesi olarak tekrar yazılabilir ve herhangi bir döngü bir `while` yapısı olarak tekrar yazılabilir