Bölüm 4 – Program Kontrolü

İçerik

4. 1 Giriş

- İçerik
  - Artışlı tekrar kontrol yapısı (döngü)
    - **`for`**
    - **`do/while`**
  - **`switch`** çoklu seçim yapısı
  - **`break`** deyimi
    - Belli bir kontrol yapısından hemen ve hızlı çıkış için kullanılır
  - **`continue`** deyimi
    - Bir tekrar yapısında döngünün geri kalanını atlamak ve döngüdeki bir sonraki adıma gitmek için kullanılır

4.2 Tekrar Temelleri

- Döngü
  - Belli bir koşul **`true`** (doğru) olduğu sürece tekrarlanan işlemler grubu
- Sayaç-kontrollü tekrar
  - Belirli tekrar: döngünün kaç kez tekrarlanacağı bellidir
  - Tekrar sayısı için bir kontrol değişkeni kullanılır
- Sezgisel-kontrollü tekrar
  - Belirsiz tekrar
  - Tekrar sayısı bilinmediğinde kullanılır
  - Sezgisel değer “veri sonu“ nu belirtir

4.3 Sayaç-Kontrollü Tekrar

- Sayaç –kontrollü tekrarda olması gerekenler
  - Bir kontrol değişkeni adı (veya döngü sayacı)
  - Kontrol değişkeninin ilk değeri
  - Kontrol değişkeninin son değerini test eden (yani döngünün devam edip etmeyeceğine karar veren) bir koşul
  - Kontrol değişkeninin döngü boyunca her defasında değiştiği bir artma (veya azalma)
- Örnek:

```
int sayac = 1;              // ilk değer
while ( sayac <= 10 ) { // tekrar koşulu
   printf( "%d\n", sayac );
   ++sayac;                 // artırma
}
```

- **for** döngüsünün formatı

  for (*ilk değer*; *döngü devam testi*; *artırma*)
  deyim

- Örnek:

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

  – 1 den 10 a kadar tamsayıları yazar

(;) olmadığına dikkat ediniz

- For döngüleri genellikle while döngüsü ile yazılır:

```
ilk değer;
while ( döngü devam testi) {
  deyim;
  artırma;
}
```

- İlk değer ve artırma
  - Virgülle ayrılan bir liste olabilir
  - Örnek:

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

4.5 `for` Yapısı: Notlar ve Gözlemler

- Aritmetik ifadeler
  - İlk değer, döngü-sürdürülmesi, ve artırma aritmetik ifade içerebilir. Eğer **`x`** = **`2`** ve **`y`** = **`10`** ise

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

ifadesi

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

ile aynıdır

- **`for`** yapısı hakkında notlar:
  - "artma" negatif olabilir(eksiltme)
  - Eğer döngü sürdürülmesi koşulu başangıçta **`false`** (yanlış) ise
    - **`for`** döngüsünden çıkılır (döngü işletilmez)
    - Kontrol **`for`** yapısından sonraki deyimle devam eder
  - Kontrol değişkeni
    - Genellikle döngü içinde yazdırılır, fakat gerekli değildir

4.6 for Yapısına Örnek

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

Çıktı:

```
Toplam= 2550
```

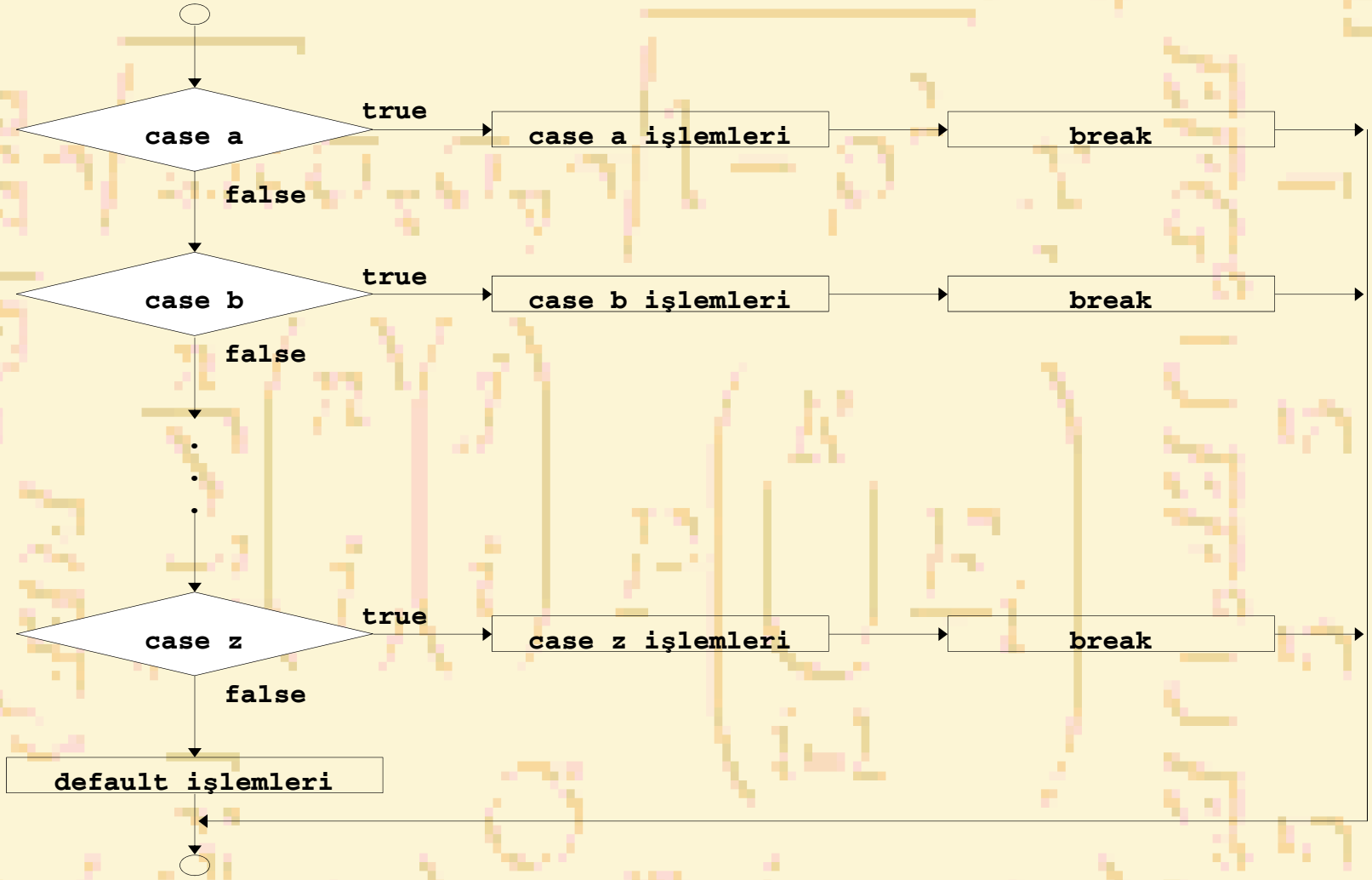
4.7 switch Çoklu-Seçim Yapısı

- **switch**
  - Bir değişken veya ifadenin alabileceği tüm değerler için test edildiği ve farklı işlemler yapıldığı durumda kullanışlıdır
- Format
  - Birçok **case** (durum) ve **default** case(diğer durumlar)

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

  - **break;** yapıdan çık

switch yapısı akış şeması



```c
/* Fig. 4.7: fig04_07.c
   Harf notları sayma */
#include <stdio.h>

int main()
{
   int not;
   int asay = 0, bsay = 0, csay = 0,
       dsay = 0, fsay = 0;

   printf(  "Harf notunu gir.\n"  );
   printf(  "Bitti ise EOF (Ctrl+z) karakterini gir.\n"  );

   while ( ( not = getchar() ) != EOF ) {

      switch ( not ) {    /* switch, while da içiçe */

         case 'A': case 'a':
            ++asay;
            break;

         case 'B': case 'b':
            ++bsay;
            break;

         case 'C': case 'c':
            ++csay;
            break;

         case 'D': case 'd':
            ++dsay;
            break;
```

```
33
34          case 'F': case 'f':
35             ++fsay;
36             break;
37
38          case '\n': case' ':/* boşluk ve enter tuşu girdi alma */
39             break;
40
41          default:         /* diğer tüm karakterler */
42             printf( "Yanlış harf girildi." );
43             printf( " Yeni not girin" );
44             break;
45       }
46    }
47
48    printf( "\Her bir not sayısı:\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 }
```

Çıktı:

```
Harf notunu gir.
Bitti ise EOF (Ctrl+z) karakterini gir.
A
B
C
C
A
D
F
C
E
Yanlış harf girildi Yeni not girin.
D
A
B

Her bir not sayısı:
A: 3
B: 2
C: 3
D: 2
F: 1
```

4.8 do/while Döngü Yapısı

- **do/while** yapısı
  - **while** yapısına benzer
  - İlk işlemden sonra döngü koşulu test edilir
    - Tüm işlemler en az bir kez yapılır
  - Format:

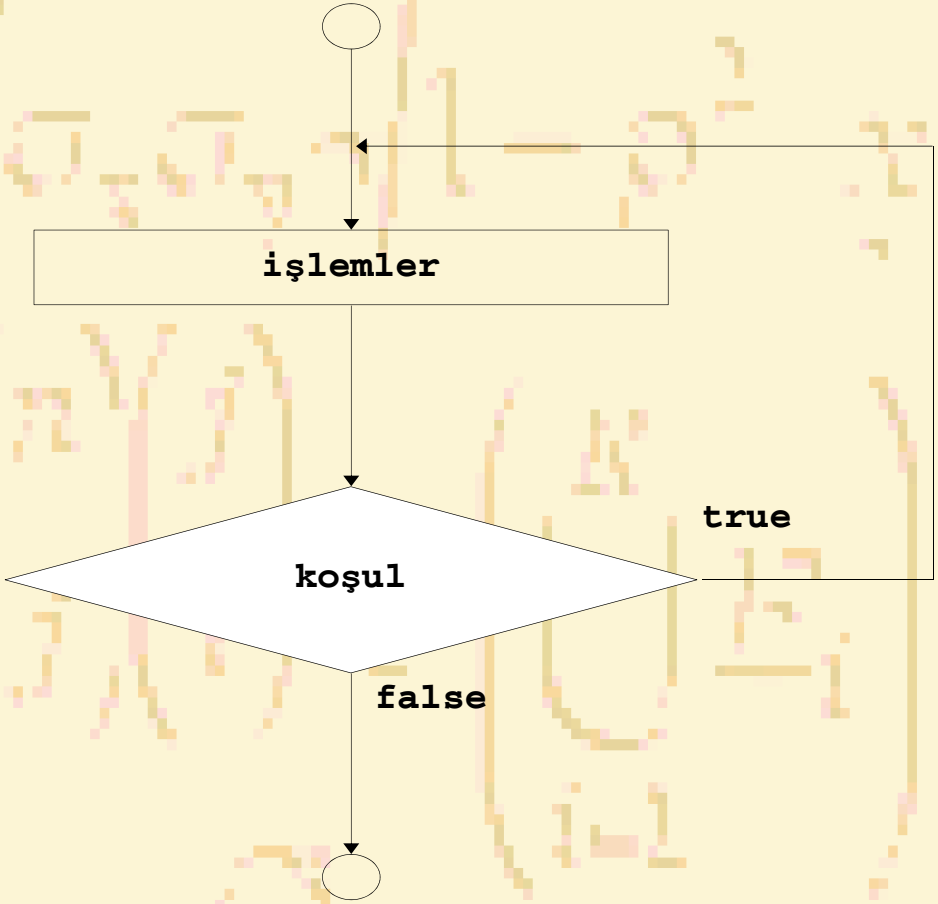
```
do {
   deyim;
} while ( koşul );
```

- Örnek :(sayac=1 olmak üzere)

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

  - **1** den **10** a tamsayıları yazar

- **do/while** akış şeması



4.8 do/while Döngü Yapısı 3

```
1  /* Örnek 4.9
2     do/while kullanma */
3  #include <stdio.h>
4
5  int main()
6  {
7     int sayac = 1;
8
9     do {
10       printf( "%d  ", sayac );
11    } while ( ++sayac <= 10 );
12
13    return 0;
14 }
```

Çıktı:

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

4.9 break ve continue Deyimleri

- **break**
  - **while**, **for**, **do/while** veya **switch** yapısından hemen çıkar
  - Program çalışması yapıdan sonraki ilk deyimle devam eder
  - **break** deyiminin genel kullanımı
    - Bir döngüden erken çıkmak
    - **switch** yapısının kalanını atlamak

- **continue**
  - **while**, **for** veya **do/while** yapısının geriye kalanını atlar
    - Döngünün bir sonraki iterasyonu ile devam eder
  - **while** ve **do/while**
    - **continue** deyiminden hemen sonra Döngü-devam koşulu test edilir
  - **for**
    - Artırma ifadesi işlenir, daha sonra Döngü-devam koşulu test edilir

4.9 break ve continue Deyimleri 2

```
1  /* Örnek 4.12
2     for yapısı içinde continue kullanma */
3  #include <stdio.h>
4
5  int main()
6  {
7     int x;
8
9     for ( x = 1; x <= 10; x++ ) {
10
11       if ( x == 5 )
12          continue;  /* Sadece x==5 ise devam eden işlemler
13                        atlanır  */
14
15       printf( "%d ", x );
16    }
17
18    printf( "\n  5 değerini yazdırmamak için continue kullanıldı\n" );
19    return 0;
20 }
```

Çıktı:

```
1 2 3 4 6 7 8 9 10
5 değerini yazdırmamak için continue kullanıldı
```

4.10 Mantıksal Operatörler

- **`&&`** (mantıksal AND (VE))
  - Her iki koşul da **`true`** ise **`true`** gönderir
- **`||`** (mantıksal OR (VEYA))
  - Koşullardan biri **`true`** ise **`true`** gönderir
- **`!`** (mantıksal NOT, mantıksal tümleyen)
  - Koşulun doğru/yanlış lığını tersine çevirir
  - Birim(unary) operator, bir operanda sahiptir
- Döngüdeki koşullar için kullanışlıdır

ifade	Sonuç
`true && false`	`false`
`true || false`	`true`
`!false`	`true`

4.11 Eşitlik (==) ve Atama (=) Operatorleri

- Tehlikeli hata
  - Derleyici hatası vermez
  - Kontrol yapısında bir değer üreten her hangi bir ifade kullanılabilir
  - Sıfırdan farklı değerler **`true`**, sıfır değeri **`false`** alınır
  - **`==`** kullanımına örnek:

```
if ( kod == 4 )
    printf( "Bir bonus kazandınız!\n" );
```

    - **`Kod-`** u kontrol eder ve eğer **`4`** ise bonus verir

  - Örnek: (**`==`** yerine **`=`** kullanma)

```
if ( kod = 4 )
    printf( "Bir bonus kazandınız!\n" );
```

    - **`Kod-`** u **`4`** alır
    - **`4`** sıfırdan farklı olduğundan, ifade **`true`**- dur. Böylece **`kod`** ne olursa olsun bonus verilir
  - Mantık hatası olup- yazılım hatası oluşmaz

4.11 Eşitlik (==) ve Atama (=) Operatorleri 2

- Sol-değerler
  - Bir denklemin sol tarafında görünen ifadeler
  - Değerleri değişebilir (değişken adları gibi)
    - `x = 4;`
- Sağ-değerler
  - Bir denklemin sadece sağında bulunan ifadeler
  - Sabitler (sayılar gibi)
    - `4 = x;` yazılamaz
    - `x = 4;` yazılmalıdır
  - Sol-değerler sağ-değerler gibi kullanılabilir, fakat tersi doğru değildir
    - `y = x;`

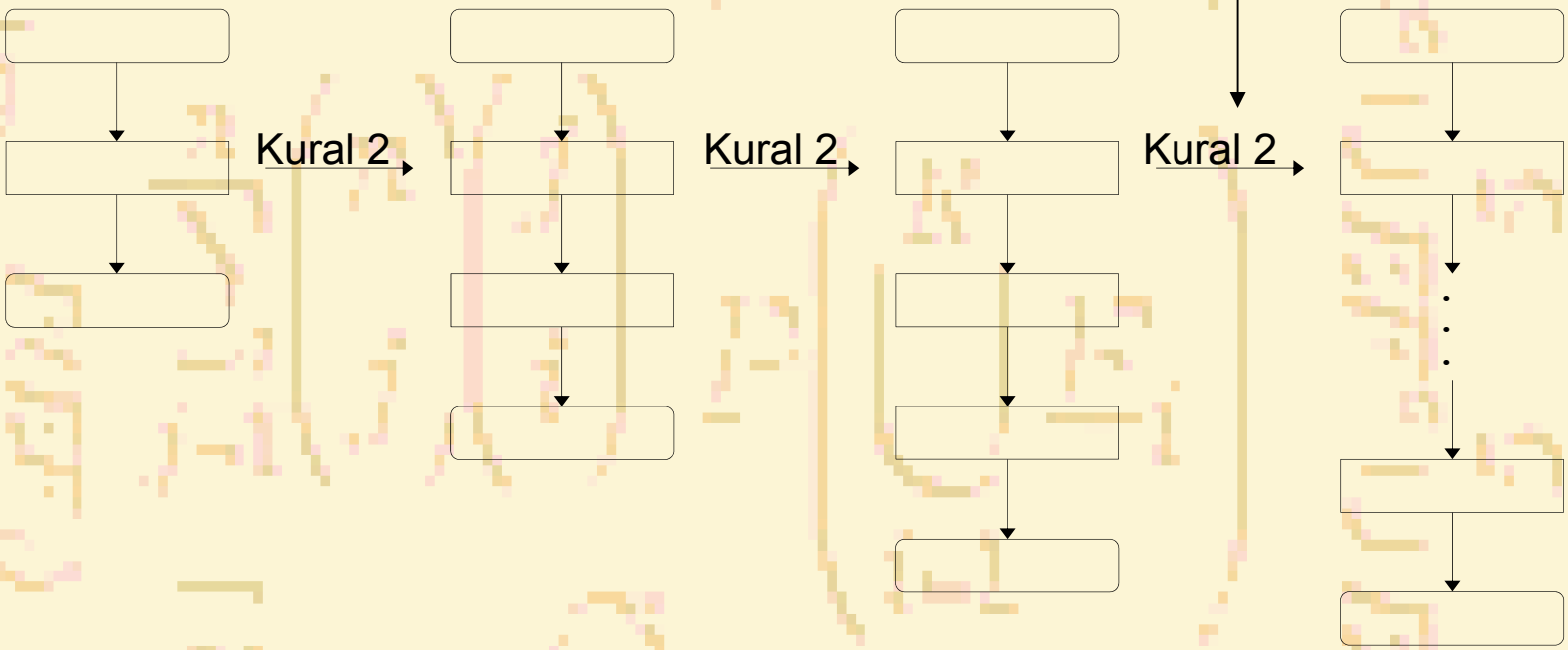
4.12 Yapısal Programlama Özeti

- Yapısal programlama
  - Anlamak, test etmek, düzeltmek ve değiştirmek için yapısal olmayanlardan daha kolaydır
- Yapısal programlama kuralları
  - Programcılar tarafından geliştirilen kurallar
  - Sadece tek-giriş/ tek-çıkış kontrol yapısı kullanılır
  - Kurallar:
    1. "en basit akış şeması" ile başla
    2. Herhangi bir dikdörgen(işlem) yerine daha sonra birden çok dikdörtgen alınabilir
    3. Herhangi bir dikdörgen(işlem) bir kontrol yapısı ile değiştirilebilir (sırası, **`if`**, **`if/else`**, **`switch`**, **`while`**, **`do/while`** veya **`for`**)
    4. 2. ve 3. kurallar herhangi bir sırada birden çok tekrarlanabilir

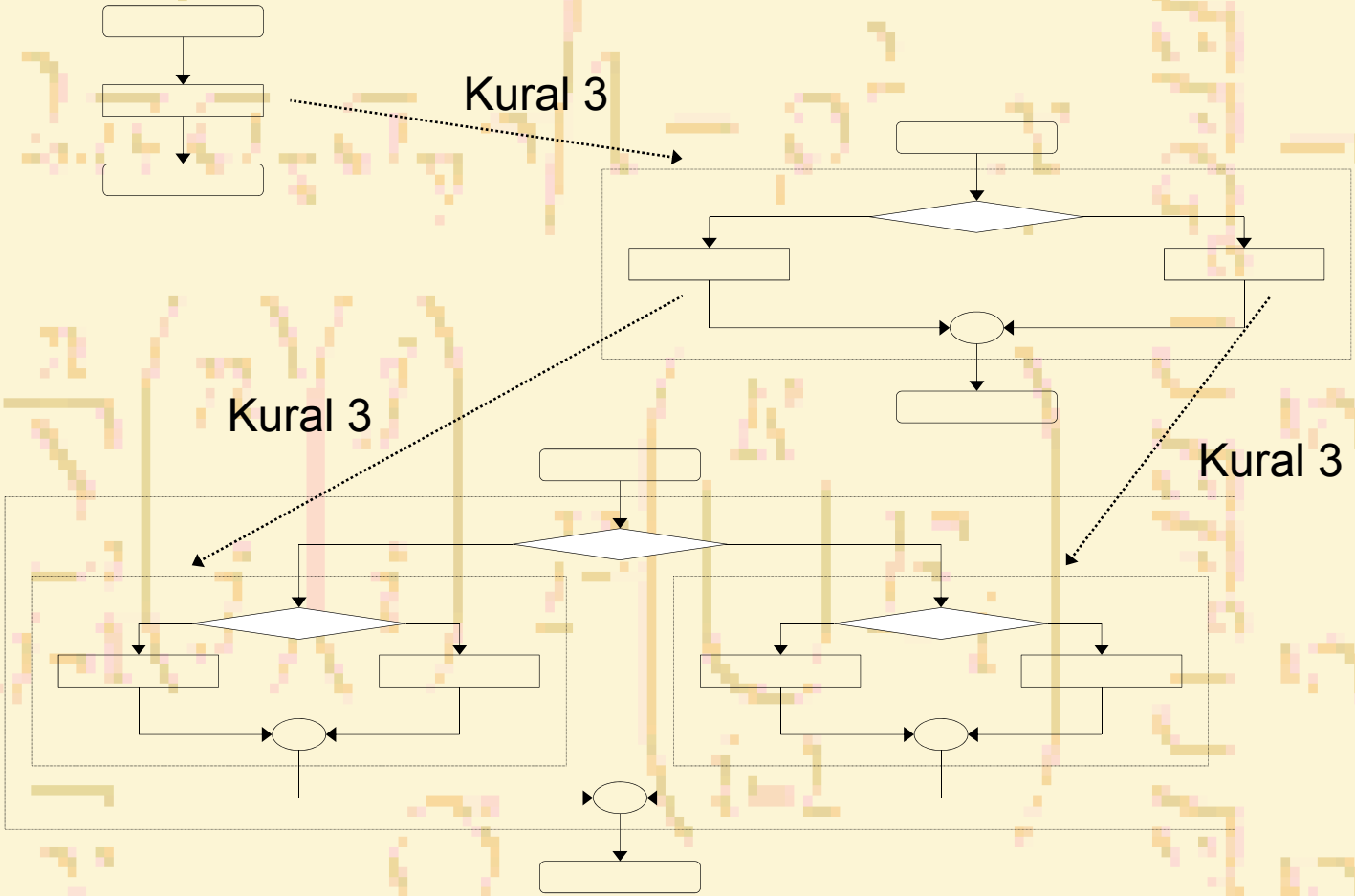
4.12 Yapısal Programlama Özeti 2

Kural 1 – En basit akış şeması ile başla

Kural 2 - Herhangi bir dikdörgen(işlem) yerine ardışık dikdörtgen al



Kural 3 - Herhangi bir dikdörgeni(işlemi) bir kontrol yapısı ile değiştir



- Tüm programlar üç kontrole ayrılabilir
  - Dizisel – derleyici otomatik olarak işler
  - Seçim – **`if`**, **`if`**/**`else`** veya **`switch`**
  - Tekrar – **`while`**, **`do`**/**`while`** veya **`for`**
    - Sadece iki şekilde birleştirilebilir
      - İçiçe (kural 3)
      - Yığma (kural 2)
  - Herhangi bir seçim bir **`if`** deyimi olarak yazılabilir, ve herhangi bir tekrar bir **`while`** deyimi olarak yazılabilir