Bölüm 3 – Yapısal Program Geliştirme

İçerik

3.1 Giriş

- Bir program yazmadan önce:
  - Problem tam anlamı ile anlaşılmalıdır
  - Çözüm için dikkatli bir yaklaşım planı yapılmalıdır
- Bir programı yazarken:
  - Ne tür “inşa bloklarına ” sahip olunduğu bilinmelidir
  - İyi programlama prensipleri kullanılmalıdır

3.2 Algoritmalar

- Problemlerin çözümü
  - Tüm problemler özel bir sıra takip edilerek adım adım çözülür
- Algoritma:
  - Yapılacak işlemler
  - Bu işlemlerin sırası
- Program kontrolü
  - Deyimlerin işlem sırasını belirtiniz

3.3 Önkodlar

- Önkod
  - Algoritmayı geliştirmek için kullanılan yapay bir dildir
  - Günlük komut diline benzer olmalı
  - Gerçekte bilgisayarda kullanılmaz
  - Bir programın yazımında kolaylık sağlar
    - Karşılık gelen C++ yazılımına dönüştürmek kolay olur
    - Sadece işletilebilir deyimler içerir

3.4 Kontrol Yapıları

- Dizisel kontrol
  - Deyimler yazıldıkları sırada ardışık olarak çalıştırılır
- Kontrol transferi
  - Çalıştırılacak bir sonraki deyim, sıradaki deyim değildir
  - **`goto`** deyiminin fazla kullanımı bir çok sorun yaratabilir
- Üç Kontrol Yapısı
  - Tüm programlar üç kontrol yapısı içerir
    - Dizisel yapı: Programlar ardışık adımlarla çalıştırılır
    - Seçim yapısı: C de üç türdür: **`if`**, **`if`**/**`else`**, ve **`switch`**
    - Tekrar yapısı: C de üç türdür: **`while`**, **`do`**/**`while`** ve **`for`**

- Akış şeması
  - Algoritmanın grafiksel gösterimi
  - Belli anlam taşıyan özel şekiller içerir
  - Dikdörtgen sembolü (işlem sembolü):
    - Her hangi bir işlemi belirtir
  - Oval sembolü:
    - Programda başlama ve bitişi belirtir
- Tek-giriş/tek-çıkış kontrol yapıları
  - Bir kontrol yapısının çıkış noktasını bir başkasının giriş noktasına bağlar
  - Programların inşasını kolaylaştırır

- Seçim yapısı:
  - Değişik seçeneklerden birini seçmek için kullanılır
  - Önkod:

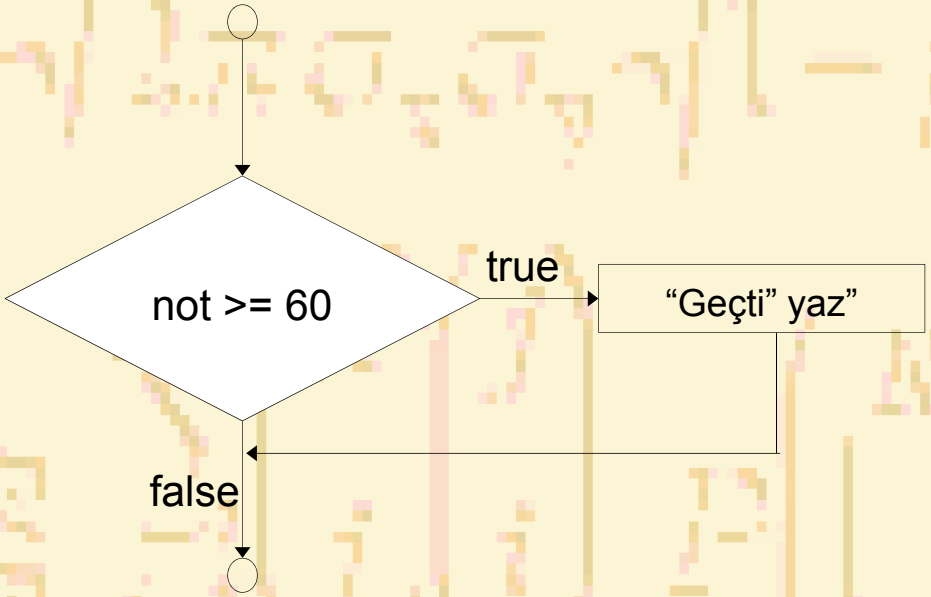
    Eğer öğrenci notu büyük eşit 60 ise "Geçti" yaz
- Eğer koşul doğru ise
  - Yazma işlemi gerçekleşir ve program bir sonraki deyime gider
  - Eğer **`yanlış`** ise yazma işlemi iptal edilir ve program bir sonraki deyime gider
  - Paragraf girişleri programın okunmasını kolaylaştırır
    - C de boşluk karakterleri işleme girmez

- Önkodun C deki karşılığı:

```
if ( not >= 60 )
    printf( "Geçti\n" );
```

- Dörtgen sembolü (karar sembolü)
  - Karar verileceğini belirtir
  - **`True(doğru)`** yada **`false(yanlış)`** olan bir ifade içerir
  - Koşulu test et, uygun yola sap

- **`if`** yapısı bir tek-giriş/tek-çıkış yapısıdır



Bir karar verilmelidir.
sıfır - **`false`** (yanlış)
Sıfırdan farklı – **`true (doğru)`**
Örnek:
`3 - 4 true`

3.6 if-else Seçim Yapısı

- **`if`**
  - Eğer koşul **`true`** (doğru) ise bir işlem yapar
- **`if/else`**
  - Gerçeklenen duruma göre (**`true`** veya **`false`**) farklı işlemler yapar
  - Önkod:

    Eğer öğrencinin notu >= 60 ise
    "Geçti" yaz
    değilse
    "Kaldı" yaz
  - Paragraf girişlerine dikkat ediniz!

- C kodu:

```
if ( not >= 60 )
   printf( "Geçti\n");
else
   printf( "Kaldı\n");
```

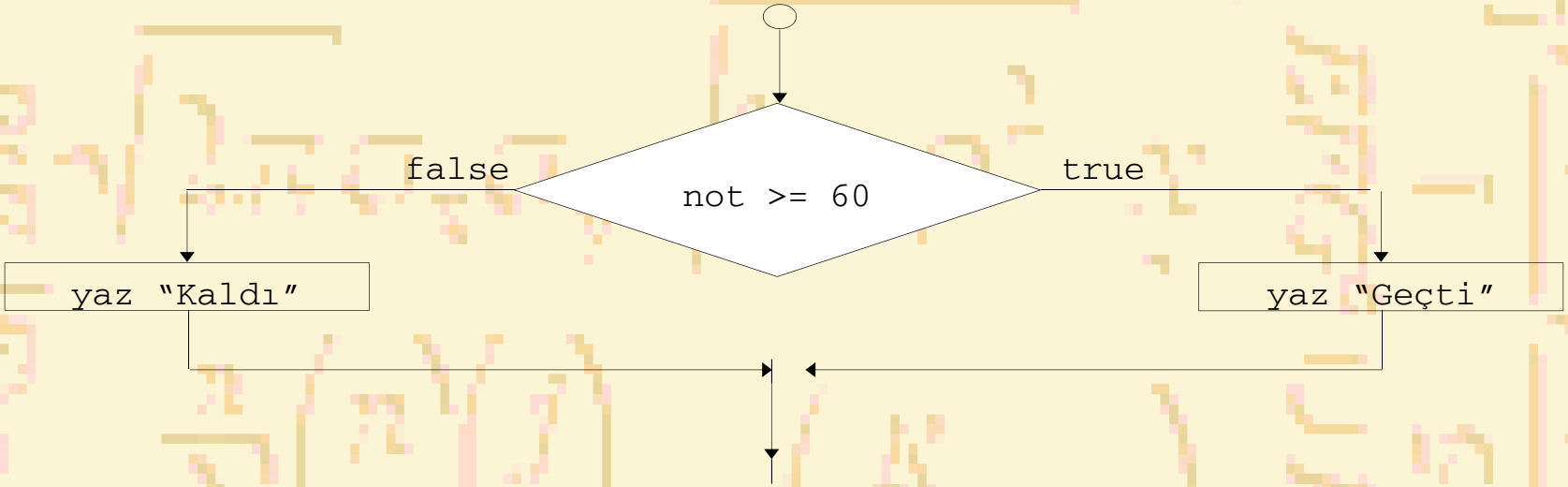
- Üçlü Koşul operatörü (**?:**)
  - Üç argüment içerir (koşul**?true** ise değer**:false** ise değer)
  - Kod:

```
printf( "%s\n", not >= 60 ? "Geçti" : "Kaldı" );
```

  - veya:

```
not>= 60 ? printf( "Geçti\n" ): printf( "Kaldı\n");
```

- **`if`/`else`** seçim yapısı akış şeması



- İçiçe **`if`/`else`** yapıları
  - **`if`/`else`** seçimi ile çoklu yapıları test eder
  - Koşul sağlandığında, deyimlerin geri kalanı atlanır

– İçiçe **`if/else`** yapısı için önkod:

Eğer öğrencinin notu >= 90 ise
"A" yaz
değilse
Eğer öğrencinin notu >= 80 ise
"B" yaz
değilse
Eğer öğrencinin notu >= 70 ise
"C" yaz
değilse
Eğer öğrencinin notu >= 60 ise
"D" yaz
değise
"F" yaz

- Bileşik Deyim:
  - Küme parantezleri arasındaki deyimler grubudur
  - Örnek:

```
if ( not >= 60 )
   printf( "Geçti.\n" );
else {
   printf( "Kaldı.\n" );
   printf( "Bu dersi tekrar alacaksınız.\n" );
}
```

  - Parantezler olmasaydı her koşulda

```
   printf( "Bu dersi tekrar alacaksınız.\n" );
```

    deyimi çalıştırılacaktı

- Blok:
  - Deklare edilmiş bileşik deyimler
- Kodlama hataları
  - Derleyici uyarır
- Mantıksal hatalar:
  - Program çalışma zamanını etkiler
  - Ciddi-olmayan(non-fatal): Program çalışır, fakat yanlış çıktı verir
  - Ciddi (Fatal): Program kesilir veya kitlenir

3.7 `while` Tekrar Yapısı

- Tekrar (Döngü) yapısı
  - Bazı koşulların gerçeklendiği (**`true`** olduğu) sürece bir işlemin tekrarlanması istenir
  - Önkod:

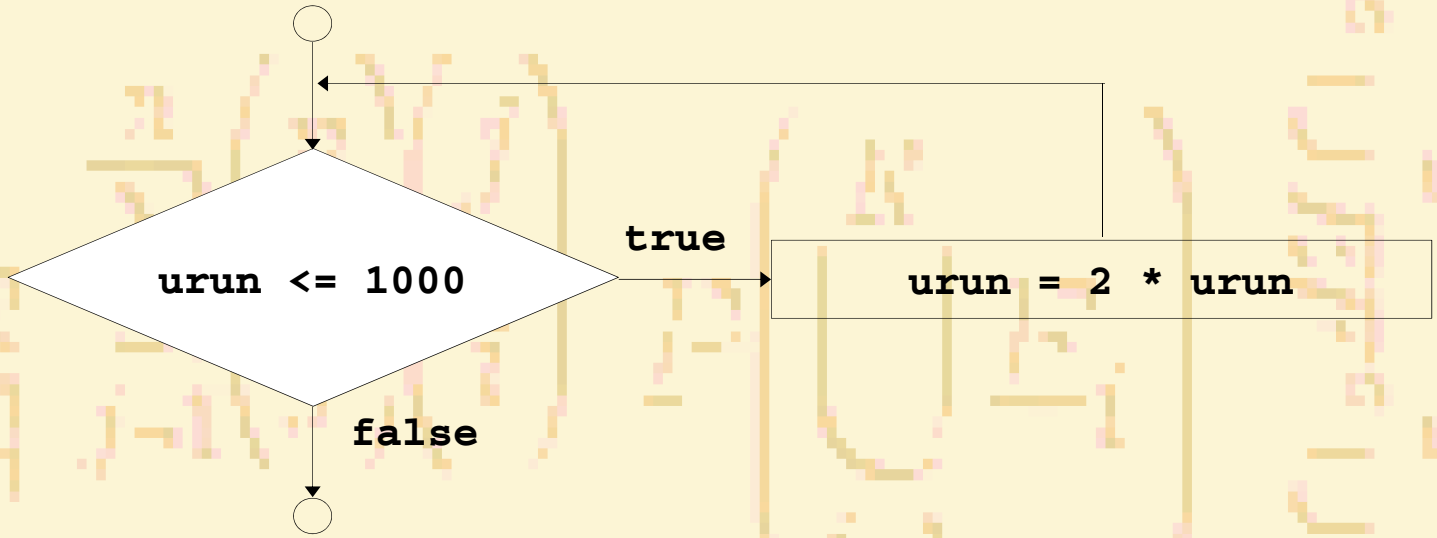
    Alışveriş listesindeki alacaklar bitinceye kadar

    Aldığının üzerini çiz ve alışverişe devam et

  - **`while`** döngüsü koşul **`false`** olana kadar devam eder

- Örnek:

```
int urun = 2;
while ( urun <= 1000 )
    urun = 2 * urun;
```



Sayaç-kontrollü tekrar

- Sayaç kesin bir değere ulaşıncaya kadar döngü devam eder
- Belirli tekrar: tekrar sayısı bilinmektedir
- Örnek: 10 öğrenciye sınav yapılır. Notların (0 - 100 arası tamsayılar) ortalamasını bulunuz.
- Önkod:

```
Toplamı sıfır al
  not sayacını 1 al
Not sayacı 10 oluncaya kadar
  Bir sonraki notu gir
  toplama ekle
  sayacı 1 artır
Toplamı 10 a böl ve ortalamayı bul
  Ortalamayı yaz
```

Sayaç-kontrollü tekrar

```
/* Örnek 3.6
   Sayac kontrollü tekrar ile sınıf ortalaması*/

#include <stdio.h>
int main()
{
   int sayac;
   int not;
   int toplam;
   int ort;

   toplam = 0;   /*başlangıç verileri*/
   sayac = 1;

   //işlem aşaması
   while ( sayac <= 10 ) {
      printf( "Not gir: " );
      scanf( "%d", &not );
      toplam = toplam + not;      /* notları topla */
      sayac = sayac + 1;      /* sayac artır */
   } /* while sonu*/

   /* bitiş aşaması*/
   ort = toplam / 10; /* tamsayı bölümü*/

   printf( "Sınıf Ortalaması:%d\n", ort ); /* sonucu yaz*/

   return 0;

}
```

Program Çıktısı

```
Not gir: 98
Not gir: 76
Not Gir: 71
Not gir: 87
Not gir: 83
Not gir: 90
Not gir: 57
Not gir: 79
Not gir: 82
Not gir: 94
Sınıf Ortalama= 81
```

Sezgisel-kontrollü tekrar

- Problem şu şekle dönüşür:

  Her çağrıldığında rasgele sayıda not girilen bir sınıfın ortalmasını bulan bir program geliştiriniz

  – Bilinmeyen sayıda öğrenci

  – Program bittiğini nasıl anlayacak?

- Sezgisel değer kullan

  – Sinyal değeri, geçici değer veya bayrak değeri olarak da bilinir

  – "Veri sonu girişi" ni belirtir

  – Sezgisel değer girildiğinde döngü sona erer

  – Sezgisel değer diğer verilerle karmaşa yaratmayacak şekilde seçilir (örneğin bu durumda `-1` olabilir)

3.8 Algoritma Formülleştirme 4

Sezgisel-kontrollü tekrar

- Yukarıdan-aşağı adım adım düzenleme
  - En üstü temsil eden bir önkod ile başla:

    Bu sınav için bir oralama belirle
  - En üstü küçük görevlere ayır ve sırala:

    Değişkenleri belirle
    Toplamı gir ve notları say
    Ortalamayı hesapla ve yaz
- Çoğu problem üç aşama içerir:
  - Başlangıç: Programdaki değişkenleri belirle
  - İşlem: Girdileri al ve program değişkenlerini ona göre ayarla
  - Durma: Çıktıları hesapla ve göster

Sezgisel-kontrollü tekrar

- Başlangıç aşamasını *(değişkenleri belirle)* yenile:

  Toplamı sıfır al
  sayacı sıfır al

- *Girdi, toplam ve sayacı* yenile:

  İlk notu gir (muhtemelen sezgisel)
  Kullanıcı sezgisel değeri girmediği sürece;
  bu notu toplama aktar
  sayacı 1 artır
  bir sonraki notu gir (muhtemelen sezgisel)

- *Sınıf ortalamasını hesapla ve yaz* ı yenile:

  Eğer sayaç sıfır değilse
  toplam bölü sayac'ı ortalama al
  ortalamayı yaz
  değilse
  "Hiç bir not girilmedi " yaz

Sezgisel-kontrollü tekrar

```
1  /* Örnek 3.8
2     Sezgisel kontrol tekrarlı
3     sınıf ortalaması */
4  #include <stdio.h>
5
6  int main()
7  {
8     float ort;            /* reel veri tipi */
9     int sayac, not, top;
10
11    /* başlangıç aşaması */
12    top = 0;
13    sayac = 0;
14
15    /* işlem aşaması */
16    printf( "Not gir, durmak için -1: " );
17    scanf( "%d", &not );
18
19    while ( not != -1 ) {
20       top = top+not;
21       sayac = sayac + 1;
22       printf( "Not gir, durmak için -1: " );
23       scanf( "%d", &not );
24    }
25
26    /* durma aşaması */
27    if ( not != 0 ) {
28       ort = ( float ) top / sayac;
29       printf( "Ortalama= %.2f", ort );
30    }
31    else
32       printf( "Hiç bir not girilmedi \n" );
33
34    return 0;    /* program sonu */
35 }
```

Program çıktısı

```
Not gir,durmak için -1: 75
Not gir,durmak için -1: 94
Not gir,durmak için -1: 97
Not gir,durmak için -1: 88
Not gir,durmak için -1: 70
Not gir,durmak için -1: 64
Not gir,durmak için -1: 83
Not gir,durmak için -1: 89
Not gir,durmak için -1: -1
Ortalama= 82.50
```

İçiçe-kontrol yapısı

- Problem
  - Bir kolejde 10 öğrencinin test sonuçları mevcuttur (**1** = geçer, **2** = kalır)
  - Sonucu analiz eden bir program yazınız
    - Eğer 8 den fazla öğrenci geçti ise, "Harcı artır" yaz
- Dikkat edilirse
  - Program 10 sonucu işlemelidir
    - Sayaç-kontrol döngüsü kullanılabilir
  - İki sayaç kullanılabilir
    - Biri geçenlerin sayısı, diğeri de kalanların sayısı için
  - Her bir test sonucu bir sayıdır — **1** veya **2**
    - Eğer sayı **1 değilse 2** olduğunu kabul ediyoruz

İçiçe-kontrol yapısı

- Üst düzey taslak

  Sınav sonuçlarını analiz et ve harç artırılacak mı karar ver

- İlk düzenleme

  Değişkenleri belirle

  10 notu gir ve geçti kaldıları say

  Test sonuçlarının özetini yaz ve harç artımına karar ver

- *Başlangıç değişkenlerini* düzenle

  Geçerleri sıfırla

  Kalanları sıfırla

  Öğrenci sayacını 1 al

İçiçe-kontrol yapısı

- *10 notu gir ve geçti-kaldıları say*

  Öğrenci sayacı 10 dan küçük veya eşit oldukça;
  Bir sonraki sınav sonucunu gir
  Eğer öğrenci geçti ise
  Geçti yi bir artır
  değilse
  Kaldı yı bir artır
  Öğrenci sayacı nı bir artır

- *Test sonuçlarının özetini yaz ve harç artımına karar ver*

  Geçer sayısını yaz
  Kalır sayısını yaz
  Eğer 8 den fazla öğrenci geçti ise
  "Harcı artır" yaz

İçiçe-kontrol yapısı

```
   /* Örnek 3.10
   Sınav sonucları analizi */
#include <stdio.h>

int main()

   {
      /* değişkenleri belirle */
      int gecen = 0, kalan = 0, ogr = 1, sonuc;
      /* 10 öğrenci işle-sayaç kontrol döngüsü */
      while ( ogr <= 10 ) {
         printf( "Sonucu gir (1=Gecti,2=Kaldı ): " );
         scanf( "%d", &sonuc );
      if ( sonuc == 1 )          /* while içinde if/else  */
         gecen = gecen + 1;
      else
         kalan = kalan + 1;

      ogr = ogr + 1;
   }
   printf( "Gecen %d\n", gecen );
   printf( "Kalan %d\n", kalan );
   if ( gecen > 8 )
      printf( "Harcı artır\n" );
   return 0;    /* bitiş */
}
```

Program Çıktısı

```
Sonucu gir (1=Geçti,2=Kaldı): 1
Sonucu gir (1=Geçti,2=Kaldı): 2
Sonucu gir (1=Geçti,2=Kaldı): 2
Sonucu gir (1=Geçti,2=Kaldı): 1
Sonucu gir (1=Geçti,2=Kaldı): 1
Sonucu gir (1=Geçti,2=Kaldı): 1
Sonucu gir (1=Geçti,2=Kaldı): 2
Sonucu gir (1=Geçti,2=Kaldı): 1
Sonucu gir (1=Geçti,2=Kaldı): 1
Sonucu gir (1=Geçti,2=Kaldı): 2
Geçen 6
Kalan 4
```

+(toplama), - (çıkarma), / (bölme), * (çarpma), % (bölümden kalan)

- Atama Operatörleri ve Kısa Gösterimleri

  `c = c + 3;`

  `c += 3;` şeklinde kısaltılabilir

- Deyim Formu

  değişken =*değişken* ***operatör*** *ifade*;

  veya

  değişken **`operator =`** *ifade*;

  formunda yazılabilir

- Örnekler:

```
d -= 4      (d = d - 4)
e *= 5      (e = e * 5)
f /= 3      (f = f / 3)
g %= 9      (g = g % 9)
```

3.10 Artırma ve Eksiltme Operatorleri

- Artırma operatorü (`++`)
  - `c+=1` yerine kullanılabilir
- Eksiltme operatorü (`--`)
  - `c-=1` yerine kullanılabilir
- Önce artırma/eksiltme
  - Operator değişkenin önünde kullanılır (`++c` veya `--c`)
  - Değişken, içinde bulunduğu ifade hesaplanmadan önce değişir
- Sonra artırma/eksiltme
  - Operator değişkenin ardında kullanılır (`c++` veya `c--`)
  - Değişken, ifade hesaplandıktan sonra değişir

3.10 Artırma ve Eksiltme Operatorleri 2

- Eğer **c** eşit **5 ise,** bu durumda

```
printf( "%d", ++c );
```

  - **6** yazar

```
printf( "%d", c++ );
```

  - **5** yazar
  - Her iki komuttan sonra , **c** nin yeni değeri **6** dır
- Değişken, bir ifade içinde değilse
  - önce yada sonra artırma aynı etkiye sahiptir

```
++c;
printf( "%d", c );
```

veya

```
c++;
printf( "%d", c );
```

aynıdır