190600213201 5
Programlama Dilleri Temelleri

BAİBÜ Bilgisayar Müh.

Ders 6

Dr. Öğr. Üyesi İsmail Hakkı Parlak

Kaynaklar: Watt, David A., Programming Language Design Concepts, Wiley

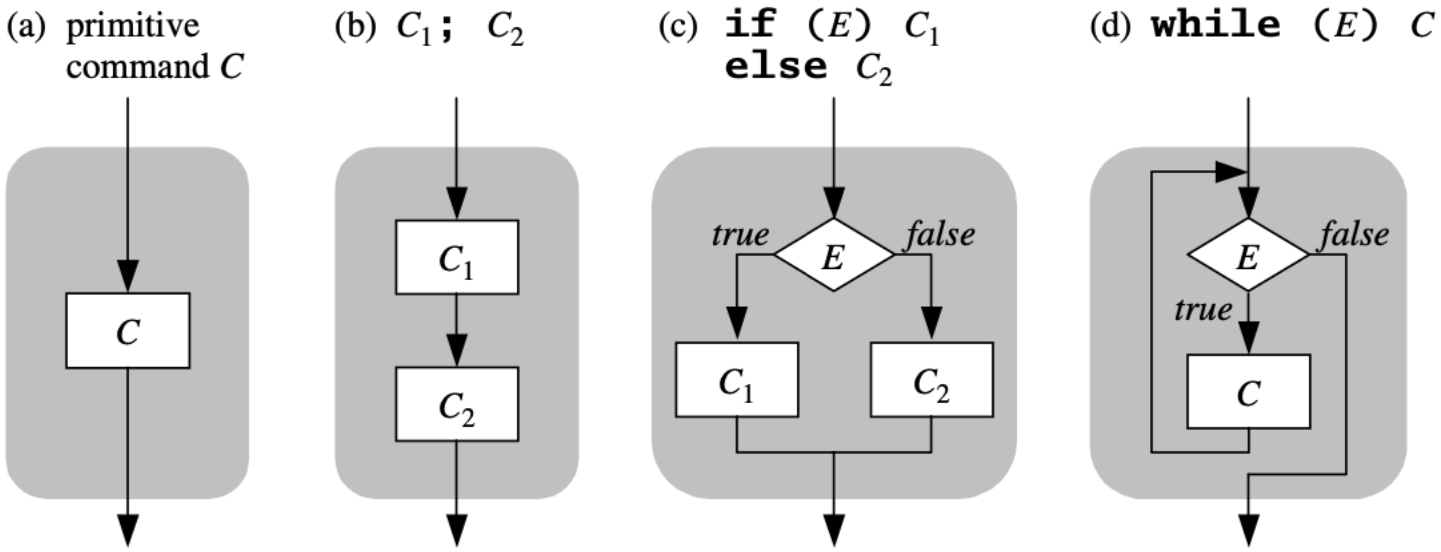
Bazı şekiller kaynak kitaptan kopyalanmıştır.

9. Akış Kontrolü (Control Flow)

- Sıralı (sequential), koşullu (conditional) ve yinelemeli (iterative) komutları kullanarak, her biri tek bir girişi ve tek bir çıkışı olan çeşitli kontrol akışlarını uygulayabiliriz.
- Bunların dışında daha farklı akış kontrolleri elde etmemizi sağlayan yapılar da bulunur:
  - Kontrol akışını etkileyen yapılar olan sıralayıcılar (sequencers)
  - Kontrolü hemen hemen her yere aktarabilen düşük seviyeli sıralayıcılar olan atlamalar (jumps)
  - Çevreleyici komutlar veya prosedürlerden kontrolü alan kaçışlar (escapes)
  - Anormal durumları bildirmek için kullanılabilen sıralayıcılar olan istisnalar (exceptions)

9.1 Sıralayıcılar (Sequencers)

- Aşağıdaki akış diyagramlarının her birinin tek bir giriş ve tek bir çıkış noktası vardır.
- Bazen tek veya çok girişli, tek veya çok çıkışlı kontrol akışları gereksinimi ortaya çıkar.
- Sıralayıcı (sequencer), kontrolü programdaki sıralayıcının hedefi (destination) olarak adlandırılan başka bir noktaya aktaran bir yapıdır.



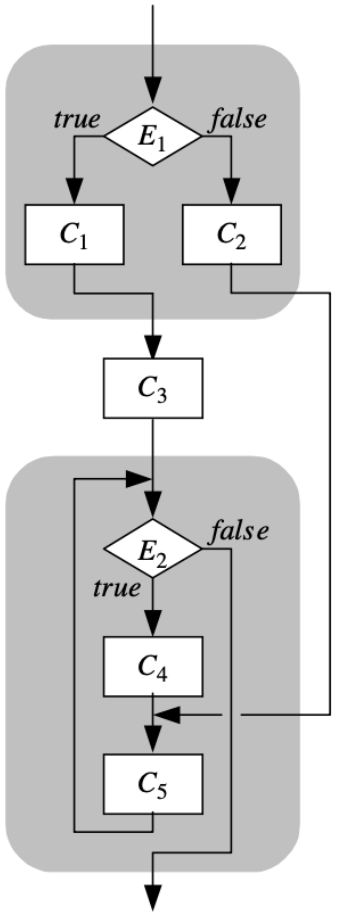
9.2 Atlamalar (Jumps)

- Atlama, kontrolü belirli bir program noktasına aktaran bir sıralayıcıdır.
- Bir atlama tipik olarak “goto L;” şeklindedir ve bu, kontrolü doğrudan bir etiket (label) olan L ile gösterilen program noktasına aktarır.

```
if (E1)
     C1
else {
     C2
     goto X;
}
C3
while (E2)
     C4
 X:  C5
}
```

9.2 Atlamalar (Jumps)

- Sınırlandırılmadan kullanılan atlamalar, herhangi bir komutun birden çok girişe ve birden çok çıkışa sahip olmasına izin verir.
- Akış şeması karışık olduğu için "spagetti" koda yol açma eğilimindedirler.
- Çoğu programlama dili atlamaları destekler (ancak atlamalar artık eskide kalmış olarak kabul edilir). JAVA, uygulamada üst düzey kaçışların ve istisnaların yeterli olduğu gerekçesiyle atlamaları desteklemez.
- Yapılandırılmış program teoremi (structured program theorem), akış şemaları olarak ifade edilebilecek programlar yazmak için goto ifadesinin gerekli olmadığını kanıtlar.



9.3 Kaçışlar (Escapes)

- Bir kaçış, metinsel olarak çevreleyen bir komut veya prosedürün yürütülmesini sonlandıran bir sıralayıcıdır.
- `break` ve `return` sıralayıcıları C, Java, Python gibi dillerde döngüleri sonlandırmak için kullanılabilirler.
- `break` sıralayıcısı ile bir döngü veya `switch` bloğundan çıkıp, akışı, çıkılan bloktan sonra gelen operasyonlara aktarabiliriz.
- Bir fonksiyon içerisinde birden çok `return` sıralayıcısı kullanarak fonksiyona birden çok çıkış ihtimali sağlayabiliriz.

9.4 İstisnalar (Exceptions)

- Anormal bir durum, bir programın normal şekilde devam edemediği durumdur.
- Tipik örnekler: bir aritmetik işlemin taşması (overflow), bir giriş/çıkış işleminin tamamlanamaması...
- Böyle anormal bir durum ortaya çıktığında ne olmalı? Çok sık olarak, program bir teşhis mesajı ile sonlanır. Programın kontrolü bir işleyiciye (handler), programın durumdan kurtulmasını sağlayan bir kod parçasına aktarması çok daha iyidir. Bu gibi durumlardan makul ölçüde kurtulan bir programın sağlam (robust) olduğu söylenir.

9.4 İstisnalar (Exceptions)

- İstisna (exception), anormal bir durumu (veya anormal durumlar ailesini) temsil eden bir varlıktır.
- Anormal bir durum algılayan herhangi bir kod, uygun bir istisna oluşturabilir (`throw, raise`). Bu istisna daha sonra, istisna işleyicisi olarak adlandırılan bir yapının anormal durumdan kurtulduğu programın başka bir bölümünde yakalanabilir (`catch`).
- Programcı her bir istisnanın nerede ve nasıl işleneceği üzerinde tam kontrole sahiptir.

9.4 İstisnalar (Exceptions)

- İstisnaların önemli özellikleri:
  - Bir alt komut bir istisna atarsa, o özel istisnayı yakalayabilecek bir istisna işleme komutu olmadığı sürece, çevreleyen komut da o istisnayı atar. Bir prosedürün gövdesi bir istisna atarsa, ilgili prosedür çağrısı da bu istisnayı atar.
  - istisna atan bir komut aniden durdurulur.
  - Belirli istisnalar kullanılan dile yerleşiktir ve yerleşik işlemler tarafından atılabilir. Örn: aritmetik taşma, aralık dışı dizi indekslemeler.
  - Programcı tarafından başka istisnalar deklare edilebilir ve programın kendisi anormal bir durum algıladığında açıkça atılabilir.

9.4 İstisnalar (Exceptions)

```
public static int divideNums(int a, int b) {
  int result = a / b;
  System.out.println("Sonuc hesaplandi.");
  return result;
}

public static void main(String[] args) {
  int c = divideNums(3, 0);
  System.out.print("3 / 0 = " + c);
}
```

9.4 İstisnalar (Exceptions)

```
3 public static int divideNums(int a, int b) {
    int result = a / b;
    System.out.println("Sonuc hesaplandi.");
    return result;
  }

9 public static void main(String[] args) {
    int c = divideNums(3, 0);
    System.out.print("3 / 0 = " + c);
  }
```

```
Exception in thread "main" java.lang.ArithmeticException: / by
zero
    at Main.divideNums(Main.java:3)
    at Main.main(Main.java:9)
```

9.4 İstisnalar (Exceptions)

```
public static int divideNums(int a, int b) {
   int result = -1;
   try {
     result = a / b;
     System.out.println("Sonuc hesaplandi.");
   } catch (Exception e) {
     System.out.println("X");
   }
   return result;
 }
 public static void main(String[] args) {
   int c = -2;
   try {
     c = divideNums(3, 0);
   } catch (Exception e) {
     System.out.println("Y");
   }
   System.out.print("3 / 0 = " + c);
 }
```

9.4 İstisnalar (Exceptions)

```
public static void kaydet(String ad) {
    if (ad.isEmpty()) {
        throw new IllegalArgumentException("Ad bos gelmis.");
    }
    FileWriter myWriter = new FileWriter("kullanicilar.txt");
    myWriter.write(ad);
    myWriter.close();
}

public static void main(String[] args) {
    try {
        kaydet("Ali");
    } catch (IllegalArgumentException e) {
        System.out.println("Yanlis arguman.");
    } catch (IOException e) {
        System.out.println("Dosya bulunamadi.");
    } finally {
        System.out.println("Program sonalniyor.");
    }
}
```