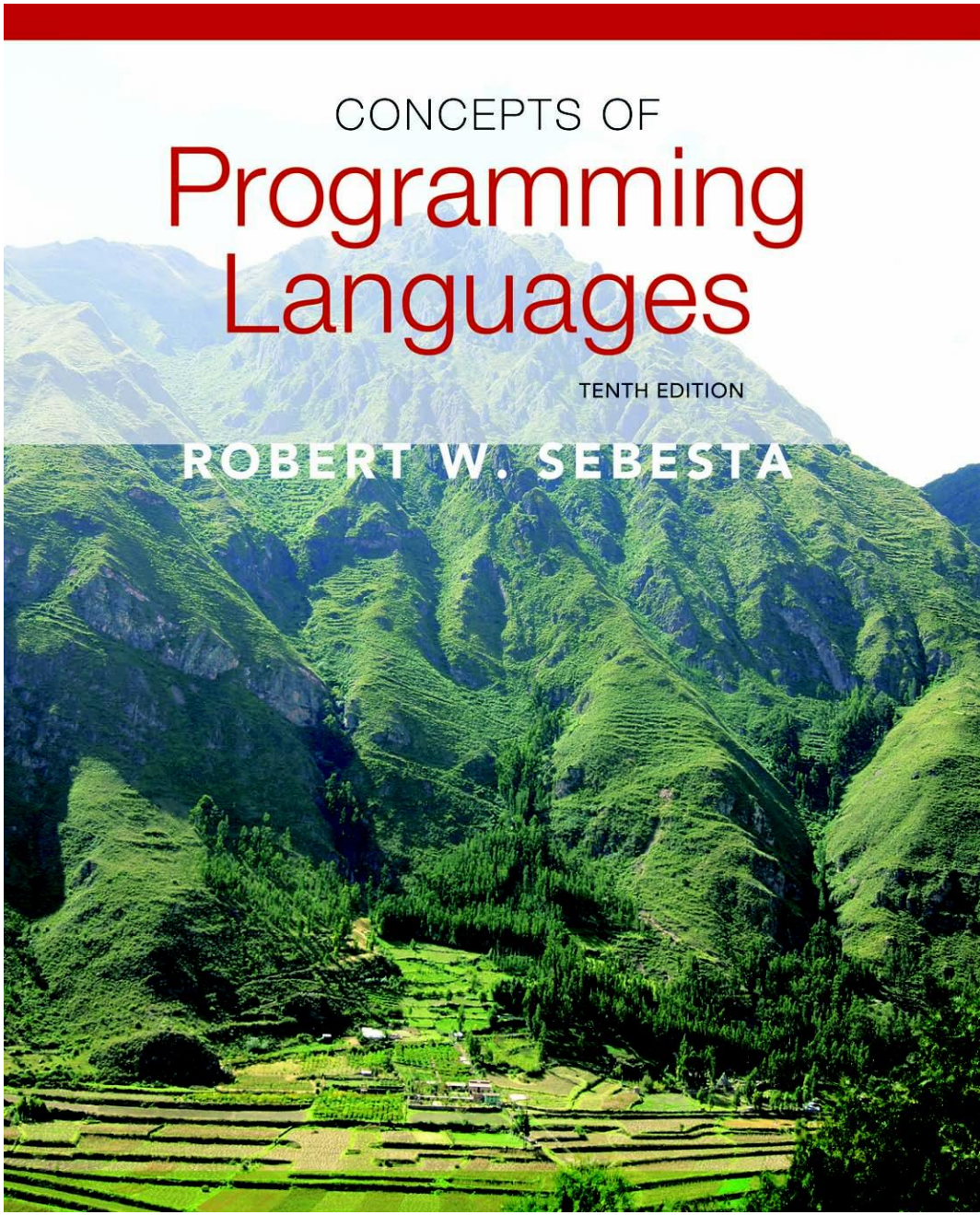
Bölüm 8

Komut Seviyeli Kontrol Yapıları



Bölüm 8'in Başlıkları

- Giriş
- Seçme Komutları
- Döngü (iteratif) Komutlar
- Koşulsuz Atlama Komutları
- Korunan Komutlar
- Sonuçlar

Akış Kontrolünün Seviyeleri

- İfadeler içinde(Bölüm 7)
- Program Birimleri Arasında(Bölüm 9)
- Program Komutları Arasında(Bu Bölümde)

Kontrol Komutları: Gelişimi

- FORTRAN I kontrol komutları (aritmetik if) doğrudan IBM 704 donanımını tasarlayanlar tarafından hazırlanmıştır.
- 1960lardan 70lerin ortalarına kadar bu konudaki çalışmalar devam etmiştir.
- Önemli teorem: Bütün akış diyagramlarının bir mantıksal döngü ve bir de iki yön
- seçmeli mantıksal ifadelerle kodlanabileceği ispat edilmiştir (Böhm ve Jacopini,
- 1966)

Kontrol Yapısı

- Bir kontrol yapısı (control structure) bir kontrol komutu ve onunkontrolündeki komutlardan oluşur.
- Tasarım Sorunu
  - Kontrol yapısının birden çok girişi var mıdır ?

Seçme Komutları

- Bir seçme komutu (selection statement) yürümekte olan programda iki veya daha fazla yoldan birini seçmemizi sağlar.

İki sınıfa ayrılır:

- İki yollu seçiciler
- Çok yollu seçiciler

İki Yollu Seçme Komutları

Genel şekli:

if <kontrol ifadesi>

then <ifade> else <ifade>

Tasarımla ilgili hususlar:

- Kontrol ifadesinin şekli ve tipi ne olacak?
- **"then" ve "else" terimleri nasıl belirlenecek?**
- İç içe geçmiş seçicilerin anlamları nasıl belirlenecek?

Kontrol İfadeleri

- Eğer ayrılmış sözcükler (reserved word) yada diğer sentatik işaretleyiciler kullanılmamışsa, kontrol ifadeleri parantez içerisinde belirtilir.Örn: if(stmnts)
- C89, C99, Python, ve C++, kontol ifadeleri aritmetik ifadelerden oluşabilir.
- Diğer dillerin çoğunda kontrol ifadeleri Boolean veri tipi olmalıdır.

Cümle Formu

- Çoğu çağdaş dilde, then ve else bloklarının içerisinde basit veya birleşik komutlar kullanılabilir.
- Perl'de tüm cümleler ayraçlarla sınırlandırılmış olmalıdır. (Ayraçların içerisindeki kodlar birleşik olmalıdır.)
- Fortran 95, Ada, Python, ve Ruby'de cümleler kod dizilerinden oluşurlar.
- Python cümleleri tanımak için çentik(“”) kullanır

```
if x > y :
  x = y
  print "x was greater than y"
```

Yuvalama Seçiciler

- Java örneği

```
if (sum == 0)
  if (count == 0)
      result = 0;
else result = 1;
```

- **Else hangi if'e ait**?
- Java'nın statik semantik kuralı: Else kendinden bir önceki (en yakın) if'e aittir.

Yuvalama Seçiciler (devamı)

- Alternatif bir semantik elde edilmek istenirse, birleşik komutlar (iç içe komutlar) kullanılabilir.

```
if (sum == 0) {
 if (count == 0)
      result = 0;
}
else result = 1;
```

- C, C++, ve C#'da üstteki çözüm kullanılır.

Yuvalama Seçiciler (devamı)

- Ruby'deki komut dizileri ise aşağıdaki gibidir.

```
if sum == 0 then
   if count == 0 then
      result = 0
   else
      result = 1
   end
end
```

Yuvalama Seçiciler (devamı)

- Python

```
if sum == 0 :
   if count == 0 :
      result = 0
   else :
      result = 1
```

Seçme İfadeleri

- ML, F#, ve LISP seçici bir ifadedir.
- F#

```
let y =
  if x > 0 then x
  else 2 * x
```

  - Eğer if ifadesi bir değer döndürüyorsa else ifadesi de olmalıdır.(ifade değerdense bir çıktı üretmelidir.)

Çoklu Seçme Komutları

- Bir programdaki akışı belirlemek için ikiden fazla yol olduğu zaman **çoklu seçim komutları kullanılır.**

Tasarımla ilgili hususlar:

1. Kontrol ifadesinin tipi ve şekli nasıl olacak?
2. Seçilebilir bölümler nasıl belirlenecek?
3. Programın çoklu yapıdaki akışı sadece bir bölge ile mi sınırlı olacak?
4. Seçimde temsil edilmeyen ifadelerle ilgili ne yapılacak?

Çoklu Seçme Komutları: Örnekler

- C, C++, Java ve JavaScript

```
switch (expression) {
  case const_expr1: stmt1;
  ...
  case const_exprn: stmtn;
  [default: stmtn+1]
}
```

Çoklu Seçme Komutları: Örnekler

- Tasarım yaparken C'nin switch kodu seçilirse
  1. Kontrol ifadeleri yalnızca tamsayı olabilir.
  2. Seçilebilir segmentler komut dizileri,bloklar veya bileşik komutlar olabilir.
  3. Herhangi bir segment numarası çalıştırılabilir bir yapı olabilir.
  4. `Default` cümlesi tanımlanmayan değerler için kullanılır.

Çoklu Seçme Komutları: Örnekler

- 1. C gibidir, sadece birden çok kısmın yürütülmesine izin vermez.
- 2. Her kısmın mutlaka "break" veya "goto" ile sonlandırılması gerekir.

```
switch (indeks) {
case 1: goto case 3;
case 3: tek +=1;
toplamtek += indeks;
break;
case 2: goto case 4;
case 4: cift += 1;
toplamcift += indeks;
break;
default: Console.WriteLine("switch içinde hata, indeks = %d\n",
indeks);
}
```

Çoklu Seçme Komutları: Örnekler

- Ruby'nin iki farklı case komutu vardır–Yalnız birinden bahsedeeğiz.

```
leap = case
    when year % 400 == 0 then true
    when year % 100 == 0 then false
    else year % 4 == 0
    end
```

Çoklu Seçicilerin Uygulanması

- Yaklaşımlar:
  - Çok koşullu dallanmalar
  - Bir tabloda durum değerleri(case değerleri) saklanır ve doğrusal arama kullanılarak değerler getirilir.
  - Eğer birden fazla case varsa Hash tablosunun case değerleri kullanılabilir.
  - If the number of cases is small and more than half of the whole range of case values are represented, an array whose indices are the case values and whose values are the case labels can be used

Çoklu Seçmede IF Kullanımı

- Çoklu seçicilerde if kullanımı aşağıdaki Python örneğindeki gibi if-else if yapısı kullanılarak yapılır.

```
if count < 10 :
  bag1 = True
elif count < 100 :
  bag2 = True
elif count < 1000 :
  bag3 = True
```

Çoklu Seçmede IF Kullanımı

- Python örneğini Ruby ile geröekleştirmek.

```
case
  when count < 10 then bag1 = true
  when count < 100 then bag2 = true
  when count < 1000 then bag3 = true
end
```

Çoklu Seçicilerin Şeması

- Genel çağırma formu ŞART:

```
(ŞART
  (YÜKLEM1 İFADE1)
   …
  (YÜKLEMn İFADEn)
  [(ELSE expressionn+1)]
 )
```

  - Else deyimi isteğe bağlıdır; yukarıda kullanılan else deyimi true ile eşanlamlıdır.
  - Her yüklem-ifade çifti parametredir.
  - Anlambilim:Şart ifadesinin değeri ilk yüklem ifadesinin değeriyle ilgiliyse şart doğrudur.

Döngülü Komutlar (Iterative Statements)

- Bir komutun veya bir komut grubunun tekrarlanan yürütülmesi döngü veya özyineleme ile elde edilir.
- Döngünün komutunun ne şekilde olacağı iki temel sorunun cevabına göre belirlenir:

1. Döngü kontrolü nasıl yapılacak? Mantıksal, sayarak veya ikisi birden.
2. Döngü mantıksal ifadesi nerede olacak? Başlangıçta mı, sonda mı, programcı mı karar verecek?

Sayaç Kontrollü Döngüler

- Sayaç kontrollü döngülerde, başlangıç değeri ,bitiş değeri ve artış miktarı belirtilerek adım kontrolü yapılır.
- Tasarım Sorunları:

1. Döngü değişkeninin tipi ve kapsamı nedir?
2. Döngü değişkeninin döngü bittiğinde değeri nedir?
3. Döngü değişkeninin değeri döngü içinde değiştirilebilmeli midir? Değiştirilebilirse bu döngü kontrolünü etkilemeli midir?
4. Döngü parametreleri bir kez mi değerlendirilmelidir yoksa her döngüde mi?

Sayaç Kontrollü Döngüler: Örnekler

- Ada

```
for var in [reverse] discrete_range
loop                    ...
end loop
```

- Tasarım Seçenekleri:
  - Döngü değişkeninin veri tipi hangi aralıkta.
  - Döngü değişkenin döngü dışına çıkmaması
  - Döngü değişkeni döngü içinde değiştirilemez (Örn: foreach döngüleri) fakat aralığı değiştirilebilir, aralığın değişmesi döngü kontrolüne etki etmez.
  - Döngü aralığı yalnızca bir defalığına değerlendirilebilir.
  - Döngü içerisinde dallanma komutları kullanılamaz.

Sayaç Kontrollü Döngüler: Örnekler

- C-tabanlı diller

  `for ([expr_1] ; [expr_2] ; [expr_3]) statement`

  – The expressions can be whole statements, or even statement sequences, with the statements separated by commas Yukarıda C tabanlı dillerde for yapısını vermiştir. İfadelerin arasına virgül konarak ifade sayısı artırılabilir. Bloklar noktalı virgülle ayrılmıştır.
  - Birden çok komutlu ifadelerin değeri, son durumdaki ifadenin değerine eşittir.
  - İkinci ifadede herhangi bir değer yoksa döngü sonsuza dek döner.
- Tasarım Seçenekleri:
  - Belirgin bir döngü değişkeni yoktur.
  - Döngü içerisinde her şey değişebilir.
  - İlk ifade yalnızca bir kez değerlendirilirken şart ifadesi adım sayısı kadar değerlendirilir.
  - C Döngü içerisinde dallanma komutlarına izin verir (goto gibi)

Sayaç Kontrollü Döngüler: Örnekler

- C++ iki şekilde C ile farklılık gösterir :
  1. Kontrol ifadesi Boolean olabilir.
  2. Başlangıç ifadesi değişken tanımları içerebilir.
- Java ve C#
  - C++'tan farkı,kontrol ifadesinin Boolean olma zorunluluğudur.

Sayaç Kontrollü Döngüler: Örnekler

- Python

  `for` döngü değişkeni **`in`** nesne**`:`**
  - döngü gövdesi

  [**`else`**`:`
  - else cümlesi]

  - Nesne sıklıkla bir aralığı(range) temsil eder.Liste değerleri ise parantez içersinde (`[2, 4, 6]`), yada **range** fonksiyonu kullanılarak ifade edilir.(**`range`**`(5)`, `0, 1, 2, 3, 4 değerlerin döndürür.`

  - The loop variable takes on the values specified in the given range, one for each iteration Döngü değişkeni aralıkta(range) gösterilen değerleri alırlar.

  - Döngü normal bir şekilde sona ererse isteğe bağlı olarak else bloğu yürütülür.

Sayaç Kontrollü Döngüler: Örnekler

- F#
  - Sayaçları değişkenler gerektiren ve fonksiyonel dillerde değişkenleri yok olduğundan, sayaç kontrollü döngü özyinelemeli fonksiyonlar ile simüle edilmelidir.

```
let rec forLoop loopBody reps =
  if reps <= 0 then ()
  else
    loopBody()
    forLoop loopBody, (reps - 1)
```

forloop adında loopbody parametrelerini kullanan recursive bir fonsiyon tanımlanır.

  - `()` Hiçbir olay gerçekleşmiyor ve hiçbir değer dönmüyor anlamına gelmektedir.

Mantıksal-Kontrollü Döngüler

- Tekrarlamalar Boolean tabanlı ifadelerle kontrol edilir.
- Tasarım Sorunları:
  - Öntest mi yoksa sontest mi?
  - Sayaç kontrollü döngülerin özel bir halimi olacak yoksa farklı spesifik bir döngü yapısı mı?

Mantıksal-Kontrollü Döngüler: Örnekler

- C ve C++ dillerinde hem öntest hemde sontest yapısını kullanan döngü ifadeleri vardır.

```
while (control_expr)          do
    loop body                     loop body
                              while (control_expr)
```

  - C ve C++'ta parantezin içerisine mantıksal kontrol ifadesi yazılır.

- Java kontrol ifadesinin Boolean olma zorunluluğu dışında C ve C++'a benzer.(Ve döngü yapısına sadece başlangıçta girilir.- Java'da `goto` deyimi yoktur.C# ta ise döngü yapısına ortadan da girilebilir.)

Mantıksal-Kontrollü Döngüler: Örnekler

- F#
  - Sayaç kontrollü döngülerde olduğu gibi mantıksal kontrollü döngülerde de recursive fonksiyonlar kullanılır.

```
let rec whileLoop test body =
   if test() then
     body()
     whileLoop test body
   else ()
```

  - Whileloop özyinelemeli fonksiyonu test ve body parametreleriyle tanımlanır. Test parametresi mantıksal kontrolu yapar body ise kontrolün doğru olduğu durumdaki yapılacak işlemi temsil eder.

Kullanıcı Tarafından Yerleştirilen döngü Kontrol Düzenekleri

- Programcılar döngüyü farklı bir şekilde kontrol etmek için komutları döngünün farklı bölgelerine yerleştirebilirler.
- Döngüler için basit tasarımlar yapılabilir. (örn, `break`)
- Tasarım Problemleri:

  1. Koşul ve döngüden çıkış tek bir kısım mı olmalı?
  2. Kontrol birden çok döngüden dışarı çıkabilmeli mi?

Kullanıcı Tarafından Yerleştirilen döngü Kontrol Düzenekleri

- C , C++, Java, Perl ve C#'ta koşulsuz, etiketsiz bir kademe çıkış **break.**
- Java, C#: bir öncekine ilaveten, koşulsuz etiketli birkaç kademeli çıkış **break.**
- Perl: koşulsuz etiketli birkaç kademeli çıkış **last.**
- Bütün bu dillerde ayrıca, döngüyü bitirmeyen, ancak kontrol kısmına gönderen, **break ile aynı özelliklerde continue.**
- Java ve Perl de **continue komutlarının etiketi de olabilir.**

C# örneği:

```
dongu1:
for(satir = 0; satir<satirsayi; satir++)
for(sutun = 0; sutun<sutunsayi; sutun++) {
toplam += mat[satir][sutun];
if(toplam > 1000.0) break dongu1;
}
```

C örneği

```
while (toplam < 1000) {
sonraki(deger);
if (deger < 0) continue;
toplam += deger;
}
```

Veri Yapılarına Dayalı Döngüler

- Kavram: bir veri yapısını (data structure) ve sırasını döngünün kontrolü için kullanmak.
- Kontrol mekanizması: varsa veri yapısının bir sonraki elemanını dönen bir fonksiyon, yoksa döngü biter.
- C'de **for bu amaçla kullanılabilir**:
- örneğin: **for (p=hdr; p; p=sonraki(p)){ ... }**

```
    for (p=root; p==NULL; traverse(p)){
  ...
}
```

Veri Yapılarına Dayalı Döngüler (devamı)

- PHP
  - `current` pointer'ın o andaki işlediği dizi elemanını temsil eder.
  - `next current` değerini bir sonraki elemana taşır.
  - `reset current` değerini dizinin ilk elemanına taşır.

- Java 5.0 (Foreach gibi davranan For kullanımı)

  Diziler ve diğer sınıflarda kullanılan döngü arayüzleri örn., `ArrayList`

```
for (String myElement : myList) { … }
```

Veri Yapılarına Dayalı Döngüler (devamı)

- C# ve F# (ve diğer .NET dilleri)'ta Java 5.0'a benzeyen kapsamlı kütüphane sınıfları vardır. (diziler,listeler,yığınlar ve kuyruklar).Bu yapılarda bulunan elemanların tümünü **`foreach`** döngüsüyle gezebiliriz.

```
List<String> names = new List<String>();
names.Add("Bob");
names.Add("Carol");
names.Add("Ted");
foreach (Strings name in names)
    Console.WriteLine ("Name: {0}", name);
```

Veri Yapılarına Dayalı Döngüler (devamı)

- Ruby blokları kod dizileridir ve bloklar **`do end`** kodları arasında tanımlanmıştır
  - Bloklar döngü oluşturabilmek için metotlarla beraber kullanılabilirler.
  - Önceden tanımlanmış döngü metotları (`times`, `each`, `upto`):

```
3.times {puts "Hey!"}
list.each {|value| puts value}
```

(`list` bir dizi; `value` ise bir blok parametresi)

```
1.upto(5) {|x| print x, " "}
```

  - Ruby bir for komutuna sahiptir fakat for komutunu çalıştırabilmek için upto metoduna dönüştürmesi gerekmektedir.

Veri Yapılarına Dayalı Döngüler (devamı)

- Ada
  - Ada dilinde döngü aralığı ile dizi indisi arasında ilişki kurulabilir.

```
subtype MyRange is Integer range 0.99;
MyArray: array (MyRange) of Integer;
for Index in MyRange loop
 ...MyArray(Index) ...
end loop;
```

Koşulsuz Dallanma

- Programın akışı değiştirilebilir, her türlü kontrol komutu "goto" ve seçici ile yapılabilir. Çok etkili bir komut.
- 60 ve 70'li yılların en ateşli tartışma konusu olan goto komutu bazı programcılara göre kaos komutudur. Çünkü programı rastgele dallandırdığı için olası hatalara sebep olabiliyor.
- Temel sorun: Okunabilirlik
- Bazı dillerde goto komutu kullanılamaz. (Java)
- C#'ta `goto` komutu kullanılabilir. (switch bloguyla beraber)
- Döngü çıkış komutları (**break, last) kamufle edilmiş goto'lardır.**

Güvenlikli Komutları

- Dijkstra tarafından tasarlanmıştır.
- Yeni programlama metodolojilerini geliştirme esnasında desteklemek ve onlara kaynak sunmak.
- Eşzamanlı programlama için iki dilsel mekanizmayı temel alır. (CSP ve Ada)
- Temel Fikir: Değerlendirme sırası önemli değilse, programı tek belirtmeniz gerekir

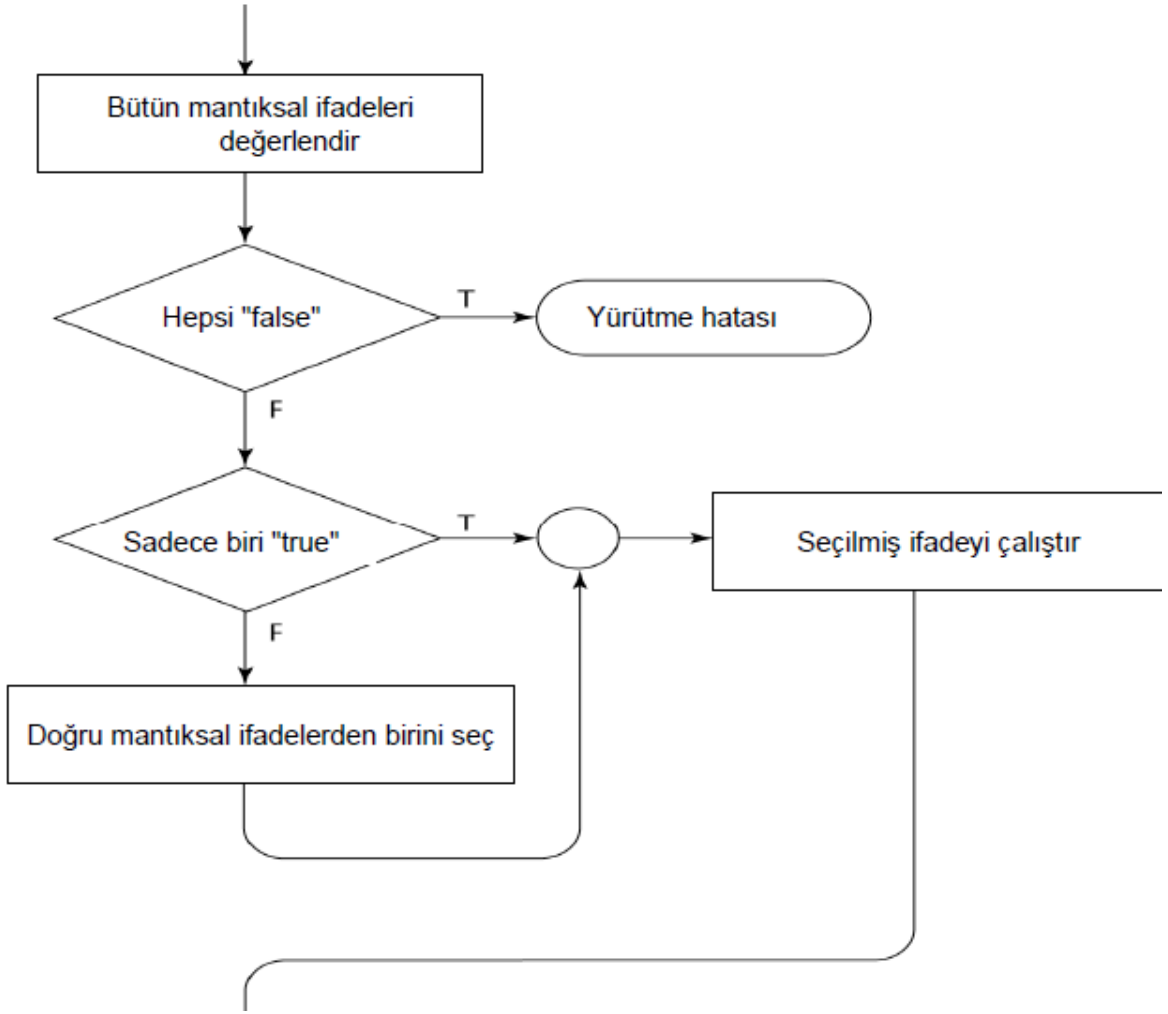
Güvenlikli Seçme Komutları

- Form

```
if <Boolean expr> -> <statement>
[] <Boolean expr> -> <statement>
 ...
[] <Boolean expr> -> <statement>
fi
```

- Anlambilim: yapıya ulaşıldığında
  - Tüm boolean ifadeleri değerlendirilir.
  - Eğer birden fazla doğru ifade varsa non-deterministik bir algoritma seçilmelidir.
  - Eğer doğru ifade yoksa çalışma zamanı hatası verilir.

Güvenlikli Seçme Komutları



Güvenlikli Döngü Komutları

- **Formu**

  do <Boolean> -> <statement>
  [] <Boolean> -> <statement>
  ...
  [] <Boolean> -> <statement>
  od

- Anlambilim: Her bir adım için
  - Tüm Boolean ifadeleri değerlendirilir.
  - Eğer birden fazla doğru varsa seçme komutlarındaki gibi non-deterministik bir seçim yapılır ve döngünün başına geri dönülür.
  - Eğer hepsi yanlışsa döngüden çıkılır.

Güvenlikli Komutlar: Gerekçe

- Kontrol deyimleri ve program doğrulama arasında güçlü bir bağlantı vardır.
- Goto komutlarının doğrulanması imkansızdır.
- Doğrulama seçim ve mantıksal öntest döngüleri için mümkündür.
- Güvenlikli kontrollerin doğrulanması daha basittir.

Sonuçlar

- Bu kısımda bahsettiğimiz seçme ve ön kontrollü döngüler dışındaki diğer kontrol komutları dilin büyüklüğü ile kolay yazılabilirlik arasındaki tercih sorunudur.
- Fonksiyonel ve mantıksal dillerdeki kontrol yapıları bu kısımda bahsettiğimiz yapılardan farklıdır.