BMB214 Programlama Dilleri Prensipleri

Ders 8. Statement-Seviye Kontrol Yapıları (Statement-Level Control Structures)

Erdinç Uzun
Tekirdağ Namık Kemal Üniversitesi
Bilgisayar Mühendisliği Bölümü

Konular

- Selection Statements
- Iterative Statements
- Unconditional Branching (Koşulsuz Dallanma)
- Guarded Commands (Korunan Komutlar)

Kontrol (Control) Statements: Evrim Başlıyor...

- FORTRAN I kontrol statement'ları doğrudan IBM 704 donanımına dayanıyordu
- 1960'larda konu hakkında birçok araştırma ve tartışma
  - Önemli bir sonuç: Akış şemaları tarafından temsil edilen tüm algoritmaların yalnızca iki yönlü seçim ve ön test mantıksal döngülerle kodlanabildiği kanıtlanmıştır.

Kontrol Yapısı (Control Structure)

- Bir kontrol yapısı, bir kontrol statement'ı ve yürütülmesini kontrol ettiği ifadelerdir.
  - Tasarım sorusu
  - Bir kontrol yapısının birden fazla girişi olmalı mı?

Selection Statements

- Bir selection (seçim) statement, iki veya daha fazla yürütme yolu arasında seçim yapma imkanı sağlar
- İki genel kategori:
  - İki yönlü seçiciler (Two-way selectors)
  - Çok yollu seçiciler (Multiple-way selectors)

İki yönlü seçiciler (Two-way selectors)

- Genel Formu

  `if` control_expression

  `then` clause

  `else` clause

- Tasarım zorlukları
  - Kontrol statement'ının biçimi ve türü nedir?
  - Then ve else cümleleri nasıl belirtilir?
  - İç içe geçmiş seçicilerin anlamı nasıl belirtilmelidir?

Control Expression (Kontrol İfadesi)

- O zaman ayrılmış kelime (reserved Word) veya başka bir sözdizimsel işaretçi then clause'lar tanıtmak için kullanılmazsa, kontrol ifadesi parantez içine yerleştirilir
- C89, C99, Python ve C++'da kontrol ifadesi aritmetik olabilir
- Diğer dillerin çoğunda, kontrol ifadesi Boolean olmalıdır

Clause Formu

- Birçok çağdaş dilde, then ve else clause'ları tek ifadeler veya bileşik ifadeler olabilir
- Perl'de tüm clause'lar ayraçlarla sınırlandırılmalıdır (bileşik olmalıdır)
- Python ve Ruby'de yan clause'lar ifade dizileridir
- Python, cümleleri tanımlamak için girinti kullanır

```
if x > y :
    x = y
    print "x, y'den büyüktür…"
```

Nesting Selectors (İç içe geçmiş Seçiciler)

- Java Örneği

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

- Hangi if else'i alır
- Java statik anlambilim kuralı: else en yakın önceki if ile eşleşir

Nesting Selectors…

- Alternatif bir semantiği zorlamak için bileşik ifadeler kullanılabilir:

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

- Hangi if else'i alır
- Yukarıdaki çözüm C, C++ ve C#'da kullanılır

Nesting Selectors…

- Statement dizisi olarak clause'lar: Ruby

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

Nesting Selectors...

- Python - Girinti Çözümü

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

Selector Expressions (Seçici İfadeleri)

- ML, F # ve Lisp'de seçici bir ifadedir.
  - F# örneği:

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

  - If ifadesi bir değer döndürürse, başka bir else clause'u olması gerekir (ifade, değeri olmayan bir birim türü üretebilir). Then ve else cümleleri tarafından döndürülen değerlerin türleri aynı olmalıdır.
    - Java'da if içeren bir fonksiyon tanımlayıp, id bölümüne return tanımlayın else bölümüne return tanımlamayın. Sonuç?

Çok yollu seçiciler (Multiple-way selectors)

- Herhangi bir sayıda stamement'tan veya statement grubundan birinin seçilmesini sağlayın.
- Tasarım Zorlukları:
  - Kontrol ifadesinin biçimi ve türü nedir?
  - Seçilebilir segmentler nasıl belirtilir?
  - Yapı boyunca yürütme akışı, yalnızca tek bir seçilebilir segmenti içerecek şekilde mi sınırlı?
  - Durum değerleri nasıl belirtilir?
  - Temsil edilmeyen ifade değerleri hakkında ne yapılır?

Multiple-Way Selection…
Örnek

- C, C++, Java ve JavaScript

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

Multiple-Way Selection...
Örnek

- C'de switch statement'ı ifadesi için tasarım kuralları
  - Kontrol ifadesi yalnızca bir tamsayı türü olabilir
  - Seçilebilir segmentler statement dizileri, bloklar veya bileşik statement'lar olabilir
  - Yapının bir uygulamasında herhangi bir sayıda segment yürütülebilir (seçilebilir segmentlerin sonunda örtük bir dal yoktur)
  - default clause, temsil edilmeyen değerler içindir (defult yoksa, tüm statement'lar hiçbir şey yapmaz)

Multiple-Way Selection...
Örnek

- C#
- Birden fazla segmentin örtülü (implicit) yürütülmesine izin vermeyen statik bir anlam kuralına sahip olmasıyla C'den farklıdır.
- Her seçilebilir bölüm koşulsuz bir dalla (unconditional branch) bitmelidir (goto veya break)
- Ayrıca, C#'da kontrol ifadesi ve case sabitleri string olabilir

Multiple-Way Selection...
Örnek

- Ruby'nin iki tür case statement'ı vardır - yalnızca birini ele alınmıştır.

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

Birden Çok Seçici Uygulaması

- Yaklaşımlar:
  - Birden çok koşullu dal
  - Case değerlerini bir tabloda saklayın ve tabloda doğrusal bir arama kullanın
  - Ondan fazla durum olduğunda, durum değerlerinin bir hash tablosu kullanılabilir (Hızlandırmak için)
  - Case sayısı küçükse ve tüm vaka değerleri aralığının yarısından fazlası temsil ediliyorsa, indisleri case değerleri ve değerleri case etiketleri olan bir array kullanılabilir

Multiple-Way Selection
if

- Birden çok seçici, else-if clasuse'larını kullanılarak iki yönlü seçicilere doğrudan uzantılar yaparak kullanabilir. Örneğin Python'da (if, elif, else):

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

Multiple-Way Selection
Ruby

- Ruby örneği (case, when, then, end):

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

Multiple-Way Selection Scheme

- Scheme'da COND'un genel formu

```
(COND
  (predicate1 expression1)
   …
  (predicaten expressionn)
  [(ELSE expressionn+1)]
 )
```

  - ELSE clause isteğe bağlıdır; ELSE, true ile eşanlamlıdır
  - Her bir predicate-expression çifti bir parametredir
  - Anlambilim: COND değerlendirmesinin değeri, doğru (true) olan ilk predicate ifadesiyle ilişkili ifadenin değeridir

Iterative Statements

- Bir statement veya compund statement tekrarlayarak yürütülmesi, iterasyon veya özyineleme ile gerçekleştirilir.
- İterasyon kontrol ifadeleri için genel tasarım sorunları:
  - İterasyon nasıl kontrol edilir?
  - Döngüde (loop) kontrol mekanizması nerede?

Counter-Controlled Loops (Sayaç Kontrollü Döngüler)

- Sayma iterasyon statement'ı bir döngü değişkeni, başlangıç (initial) ve sonun belirtilmesi için bir terminal ve adım boyutu (stepsize) değerlerine sahiptir.
- Tasarım Zorluğu:
  - Döngü değişkeninin türü ve kapsamı nedir?
  - Döngü değişkeninin veya döngü parametrelerinin döngü gövdesinde değiştirilmesi yasal olmalı mı ve öyleyse, değişiklik döngü kontrolünü etkiliyor mu?
  - Döngü parametreleri yalnızca bir kez mi yoksa her iterasyon için bir kez mi değerlendirilmelidir?

Counter-Controlled Loops
Örnekler

- C tabanlı diller

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

  - İfadeler, statement'ların virgülle ayrılmış olduğu tam ifadeler veya hatta ifade dizileri olabilir
  - Çoklu ifadenin değeri, ifadedeki son ifadenin değeridir
  - İkinci ifade yoksa, sonsuz bir döngüdür
- Tasarım seçenekleri:
  - Açık (explicit) döngü değişkeni yok
  - Döngüde her şey değiştirilebilir
  - İlk ifade bir kez değerlendirilir, ancak diğer ikisi her iterasyonda değerlendirilir
  - C'deki bir for döngüsünün gövdesine dalmak yasaldır

Counter-Controlled Loops
Örnekler

- C++, C'den iki yönden farklıdır:
  - Kontrol ifadesi ayrıca Boole olabilir
  - İlk ifade değişken tanımları içerebilir (kapsam, tanımdan döngü gövdesinin sonuna kadardır)
- Java ve C# kontrol ifadesinin Boole olması gerektiğinden C++'dan farklıdır

Counter-Controlled Loops
Örnekler

- Python

```
for loop_variable in object:
    - loop body
   [else:
    - else clause]
```

  - Nesne (Object) genellikle, parantez içindeki bir değer listesi ([2, 4, 6]) veya aralık (range) fonksiyonuna (range(5) → 0, 1, 2, 3, 4 döndüren) değerdir.
  - Döngü değişkeni, her iterasyon için bir tane olmak üzere, verilen aralıkta belirtilen değerleri alır
  - İsteğe bağlı olan else clause, döngü normal şekilde sona ererse yürütülür.

Counter-Controlled Loops
Örnekler

- F#'ta sayaçlar (counters) değişken gerektirdiğinden ve fonksiyonel diller değişkenlere sahip olmadığından, karşı kontrollü döngülerin özyinelemeli fonksiyonlarla simüle edilmesi gerekir

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

- Bu, döngü için özyinelemeli fonksiyon, loopBody (döngünün gövdesini tanımlayan bir fonksiyon) parametreleri ve yineleme sayısı ile tanımlar.
- () hiçbir şey yapma ve hiçbir şey döndürme anlamına gelir

Mantıksal Kontrollü Döngüler (Logically-Controlled Loops)

- Tekrarlama kontrolü, bir Boole ifadesine dayanır
- Tasarım Zorlukları:
  - Ön test mi yoksa son test mi?
  - Mantıksal olarak kontrol edilen döngü, sayma döngüsü statement'ının özel bir durumu mu yoksa ayrı bir statement mi olmalıdır?

Mantıksal Kontrollü Döngüler...
Örnekler

- C ve C++, kontrol ifadesinin aritmetik olabileceği hem ön test hem de son test formlarına sahiptir:
  - while (control_expr) loop body
  - do loop body
    while (control_expr)
- Java, C ve C++ gibidir, ancak kontrol ifadesinin Boolean olması gerekir
  - Java'da goto yoktur

Mantıksal Kontrollü Döngüler...
Örnekler

- F #
- Karşı kontrollü döngülerde olduğu gibi, mantıksal olarak kontrol edilen döngüler özyinelemeli fonksiyonlarla simüle edilebilir

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

- Bu, her iki fonksiyonun test ve body parametreleriyle whileLoop özyinelemeli fonksiyonunun tanımlar. test, kontrol ifadesini tanımlar.

Kullanıcı Tarafından Yerleştirilen Döngü Kontrol Mekanizmaları (User-Located Loop Control Mechanisms)

- Bazen programcıların döngü kontrolü için bir konuma karar vermesi uygundur (döngünün üstü veya altı dışında)
- Tek döngüler için basit tasarım (ör. break)
- İç içe döngüler için tasarım zorlukları
  - Koşullu çıkışın bir parçası olmalı mı?
  - Kontrol birden fazla döngüden aktarılabilir mi olmalı?

User-Located Loop Control Mechanisms Örnekler

- C, C++, Python, Ruby ve C# koşulsuz etiketlenmemiş çıkışlara sahiptir (break)
- Java ve Perl koşulsuz etiketli (unconditional labeled) çıkışlara sahiptir (Java'da break, Perl'de last olarak)
- C, C++ ve Python, geçerli yinelemenin geri kalanını atlayan, ancak döngüden çıkmayan etiketlenmemiş bir kontrol ifadesine sahiptir (continue).
- Java ve Perl, devam etmenin (continue) etiketlenmiş sürümlerine sahiptir.

İterasyon Tabanlı Veri Yapıları (Iteration Based on Data Structures)

- Bir veri yapısındaki öğe sayısı döngü yinelemesini kontrol eder
- Kontrol mekanizması, bir sonraki öğeyi, eğer varsa, seçilen bir sırayla döndüren bir yineleyici fonksiyona yapılan bir çağrıdır; yoksa döngü sonlandırılır
- C'ler, kullanıcı tanımlı bir iteratör (user-defined iterator) oluşturmak için kullanılabilir:

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

İterasyon Tabanlı Veri Yapıları...

- PHP
  - Dizide geçerli (current) değer bir öğeyi (element) işaret eder.
  - next bir sonraki öğeyi geçerli yapar
  - reset geçerli değeri ilk öğeye taşır
- Java 5.0 (for kullanır)
  - Örneğin ArrayList dolaşılabilir
  - for (String myElement : myList) { … }

İterasyon Tabanlı Veri Yapıları…

- C # ve F # (ve diğer .NET dilleri), Java 5.0 gibi generic library sınıfları (arrays, lists, stacks ve queues).
- Forach statement'ı ile öğeleri dolaşabiliriz
- Kullanıcı tanımlı koleksiyonlar IEnumerator interface'i üzerinden döngü kurulabilir. (foreach)

```
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);
```

İterasyon Tabanlı Veri Yapıları...

- Ruby blokları, parantez veya **do** ve **end** ile ayrılmış kod dizileridir.
- Bloklar, iterasyon oluşturmak için metotlar ile kullanılabilir
- Önceden tanımlanmış iterasyon metotları (times, each, uptu):

```
3.times {puts "Hey!"}
list.each {|value| puts value}
(list bir dizidir, value )
1.upto(5) {|x| print x, " "}
```

- İterator'ler, uygulamalar tarafından da tanımlanabilen bloklarla gerçekleştirilir.

İterasyon Tabanlı Veri Yapıları…

- Ruby blokları ekli metot çağrılarıdır; parametreleri olabilir; metot bir **yield** statement'ı yürüttüğünde çalıştırılırlar

```
def fibonacci(last)
  first, second = 1, 1
  while first <= last
    yield first
    first, second = second, first + second
  end
end
puts "Fibonacci numbers less than 100 are:"
fibonacci(100) {|num| print num, " "}
puts
```

Ruby bir for statement'a sahip, ama Ruby **upto** metot çağrısı ile bunları dönüştürür.

Koşulsuz Dallanma (Unconditional Branching)

- Yürütme kontrolünü programda belirli bir yere aktarır
- 1960'lar ve 1970'lerde en hararetli tartışmalardan birini temsil etti
- Büyük endişe: Okunabilirlik
- Bazı diller goto statement desteklemez (ör. Java)
- C# goto statement sunar (switch statement'larda da kullanılabilir)
- Döngü çıkış (exit) statement'ı kısıtlanmıştır ve bir şekilde kamufle edilmiş goto'lar

Korunan Komutlar (Guarded Commands)

- Dijkstra tarafından tasarlandı (1975)
- Amaç: geliştirme sırasında doğrulamayı (doğruluğu) (verification (correctness)) destekleyen yeni bir programlama metodolojisini desteklemek
- Eşzamanlı programlama (concurrent programming) için iki dil mekanizmasının temeli (CSP'de)
- Temel Fikir: Eğer değerlendirme sırası önemli değilse, program bir tane belirtmemelidir

Korunan Komutlar
Selection

- Formu

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

- Anlambilim: yapıya ulaşıldığında,
  - Tüm Boole ifadelerini değerlendirin
  - Birden fazla doğruysa, belirleyici olmayan birini seçin
  - Hiçbiri doğru değilse, bu bir çalışma zamanı hatasıdır

Korunan Komutlar

- Formu

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

- Anlambilim: her iterasyon için
  - Tüm Boole ifadelerini değerlendirin
  - Birden fazlası doğruysa, belirleyici olmayan bir şekilde birini seçin; sonra döngüyü tekrar başlat
  - Hiçbiri doğru değilse, döngüden çık

Korunan Komutlar
Gerekçe

- Kontrol statement'ları ve program doğrulama arasındaki bağlantı birebirdir
- Goto statement'ı ile doğrulama imkansızdır
- Doğrulama yalnızca seçim ve mantıksal ön test döngüleri ile mümkündür
- Doğrulama, yalnızca korumalı komutlarla nispeten basittir