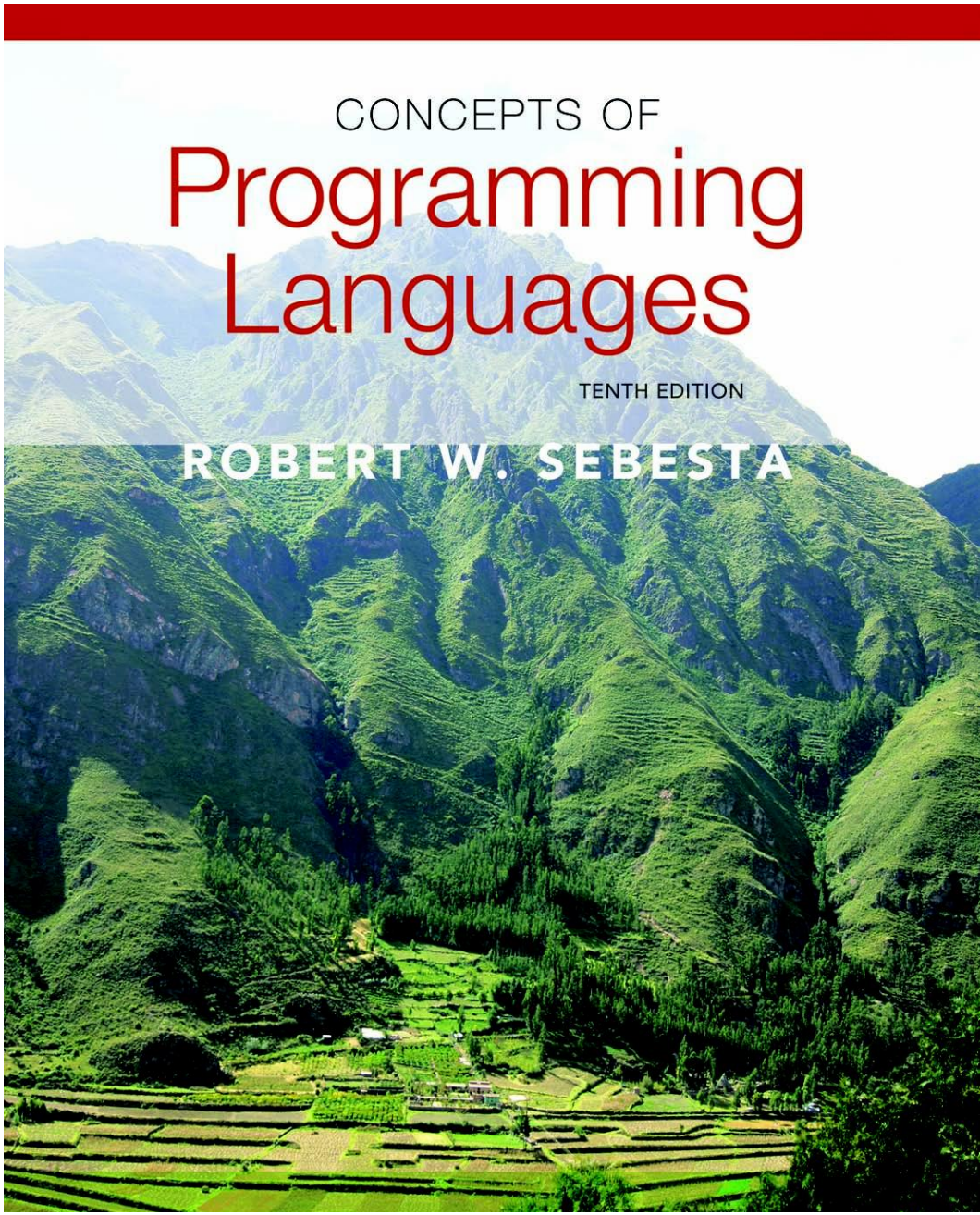
Bölüm 7

İfadeler ve Atama İfadeleri



Bölüm 7 Konular

- Giriş
- Aritmetik İfadeler
- Operatörlerin Aşırı Yüklenmesi
- Tip Dönüşümleri
- İlişkisel ve Mantıksal İfadeler
- Kısa Devre Tespiti
- Atama İfadeleri
- Karışık-Biçim Atamaları

Giriş

- İfadeler bir programlama dilinde hesaplamaları belirtmede temel araçtır.
- İfadelerin değerlendirmesini anlamak için, operatörlerin sırası ve işlenenlerin(operant) değerlendirmesine aşina olmamız gerekir.
- Emirsel dillerin temeli atama ifadeleridir.

Aritmetik İfadeler

- Aritmetik ölçüm ilk programlama dilinin gelişiminde kullanılan motivasyonlarından biri olmuştur.
- Aritmetik ifadeler; operatörler,operantlar,parantezler ve fonksiyon çağrılarından oluşur

Aritmetik İfadeler: Tasarım Sorunları

- Aritmetik İfadeler için Tasarım Sorunları
  - Operatörlerin öncelik kuralları?
  - Operatörlerin birleşilirlik kuralları?
  - Operantların sırasının değerlendirilmesi?
  - Operant değerlendirmenin yan etkileri?
  - Operatörlere aşırı yükleme?
  - İfadelerdeki tip karıştırılması?

Aritmetik İfadeler: Operatörler

- Unary(tekli) operatorün tek operantı vardır.
- Binary(ikili) operatorün iki operantı vardır.
- Ternary(üçlü) operatorün iki operantı vardır.
- N-ary(nli) operatörün n tane operantı vardır.

Aritmetik İfadeler: Operatör Öncelik Kuralları

- Operatör öncelik kuralları farklı öncelik seviyesindeki bitişik operatörlerdeki operatörlerin işlenme sırasını belirtir.
- Klasik öncelik seviyeleri
  - Parantezler
  - Tekli operatörler
  - ** (Eğer dil destekliyorsa, üs alma)
  - *, /(çarpma,bölme)
  - +, -(toplama,çıkarma)

Aritmetik İfadeler: Operatör Birleşilirlik Kuralı

- Bu kural aynı öncelik seviyesindeki bitişik operatörlerin işlenmesi sırasını belirtir.
- Tipik birleşilirlik kuralları
  - Soldan sağa, **(hariç, burada sağdan sola),
  - Zaman zaman tekli operatörlerin birleşilirliği sağdan sola olabilir (ör., FORTRAN)
- APL dili farklıdır; bu dilde tüm operatörler eşit önceliklere sahiptir ve operatörlerin birleşilirliği sağdan soladır.
- Öncelik ve birleşilirlik kuralları parantezlerle geçersiz kılınabilir.

Ruby ve Scheme'de İfadeler

- Ruby
  - Tüm aritmetik, ilişkisel, atama operatörleri, ve hatta dizi indeksleme,kaydırma ve bit şeklindeki mantık operatörleri metotlar olarak sağlanır
  - Bunun sonuçlarından biri bu operatörlerin uygulama programları tarafından geçersiz kılınabilmesidir.
- Scheme (ve Common LISP)
  - Tüm aritmetik ve mantık işlemleri belirgin bir şekilde alt programlar tarafından çağrılır.
  - `a + b * c` ifadesi `(+ a (* b c))` olarak kodlanır.

Aritmetik İfadeler: Şartlı İfadeler

- Şartlı İfadeler
  - C-tabanlı diller (ör., C, C++)
  - Bir örnek:

```
average = (count == 0)? 0 : sum / count
```

  - Aşağıdakine eş değerdir:

```
if (count == 0)
   average = 0
else
   average = sum /count
```

Aritmetik İfadeler: operant Değerlendirme Sırası

- *operant değerlendirme sırası*
  1. Değişkenler: Bellekten değerini al
  2. Sabitler: bazen bellekten alınır bazen makine dili komutundadır.
  3. Parantezli ifadeler: İçindeki tüm operant ve operatörler ilk olarak işlenir
  4. En ilginç durum bir operantın bir fonksiyon çağrısı yapması durumudur.(İşlenme sırası çok önemli)(yan etkilerinden dolayı önemli)

Aritmetik İfadeler: Fonksiyonel Yan Etkiler

- Fonksiyonel Yan Etkiler*:* Bir fonksiyon iki yönlü bir parametreyi veya lokal olmayan bir değişkeni değiştirdiğinde meydana gelir.
- Örnek:
  - Bir ifadede çağrılmış bir fonksiyon ifadenin başka bir operantını değiştirdiğinde ortaya çıkar; bir parametre değişim örneği:
  - ```
    a = 10;
    /* fun, parametresini değiştiriyor */
    b = a + fun(&a);
    ```
    ```

Fonksiyonel Yan Etkiler

- Bu problem için 2 muhtemel çözüm:
  1. Fonksiyonel yan etkileri iptal etmek için dil tanımlaması yapılır
     - Fonksiyonlarda 2 yönlü parametre olmayacak
     - Fonksiyonlarda global değişken olmayacak
     - **Avantajı**: o çalışıyor!
     - **Dezavantajı**: tek yönlü parametrelerin kararlılığı ve global değişkenlerin olmayışı(fonksiyonların birden çok değer döndürmeleri ihtiyacından dolayı pratik değil)
  2. operantların işlem sırasını belirlemek için dil tanımlaması yapılır
     - **Dezavantajı** : Bazı derleyicilerin optimizasyonunu sınırlar
     - Java operantların soldan sağa işlenmesine izin verdiğinden bu problem oluşmaz.

İmalı Şeffaflık

- Eğer bir programdaki aynı değere sahip herhangi iki ifade programın akışını etkilemeksizin birbiri yerine kullanılabilirse bu program imalı şeffaflık özelliğine sahiptir.

```
result1 = (fun(a) + b) / (fun(a) - c);
temp = fun(a);
result2 = (temp + b) / (temp - c);
```

Eğer `fun` fonksiyonu yan etkiye sahip değilse,

`result1 = result2` olacaktır.

Aksi taktirde,olmayacak , ve imalı şeffaflık bozulur.

İmalı Şeffaflık(devam)

- İmalı Şeffaflığın Avantajı
  - Eğer bir program imalı şeffaflığa sahipse programın anlamı daha kolay anlaşılır
- Onlar değişkenlere sahip olmadığı için teorik fonksiyonel dillerdeki programlar imalı şeffaftırlar.
  - Fonksiyonlar yerel değişkenler içinde saklanacak durumlara sahip olamazlar.
  - Eğer bir fonksiyon yabancı bir değer kullanırsa, o bir sabit olmalıdır(değişkenler değil).Bu yüzden bir fonksiyonun değeri sadece onun parametrelerine bağlıdır.

Aşırı Yüklenmiş Operatörler

- Bir operatörün birden fazla amaç için kullanımı *operatörün aşırı yüklenmesi* olarak adlandırılır.
- Yaygın olanlardan bazısı(örn., int ve float için '+',string ifadelerin birleştirilmesi)
- Bazısı potansiyel olarak sorunludur(örn.,C ve C++ da '*' ikili olarak çarpı,tekli olarak adresleme(pointer))
  - Derleyicinin hata belirlemesindeki kayıplar(derleyici operant eksiklerini fark etmeli) (a*b yerine *b yazmak gibi)
  - Bazı okunabilirlik kayıpları(okunabilirliği zorlaştırır)

Aşırı Yüklenmiş Operatörler(devam)

- C++, C#, ve F# kullanıcı tarafından tanımlanan operatörlere izin verir.
  - Böyle operatörler anlamlı kullanıldığında okunabilirliğe bir yardımı olabilir ,(metot çağrılarından kaçınmak, ifadeler doğal görünür)
  - Potansiyel problemler:
    - Kullanıcılar anlamsız işlemler tanımlayabilir
    - Operatörler anlamlı bile olsa okunabilirlik zarar görebilir

Tip Dönüşümleri

- **Daraltıcı dönüşüm**: Dönüştürülecek tip orijinal tipin tüm değerlerini içermiyorsa bu dönüşüme daraltıcı dönüşüm denir (örnek, `float` → `int`)
- **Genişletici dönüşüm**: Dönüştürülecek tip orijinal tipin tüm değerlerinden fazlasını içeriyorsa bu dönüşüm genişletici dönüşümdür.(örnek,`int` → `float`)

Tip Dönüşümleri: Karışık Biçim

- Eğer bir işlemin operantları farklı türden ise bu ifadeye karışık biçimli ifade denir.
- Zorlama(istemsiz) kapalı tip dönüşümdür.
- Zorlamanın Dezavantajları:
  - Derleyicinin hata bulma kabiliyetini azaltır.
- Çoğu dillerde, tüm sayısal tipler genişletici dönüşüm kullanılarak zorlanır.
- Ada'da, ifadelerde zorlama yoktur.
- ML ve F#' ta, ifadelerde zorlama yoktur.

Açık Tip Dönüşümleri

- C tabanlı dillerde *veri tipleri dönüşümü*(*casting)*
- Örnekler
  - `C: (int)angle`
  - `F#: float(sum)`

Not: F#'ın sentaksı fonksiyon çağırmaya benzer.

İfadelerdeki Hatalar

- Sebepler
  - Aritmetiğin doğal sınırları örn: sıfıra bölme
  - Bilgisayar aritmetik sınırları örn: taşma(overflow)
- Çalışma zamanında sık sık ihmal edilir.

İlişkisel ve Mantıksal İfadeler

- İlişkisel İfadeler
  - İlişkisel operatörler ve çeşitli tipteki operantların kullanımı
  - Bazı mantıksal işaretlerin ölçümü
  - Operatör sembolleri dillere göre değişiklik gösterir. (`!=`, `/=`, `~=`, `.NE.`, `<>`, `#`)
- JavaScript ve PHP 2 ek İlişkisel operatöre sahiptir, `===` and `!==`
  - Operantlarını zorlamamaları dışında kuzenlerine benzer, `==` ve `!=`,
  - Ruby eşitlik ilişki operatörü için `==` kullanır

İlişkisel ve Mantıksal İfadeler

- Mantıksal İfadeler
  - Hem operantlar hem de sonuçlar mantıksaldır
  - Örnek operatörler:
- C89 mantıksal tipe sahip değil ve bunun için int tipini kullanır.(0➔yanlış,değilse doğru)
- C ifadelerinin tuhaf bir özelliği:
- `a < b < c` doğru bir ifade, ama sonuç umduğumuz şeyi vermeyebilir:
  - Soldaki operatörler işlendiğinde,0 veya 1 üretir
  - Ölçülen sonuç o zaman 3. operant ile karşılaştırılır (ör:, `c`)

Kısa Devre Tespiti

- Bir ifadede operant/operatörlerin tüm hesaplamalarını yapmaksızın sonucun bulunmasıdır.
- Örnek: `(13 * a) * (b / 13 - 1)`

  Eğer `a==0 ise` , diğer kısmı hesaplamaya gerek yok `(b /13 - 1)`
- Kısa Devre Olmayan Problem

```
index = 0;
while (index <= length) && (LIST[index] != value)
      index++;
```

  - `index=length` **olduğunda,** `LIST[index]` **indeksleme problemi ortaya çıkaracak(**`LIST` dizisi `length - 1` uzunluğunda varsayılmış)

Kısa Devre Tespiti(devam)

- C, C++, ve Java: kısa devre tespitinin bütün mantıksal operatörler(`&&` ve `||`) için yapar, ama bit düzeyinde mantıksal operatörler(`&` and `|`) için yapmaz.
- Ruby, Perl, ML, F#, ve Python'da tüm mantık operatörleri için kısa devre tespiti yapılır.
- Ada: Programcının isteğine bağlıdır(kısa devre '**`and then`**' ve '**`or else`**' `ile belirtilir`)
- Kısa devre tespiti ifadelerdeki potansiyel yan etki problemini ortaya çıkarabilir
  örnek. `(a > b) || (b++ / 3)`
- `a>b olduğu sürece b artmayacak`

Atama İfadeleri

- Genel Sentaks

  `<target_var> <assign_operator> <expression>`

- Atama operatörü

  = Fortran, BASIC, C-tabanlı diller

  := Ada

- = eşitlik için ilişkisel operatörler aşırı yüklendiğinde kötü olabilir(o zaman C-tabanlı diller ilişkisel operatör olarak neden '==' kullanır?)

Atama İfadeleri: Şartlı Amaçlar

- Şartlı Amaçlar(Perl)

```
($flag ? $total : $subtotal) = 0
```

Hangisi eşittir

```
if ($flag){
  $total = 0
} else {
  $subtotal = 0
}
```

Atama İfadeleri : Birleşik Atama Operatörleri

- Atama formu için yaygın olarak bir stenografi metodu belirtmek gerekir
- ALGOL'de tanımlı; C ve C-tabanlı diller de benimsemiş.
  - Örnek:

```
a = a + b
```

Aşağıdaki gibi yazılabilir.

```
a += b
```

Atama İfadeleri : Tekli Atama Operatörleri

- C-tabanlı dillerdeki tekli atama operatörleri atama ile artış ve azalış işlemlerini birleştirir
- Örnekler
  - `sum = ++count` (`count` arttırıldı, daha sonra `sum`'a **aktarıldı**)
  - `sum = count++` (`count` `sum`'a **aktarıldı**, ondan sonra arttırıldı
  - `count++` (`count` arttırıldı)
  - `-count++` (`count` arttırıldı ondan sonra negatifi alındı)

Bir İfade olarak Atama

- C-tabanlı diller, Perl, ve JavaScript'te atama durumu bir sonuç üretir ve bir operant olarak kullanılabilir,

```
while ((ch = getchar())!= EOF){...}
```

`ch = getchar()` başarılı; sonuç(`ch a aktar`) while döngüsü için şartsal bir değer olarak kullanılır

- Dezavantaj: Başka tip yan etkiler.

Çoklu Atamalar

- Perl, Ruby, ve Lua çok hedefli ve çok kaynaklı atamalara izin verir

```
($first, $second, $third) = (20, 30, 40);
```

Hatta,aşağıdaki geçerli ve bir yer değiştirme uygulanır:

```
($first, $second) = ($second, $first);
```

Fonksiyonel Dillerde Atama

- Fonksiyonel dillerde tanıtıcılar(identifier) sadece değer adlarıdır.
- ML
  - İsimler **`val`** ve değer ile sınırldır İsimler

```
val fruit = apples + oranges;
```

  - Eğer gruit için başka bir val izlenecekse, o yeni ve farklı bir isimde olmalıdır.
- F#
  - F#'s yeni bir skop(scope) yaratmanın dışında ML 'in **`val`** ile aynıdır.

Karışık Biçim Ataması

- Atama ifadeleri karışık biçimde olabilir
- Fortran, C, Perl, ve C++'ta ,her tip sayısal değer her tip sayısal değişkene atanabilir
- Java ve C#'ta, sadece genişletici atama zorlaması yapılır
- Ada'da, atama zorlaması yoktur.

Özet

- İfadeler
- Operatör önceliği ve birleşilirliği
- Operatörlerin aşırı yüklenmesi
- Karışık tipli ifadeler
- Atamaların çeşitli formları