Programlama Dilleri Prensipleri

Ders 7. Expressions ve Assignment (Atama) Statements

Konular

- Arithmetic Expressions (Aritmetik İfadeler)
- Overloaded Operators (Aşırı Yüklenmiş Operatörler)
- Type Conversions (Tür Dönüşümleri)
- Relational and Boolean Expressions (İlişkisel ve Boolean İfadeler)
- Short-Circuit Evaluation (Kısa-Devre Değerlendirmesi)
- Assignment (Atama) Statements
- Mixed-Mode Assignment (Karma-Mod Atama)

Giriş

- Expression, bir programlama dilinde hesaplamaları belirlemenin temel yoludur
- Expression değerlendirmesini anlamak için, operatörün sıralarına ve operand'ların değerlendirmesine anlamamız gerekir
- Emir esaslı dillerin özü, atama ifadelerinin baskın rolüdür
  - X = X + Y'nin matematiksel anlamı ile programlama anlamı birbirinden tamamen farklıdır.

Aritmetik İfadeler (Arithmetic Expressions)

- Aritmetik değerlendirme, ilk programlama dillerinin geliştirilmesi için motivasyonlardan biriydi
- Aritmetik ifadeler operatörlerden, operand'lar, parantezlerden ve fonksiyon çağrılarından oluşur

Aritmetik İfadeler
Tasarım Zorlukları

- Operatör öncelik kuralları?
- Operatör birliktelik kuralları?
- Operand değerlendirme sırası?
- Operand değerlendirmesinin yan etkileri?
- Operatör aşırı mı yüklüyor?
- İfadelerde tür karıştırma?

Aritmetik İfadeler : Operators

- Unary operatör: Tek operanda sahip
  - Javascript: +, -, ++, --, ~
    - JavaScript Unary Operators: Simple and Useful | DigitalOcean
- Binary operatör: İki operanda sahip
- Ternary operatör: Üç Operand'a sahip
  - Condition ? True : False
    - Conditional (ternary) operator - JavaScript | MDN (mozilla.org)

Aritmetik İfadeler
Operatör Öncelik Kuralları

- İfade değerlendirmesi için operatör öncelik kuralları, farklı öncelik seviyelerine sahip "bitişik" (adjacent) operatörlerin değerlendirilme sırasını tanımlar
- Tipik öncelik seviyeleri
  - parantezler
  - tekli operatörler
    - ** (dil destekliyorsa)
  - *, /
  - +, -

Aritmetik İfadeler
Operatör İlişkilendirme Kuralı (Operator Associativity Rule)

- İfade değerlendirmesi için operatör ilişkilendirme kuralları, aynı öncelik düzeyine sahip bitişik operatörlerin değerlendirilme sırasını tanımlar
  - 7 − 4 + 2

- Tipik ilişkilendirilebilirlik kuralları
  - Soldan sağa,
  - Sağdan sola (**)
  - Bazen tekli operatörler sağdan sola ilişkilendirir (örneğin, FORTRAN'da)
- APL farklıdır; tüm operatörler eşit önceliğe sahiptir ve tüm operatörler sağdan sola ilişkilendirilir
- Öncelik ve ilişkilendirilebilirlik kuralları parantezlerle geçersiz kılınabilir
  - (7 − 4) + 2 = 5
  - 7 − (4 + 2) = 1

Expressions
Programlama Dilleri

- Ruby
  - Tüm aritmetik, ilişkisel ve atama operatörlerinin yanı sıra dizi indeksleme, kaydırmalar ve bit bazlı mantık operatörleri method olarak uygulanır.
  - Bunun bir sonucu, bu operatörlerin tümünü uygulama programları tarafından yeni görevler verilebilir
    - Operator Overloading in Ruby – GeeksforGeeks
- Scheme (ve Common Lisp)
  - Tüm aritmetik ve mantıksal işlemler açıkça alt programlar olarak adlandırılır.
  - a + b * c, (+ a (* b c)) olarak kodlanır

Arithmetic Expressions : Conditional Expressions

- Conditional Expressions
  - C-based languages (e.g., C, C++)
  - Örnek:
    average = (count == 0) ? 0 : sum / count
- Klasik yöntem:

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

Aritmetik İfadeler:
Operand Değerlendirme Sırası

- Değişkenler: değeri bellekten getir
- Sabitler: bazen bellekten bir getirme; bazen sabit makine dili komutundandır
- Parantezli ifadeler: önce tüm operand'ları ve operatörleri değerlendirin
- En ilginç durum, bir operand'ın bir fonksiyon çağrısı olduğu durumdur.

Aritmetik İfadeler:
Yan Etkiler İçin Potansiyeller

- Fonksiyonel yan etkiler: bir fonksiyon iki yönlü bir parametreyi veya yerel olmayan bir değişkeni değiştirdiğinde
- Fonksiyonel yan etkilerle ilgili sorun:
  - Bir ifadede referans verilen bir fonksiyon, ifadenin başka bir operandını değiştirdiğinde; ör. bir parametre değişikliği için:

    a = 10;
- / * fonksiyon parametresini değiştirdiğini varsayalım * /

  b = a + fonksiyon(a);

Sonuç ne olur?

```
#include <iostream>

using namespace std;

int deg1(int* p){
    *p = 20;
    int i = *p;
    return i;
}

int deg2(int x){
    return x * 2;
}

int main()
{
    int a = 10;

    a = a + deg1(&a);

    cout<<a<<'\n';

    int b = 20;
    b = b + deg2(b);

    cout<<b;

    return 0;
}
```

Fonksiyonel Yan Etkiler

- Soruna iki olası çözüm
  - Fonksiyonel yan etkilere izin vermemek için programlama dili tanımlarımız:
    - Fonksiyonlarda iki yönlü parametre yok
    - Fonksiyonlarda yerel olmayan referans yok
    - Avantaj: işe yarıyor!
    - Dezavantaj: tek yönlü parametrelerin esnekliği ve yerel olmayan referansların olmaması
  - Operand değerlendirme sırasının sabitlenmesini talep edilen programlama dili tanımlarımız:
    - Dezavantaj: bazı derleyici optimizasyonlarını sınırlar
    - Java, operand'ların soldan sağa sırayla değerlendirilmiş görünmesini gerektirir

Referans Şeffaflığı (Referential Transparency)

- Bir program, programdaki aynı değere sahip herhangi iki ifade, programın eylemini etkilemeden programın herhangi bir yerinde birbiriyle ikame edilebiliyorsa, referans şeffaflığı özelliğine sahiptir.

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

- Eğer result1 = result2 ise referans şeffalığı sahiptir.

Referans Şeffaflığı…

- Referans şeffaflığının avantajı
  - Bir programın anlambilimini, referans şeffaflığa sahipse anlamak çok daha kolaydır
- Değişkenlere sahip olmadıkları için, saf fonksiyonel dillerdeki programlar referans olarak şeffaftır
  - Fonksiyonlar, yerel değişkenlerde saklanacak duruma sahip olamaz
  - Bir fonksiyon bir dış değer kullanıyorsa, bu bir sabit olmalıdır (değişken yoktur). Dolayısıyla, bir fonksiyon değeri yalnızca parametrelerine bağlıdır

Overloaded Operators (Aşırı Yüklenmiş Operatörler)

- Bir operatörün birden fazla amaç için kullanımına operatörün aşırı yüklenmesi denir
- Bazıları yaygındır (ör. int ve float için +)
- Bazıları potansiyel sorunlardır (örneğin, C ve C ++ 'da *)
  - Derleyici hata tespiti kaybı (bir operand'ın ihmal edilmesi tespit edilebilir bir hata olmalıdır)
  - Okunabilirlikte bir miktar kayıp

Overloaded Operators (Aşırı Yüklenmiş Operatörler)

- C ++, C# ve F#, kullanıcı tanımlı aşırı yüklenmiş operatörlere izin verir
  - Makul bir şekilde kullanıldığında, bu tür operatörler okunabilirliğe yardımcı olabilir
  - Olası sorunlar:
    - Kullanıcılar saçma işlemler tanımlayabilir
    - Operatörler mantıklı olsa bile okunabilirlik zarar görebilir
- Ruby
  - Operator Overloading in Ruby – GeeksforGeeks

Type Conversions (Tür Dönüşümleri)

- Daraltma dönüştürme (*narrowing conversion*), bir nesneyi orijinal türün tüm değerlerini içeremeyen bir türe dönüştüren bir dönüştürmedir
  - örneğin float'tan int'e
- Genişletme dönüşümü (*widening conversion*), bir nesnenin orijinal türün tüm değerlerine en azından yaklaşıklıkları içerebilen bir türe dönüştürülmesidir
  - örneğin int'ten float'a

Type Conversions
Mixed Mode (Karma -Mod)

- Karma modlu ifade, farklı türlerde operand'lara sahip olandır.
  - Örneğin bir int bir de float
- Bir zorlama (*coercion)*, **örtük (implicit)** bir tür dönüştürmesidir
- Zorlamaların dezavantajı:
  - Derleyicinin tür hatası algılama yeteneğini azaltır
- Çoğu dilde, tüm sayısal türler genişleyen dönüşümler kullanılarak ifadelerde zorlanır.
  - ML ve F# 'de ifadelerde zorlama yoktur

Explicit (Açık) Type Conversions

- C tabanlı dillerde **casting** denir
- Örnekler
  - C: (int) angle
  - F#: float (sum)
  - F# 'nin sözdiziminin fonksiyon çağrılarınınkine benzer olduğunu unutmayın
  - C#
    - Console.WriteLine((double)5 / 2);
    - Console.WriteLine(5 / 2);

Errors in Expressions (İfadelerdeki Hatalar)

- Nedenleri
  - Aritmetiğin doğal sınırlamaları, örneğin sıfıra bölme (division by zero)
  - Bilgisayar aritmetiğinin sınırlamaları, ör. taşma (overflow)
- Genellikle çalışma zamanı sistemi tarafından göz ardı edilir

Relational and Boolean Expressions (İlişkisel ve Boolean İfadeler)

- İlişkisel İfadeler
  - İlişkisel operatörleri ve çeşitli türlerdeki operand'ları kullanır
  - Bazı Boole temsillerine göre değerlendirir
  - Kullanılan operatör sembolleri diller arasında biraz farklılık gösterir (! =, / =, ~ =, .NE., <>, #)
- JavaScript ve PHP'nin iki ek ilişkisel operatörü vardır
  - === ve !== hem değer hem tür karşılaştırması yapar
  - == ve !=, operandları değer karşılaştırması yapar, tür karşılaştırması yapar
  - Ruby'de == vardır. Ancak, tür karşılaştırması için eql? vardır.
    - Difference Between ==, eql?, equal? in ruby | by Khalidh Sd | Medium

Relational and Boolean Expressions (İlişkisel ve Boolean İfadeler)

- Boolean İfadeler
  - Operand’lar Boole'dir ve sonuç Boole'dur
  - Örnek operatörler
- C89'da Boolean türü yoktur - yanlış için 0 ve doğru için sıfır olmayan int türü kullanır
- C’nin ifadelerinin tuhaf bir özelliği: a < b < c yasal bir ifadedir, ancak sonuç beklediğiniz gibi değildir:
  - Sol operatör değerlendirilir, 0 veya 1 üretir
  - Değerlendirme sonucu daha sonra üçüncü operatörle karşılaştırılır (Örneğin, C)

Short Circuit Evaluation (Kısa Devre Değerlendirmesi)

- Tüm operand'lar ve / veya operatörler değerlendirilmeden sonucun belirlendiği bir ifade
  - Örneğin (13 * a) * (b / 13 – 1)
  - a sıfır ise, (b / 13 – 1) değerlendirmeye gerek yoktur.
- Kısa devre dışı kalma sorunu

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

  - Burada and (&&) sebebiyle döngü hiç bitmeyebilir.

Short Circuit Evaluation…

- C, C ++ ve Java: normal Boole operatörleri (&& ve ||) için kısa devre değerlendirmesini kullanır, ancak aynı zamanda kısa devre (& ve |) olmayan bitsel Boole operatörleri de sağlar.
- Ruby, Perl, ML, F# ve Python'daki tüm mantık operatörleri kısa devre olarak değerlendirilir
- Kısa devre değerlendirmesi, ifadelerdeki potansiyel yan etki problemini ortaya çıkarır.
  - (a > b) || (b++ / 3)

Assignment (Atama) Statements

- Genel sözdizimi
- <target_var> <assign_operator> <expression>
- Atama operatörü
  - = Fortran, BASIC, C tabanlı diller
  - := Ada
- = eşitlik için ilişkisel operatör için aşırı yüklendiğinde kötü durumlar oluşabilir (bu nedenle C tabanlı diller ilişkisel operatör olarak == kullanır)

Assignment Statements
Conditional Targets (Şartlı Hedefler)

- Conditional targets (Perl)

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

Which is equivalent to

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

Assignment Statements
Compound Assignment Operators (Bileşik Atama Operatörleri)

- Yaygın olarak ihtiyaç duyulan bir atama biçimini belirtmenin kısa bir yöntemi
- ALGOL'da tanıtılan; C ve C tabanlı diller tarafından benimsenmiştir
  - Örneğin
    - `a = a + b`

    yerine
    - `a += b`

Assignment Statements
Unary Assignment Operators (Tekli Atama Operatörleri)

- C tabanlı dillerdeki tekli atama operatörleri, atama ile artırma ve azaltma işlemlerini birleştirir
- Örneğin

  `sum = ++count` (`count` artar, sonra `sum`'a atanır)

  `sum = count++` (`count` `sum`'a atanır, sonra `count` artar

  `count++` (`count` artar)

  `-count++` (`count` artar sonra negatif olur)

Expression olarak Atama

- C tabanlı dillerde, Perl ve JavaScript'te, atama ifadesi bir sonuç üretir ve bir işlenen olarak kullanılabilir

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

- Öncelikle ch = getchar() gerçekleştirilir; sonuç (ch'ye atanan) while ifadesi için koşullu bir değer olarak kullanılır
- Dezavantaj: başka bir ifade yan etkisi türü

Multiple Assignments (Çoklu Atamalar)

- Perl, Ruby ve Lua, çok hedefli çok kaynaklı atamalara izin verir
  ($ birinci, $ ikinci, $ üçüncü) = (20, 30, 40);
- Ayrıca, iki değişkenin değeri aşağıdaki gibi değiştirilebilir. (swap işlemi)
  - ($ birinci, $ saniye) = ($ ikinci, $ birinci);
- Python'da benzer kulanımlar vardır.

Assignment in Functional Languages (Fonksiyonel Dillerde Atamalar)

- Fonksiyonel dillerdeki tanımlayıcılar yalnızca değerlerin isimleridir
- ML
  - İsimler val ile değerlere bağlıdır
  - val x = y + z;
  - x için başka bir değer gelirse, bu yeni ve farklı bir isimdir
- F#
  - F# - let, ML'nin değeri gibidir, ancak yeni bir kapsam da oluşturur

Mixed -Mode Assignment (Karma -mod Atama)

- Atama ifadeleri ayrıca karışık modda olabilir
- Fortran, C, Perl ve C++'da, herhangi bir sayısal tür değeri herhangi bir sayısal tür değişkene atanabilir
- Java ve C#'de yalnızca genişleyen atama zorlamaları yapılır
- Ada'da atama zorlaması yoktur