# 5-Expressions and Assignment Statements

**First, the semantic rules that determine the order of evaluation of operators in expressions are discussed. Overloaded operators and short-circuit evaluation will be discussed. Finally, the assignment statement will be studied.**

## 5.1 Excercises

### 5.1.1 Short Circuit Evaluation Examples

**1. Write a Pascal program segment, using a while construct to search an array of integers for a particular integer, that would work even if short circuit evaluation of Boolean expressions were not done.**

The following code works false if short circuit evaluation is not done:

index:=1;

while (index<=listlen) and (list[index]<>key) do

index:=index+1;

**.**

**.**

**.**

index:=1;

while (index<listlen) and (list[index]<>key) do

index:=index+1;

If (list[index]<>key) then

index:=index+1;

**.**

**.**

**.**

2. Describe and exemplify how ambiguity may result, due to ignorance of shortcut evaluation policy of a C compiler.

int stringSize(char \*c)

{ int i;

while(c && (i++<500));

/\* Does i contain what we want? \*/

if (i=500) printf(“String too big !”);

return i;

}

# 6-Control Structures

**The statements that form the flow of control in a programming language will be studied. These are, single way, two-way and multiple selection constructs. Following this, various looping constructs will be studied.**

## 6.1 Exercises

### 6.1.1 Loop Examples

**1. Rewrite the following code segment using a loop in Pascal or C.**

**k:=(j+13)/27;**

**loop:**

**If k>10 then goto out**

**k:=k+1;**

**i:=3\*k-1;**

**goto loop**

**out:**

k:=(j+13)/27;

while k<=10 do

begin

k:=k+1;

i:=3\*k-1;

end;

**2. Write C code to print numbers from 15 to 50 in order. Do not use comparison operators.**

**a) Use while loop**

int i=15;

while (51-i)

printf(“%d \n”,i++);

OR

int i=14;

while (50-i)

printf(“%d \n”,++i);

**b) Use for loop**

int i;

for(i=15; 51-i;i++)

printf(“%d \n”,i);

OR

for(int i=14; 50-i;)

printf(“%d \n”,++i);

**c) Use unconditional loop and break**

int i=15;

while(1)

{ printf(“%d \n”,i++);

if(!(51-i)) break; //same as i==51

}

### 6.1.2 Multiple Way Selection Examples

**1. Consider the following C program segment. Rewrite it using no gotos or breaks.**

j:=-3;

for (i=0; i<3; i++)

{

switch (j+2)

{

case 3:

case 2: j--; break;

case 0: j+=2; break;

default: j=0;

}

if (j>0) break;

j=3-i;

}

A first simplification gives:

j = -3;  
for ( i = 0 ; i < 3 ; i++) {  
    if ( j == 0 || j == 1 ) j--;  
    else if (j == -2) j = 0;  
        else j = 0;  
    if ( j > 0 ) break;  
    j = 3 - i;  
}

Hence, j is always either 0 or -1 after the if statement; if (j>0) is useless.

j = -3;  
for ( i = 0 ; i < 3 ; i++)

{ if (j == 0) j = -1;  
  else j = 0;  
  j = 3 - i;  
}

**2. Rewrite the following code segment using a multiple selection statement in Pascal or in C. All variables are integer type.**

**if (x=1) or (x=2) then y:=2\*x-1**

**if (x=3) or (x=5) then y:=3\*x+1**

**if (x=4) y:=4\*x-1**

**if (x=6) or (x=7) or (x=8) then y:=x-2**

case x of

1,2 : y:=2\*x-1;

3,5 : y:=3\*x+1;

4 : y:=4\*x-1;

6, 7, 8 : y:=x-2;

end;

**3. Rewrite the following Pascal case statement using only two-way selection.**

  Case index – 1 of

   2, 4: even := even + 1;

   1, 3: odd := odd + 1;

      0: zero := zero + 1;

   else error := true

 end;

My **solution** compares against index rather than index – 1, to avoid unnecessary arithmetic at run time:

  if (index = 3) or (index = 5)

     then even := even + 1

     else if (index = 2) or (index = 4)

          then odd := odd + 1

          else if (index = 1)

               then zero := zero + 1

                    else error := true;

## 6.2 Assignments

### 6.2.1 Loop Examples

**1. Assume that a program contains a posttest loop that repeats the loop body instructions until the user enters a negative number. How many times will the loop condition be evaluated if the user enters the numbers 5, 8, 9 and –1 respectively?**

**2. Write a program fragment that skips over a sequence of positive integers read as data until it reaches a negative value. Write two versions: one using repeat, one using while.**

**3. Consider the following program segment:**

**Count:=0;**

**For I:=1 to N do**

**Begin**

**Read(x);**

**If x=I then**

**Count:=count+1;**

**End;**

**Write a while loop equivalent to the for loop.**

**Kısa Devre Değerlendirme**

Kısa devre değerlendirme, ne yazık ki pek iyi bir çeviri olmadı ve bu yüzden hiçbir anlam ifade etmeyebilir. İngilizce’de, Short Circuit Evaluation olarak geçen bu konu, mantıksal ifadelerle ilgilidir.

Hatırlarsanız, daha önce ki derslerimizde iki farklı AND ve OR operatörü görmüştük. Bu yapılardan biri AND için && işaretini kullanıyorken, diğeri sadece tek & simgesini kullanıyordu. Keza, OR ifadesi bir yerde, || şeklinde gösteriliyorken, diğer yerde, tek bir | simgesiyle ifade edilmekteydi. Bu işaretler, aynı sonucu üretiyor gibi görünseler de, farklı şekilde çalışırlar.

Çift şekilde yazılan operatörler, ( yani && ve || ) kısa devre operatörleridir. İngilizce, Short Circuit Operator olarak isimlendirilirler. Tek sembolle yazılan operatörlerden farkı, işlemleri kısaltmalarıdır.

Bir koşul içersinde AND ( && ) operatörü kullandığınızda, koşulun sol tarafı yanlışsa, sağ tarafı kontrol edilmez. Çünkü artık sağ tarafın doğru veya yanlış olmasının önemi yoktur; sonuç her şekilde yanlış olacaktır.

Benzer bir mantık OR ( || ) operatörü içinde geçerlidir. Eğer sol taraf doğruysa, sağ tarafın kontrol edilmesine gerek yoktur. Çünkü OR operatöründe taraflardan birinin doğru olması durumunda, diğerinin önemi kalmaz ve sonuç doğru döner.

Aşağıdaki örnekleri inceleyelim:

**&& Operatörü**

#include<stdio.h>

int main( void )

{

int i, j;

i = 0;

j = 5;

if( i == 1 **&&** j++ ) {

printf( "if içersine girdi\n" );

}

else {

printf( "if içersine girmedi\n" );

printf( "i: %d, j: %d\n", i, j );

}

return 0;

}

if içersine girmedi

i: 0, j: 5

**& Operatörü**

#include<stdio.h>

int main( void )

{

int i, j;

i = 0;

j = 5;

if( i == 1 & j++ ) {

printf( "if içersine girdi\n" );

}

else {

printf( "if içersine girmedi\n" );

printf( "i: %d, j: %d\n", i, j );

}

return 0;

}

if içersine girmedi

i: 0, j: 6

Gördüğünüz gibi, program çıktıları birbirinden farklıdır. Bunun sebebi, ilk örnekte, *i == 1* koşulu yanlış olduğu için, && operatörünün ifadenin sağ tarafına bakmamasıdır. İkinci örnekteyse, & operatörü, koşulun her iki tarafına da bakar. Bu yüzden, j değişkenine ait değer değişir. Benzer bir uygulamayı, OR için || ve | kullanarak yapabilirsiniz.

**ÖNEMLİ NOT:**Özetle işlemlerinizi hızlandırmak istiyorsanız; AND kullanacağınız zaman, && operatörüyle çalışın ve yanlış olması muhtemel olan koşulu sol tarafa koyun. Eğer OR operatörü kullanacaksanız, doğru olma ihtimali fazla olan koşulu, ifadenin soluna koyun ve operatör olarak || ile çalışın. Bu şekillde yazılan bir program, kısa devre operatörleri sayesinde, gereksiz kontrolden kaçınarak işlemlerinizi hızlandıracaktır.

Elbette & veya | operatörlerinin kullanılması gereken durumlarda olabilir. Her n’olursa olsun, koşulun iki tarafınında çalışmasını istiyorsanız, o zaman & ve | operatörlerini kullanmanız gerekmektedir.

Mantıksal İşleçler

Bu işleçler, terimleri üzerinde mantıksal işlem yapar. Terimlerini doğru (true) ya da

yanlış (false) olarak yorumladıktan sonra işleme sokar. C'de öncelikleri farklı

seviyede olan üç mantıksal işleç vardır:

(!) mantıksal değil işleci (logical not)

(&&) mantıksal ve işleci (logical and)

(||) mantıksal veya işleci (logical or)

C'de mantıksal veri türü olmadığını biliyorsunuz. Mantıksal veri türü olmadığı için

bu türün yerine int türü kullanılır ve mantıksal doğru olarak 1, mantıksal yanlış

olarak da 0 değeri kullanılır.

C dilinde herhangi bir ifade, mantıksal işleçlerin terimi olabilir. Bu durumda söz

konusu ifade, mantıksal olarak yorumlanır. Bunun için ifadenin sayısal değeri

hesaplanır. Hesaplanan sayısal değer, 0 dışı bir değer ise doğru (1), 0 ise yanlış

(0) olarak yorumlanır. Örneğin:

25 Doğru (Çünkü 0 dışı bir değer)

-12 Doğru (Çünkü 0 dışı bir değer)

0 Yanlış (Çünkü 0) ifadesi mantıksal bir işlecin terimi olduğu zaman yanlış olarak

yorumlanır. Çünkü sayısal değeri sıfıra eşittir.

İstanbul Üniversitesi

Elektrik Elektronik Mühendisliği

27

Mantıksal Değil İşleci

Mantıksal değil işleci, önek konumunda bulunan tek terimli bir işleçtir. Bu işleç, teriminin

mantıksal değerinin tersini üretir. Yani terimi mantıksal olarak "doğru" biçiminde yorumlanan

bir değer ise işleç yanlış anlamında int türden 0 değerini üretir. Terimi, mantıksal olarak

"yanlış" biçiminde yorumlanan bir değer ise işleç doğru anlamında int türden 1 değerini

üretir.

x !x

Doğru (0 dışı değer) Yanlış (0)

Yanlış (0) Doğru (1)

Örnekler :

a = !25; /\* a değişkenine 0 değeri atanır \*/

b = 10 \* 3 < 7 + !2

İşlem sırası:

!2 = 0

10 \* 3 = 30

7 + 0 = 7

30 < 7 = 0

b = 0 (atama işleci en düşük öncelikli işleçtir)

y = 5;

x = !++y < 5 != 8;

İşlem sırası:

++y 6

!6 0 /\* ++ ve ! işleçleri aynı öncelik seviyesindedir ve öncelik yönü sağdan soladır. \*/

0 < 5 1

1 != 8 1

x = 1 İstanbul Üniversitesi

Elektrik Elektronik Mühendisliği

28

Mantıksal ve (&&) işleci

Bu işleç ilişkisel işleçlerin hepsinden düşük, || (veya / or) işlecinden yüksek

önceliklidir. Terimlerinin ikisi de doğru ise doğru (1), terimlerinden biri yanlış ise

yanlış (0) değerini üretir.

x = 3 < 5 && 7;

3 < 5 1

7 1

1 && 1 1

x = 1

&& işlecinin, önce sol tarafındaki işlemler öncelik sırasına göre tam olarak yapılır.

Eğer bu işlemlerde elde edilen sayısal değer 0 ise, && işlecinin sağ tarafındaki

işlemler hiç yapılmadan, yanlış (0) sayısal değeri üretilir. Örneğin:

x = 20;

b = !x == 4 && sqrt(24);

!20 0

0 == 4 0

Sol taraf 0 değeri alacağından işlecin sağ tarafı hiç yürütülmez dolayısıyla da sqrt

işlevi çağrılmaz. Sonuç olarak b değişkenine 0 değeri atanır.

İstanbul Üniversitesi

Elektrik Elektronik Mühendisliği

29

Mantıksal veya (||) İşleci

Önceliği en düşük olan mantıksal işleçtir. İki teriminden biri doğru ise doğru

değerini üretir. İki terimi de yanlış ise yanlış değerini üretir.

a = 3 || 5 /\* a = 1 \*/

x = 0 || -12 /\* x = 1 \*/

sayi = 0 || !5 /\* sayi = 0 \*/

&& ve || İşleçlerinin Kısa Devre Davranışı

"Mantıksal ve", "mantıksal veya" işleçlerinde önce soldaki terimlerinin

değerlendirilmesi güvence altına alınmıştır. "Mantıksal ve" işlecinin soldaki terimi

yanlış olarak yorumlanırsa işlecin sağ terimi hiç ele alınmaz. Aynı durum

"mantıksal veya" işleci için de geçerlidir. "Mantıksal veya" işlecinin önce soldaki

terimine bakılır. Sol terimi doğru olarak yorumlanırsa işlecin sağ terimi hiç dikkate

alınmaz. C dili tarafından güvence altına alınan bu özelliğe "kısa devre davranışı"

(short circuit behavior) denir. Kısa devre davranışına neden gerek duyulmuştur?

Çünkü bu özellik bazı kodların çok daha verimli yazılmasını sağlar. C'nin ileride

göreceğimiz birçok kalıp kodu kısa devre davranışının kullanımına bağlıdır.

İstanbul Üniversitesi

Elektrik Elektronik Mühendisliği

30

Atama İşleçleri

Atama işleçleri, C dilinde öncelik tablosunun en alttan ikinci seviyesinde, yani 14.

seviyesinde bulunur ve yalnızca virgül işlecinden daha yüksek önceliklidir. Atama

işleçlerinin bulunduğu 14. seviye, sağdan sola öncelik yönüne sahiptir.

Yalın Atama İşleci

Diğer işleçler gibi atama işleci de, yaptığı atama işleminin yanısıra, bir değer üretir.

Atama işlecinin ürettiği değer, nesneye atanan değerin kendisidir.

#include <stdio.h>

int main()

{

int x;

printf("ifade degeri = %d\n", x = 5);

printf("x = %d\n", x);

return 0;

}

main işlevi içinde yapılan birinci printf çağrısı ile, x = 5 ifadesinin değeri

yazdırılıyor. x = 5 ifadesinin değeri atama işlecinin ürettiği değer olan 5 değeridir.

Yani ilk printf çağrısı ile ekrana 5 değeri yazdırılır. Atama işleci yan etkisi sonucu x

nesnesinin değerini 5 yapar. Bu durumda ikinci printf çağrısı ile x değişkeninin

değeri ekrana yazdırıldığından ekrana yazılan, 5 değeri olur.

İstanbul Üniversitesi

Elektrik Elektronik Mühendisliği

31

İşlemli Atama İşleçleri

Bir işlemin terimi ile, işlem sonucunda üretilen değerin atanacağı nesne aynı ise,

işlemli atama işleçleri kullanılabilir.

<nesne1> = <nesne1> işlem <terim2>

ile

<nesne1> işlem= <terim2>

aynı anlamdadır.

İşlemli atama işleçleri, atama işleciyle aynı öncelik seviyesindedir. İşlemli atama

işleçleri, hem okunabilirlik hem de daha kısa yazım için tercih edilir.

Aşağıdaki ifadeler eşdeğerdir:

deger1 += 5; deger1 = deger1 + 5;

sonuc \*= yuzde; sonuc = sonuc \* yuzde;

x %= 5 x = x % 5;

katsayi = katsayi \* (a \* b + c \* d);

ifadesi de yine

katsayi \*= a \* b + c \* d;

İstanbul Üniversitesi

Elektrik Elektronik Mühendisliği

32

Şimdi de aşağıdaki main işlevini inceleyin:

#include <stdio.h>

int main()

{

int x = 3;

int y = 5;

x += y \*= 3;

printf("x = %d\n", x);

printf("y = %d\n", y);

return 0;

}

x += y \*= 3;

deyimiyle önce y değişkenine 15 değeri atanır. Bu durumda \*= işleci 15 değerini üretir

ve üretilen 15 değeri bu kez += işlecinin terimi olur. Böylece x değişkenine 18 değeri

atanır.

Özellikle += ve -= işleçlerinin yanlış yazılması, bulunması zor hatalara neden olabilir.

x += 5;

deyimi x değişkeninin değerini 5 artırırken, işlecin yanlışlıkla aşağıdaki gibi yazılması

durumunda

x =+ 5;

x değişkenine 5 değeri atanır. Çünkü burada iki ayrı işleç söz konusudur: Atama işleci

olan = ve işaret işleci olan +.

Öncelik İşleci

Öncelik işleci ( ), bir ifadenin önceliğini yükseltmek amacıyla kullanılır.

x = (y + z) \* t;

Öncelik işleci, C'nin en yüksek öncelikli işleçler grubundadır. Öncelik işleci de,

kendi arasında soldan sağa öncelik kuralına uyar. Örneğin:

a = (x + 2) / ((y + 3) \* (z + 2) – 1);

ifadesinde işlem sırası şöyledir :

i1 : x + 2

i2 : y + 3

i3 : z + 2

i4 : i2 \* i3

i5 : i4 – 1

i6 : i1 / i5

i7 : a = i6

Öncelik işlecinin terimi nesne gösteren bir ifade ise, işlecin ürettiği ifade de nesne

gösterir:

int x;

(x) = 20; /\* Geçerli \*/

İlişkisel ve Mantıksal Deyimler: İşleç Önceliği

•

C-temelli işleçlerdeki öncelik

prefix ++, --unary +, -, prefix ++, --, !

\*,/,%

binary +, -<, >, <=, >=

=, !=

&&

||

Copyright © 2007 AddisonWesley. All rights reserved. 1-23

Kısa devre hesaplama(Short Circuit Evaluation)

•

Tüm işleçler hesaplanmadan sonucun

belirlenebildiği deyimlerdir.

•

örnek: (13\*a) \* (b/13–1)

eğer asınıfrsa,(b/13-1)hesaplamaya gerek yoktur.

•

Kısa devre olmayan hesaplamadaki sorun

index = 1;

while (index <= length) && (LIST[index] != value)

index++;

–

index=length olduğunda, LIST [index]indeksleme

sorunu oluşturur ( LISTde length -1eleman olduğunu

varsayıyoruz)

Copyright © 2007 AddisonWesley. All rights reserved. 1-24

Kısa devre hesaplama

•

C, C++, ve Java: Mantıksal işleçler(Boolean

operators) (&&ve ||) için kısa devre hesaplama

kullanır, ayrıca kısa devre hesaplamanın

yapılmadığı bitsel işleçlerede olanak verirler.

(&ve |)

•

Ada: programcılar kısa devre hesaplama yapılıp

yapılmayacağına izin verebilir(kısa devre and then

ile belirtilebilir veya or else)

•

Kısa devre hesaplama deyimlerde yan etkilere yol

açabilir.

örn. (a > b) || (b++ / 3)

Copyright © 2007 AddisonWesley. All rights reserved. 1-25

Atama İfadeleri

•

Genel sözdizimi

<target\_var> <assign\_operator> <expression>

•

Atama işleci

=FORTRAN, BASIC, PL/I, C, C++, Java

:=ALGOLs, Pascal, Ada

•

= işleci aşırı yüklenirse problem olabilir.

Copyright © 2007 AddisonWesley. All rights reserved. 1-26

Atama ifadeleri: Şartlı hedefler

•

Şartlı hedefler (C, C++, and Java)

(flag)? total : subtotal = 0

bu aşağıdakiyle eşleniktir.

if (flag)

total = 0

else

subtotal = 0

Copyright © 2007 AddisonWesley. All rights reserved. 1-27

Atama İfadeleri: Bileşik

İşleçler(compound operators)

•

Sıklıkla ihtiyaç duyulan bir deyim yerine kısa

yoldur.

•

ALGOL ile geldi; C tarafından benimsendi

•

Örnek

a = a + b

şöyle yazılır

a += b

Copyright © 2007 AddisonWesley. All rights reserved. 1-28

Atama ifadeleri: tekil atama işleçleri

•

C temelli dillerde tekil arttırma ve azaltma

işleçleri atamalar ile kullanılabilir.

•

Örnekler

sum = ++count

sum = count++

count++

-count++

Copyright © 2007 AddisonWesley. All rights reserved. 1-29

Deyim olarak atama

•

C, C++, ve Java, da atama ifadesi bir sonuç

üretir ve işlenen(operand) gibi kullanılabilir.

•

Örnek:

while ((ch = getchar())!= EOF){…}

ch = getchar()yapılır; sonuç (ch ye

atanır) whileifadeyi için karşılaştırma

koşulu olarak kullanılır.

Copyright © 2007 AddisonWesley. All rights reserved. 1-30

Karışık kipli atama

•

Atama ifadeleri karışık kipli olabilir örneğin

int a, b;

float c;

c = a / b;

•

Pascal da, integer değişkenler real değişkenlere

atanır, fakat real değişkenler integer lara atanmaz

•

Java da, sadece genişletme zorlaması atamalarda

yapılır.

•

Ada da, atamada zorlama yoktur.