CS10003: Programming & Data Structures

Dept. of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Autumn 2020

Operations and Conditional Assignments

Operator Precedence and Associativity

An explicitly parenthesized arithmetic (and/or logical) expression clearly indicates the sequence of operations to be performed on its arguments.

However, it is quite common that we do not write all the parentheses in such expressions.

Instead, we use some rules of **precedence** and **associativity**, that make the sequence clear.

For example, the expression

a + b * c conventionally stands for

a + (b * c)

and not for (a + b) * c

Another ambiguity

Let us look at the expression a - b - c

Now the *common* operand b belongs to two same operators (subtraction).

They have the same precedence. Now we can evaluate this as

- (a - b) - c or as
- a - (b - c)
- Again the two expressions may evaluate to different values.
- The convention is that the first interpretation is correct.

In other words, the subtraction operator is *left-associative*.

Associativity and Precedence

Operator(s)	Type	Associativity
++ --	unary	non-associative
- ~	unary	right
* / %	binary	left
+ -	binary	left
<< >>	binary	left
&	binary	left
| ^	binary	left
= += -= *= etc.	binary	right

Unary operators

- Consider ++a and a++
  - there is a subtle difference between the two.
  - Recall that every assignment returns a value.
  - The increment (or decrement) expressions ++a and a++ are also assignment expressions.
- Both stand for "increment the value of a by 1". But then which value of a is returned by this expression? We have the following rules:
  - For a++ the older value of a is returned and then the value of a is incremented. This is why it is called the post-increment operation.
  - For ++a the value of a is first incremented and this new (incremented) value of a is returned. This is why it is called the pre-increment operation.

A sample code

```
#include<stdio.h>
main()
{
  int a, s;
  a=1;
  printf("a++=%d\n",a++);
  printf("++a=%d\n",++a);
}
```

Can lead to ambiguities…

```
#include<stdio.h>
main()
{
  int a, s;
  a=1;
  printf(“++a=%d,a++=\n",++a,a++);
  }
```

Conditions and Branching

Think about mathematical definitions like the following. Suppose we want to assign to y the absolute value of an integer (or real number) x. Mathematically, we can express this idea as:

y=0 if x = 0,

y = x if x > 0,

-x if x < 0.

Fibonacci numbers

$F_n = 0$ if $n = 0$,

$F_n = 1$ if $n = 1$,

$F_n = F_{n-1} + F_{n-2}$ if $n >= 2$.

Conditional World

If your program has to work in such a conditional world, you need two constructs:

- A way to specify conditions (like x < 0, or n >= 2).
- A way to selectively choose different blocks of statements depending on the outcomes of the condition checks.

Logical Conditions

Let us first look at the rendering of logical conditions in C.

A logical condition evaluates to a **Boolean value**, i.e., either "true" or "false".

For example, if the variable x stores the value 15, then the logical condition x > 10 is true, whereas the logical condition x > 100 is false.

Mathematical Relations

Relational operator	**Usage**	**Condition is true iff**
==	E_1 == E_2	E_1 and E_2 evaluate to the same value
!=	E_1 != E_2	E_1 and E_2 evaluate to different values
<	E_1 < E_2	E_1 evaluates to a value smaller than E_2
<=	E_1 <= E_2	E_1 evaluates to a value smaller than or equal to E_2
>	E_1 > E_2	E_1 evaluates to a value larger than E_2
>=	E_1 >= E_2	E_1 evaluates to a value larger than or equal to E_2

Examples

Let x and y be integer variables holding the values 15 and 40 at a certain point in time. At that time, the following truth values hold:

x == y False

x != y True

y % x == 10 True

600 < x * y False

600 <= x * y True

'B' > 'A' True

x / 0.3 == 50 False (due to floating point errors)

A funny thing about C is that it does not support any Boolean data type.

Instead it uses any value (integer, floating point, character, etc.) as a Boolean value.

Any non-zero value of an expression evaluates to "true", and the zero value evaluates to "false". In fact, C allows expressions as logical conditions.

Example:

0 False

1 True

6 - 2 * 3 False

(6 - 2) * 3 True

0.0075 True

0e10 False

'A' True

'\0' False

x = 0 False

x = 1 True

The last two examples point out the potential danger of mistakenly writing = in place of ==. Recall that an assignment returns a value, which is the value that is assigned.

Logical Operators

Logical operator	**Syntax**	**True if and only if**
AND	C_1 && C_2	Both C_1 and C_2 are true
OR	C_1 || C_2	Either C_1 or C_2 or both are true
NOT	!C	C is false

Examples

(7*7 < 50) && (50 < 8*8) True

(7*7 < 50) && (8*8 < 50) False

(7*7 < 50) || (8*8 < 50) True

!(8*8 < 50) True

('A' > 'B') || ('a' > 'b') False

('A' > 'B') || ('A' < 'B') True

('A' < 'B') && !('a' > 'b') True

Note

Notice that here is yet another source of logical bug. Using a single & and | in order to denote a logical operator actually means letting the program perform a bit-wise operation and possibly ending up in a logically incorrect answer

Associativity of Logical Operators

Operator(s)	**Type**	**Associativity**
!	Unary	Right
< <= > >=	Binary	Left
== !=	Binary	Left
&&	Binary	Left
||	Binary	Left

Examples

x <= y && y <= z || a >= b is equivalent to

((x <= y) && (y <= z)) || (a >= b).

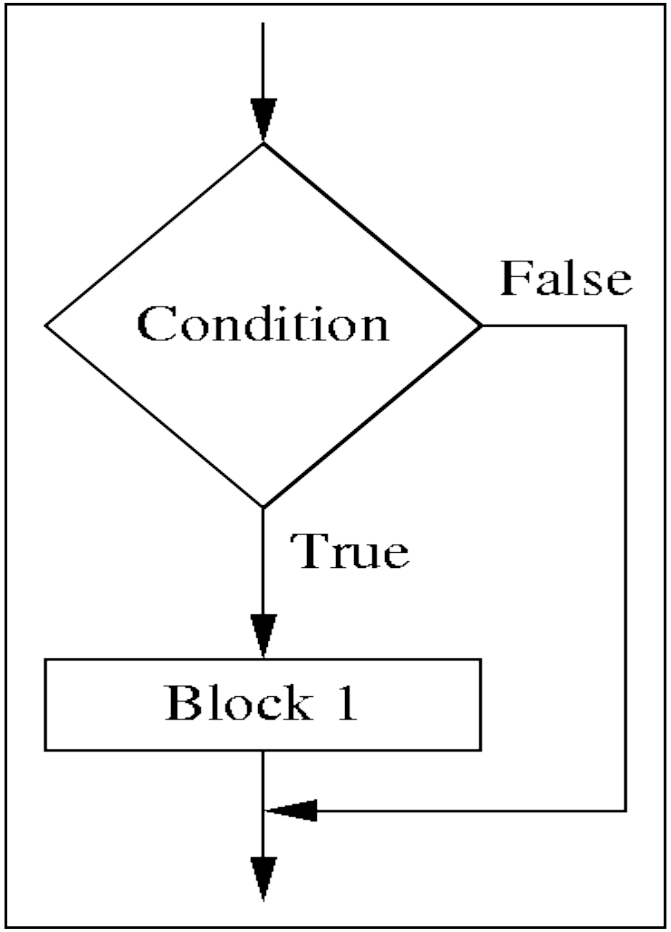
C1 && C2 && C3 is equivalent to

(C1 && C2) && C3.

a > b > c is equivalent to

(a > b) > c.

The If Statement

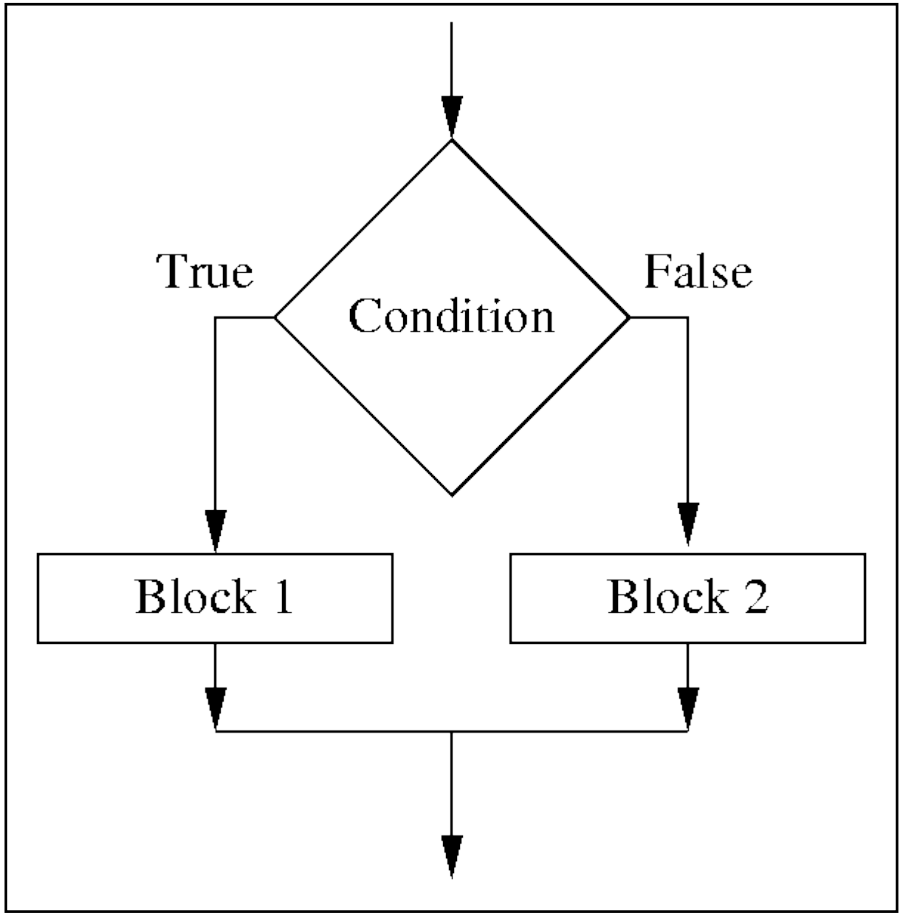


C Statement:

```
  if(Condition)
      Block1;
```

```
scanf("%d",&x);
if (x < 0) x = -x;
x=x+1;
```

The If else Statement



C Statement:

```
if (Condition)
 { Block 1 }
else { Block 2 }

scanf("%d",&x);
if (x >= 0) y = x;
else y = -x;
x=x+1;
```

Ternary Operator

Consists of two symbols: ? and :

- example,
  - larger = (i > j) : i : j;
- i and j are two test expressions.
- Depending on whether i > j, larger (the variable on the left) is assigned.
  - if (i > j), larger = i
  - else (i,e i<=j), larger = j
- This is the only operator in C which takes three operands.

The ternary statement

Consider the following special form of the if-else statement:

if (C) v = E1; else v = E2; Here depending upon the condition C, the variable v is assigned the value of either the expression E1 or the expression E2. This can be alternatively described as:

v = (C) ? E1 : E2; Here is an explicit example. Suppose we want to compute the larger of two numbers x and y and store the result in z. We can write:

z = (x >= y) ? x : y;

Comma Operator

int i, j;

i=(j=1,j+10);

What is the result? j=11.

Nested If else

Suppose that we want to compute the absolute value |xy| of the product of two integers x and y and store the value in z. Here is a possible way of doing it:

```
if (x >= 0)
        { z = x;
            if (y >= 0)  z *= y;
            else z *= -y; }
    else { z = -x;
            if (y >= 0) z *= y;
            else z *= -y; }
```

This can also be implemented as:

```
    if (x >= 0) z = x; else z = -x;
    if (y >= 0) z *= y; else z *= -y;
```

Here is a third way of doing the same:

```
if ( ((x >= 0)&&(y >= 0)) || ((x < 0)&&(y < 0)) )
          z = x * y;
  else z = -x * y;
```

Repeated if-else statements

A structure of the last figure can be translated into C as:

```
if (Condition 1)
        { Block 1 }
   else if (Condition 2)
        { Block 2 }
    else if ... ... }
   else if (Condition n)
       { Block n }
    else
      { Block n+1 }
```

Example

Here is a possible implementation of the assignment y = |x|:

```
scanf("%d",&x);
 if (x == 0) y = 0;
else if (x > 0) y = x;
 else y = -x;
```

The Switch Statement

```
switch (E)
   { case val1 : Block 1 break;
     case val2 : Block 2 break;
      ...
    case valn : Block n break;
    default:  Block n+1
   }
```

Example

```
char lang; ...
   switch (lang) {
  case ‘B’: printf(“Dhanyabad\n”); break;
  case 'E' : printf("Thanks\n"); break;
  case 'F' : printf("Merci\n"); break;
  case 'G' : printf("Danke\n"); break;
  case 'H' : printf("Shukriya\n"); break;
  case 'I' : printf("Grazie\n"); break;
  case 'J' : printf("Arigato\n"); break;
  case 'K' : printf("Dhanyabaadagaru\n"); break;
  default : printf("Thanks\n"); }
```

Switch is strange

Switch statements are strange.

It checks for the satisfying value of the condition it is checking.

Once a match is found, further checks are disabled and all the subsequent statements are done one after the other, irrespective of the condition.

Example

There are, however, situations where this odd behavior of switch can be exploited. Let us look at an artificial example. Suppose you want to compute the sum

n + (n+1) + ... + 10

Using the strangeness of Switch

```
switch (n) {
    case 0  :
    case 1  : sum += 1;
    case 2  : sum += 2;
    case 3  : sum += 3;
    case 4  : sum += 4;
    case 5  : sum += 5;
    case 6  : sum += 6;
    case 7  : sum += 7;
    case 8  : sum += 8;
    case 9  : sum += 9;
    case 10 : sum += 10;
              break;
    default : printf("n = %d is not in the desired range...\n", n);
}
```

Displaying a menu and using Switch

```
#include<stdio.h>
main()
{
  int choice;

  printf("Choice of destination:\n");
  printf("\t1 - Mercury\n");
  printf("\t2 - Venus\n");
  printf("\t3 - Mars\n");
  printf("Enter the number corresponding to your choice: ");
  scanf("%d",&choice);
```

```
switch(choice)
  {
    case 1:
     puts("Mercury is closest to the sun.");
     puts("So, the weather may be quite hot there.");
     puts("The journey will cost you 10000 IGCs.");
     //break;
   case 2:
     puts("Venus is the second planet from the sun.");
     puts("The weather is probably hot and poisonous.");
     puts("The journey will cost 5000 IGCs.");
     break;
```

The output menu

```
case 3:
     puts("Mars is the closest planet to earth in the solar system.");
     puts("There is probably some form of life there.");
     puts("The journey will cost 3000 IGCs.");
     break;
  default:
    puts("Unknown destination.\n");
    break;
 }
 puts("\n Note: IGC = Inter Galactic Currency\n");
```

```
-bash-3.2$ ./a.out
Choice of destination:
        1 - Mercury
        2 - Venus
        3 - Mars
Enter the number corresponding to your choice:
```