Control Instructions in C

As the name suggests the 'Control Instructions' enable us to specify the order in which the various instructions in a program are to be executed by the computer. In other words the control instructions determine the 'flow of control' in a program. There are four types of control instructions in C. They are:

(a) Sequence Control Instruction
(b) Selection or Decision Control Instruction
(c) Repetition or Loop Control Instruction
(d) Case Control Instruction

The Sequence control instruction ensures that the instructions are executed in the same order in which they appear in the program. Decision and Case control instructions allow the computer to take a decision as to which instruction is to be executed next. The Loop control instruction helps computer to execute a group of statements repeatedly.

The Decision Control Structure Decisions

By default the instructions in a program are executed sequentially. However, in serious programming situations, seldom do we want the instructions to be executed sequentially. Many a times, we want a set of instructions to be executed in one situation, and an entirely different set of instructions to be executed in another situation. This kind of situation is dealt in C programs using a decision control instruction. As mentioned earlier, a decision control instruction can be implemented in C using:

(a) The if statement

(b) The if-else statement

(c) The conditional operators

The if Statement

Like most languages, C uses the keyword if to implement the decision control instruction. The general form of if statement looks like this:

```
if ( Condition  )
execute this statement ;
```

The keyword if tells the compiler that what follows is a decision control instruction. The condition following the keyword if is always enclosed within a pair of parentheses. If the condition, whatever it is, is true, then the statement is executed. If the condition is not true then the statement is not executed; instead the program skips past it. But how do we express the condition itself in C? And how do we evaluate its truth or falsity? As a general rule, we express a condition using C's 'relational' operators. The relational operators allow us to compare two values to see whether they are equal to each other, unequal, or whether one is greater than the other. Here's how they look and how they are evaluated in C.

this expression	is true if
x == y	x is equal to y
x != y	x is not equal to y
x < y	x is less than y
x > y	x is greater than y
x <= y	x is less than or equal to y
x >= y	x is greater than or equal to y

The relational operators should be familiar to you except for the equality operator == and the inequality operator !=. Note that = is used for assignment, whereas, == is used for comparison of two quantities. Here is a simple program, which demonstrates the use of if and the relational operators.

```
/* Demonstration of if statement */

main( )
{
int num ;
printf ( "Enter a number less than 10 " ) ;
scanf ( "%d", &num ) ;
if ( num <= 10 )
printf ( "What an obedient servant you are !" ) ;
}
```

On execution of this program, if you type a number less than or equal to 10, you get a message on the screen through printf( ). If you type some other number the program doesn't do anything.

The if-else Statement

The if statement by itself will execute a single statement, or a group of statements, when the expression following if evaluates to true. It does nothing when the expression evaluates to false. Can we execute one group of statements if the expression evaluates to true and another group of statements if the expression evaluates to false? Of course! This is what is the purpose of the else statement that is demonstrated in the following example:

Example 2.3: In a company an employee is paid as under:If his basic salary is less than Rs. 1500, then HRA = 10% of basic salary and DA = 90% of basic salary. If his salary is either equal to or above Rs. 1500, then HRA = Rs. 500 and DA = 98% of basic salary. If the employee's salary is input through the keyboard write a program to find his gross salary.

```
main( )
{
float bs, gs, da, hra ;
printf ( "Enter basic salary " ) ;
scanf ( "%f", &bs ) ;
if ( bs < 1500 )
{
hra = bs * 10 / 100 ;
da = bs * 90 / 100 ;
}
else
{
hra = 500 ;
da = bs * 98 / 100 ;
}
gs = bs + hra + da ;
printf ( "gross salary = Rs. %f", gs ) ;
}
```

A few points worth noting...

1. The group of statements after the if upto and not including the else is called an 'if block'. Similarly, the statements after the else form the 'else block'.
2. Notice that the else is written exactly below the if. The statements in the if block and those in the else block have been indented to the right. This formatting convention is followed throughout the book to enable you to understand the working of the program better.
3. Had there been only one statement to be executed in the if block and only one statement in the else block we could have dropped the pair of braces.
4. As with the if statement, the default scope of else is also the statement immediately after the else. To override this default scope a pair of braces as shown in the above example must be used.

Nested if-elses

It is perfectly all right if we write an entire if-else construct within either the body of the if statement or the body of an else statement. This is called 'nesting'of ifs. This is shown in the following program.

```
main( )
{
int i ;
printf ( "Enter either 1 or 2 " ) ;
scanf ( "%d", &i ) ;
if ( i == 1 )
printf ( "You would go to heaven !" ) ;
else
{
if ( i == 2 )
printf ( "Hell was created with you in mind" ) ;
```

```
else
printf ( "How about mother earth !" ) ;
}
}
```

Note that the second if-else construct is nested in the first else statement. If the condition in the first if statement is false, then the condition in the second if statement is checked. If it is false as well, then the final else statement is executed.

You can see in the program how each time a if-else construct is nested within another if-else construct, it is also indented to add clarity to the program. Inculcate this habit of indentation, otherwise you would end up writing programs which nobody (you included) can understand easily at a later date.

In the above program an if-else occurs within the else block of the first if statement. Similarly, in some other program an if-else may occur in the if block as well. There is no limit on how deeply the ifs and the elses can be nested.