Decision Control Statements

The default execution sequence of a program is sequential flow. At times it is required that based upon a test condition a decision must be taken that alters the flow of program flow. We want a set of instructions to be executed in one situation and an entirely different set of instructions be executed in another situation. This kind of situation is dealt in C language with the help of decision control instructions. These can be implemented in C language using:
a) The **if** statement.
b) The if-else statement.

The if statement is one of the most powerful devices in programming. It allows the program to choose an action based upon input. For example, a program can be written to decide if a user can access the program by using an if statement to check if the password is correct or not.

Without a conditional statement such as if programs would run almost the exact same way every time. If statements allow the flow of the program to be changed.

There are many things to understand when using if statements. You must understand Boolean operators such as OR, NOT, and AND. You must also understand comparisons between numbers and most especially, the idea of true and false in the same way computers do. The Boolean operators will be discussed later.

True and false in C++ are denoted by a non-zero number, and zero. Therefore, 1 is TRUE and 8.3 is TRUE but 0 is FALSE. No other number is interpreted as being FALSE.

When programming, the program often will require the checking of one value stored by a variable against another value, to determine which is larger, smaller, or if the two are equal. To do so, there are a number of operators used.

The relational operators, as they are known, along with examples:

>	greater than	5>4 is TRUE
<	less than	4<5 is TRUE
>=	greater than or equal	4>=4 is TRUE
<=	less than or equal	3<=4 is TRUE

NOT: This just says that the program should reverse the value. For example, NOT (1) would be 0. NOT (0) would be 1. NOT (any number but zero) would be 0. In C NOT is written as !.

AND: This is another important command, it returns a one if 'this' AND 'this' are true. (1) AND (0) would come out as 0 because both are not true, only 1 is true. (1) AND (1) would come out as 1. (ANYREAL NUMBER BUT ZERO) AND (0) would be 0. (ANY REAL NUMBER BUT ZERO) AND (ANY REAL NUMBER BUTZERO) would be 1. The AND is written as && in C. Do not be confused and think that it checks equality between numbers. It does not. AND will be evaluated before OR.

OR: Very useful is the OR statement! If either (or both) of the two values it checks are TRUE then it returns TRUE. For example, (1) OR (0) would be 1! (0)OR(0) would be0. (ANY REAL NUMBER) OR (ANY REAL NUMBER BUT ZERO) would be 1! The OR

is written as || in C. Those are the pipe characters. On your keyboard, they may look like a stretched colon. Keep in mind that OR will be evaluated after AND.

The next thing to learn is to combine them... What is !(1 && 0)? Of course, it would be TRUE (1). This is because 1 && 0 evaluates two 0 and ! 0 equals 1.

Try some of these...

!(1 || 0)	ANSWER: 0
!(1 || 1 && 0)	ANSWER: 0 (AND is evaluated before OR)
!((1 || 0) && 0)	ANSWER: 1 (Parenthesis are useful)

The structure of an if statement is essentially:

```
if (TRUE)
  Do whatever follows on the next line.
```

To have more than one statement execute after an if statement (when it evaluates to true) use brackets. For example,

```
if (TRUE)
{
  Do everything between the brackets.
}
```

There is also the else statement. The code after it (whether a single line or code between brackets) is executed if the IF statement is FALSE.

It can look like this:

```
if(condition is FALSE)
{
  Not executed if its false
}
else
{
  do all of this
}
```

One use for else is if there are two conditional statements that will both evaluate to true with the same value (but only at times. For example, x<30 and x<50 will both return true if x is less than 30 but not otherwise), but you wish only one of them to be checked. You can use else after the if statement. If the if statement was true the else statement will not be checked.

Let's look at a simple program for you to try out on your own...

```
int main()                      /*Most important part of the
program! */
{
  int age;                      /*Need a variable... */

  printf("\nPlease input your age: "); scanf('"%d",&age);
     /*Asks for age*/
```

```
  if(age<100)       /*If the age is less than 100 */
  {
     printf("\n You are pretty young!");
          /*Just to show it works */
  }
  else if(age==100)
          /*I use else just to show an example */
  {
     printf("You are old");
               /*Just to show you it works...*/
  }
  else if(age>100)
  {
     printf("You are really old");
     /*Proof that it works for any condition*/
  }
}
```

Different forms of IF

Sr.	Forms of if
1.	if(condition) Do this;

2.	if(condition)
	{
	do this;
	and this;
	}

3.	if(condition)
	do this;
	else
	do this;

4.	if(condition)
	{
	do this;
	and this
	}
	else
	{
	do this;
	and this;
	}

5.	if(condition)
	do this;
	else
	{
	if(condition)
	do this;
	else
	{
	do this;
	and this;
	}
	}

6.	if(condition)
	{
	if(condition)

	do this;
	
	else
	{
	do this;
	and this;
	}
	}
	else
	do this;

What will be output of the following program:

```
main()
{
      int i;
      printf("\nEnter the value of integer");
      scanf("%d",&i);
      if(i=5)
      {
            printf("\nThe number is Five");
      }
      else
      {
            printf("\nThe number is not Five");
      }
}
```

At the first glance the program appears to say whether a number is five or not. BUT
The above program will always write "Number is Five" for any value of i because the condition is written wrong here. The assignment operator (=) is used in place of == and recall that for C anything other than 0 is TRUE the expression always evaluates to TRUE. However such types of mistakes can be prevented by a simple way of writing code.
Instead of writing

if(i==5)

develop a habit of always writing a constant first i.e.
if(5 == i)
(compare i with 5 or compare 5 with i the results are always going to be the same. BE CAREFUL IN > & < because it will change 5 > i not the same as i > 5)
this way if you ever miss one = in writing the statement
if(5=i)
will just not compile as the compiler will flash an error as you are assigning value of i to a constant 5 which is absurd.