Decision Making

Decision is a word which is normally taken in a moment where one is in a position to select one option from the available options which are obviously more than one. While writing programs this concept is always going to play an important role. We will have to provide the capabilities to our program so that it can take decisions on its own depending upon the various possible inputs from the user.

When you are going to write some program it will be always necessary that you will have to put some code which has to be executed only when a specific condition is satisfied. These conditions are evaluated by the computer itself when the input is given by the user.

In C language there are various methods which can be used to select an appropriate set of statements depending upon the user's input. On counting them, totally there are FOUR different ways to take decisions which are as follows:

(a) if Statement
(b) if-else Statement
(c) Conditional Operators
(d) Switch Statement

Confused, which one to use!!! All the above four methods can be used for decision making but their application will vary, you will feel comfortable after making a few programs using them.

If Statement

As the word itself is clarifying its meaning, it is used to check a condition and perform a set of statements according to the result of the condition checked.

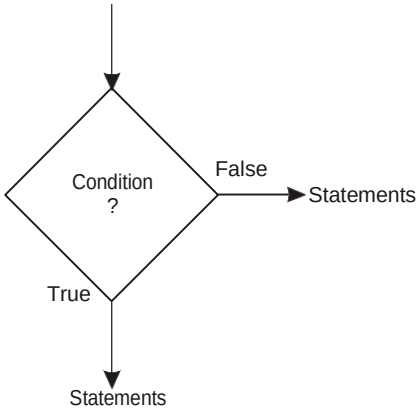


Fig. 3.1

Syntax:

```
if (condition)
{
     ...
     ...                                  // True_Block statements;
     …
}
els
e
{    …
     ...                                  // False_Block Statements;
     …

}
```

In the above given syntax "condition" means any relational condition or Any mathematical expression. If the condition given in the brackets (parenthesis) is TRUE then the statements within the BRACES executed.

In case of mathematical expressions every non-zero value (i.e. +ve or -ve) is considered to be TRUE and zero value is considered as FALSE.

The statements given in BRACES after the "else" part are executed when the condition is false (not satisfied).

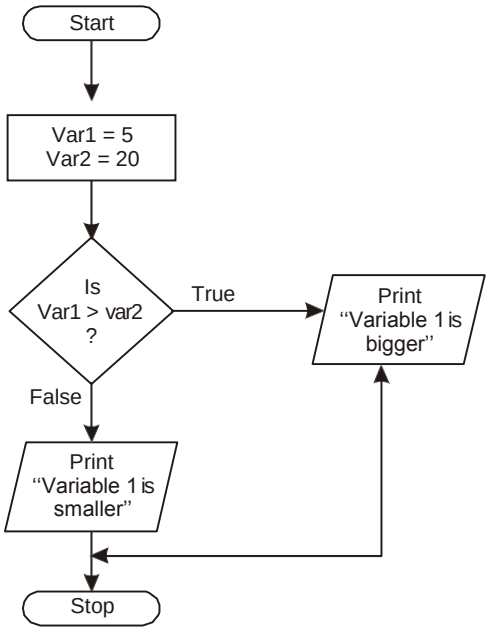
if with a relational expression

In the previous chapter there is a list of all possible relational operators. We can use any of those relational operators in the if statement.

Example 1. Write a program to print which variable is bigger in two variables.

Solution:

Flowchart:



Program:

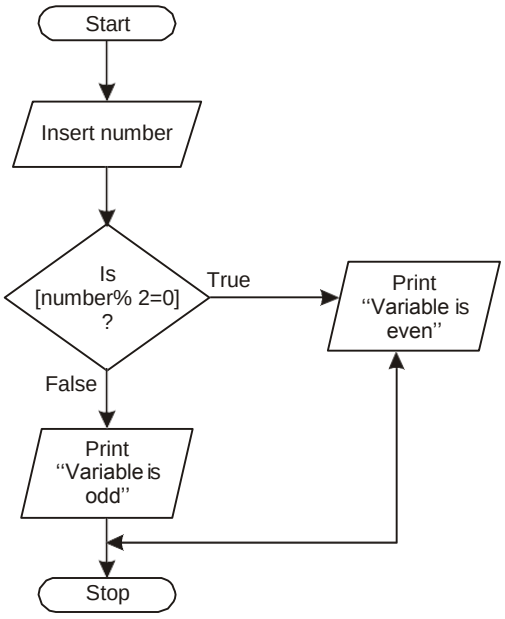
```
#include<stdio.h>
main ()
    {
        int VAR1=5, VAR2=20;
        if(VAR1 > vaR2)
                {
                    printf("Variable 1 is bigger ");
                }
        else
                {
                printf("Variable 1 is smaller");
                }
    }
```

Explanation: There are two variables in the above program which are having some values. In the given example the value of VAL1 is smaller than VAR2, so the condition for if is false. According the program is going to print the output. You guessed right. The output will be "Variable 1 is smaller".

Example 2. Write a program to check if a given number is even or odd.

Solution:

Flowchart :



Program:

```
#include<stdio.h>
main ()
{
    int var;
    printf("Enter a Number : ");
    scanf("%d",&var); if(VAR%2=
    =0)
        {
printf("You entered a Even Number");
        }
    else
        {
printf("You entered a Odd Number");
        }
    }
```

Explanation : The above program is using **"var%2"** (% - modulo operator) as a part of condition. This is going to return the remainder after dividing variable "var" with 2. If any number is fully divided by 2 it will be taken as a Even Number.

Table 3.1. Table showing results when relational operators are used with IF statement

if(x == y)	Evaluates the expressions of TRUE Block if values of

	variable “x” and “y” are equal.
if(x > y)	Evaluates the expressions of TRUE Block if value of variable “x” is greater than value of variable “y”.
if(x >= y)	Evaluates the expressions of TRUE Block if value of variable “x” is greater than or equal to value of variable “y”.
if(x < y)	Evaluates the expressions of TRUE Block if value of variable "x" is smaller than value of variable “y”.
if(x <= y)	Evaluates the expressions of TRUE Block if value of variable “x” is smaller than or equal to value of variable “y”.
if(x !=y)	Evaluates the expressions of TRUE Block if value of variable “x” is not equal to value of variable “y”.

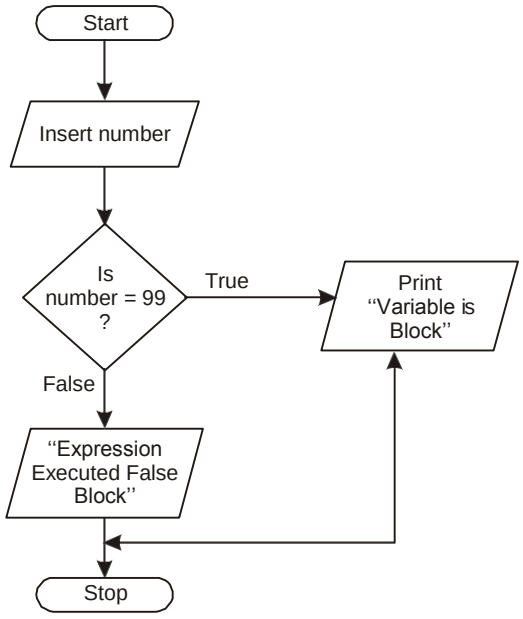
if with a mathematical expression

In case of relational operators it's clear as to which block will be executed. But we can also use a mathematical expression in IF statement. How will IF statement evaluates it?

IF statement will evaluate the True_block when the mathematical expression returns a NON-ZERO (i.e. TRUE) and if it returns a ZERO value then it will evaluate the expression given in the False_block.

Example 3. Write a program that execute true block if inserted number is equal to 99 otherwise false block.

Solution : Flowchart



Program :

```
#include<stdio.h>
main()
```

```
{
    int VAR1;
    printf("Enter a value: ");
    scanf("%d",&var1) ;
    if (VAR1 == 99)
    {
            printf("Execute True Block");
    }
    els
    e
    {       printf("Executed False Block");

    }
}
```

Explanation : The true statement will be executed when the entered value is 99. But if user enters greater than or less than 99 then it will execute the false block.

Example 4.

```
#include<stdio.h>
main()
    {
        int VAR1;
        printf("Enter a value: ");
        scanf("%d",&var1);
        if( VAR1)
        {
                printf("Expression Executed True Block");
        }
        els
        e
        {       printf("Expression Executed False Block");

        }
    }
```

Explanation : In the above example, if statement is now not having any condition. "if (VAR1)" is a valid statement and will execute the True_block if user enters a non-zero (+ve or –ve) and Flase_block will be executed if value or VAR1 is ZERO.

Example 5.

```
#include<stdio.h>
main ()
    {
        int VAR1;
        printf("Enter a value : ");
        scanf("%d", &var1);
```

```
        if( VAR1 > 90 };
        {
                printf("Expression Executed True Block");
        }
        els
        e
        {       printf("Expression Executed False Block");

        }
    }
```

Will the program work? NO! Compiler gives an error "Misplace else in Line 11"

Explanation : In the IF statement there is a problem: If we apply a SEMI-Colon at the end of IF-statement which signifies that if the condition in if statement satisfied we do not want to do any thing. In the next line there is a block of statements in braces which is out of the scope of IF-statement. So we can say that the if-statement ends in that its line itself.

When the compiler comes down there is an ELSE statement followed by a set of statements enclosed in BRACES. That's why there is an error "Misplaced else". So IF-statement is dangerous in such cases.

Example 6.

```
#include<stdio.h>
main()
{
    int VAR1;
    printf("Enter a value: ");
    scanf("%d",&var1);
    if( VAR1 > 90 };
    else
    {
        printf("Expression Executed False Block");
    }
}
```

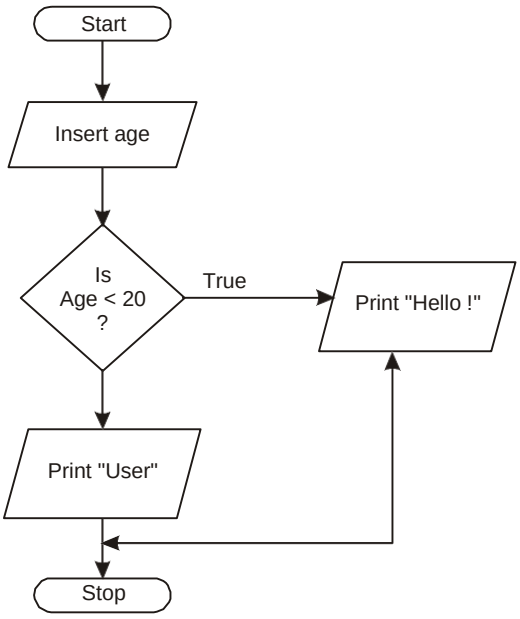
Explanation : Example 6 is an error free program. If the value for var1 is greater than 90, no message will be displayed. If the value of VAR1 is less then or equal to 90 the output will be "Expression Executed False Block".

Are Brackets going to Matter?

Example 7.

```
#include<stdio.h>
main ()
{
    int age;
    printf("Enter your age: ");
    scanf("%d",&age);
    if( age < 20 )
```

```
    printf ("Hello!”);
    printf("User”);
}
```



Execution of Program :
Input :
Enter your age: 25
Output: User
Output :
Enter your age: 2
Output: Hello! User

Explanation: If the user enters age less than 20. The condition for the above program will logically correct. There are no braces enclosing for the if-statement hence the first statement is taken as an true_block statement and the second statement is taken as a normal statement. Second statement is not effected by the condition.

Expressions	Result
if(!age)	Executes True_block if value of age is equal to 0.
if(!!age)	If there are two! (not) operators used they are going to cancel each other and final result depends upon the value of age only.
if(age-50)	Executes True_block if value of age is not 50.
if(59)	Always executes the True_block as 59 is a Non_zero constant.
if(!a==0)	Will first negate "a". If "a" is non-zero it becomes zero and vice-versa.

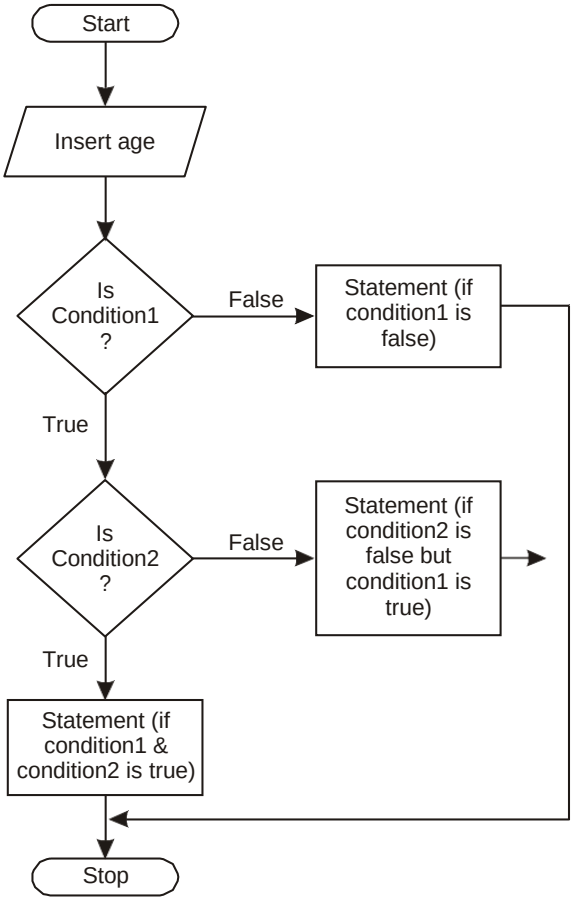
	After that "a" is compared with 0 and executed accordingly.
if (! (a==0))	Will first compare "a" with 0 and the result will be reversed. Suppose, if "a==0"

Nested IF Statements

There are times when it will be required to check more than one conditions and according to their collective results would like to perform various tasks. We can use IF statements within another if statements for any number of times required. Although there is not any separate syntax for if- else statements.

Syntax :

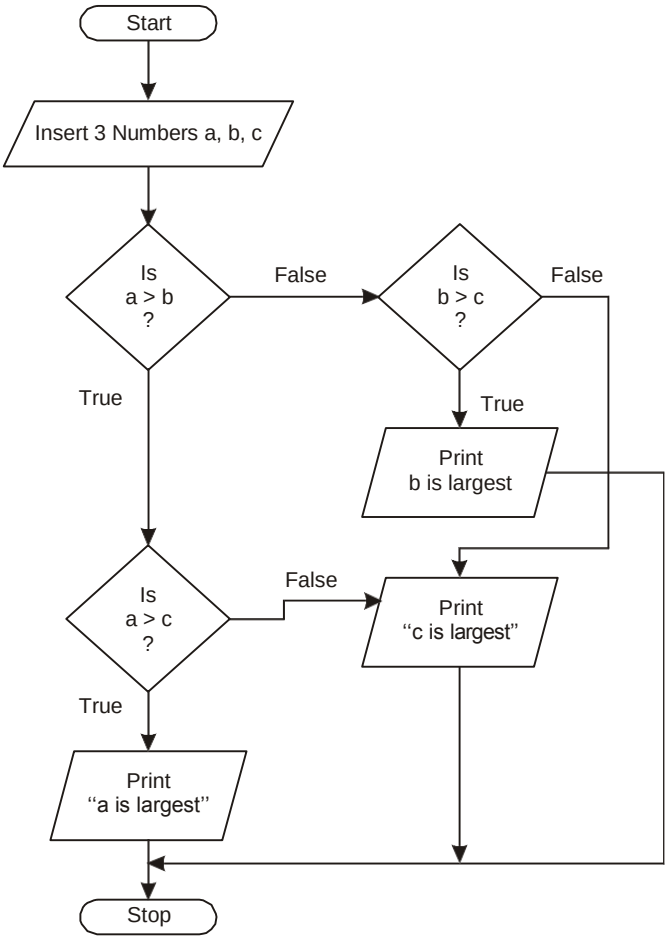
```
if(condition1)
    {
        if(condition2)
        {
        …                   //statements
        }
        else
        {
        …                   //statements
        }
    }
    else
    {
    …                       //statements
    }
```



Example 8. Write a program to find Maximum out of three numbers.

Solution :

Flowchart :



Program :

```
#include<stdio.h>
main ()
{
    int a, b, c;
    printf("Enter three numbers a, b and c respectively: ");
    scanf("%d%d%d",&a, &b, &c);
        if(a > b)
        {
                if(a > c)
                {
                        printf("a is Largest");
                }
                els
                e       printf("c is Largest");
                {

                }
```

```
        }
        else
        if(b > c)
        {
                printf("b is
        }
        els
        e
                Largest"); printf("c

        }
}
                is Largest");
```

Execution of Program :
Intput : Enter three numbers a, b and c respectively:
Output : b is largest.

Explanation : The above program looks bigger in size but it's very simple. At the start the user enters three values stored in a, b and c respectively. First IF statement checks if a is greater than b and if the condition is true it enters into the True- block otherwise in the False-block.

With the True and False blocks there is again a IF condition which further makes check for the largest number and finally prints name of the variable holding the largest value.

Condition Connectors

Operator Example	Treatment
&&	if(a>5 && a<10) Executes True block if both the conditions are satisfied, otherwise False_block is executed.
||	if(a>5 II a<10) Executes True_block if any of the condition is satisfied.

Example 9.

```
#include<stdio.h>
main()
{
int a=10, b=20;
if(a > 5 && b >
10)
{
printf("Both conditions are satisfied\n");
}
if(a > 15 || b > 15)
{
printf ("Anyone condition is compulsory true");
}
}
```

Conditional Operator

Recall Example 8, the program was written to find maximum out of three numbers and the program became big in size and complicated. C provides a

conditional operator (? :) which can be used to check conditions within one line and result of the condition can be stored somewhere else.

The syntax to be followed for the conditional operator is as follows : var = (conditional_statement) ? True_block/expression :
False_block/expression;

In the above syntax shows that the first parameter for the conditional operator will be a condition (Relational condition or mathematical expression) and depending upon the condition either True_block or the False_block is going to be executed for which the result will be taken to a variable which is at the left hand side.

```
ans=(a>b)?a:b;
printf("%d",ans);
```

In the above the condition a>b will be checked and if it is satisfied then a will be assigned to the variable ans otherwise b will be assigned to ans. We can extend this idea and find maximum for three numbers.

Example 10. Write a program to find Maximum out of three numbers (Using Conditional Operator)

Program :

```
#include<stdio.h>
main ()
{
int a, b, c, max;
printf("Enter three numbers:
"); scanf("%d%d%d",&a, &b,
&c);
max = (a>b) ? (a>c)? a:c: (b>c)? b:c;
printf("Maximum value =
%d\n",max);
}
```

The operator works with ease when there are few conditions (1-3) to be checked, but it becomes complicate when there is a need to check more than 3 or 4 conditions.

Example 11. Write a program to give bonus according to grade of employees. There are only following three grades available.

Employee Grades	Bonus
101	1000/-
102	1500/-
103	2000/-

Program :

```
#include<stdio.h>
main()
{
    int salary, grade, bonus, basic;
    printf("Enter basic salary and grade of employee
    :"); scanf("%d%d", &basic, &grade);
```

```
    salary = basic + (grade = = 101) ? 1000 : (grade = = 102) ? 1500:2000;
    printf("Total Salary for employee = %d",salary);
}
```

If conditions increased :

SWITCH Statement

C provides however another method which can be used to check a good number of conditions and will work accordingly. **SWITCH** is a method which is going to take an integer parameter and depending upon that parameter will do the given operations.

The syntax for Switch is as follows :

```
switch (integer_expression)
{
    case value:
    statements
    ; break;
    case value:
    statements
    ; break;
    default:
    statements
    ; break;
}
```

As the syntax shows that within parenthesis of switch we will always have to give an integer expression and depending upon the value of that expression switch will select the equivalent case. After executing the case statements if break statement comes then the control exit from the SWITCH. If in any case the break is missing then all the statements below that case are also executed. If there is no case value which matches the value in the switch then statements in the default block will be executed.

Example 12. Write a program to calculate grades for a student who appeared in four tests. The criteria for allocating grade are as follows :

Average Marks Obtained	Grades
>=80	Distinction
>=60	First Division
>=50	Second Division
>=40	Pass
Otherwise	Fail

Program :

```
#include<stdio.h>
main()
{
    int marks1, marks2, marks3, total, average;
    printf("Enter marks for three subjects: ");
    scanf("%d%d%d",&marksl,&marks2,&marks3
    ); total = marksl+marks2+marks3;
    average = total/3;
    switch (average/10)
    {
    case 10:
    case 9:
```

```
    case 8:
    printf("Distinction");
    break;
    case 7:
    case 6:
    printf("First Division");
    break;
    case 5:
    printf("Second
    Division"); break;
    case 4:
    printf("Pass")
    ; break;
    default:
    printf("Fail");
    }
}
```

Execution of Program :

Input : Enter marks for three subjects
: 55
66
77
First Division.

Explanation : In the program average is an integer variable holding the average marks for the student. Because of integer type it only stores the integer part of the value. We are loosing the accuracy by leaving the fractional part but that is necessary because the switch statement can only operate on integer expressions.

The value of average can be between 0-100 and putting condition for each value is a typical task. The switch statement is having average/10 as an expression which reduces the possible set of values for the switch expression from 0-100 to 0-10 only.

What will happen if the average marks for a student is 73.

The integer expression for the switch statement is = 7. So control comes on the case 7 but there is no break statement. Due to absence of break statement control comes on case 6 and print "First Division". Printf statement is followed by a break statement which brings the control out.

Example 13. Write a program to generate Electricity Bills according to the following conditions :

Units Consumed	Charges
<=200	Re. 0.50 per unit
<=400	Rs. 100 + Re. 0.65 per unit
<=600	Rs. 230 + Re. 0.75 per unit
Otherwise	Rs. 380 + Re. 1 per unit

Program :

```
#include<stdio.h>
main()
{
    int c_no, units;
```

```
    float bill;
    printf("Enter consumer number and units consumed:
    "); scanf("%d%d",&c_no, &units);
    switch((units-1)/200)
    {
        case 0:
        bill = units * 0.50;
        break;
        case 1:
        bill = 100 + (units-200) * 0.65;
        break;
        case 2:
        bill = 230 + (units-400) * 0.75;
        break;
        default:
        bill = 380 + units-600;
    }
    printf("Consumer Number:
    %d\n",c_no); printf("Units Consumed:
    %d\n",units); printf("Total Bill:
    %f\n",bill);
}
```

Execution of Program :

Input : Enter consumer number and unit consumed : 101
100

Output : Consumer Number : 101
Unit consumed :
100 Total Bill : 50

Explanation : From the specified conditions we can see that we can divide units consumed by 200 to get a smaller number of cases but the problem is there in case a consumer consumes exactly 200/400 or 600 units then the group may get changed. To overcome from that condition we wrote (units-1)/200; that mean reduced units consumed by one and then divide it by 200. The reduction is not actual because the given below two statements are entirely different.

```
x = x - l;          /* Reduces the value of x by 1. */
y = x - l;          /*Assigns y value 1 less that the value of x, x
                      remains unchanged.*/
```

Prefix and Postfix Operators

C provides prefix and postfix operators which can be used with the expressions so that more than one job is done by single expression.

Prefix: If there are more than one operation to be carried in an expression and anyone of them is a prefix operation then that will be the first operation to be carried out.

Example: ++a (i.e. It will increase the value of a by 1 then assign the value.)

Postfix: Here if there are more than one operation to be carried in an expression and any one of them is a postfix operation then that will be the last operation to be carried out.

Example: a++ (i.e. It will assign the value of a and then increment by 1.)

Program :

```
void main()
{
int i =1;
    printf("%d%d%d", i++, i++, i++);
}
```

Execution of Program :

Output :

123

Still Confusing ?

Example :

This can be confusing at first, but if x is an integer whose value is 5 and you write

```
int a = ++x;
```

You have told the compiler to increment x (making it 6) and then fetch that value and assign it to a. Thus, a is now 6 and x is now 6.

If, after doing this, you

```
write int b = x++;
```

You have now told the compiler to fetch the value in x (6) and assign it to b, and then go back and increment x. Thus, b is now 6, but x is now 7.

Example 14.

```
#include<stdio.h>
main ()
{
int a=10, b=20;
if(++a > 5 && ++b > 10)
{
printf("a = %d, b = %d", a, b);
}
```

Result: a = 11, b = 21

In the above program change the if-condition which is given in bold letter with the given below line and notice the difference.

if(++a > 5 || ++b > 10)

Result: a = 11, b = 20

(Because in || (OR) if the first condition is satisfied then the compiler is not going to check the second condition and hence the b is not incremented and remains 20)