UNIT-5
DECISION MAKING

5.1 Decision Making in Programming:

C is a set of statements which are normally executed sequentially in the order in which they appear. This happens when no options or no repetitions of certain calculations are necessary. This involves a kind of decision making to see whether a particular condition has occurred or not and then direct the computer to execute certain statements accordingly.

C language possesses such decision making capabilities by supporting the following statements:

1. **if** statement
2. **switch** statement
3. Conditional operator statement
4. **goto** statement

These statements are popularly known as Decision making statements. These statements 'control' the flow of execution, they are known as Control statements.

5.2 Decision Making Statements:

C programming language provides following types of decision making statements:

Statements	Description
If statement	An if statement consists of a Boolean expressions followed by one or more statements.
If.....else	An if statement can be followed by an expression optional else statement which executes when the Boolean expression is false.
Nested if	You can use one if or else if statement inside another if or else if statements.
Switch statement	A switch statement allows a variable to be tested for equality against a list of values.
Nested switch statement	You can use one switch statement inside another switch statement.

5.3 Relational Operators with Precedence:

Following table shows the Relational Operators:

Operator	Description
==	Checks if the values of two operands are equal or not, if yes then the condition becomes true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.
>=	Checks if the value of left operand is greater than or equal to right operand, if yes then the condition becomes true.
<=	Checks if the value of left operand is less than or equal to right operand, if yes then the condition becomes true.

Precedence:

< , > ,<= , >= 1^{st} precedence

== , != 2^{nd} precedence

Associativity:

Left to Right

5.4 Logical Operators with Precedence:

Following table shows all the Logical Operators supported by C language.

Operator	Description
&&	It is called Logical AND operator. If both the operands are non-zero, then the condition becomes TRUE.
||	It is called Logical OR operator. If one of the operand is non-zero, then the condition becomes TRUE.
!	It is called Logical NOT operator. Use of reverse the logical state of its operand. If a condition is TRUE then Logical NOT operator will be FALSE.

Precedence:

! 1st precedence
&& 2nd precedence
|| 3rd precedence

Associativity:

Left to Right

5.6 Simple If Statement:

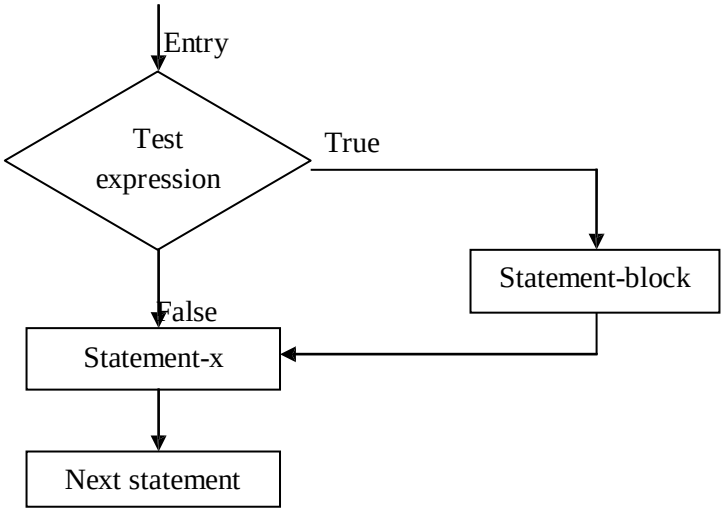
The general form of a simple **if** statement is:

```
if(test expression)
  {
        Statement-block;
  }
Statement-x;
```

The 'statement-block' may be a single statement or a group of statements. If the *test expression* is true, the *statement-block* will be executed; otherwise the statement-block will be skipped and the execution will jump to the *statement-x*.

Note: When the condition is true both the statement-block and the statement-x are executed in sequence.

The flowchart for simple **if** is given below:



Example program:

```
#include<stdio.h>
main()
{
    int  n;
    clrscr();
    printf("Enter  n value\n");
    scanf("%d",&n);
     if(n<0)
       {
          n=-n;
       }
    printf(" AbsoluteValue  =%d\n",n);
    getch();
}
```

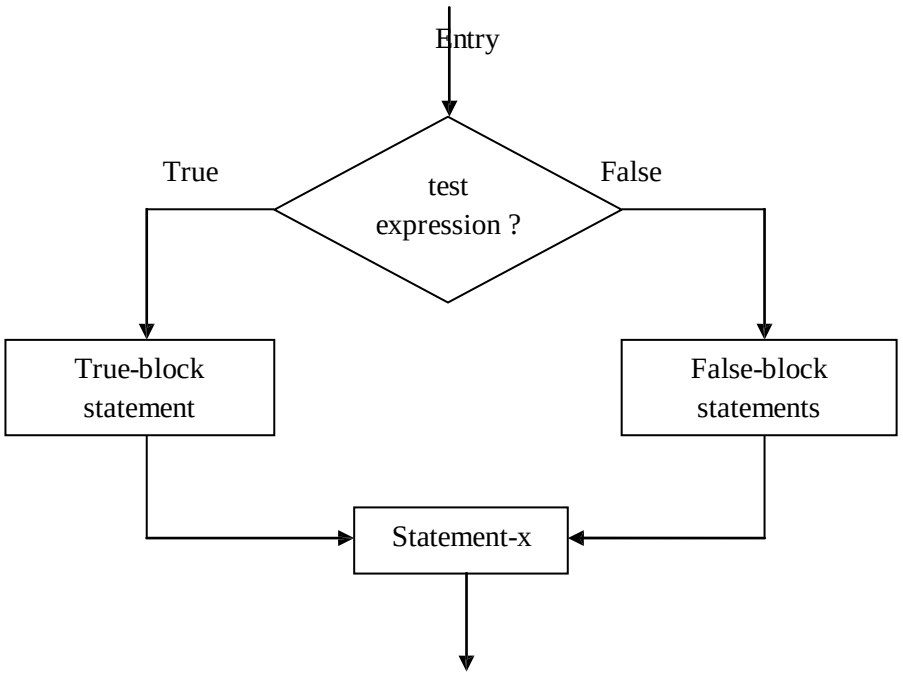
5.7 If.....Else statement:

The **if.....else** statement is an extension of the simple **if** statement. The general form is:

```
if(test expression)
    {
        True-block statements;
    }
else
    {
        False-block statements;
    }
Statement-x;
```

If the *test expression* is true, then the *true-block* statements, immediately following the **if** statements are executed; otherwise, the *false-block* statements are executed.

The flow chart for **if.....else** statement is shown below



Example program:

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

```
    printf("Enter any number\n");
    scanf("%d",&n);
    if(n%2==0)
       printf("The given number is even\n");
    else
       printf("The given number is odd\n");
    getch();
}
```

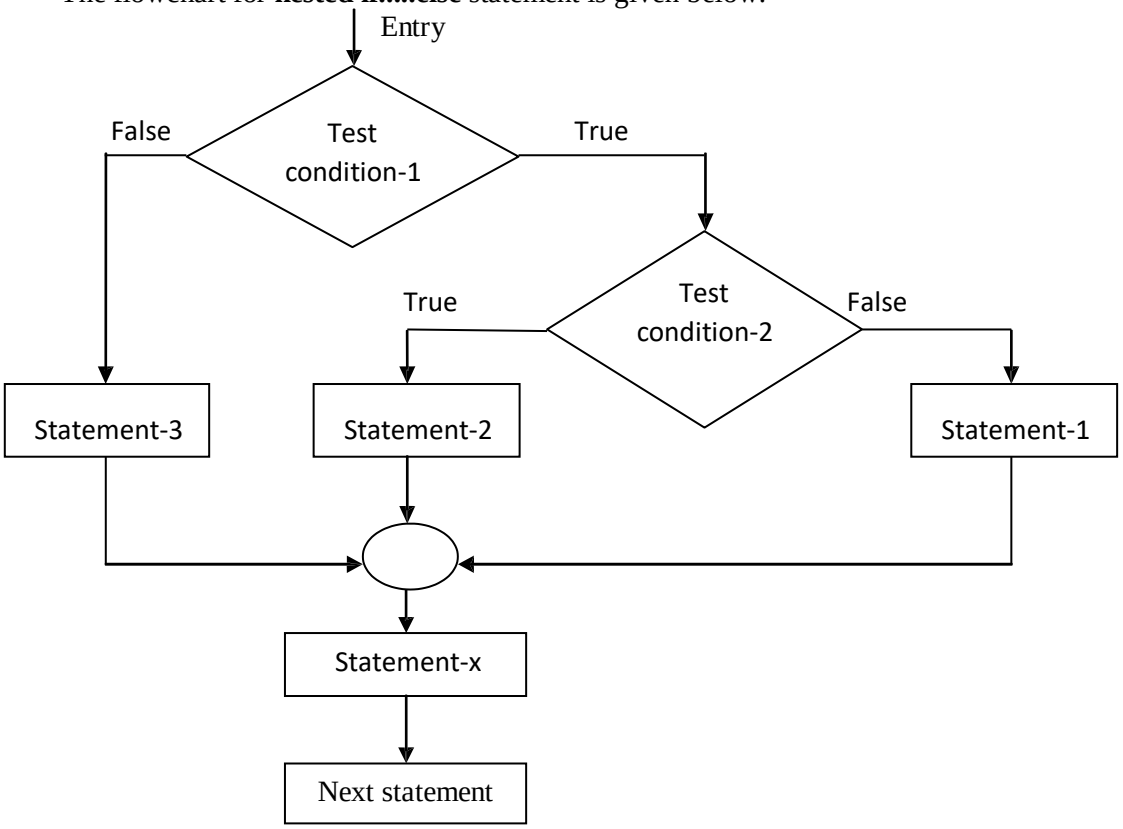
5.7.1 Nesting of if.....else statements:

When a series of decisions are involved, we may have to use more than one if....else statement in nested form. The logic of execution is shown below:

```
if(test condition-1)
  {
    if(condition-2)
      {
        Statement-1;
      }
    else
      {
        Statement-2;
      }
  }
else
{
    Statement-3;
}
Statement-x;
```

If the condition-1 is false, the statement-3 will be executed; otherwise it continues to perform the second test. If the condition-2 is true, the statement-1 will be evaluated; otherwise the statement-2 will be evaluated and then the control is transferred to the statement-x.

The flowchart for **nested if.....else** statement is given below:



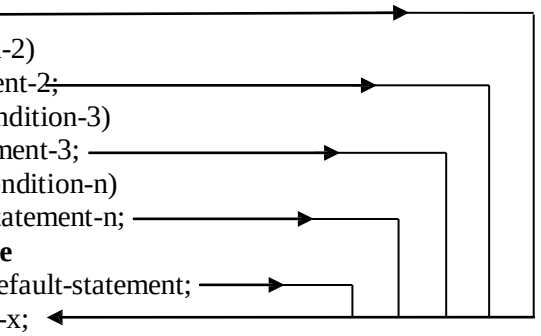
Example program:

```
#include<stdio.h>
main()
{
  int a,b,c;
  clrscr();
  printf("Enter a,b,c values\n");
  scanf("%d %d %d",&a,&b,&c);
  if(a>b)
   {
      if(a>c)
         printf("A is big\n");
      else
         printf("C is big\n");
   }
  else
   {
      if(b>c)
          printf("B is big\n");
      else
          printf("C is big\n");
   }
  getch();
}
```

5.8 The else if ladder statement:

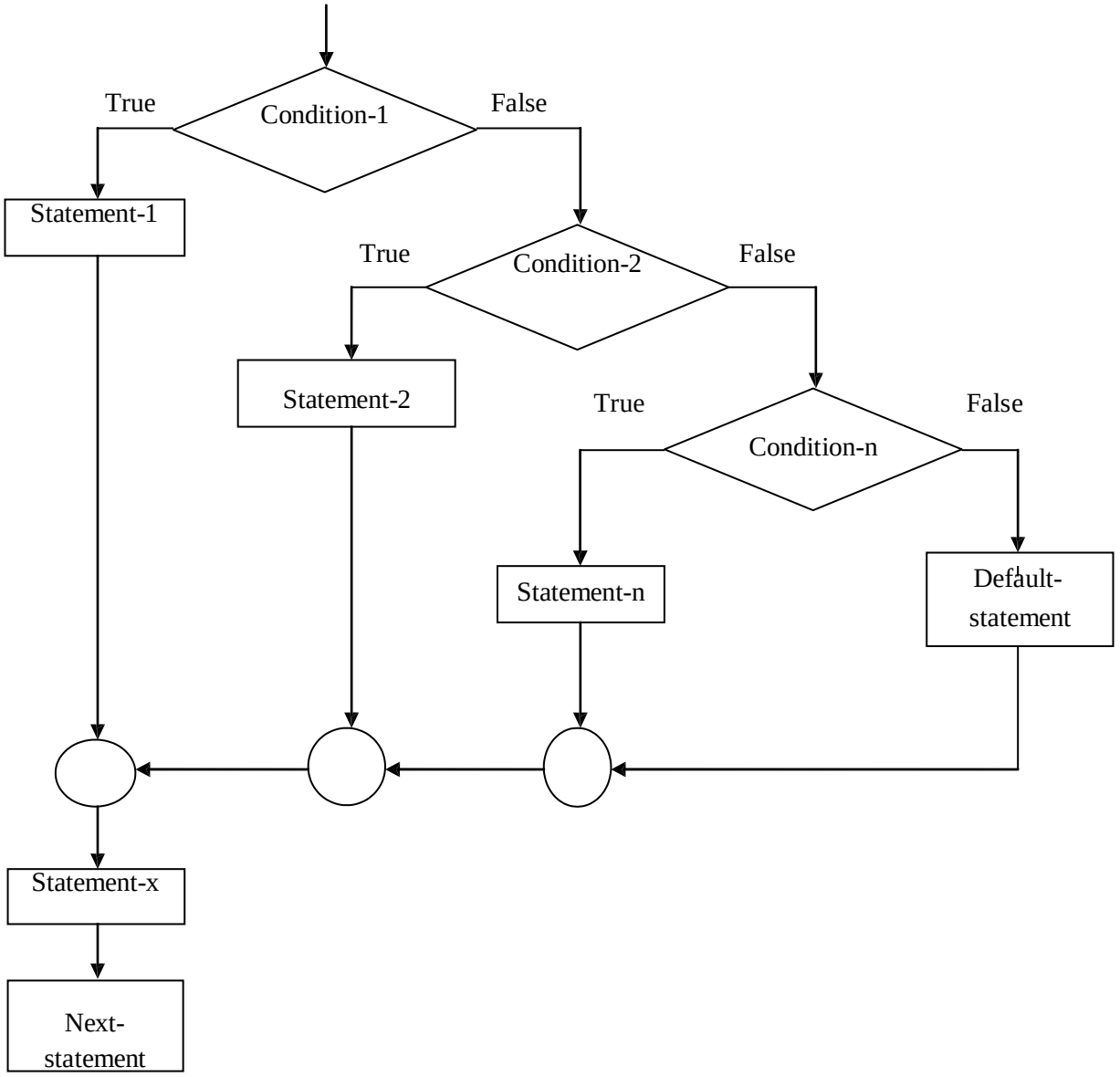
There is another of putting **if**s together when multipath decisions are involved. A multipath decision is a chain of **if**s in which the statement associated with each **else** is an **if.** It takes the following general form:

```
if(condition-1)
      statement-1;
  else if(condition-2)
          statement-2;
       else if(condition-3)
            statement-3;
        else if(condition-n)
                statement-n;
             else
                default-statement;
       statement-x;
```



This construct is known as the **else if** ladder. The conditions are evaluated from top downwards. As soon as a true condition is found, the statement associated with it is executed and the control is transferred to the statement-x. When all the n conditions become false, then the final **else** containing the *default-statement* will be executed.

The flowchart of **else if** ladder is shown below:



Example program:

```
#include<stdio.h>
main()
{
  int marks;
  clrscr();
  printf("Enter marks\n");
  scanf("%d",&marks);
  if((marks>=80)&&(marks<=100))
     printf("Honours\n");
  else if((marks>=60)&&(marks<=79))
     printf("First Division\n");
  else if((marks>=50)&&(marks<=59))
     printf("Second Division\n");
  else if((marks>=40)&&(marks<=49))
     printf("Third Division\n");
  else if((marks>=0)&&(marks<=39))
     printf("Fail\n");
  else
     printf("Marks given above 100 or below \n");
```

```
    getch();
}
```

5.9 Importance of Indentation:

The importance of indentation is given below:

- Indent statements that are dependent on the previous statements; provide at least three spaces of indentation.
- Align vertically else clause with their matching if clause.
- Use braces on separate lines to identify a block of statements.
- Indent the statements in the block by at least three spaces to the right of the braces.
- Align the opening and closing braces.
- Use appropriate comments to signify the beginning and end of blocks.
- Indent the nested statements as per the above rules.
- Code only one clause or statement n each line.

5.10 Switch statement:

C has a built-in multiway decision statement known as a **switch.** The **switch** statement tests the value of a given variable against a list of **case** values and when a match is found, a block of statements associated with that **case** is executed.

The general form of the **switch** statement is shown below:

```
switch(expression)
{
    case value-1:
                block-1;
                break;
    case value-2:
                block-2;
                break;
    .....
    .....
    .....
    default:
                default-block;
                break;
}
Statement-x;
```

The expression is an integer expression or characters. Value-1, value-2 ... are constants or constant expressions and are known as case labels. Each of these values should be unique within a **switch** statement. Block-1, block-2 ... are statement lists and may contain zero or more statements. There is no need to braces around these blocks.

NOTE: **case** labels must end with colon (:).

When the **switch** is executed, the value of the expression is successfully compared against the values value-1, value-2..... If a case is found whose value matches with the value of the expression, then the block of statements that follow the case are executed.

The **break** statement at the end of each block signals the end of a particular case and causes an exit from the **switch** statement, transferring the control to the **statement-x** following the **switch.**

The **default** is an optional case. When present it will be executed if the value of the expression does not match with any of the case values. If not present, no action takes place if all matches fail and the control goes to the **statement-x.**

The flowchart of **switch** statement is shown below:

Rules for Switch statement:

- The **switch** expression must be an integer type.
- Case labels must be constants or constant expressions.
- Case labels must be unique. No two labels can have the same value.
- Case labels must end with colon.

- The **break** statement transfers the control out of the **switch** statement.
- The **break** statement is optional.
- The default label is optional.
- There can be at most one **default** label.
- The **default** may be placed anywhere but usually placed at the end.
- It is permitted to nest **switch** statements.

Example program:

```
#include<stdio.h>
main()
{
  int a,b,c,x;
  clrscr();
  printf("Enter a,b values\n");
  scanf("%d%d",&a,&b);
  printf("1.Addition\n2.Subtraction\n3.Multiplication\n4.Division\n5.Modulo Division\n");
  printf("Enter ur choice\n");
  scanf("%d",&x);
  switch(x)
   {
       case 1:c=a+b;
              printf("C value is %d\n",c);
              break;
       case 2:c=a-b;
             printf("C value is %d\n",c);
             break;
       case 3:c=a*b;
              printf("C value is %d\n",c);
              break;
     case 4:c=a/b;
              printf("C value is %d\n",c);
              break;
      case 5:c=a%b;
            printf("C value is %d\n",c);
            break;
    default:printf("Invalid choice\n");
   }
 getch();
}
```

5.11 Conditional Operator:

The C language has an unusual operator, useful for making two-way decisions. This operator is a combination of ? and :, and takes three operands. This operator is popularly known as *conditional operator*. The general form of use of the conditional operator is as follows:

Conditional operator? expression1: expression2

The *conditional expression* is evaluated first. If the result is non-zero, *expression-1* is evaluated and is returned as the value of the conditional expression. Otherwise, *expression-2* is evaluated and its value is returned.

For example,

```
if(x<0)
    flag=0;
else
     flag=1;
```

can be written as

```
flag=(x<0)?0:1;
```

5.5 Evaluation of Logical Expression:

Logical Expression: An expression which combines two or more relational expressions using logical operators is called logical expression or compound relational expressions.

e.g.: a > b && x==10

A logical expression also yields a value of one or zero, according to the truth table shown in fig:

Op1	Op2	Op1 && op2	Op1 || op2	!op1
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

The relative precedence of relational and logical operators is as follows:

Precedence:

FIRST PRECEDENCE: !
SECOND PRECEDENCE: &&
THIRD PRECEDENCE: ||

Associativity:

Left to Right

The relative precedence of relational and logical operators is as follows:

Highest !(Right to left)
>, >= , < ,< =
= =, !=
&&
Lowest ||

Associativity:
Left to Right

Example 1: x=((-10&&10)||(-20&&20))
x=1||1
x=0

Example2: a=20,b=30
c = (!(a>b)&&(a||(!b==a)))
c = (!(20>30)&&(20||(!30==20)))
c = (1&&(20||0))
c = (1&&1)
c = 1

Example program:

```
#include<stdio.h>
main()
{
    int a=20,b=30,c;
    clrscr();
  c = (!(a>b)&&(a||(!b==a)));
  printf("c =%d\n",c);
  getch();
}
```