CONTROL FLOW

Control flow refers to the order in which the individual statements, instructions or function calls of an imperative or functional program are executed or evaluated.

Types of control instructions

1. Sequence control
   Instructions are executed in the same order in which they appear in the program
2. Selection/ Decision control
   It allows the computer to take a decision as, to which instruction is to be executed next.
3. Repetition/ Loop control
   It helps the computer to execute a group of instructions repeatedly till, a condition satisfied.
4. Case control

Decision control statements

a) The if Statement

An if statement is a selection statement that allows more than one possible flow of control. It is implemented in two forms;

(1) **Simple if statement**

Syntax->

```
        if (expression)
          {
           statements;
          }
```

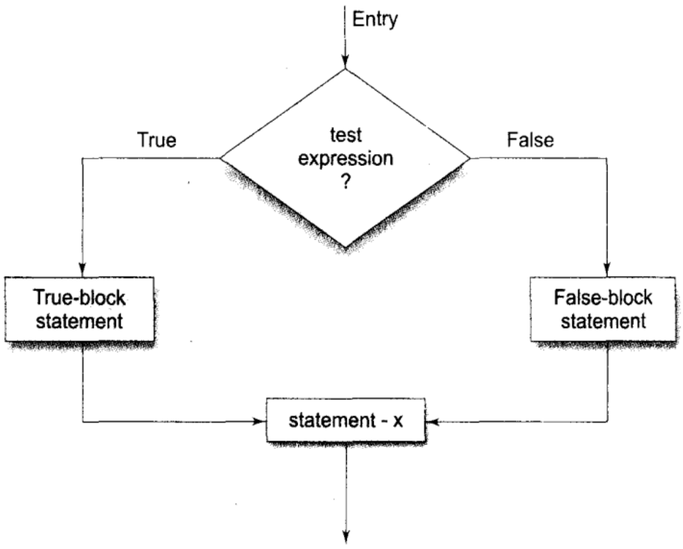
(2) i**f...else statement**

syntax ->

```
          if (expression)
```

```
    {
         statements;
    }
else
    {
          statements;
    }
```

```
Eg. -  if (a>b)
          {   printf("biggest value is a");      }
      else
          {   printf("biggest value is b");      }
```



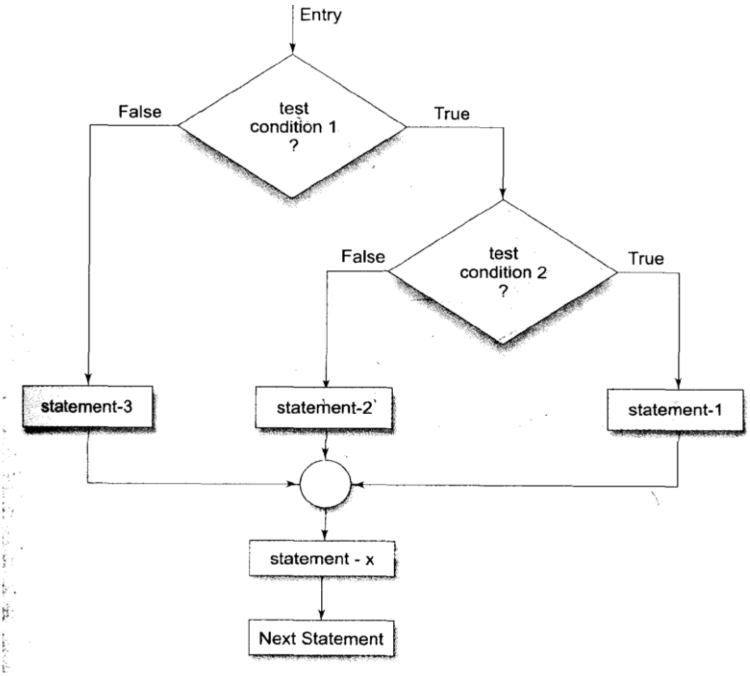
Nesting of if ... else Statements

If within if is called nested if and it is used when we have multiple conditions to check and when any if condition contains another if statement then that is called ‘nested if’.

If the external condition is true, then the internal if condition is executed and if the condition is false then the else portion of external if is executed.

Syntax->

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

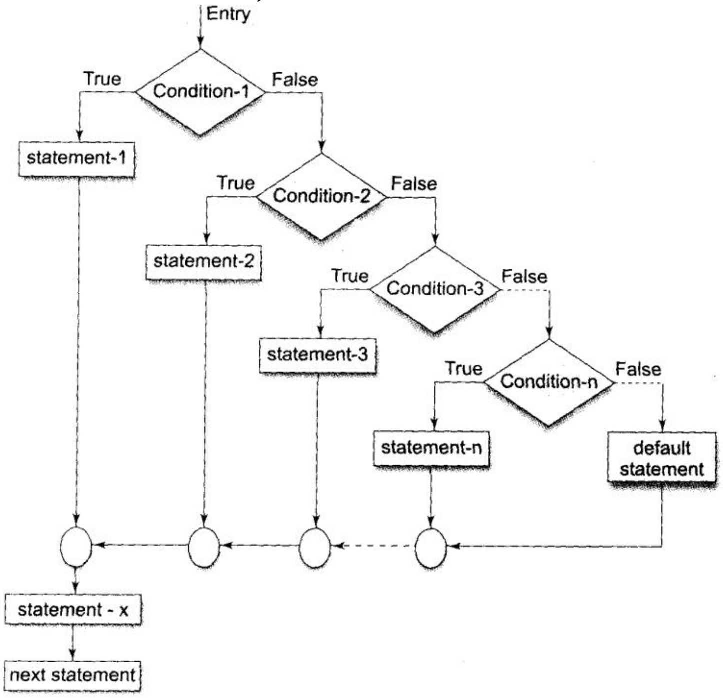


The else if Ladder

When a series of many conditions have to be checked we may use the ladder else if statement which takes the following general form.

syntax ->

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



This construct is known as if else construct or ladder. The conditions are evaluated from the top of the ladder to downwards. As soon as the true condition is found, the statement associated with it is executed and the control is transferred to the statement – x (skipping the rest of the ladder. When all the condition becomes false, the final else containing the default statement will be executed.

Eg.

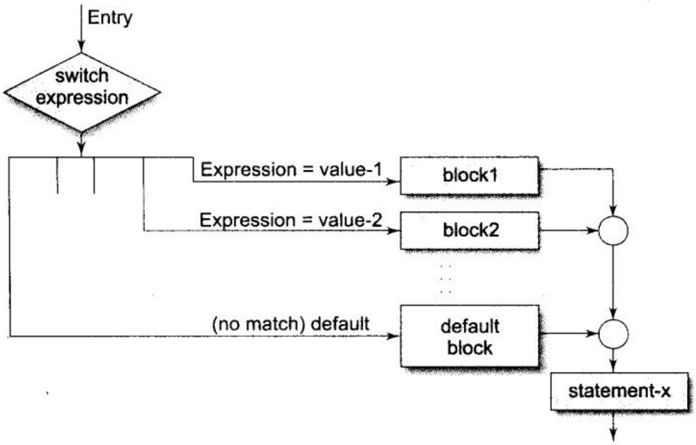
```
          main()
         {
           int magic=123;
           int guess;
           scanf(“%d”,&guess);
           if (guess==magic)
           {   printf("** right **");
               printf("%d is the magic number",magic);
           }
           else if (guess>magic)
               printf(.. Wrong .. too High");
           else
               printf(".. Wrong .. Too Low");
         }
```

b) The switch statement

The switch statement is a construct that is used when many conditions are being tested for. When there are many conditions, it becomes too difficult and complicated to use the if and else if constructs. Nested if/else statements arise when there are multiple alternative paths of execution based on some condition that is being tested for.

This is implemented as follows;

```
scanf("%d",&variable_name);
Switch (variable_name)
{
   case  1:
        statements;
        break;
  case  2:
        statements;
        break;
  default:
        statements;
 }
```



Rules for switch statement

- The switch expression must be an integral 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 semicolon.
- The break statement transfers the control out of the switch statement.
- The break statement is optional. That is, two or more case labels may belong to the same statements.
- The default label is optional. If present, it will be executed when the expression does not find a matching case label.
- 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.

Loops

A loop is a programming structure that repeats a sequence of instructions until a specific condition is met.

A nested loop is a logical structure, which is characterized by two or more repeating statements that are placed in a "nested" form (loop inside another loop).

Infinite loops : -Loops that never ends (condition never becomes false.)

Entry controlled loops

In this type test expression is checked first. The body of the loop is executed only if the test expression evaluates to true. Two entry controlled loops are as follows.

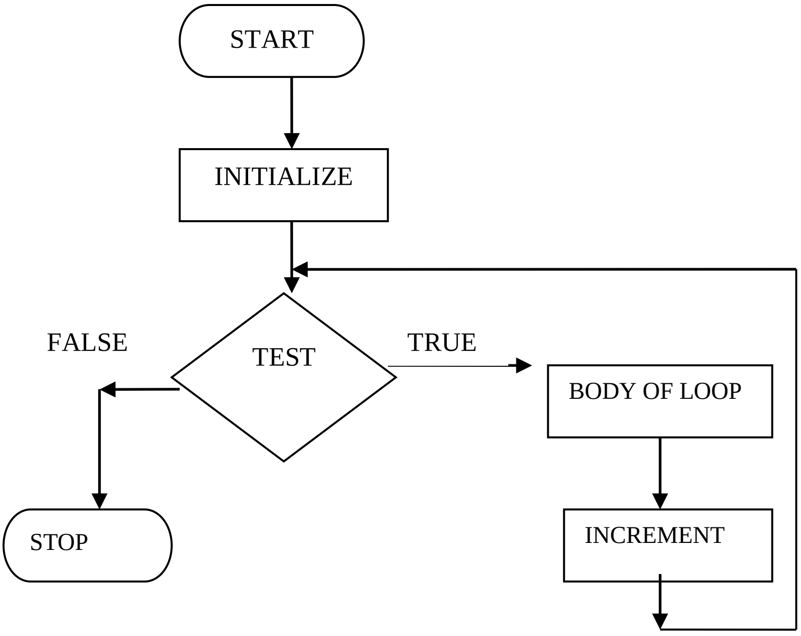
- **a) while loop**
- **b) for loop**

a) while loop

The while loop is used to execute a block of code as long as the test expression is true. This is an entry controlled loop.

Syntax->

```
  while (tested condition is satisfied)
 {
  block of code
 }
```



b) for loop

The for loop can execute a block of code for a fixed or given number of times. This is an entry controlled loop.

Syntax->.

```
for (initialization;test expression;increment/decrement)
{
   block of code
}
```

The initialization is usually an assignment statement that is used to set the loop-control variable. Test expression is a relational expression that determines when the loop will exit. The increment/decrement defines how the loop-control variable will change each time the loop is repeated. These three major sections must be separated by semicolons. The for loop will continue to execute as long as the test expression is true.

Exit controlled loops

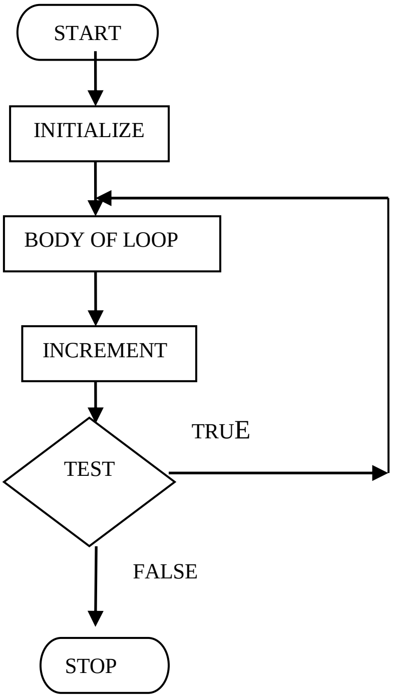
In this type, test expression is checked last. The body of the loop is executed at least once even the test expression is false.

do ....while loop

The do loop also executes a block of code as long as a condition is satisfied. The difference between a "do ...while" loop and a "while" loop is that the while loop tests its condition before execution of the contents of the loop begins; the "do" loop tests its condition after it's been executed at least once. It is an exit controlled loop.

Syntax->

```
do
{
    block of code
} while (condition is satisfied);
```



Example programs

1. /* 1 + (1/2) + (1/3) + ... (1/n) */

```
main()
{
 int nterms;
 float count=1,res=0;
 printf("Enter no of terms");
 scanf("%d",&nterms);
 for(count=1;count<=nterms;count++)
  res=res+(1/count);
 printf("result=%f",res);
 getch();
 }
```

2. /* 1 * (1+3) * (1+3+5) * .... upto n */

```
main()
{
 int nterms,row=1,col=1,sum=0,prod=1,digit;
 printf("Enter no of terms");
 scanf("%d",&nterms);
 for(row=1;row<=nterms;row++)
 {
  digit=1;
  sum=0;
  for(col=1;col<=row;col++)
 {  sum= sum+digit;
    digit=digit+2;
 }
 prod=prod*sum;
 }
 printf("result=%d",prod);
 getch();
 }
```

3. /* print all factors of a given number*/

```
main()
{
 int num,count;
 clrscr();
 printf("Enter a number");
 scanf("%d",&num);
 printf("\n Factors are \n");

 for(count=1;count<=num;count++)
 {
  if (num%count==0)
    printf("\n %d",count);
 }
 getch();
}
```

4. Summation of a set of numbers.

```
main()
{
      int I,s=0,n;
      for (I=1;I<=10;I++)
      {   scanf("%d",&n);
             s=s+n;
       }
       printf("sum = %d",s);
       getch();
 }
```

5. Generate the following series(fibonacci series-0,1,1,2,3,5,8...20 terms)

```
main()
{   int i,j=0,k=1,l=0,m=0;
     printf("%d,%d,",j,k);
      for (i=1;i<=20;i++)
      {   l=j+k;
```

```
        m=l+k;
        if (i<19)
            printf("%d,%d,",l,m);
         else
            printf("%d,%d",l,m);
         j=l;
         k=m;
         ++i;
    } getch();
}
```

6. Print all prime numbers below 100.

```
main()
{
    int n=1,r=0,x,p=0;
    while (n<=100)
    {   x=2;
        while (x<=(n/2))
        {   k=n%x;
            if (k==0)
                p=1;
            x=x+1;
        }
        if (p==0)
            printf("%d",n);
        n++;
     }
 }
```

7. Find out smallest among 10 accepted values.

```
main()
{
    int a=0,n,f;
    scanf("%d",&f);
    for (a=1;a<=10;a++)
    {   scanf("%d",&n);
        if (f>n)
            f=n;
        }
        printf("\nsmallest no %d",f);
    }
```

Addittional features of for loop

More than one variable can be intialised at a time in the for statement

```
p=1;
for(n=0;n<17;++n)
```

can be rewritten as

```
for(p=1,n=0;n<17;++n)
```

Like the intialisation section we may also have more than one increment or decrement section.

```
for(n=1,m=50;n<=m;n=n+1,m=m-1)
```

The test condition may have compound relation and testing need not be limited only to the loop control variable.

```
s=0;
for(i=1;i<20 && s<100;++i)
{
s=s+i;
printf("%d %d\n",i,sum);
}
```

It is also possible to use expression I the assignment statement of intialisation and increment sections.

```
for(x=(m+n)/2;x>0;x=x/2)
```

Another unique aspect of for loop is that one or more sections can be omitted.

```
m=5;
for(;mm!=100;)
{
printf("%d",m);
m=m+5;
}
```

Nesting of For loop

One for statement within another for statement.

```
main()
{
  long sum = 0L;
  int i = 1;     /* Outer loop control variable     */
  int j = 1;     /* Inner loop control variable     */
  int count = 10; /* Number of sums to be calculated */

  for( i = 1 ; i <= count ; i++ )
  {
    sum = 0L;  /* Initialize sum for the inner loop */

    /* Calculate sum of integers from 1 to i */
    for(j = 1 ; j <= i ; j++ )
      sum += j;

    printf("\n%d\t%ld", i, sum); /* Output sum of 1 to i */
  }

}
```

Output

```
     1        1
     2        3
     3        6
     4        10
     5        15
     6        21
     7        28
     8        36
     9        45
     10       55
```