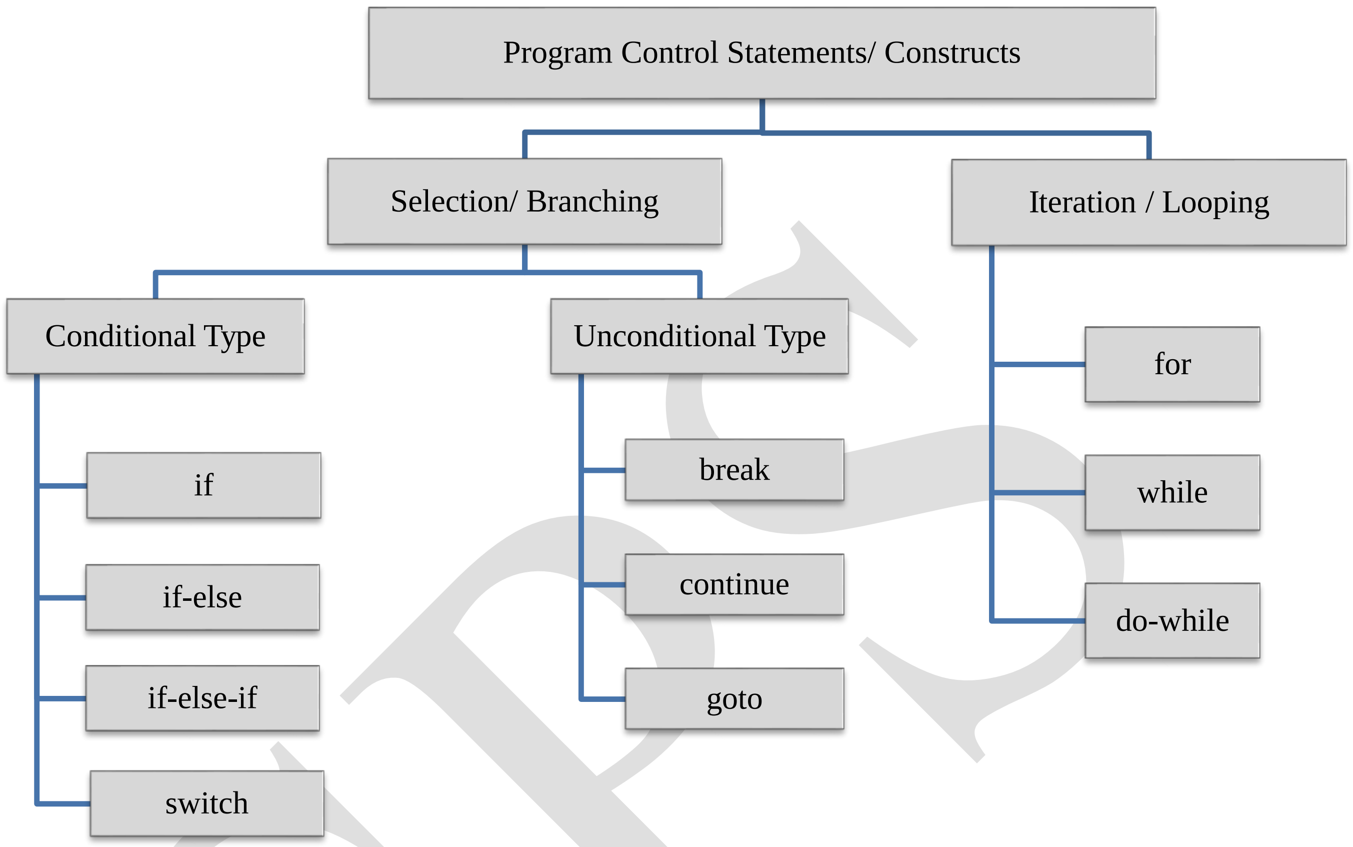
Control Statement



Selection / Branching Statements

A] Conditional Type

There are three types of decision statement

1. One-way (if Condition)
2. Two-way (if-else Condition)
3. Multi-way (if-else-if / switch-case Condition)

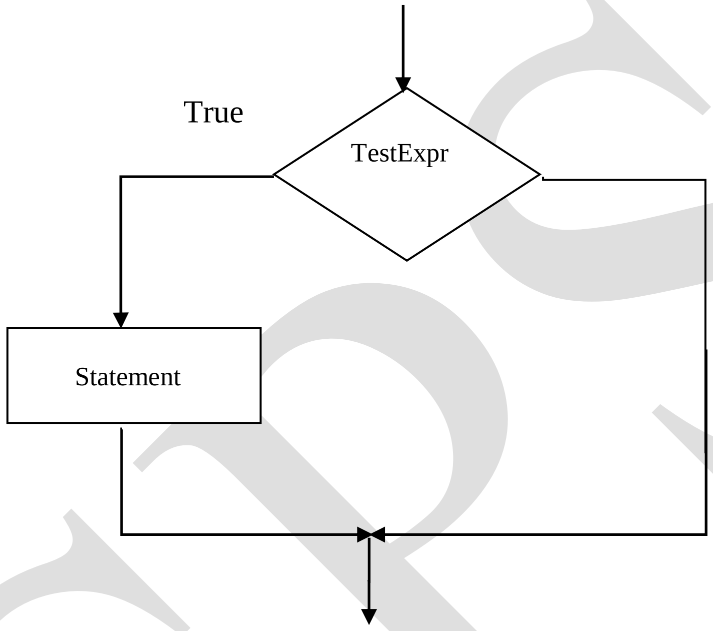
1. **One –way decision statement using if statement:**
   One-way decision are handled with an if statement that either do some particular things or do nothing. The decision based on “Test Expression” or “Condition” that evaluates true or false. If condition is true then its associate statement is executed & if false then next statement executed. Syntax of *if* statement:

```
        if(TestExpr)
                Statement;
```

If number of statements more than one then syntax is

```
        if(TestExpr)
        {
                Statement_1;
                Statement_2;
                .
                .
                Statement_N;
        }
```

Flow chart:



Example

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int a, b, c, max;
  clrscr();
  printf("\n Enter First Number");
  scanf("%d",&a);
  printf("\n Enter Second Number");
  scanf("%d",&b);
  printf("\n Enter Third Number");
  scanf("%d",&c);
  max=a;
  if(b>max)
    max=b;
  if(c>max)
    max=c;
  printf("Maximum Number out of %d, %d and %d is %d",a,b,c,max);
  getch();
}
```

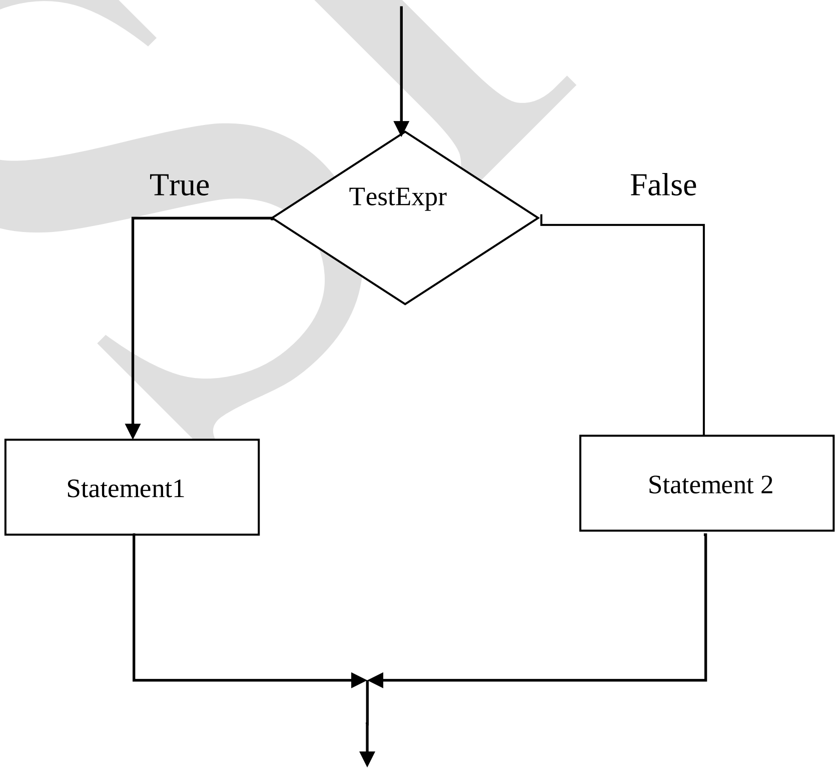
2. **Two –way decision statement using if-else statement:**
   Two-way decision handled with *if-else* statements. That either does one particular thing or does another. Syntax of *if-else* statement is

```
if(TestExpr)
        Statement_1;
else
        statement_2;
```

If number of statements more than one then syntax is

```
        if(TestExpr)
        {
                Statement_1;
                Statement_2;
                .
                .
                Statement_N;
        }
        else
        {
                Statement_1;
                Statement_2;
                .
                .
                Statement_N;
        }
```

Flow chart:



Example:

```
 if(a>b)
    printf("%d is greater",a);
  else
    printf("%d is greater",b);
```

3. **Multi-way decisions**
   Multi-way decision statements use ***if-else-if*** nested ***if*** or ***switch*** statements.

- **<u>if-else-if</u>**

```
if(TestExpr1)
     stmt1;
else if(TestExpr2)
          stmt2;
       else if(TestExpr3)
                stmt3;
                .
                .
                .
                else if(TestExprN)
                          stmtN;
                       else
                          stmt
```

- **<u>Nested if</u>**

```
if(TestExpr1)
{
     Statement
     if(TestExpr2)
         Statement 2;
     else
         Statement 3;
}
Else
{
     Statement4
     if(TestExpr3)
         Statement 5;
     else
         Statement 6;
}
```

Example

```
if(a>b)
{
        if(a>c)
                printf(" %d is greater",a);
        else
                printf("%d is greater", c);
}

else
{
        if(b>c)
                printf(" %d is greater",b);
        else
                printf("%d is greater", c);
}
```

<u>Dangling else problem</u>

The C compiler allows to use the 'if statement' in various combination. For nested 'if statements' a dangling 'else' might be encountered by the compiler.C compiler unable to locate a pair of 'if-else'.

```
if(TestExpr1)
        if(TestExpr2)
           Statement
Else
       Statement
```

Compiler unable to locate else statement is for first if or second if. Solution for this problem is:

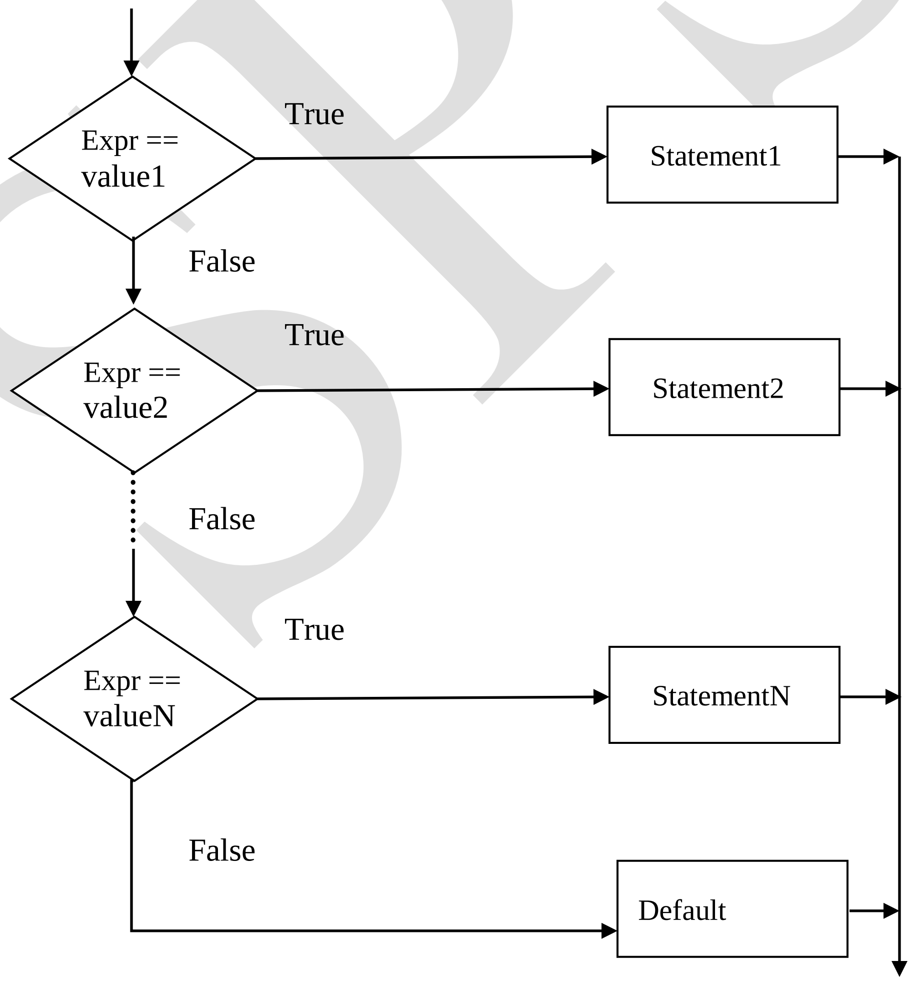
```
if(TestExpr1)
{
        if(TestExpr2)
           Statement
}
Else
       Statement
```

- **<u>Switch Statements</u>**
  Instead of use multiple else statement another in built decision structures called switch for multi-way selection. The syntax of switch-case is

```
switch(expr)
{
   case value1: Statement1;
                      break;
  case value2: Statement2;
                      break;
  case value3: Statement3;
                      break;



  case valueN: StatementN;
                      break;
  default:  Statement;
}
```

When there is switch statement is executed then it will match expr with value1, value2, ….., valueN and whenever it match with value it will execute its associate statement. If none is found then default statement is execute.

Flow chart:



Example:

```
int main()
{
   int n,number;
   printf("Enter any number");
   scanf("%d",&n);
   number=n%10;
   switch(number)
   {
        case 1:
        case 3:
        case 5:
        case 7:
        case 9: printf("%d is ODD Number",n);
                break;
        case 0:
        case 2:
        case 4:
        case 6:
        case 8: printf("%d is Even Number",n);
                break;
        default: printf("Please enter integer number");
  }
  return 0;
}
```

B] Unconditional Type

1. ***break*** statement: The break statement is use in loop statement to terminate execution of loop or switch statement. The syntax of break statement is

```
    break;
```

2. ***continue*** statement: The continue statement does not terminate the loop but goes to the test expression in *for*, *while* and *do-while* loop. The form of continue statement is

```
    continue;
```

3. ***return*** statement: The return type is used in function definition to set its return value and to terminate execution of function. There are two types return statement return type void & function with non-return value. The syntax is

```
    return;               //For void return value

    return expression;    //For non-void return value.
```

after break statement is executed within a loop or case in switch statement execution proceeds to the statement that follows the loop or switch statement.

4. **goto** statement: The goto statement is another type of control statement supported by C. The control is unconditionally transferred to the statement associated with label specified in the goto statement. The syntax of goto statement is

```
    goto label_name;
```

A statement label is defined in exactly the same way as a variable name. The statement must be followed by a colon(:); the goto statement ends with a semicolon. Syntax of goto is

```
label1:
Statement1;
Statement2;
.
.
StatementN;
goto label1;
```

OR

```
goto label1;
Statement1;
Statement2;
.
.
StatementN;
label1:
```

Iterative or Repetitive Execution:

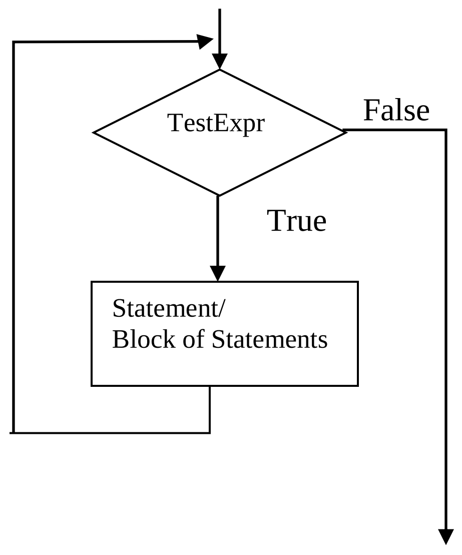
In every programming language(C programming language) there are circumstances were you want to execute a statement or block of statement repeatedly. There are mainly two types of iteration or loop.

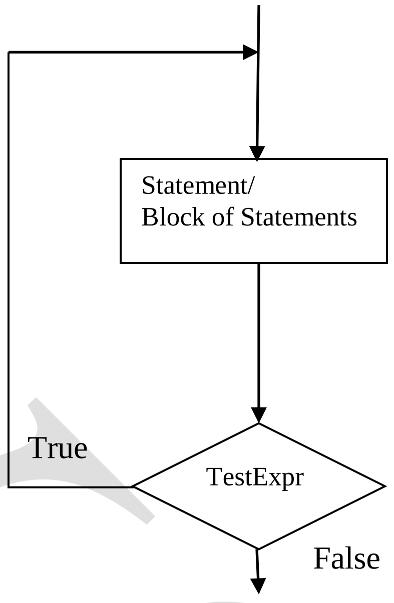
1. **Unbounded iteration/loop:** This iteration is use when we doesn't know how many times it will executed

A loop can either be a pre-test or post-test loop.

Pre-test loop: In a pre-test loop the condition is checked before the beginning of each iteration. If test expression is true then only it will execute its associate statements otherwise it will execute next statement. This process is repeated until test expression is false. The minimum execution for pre-test loop is zero.
Eg. While loop.

Post-test loop: In a post-test loop the condition is checked after execution of its associate statement. This process is repeated until test expression is false. If the test expression is false then also it will execute its associate statement at least ones. The minimum execution for post-test loop is one.
Eg. Do-While loop.



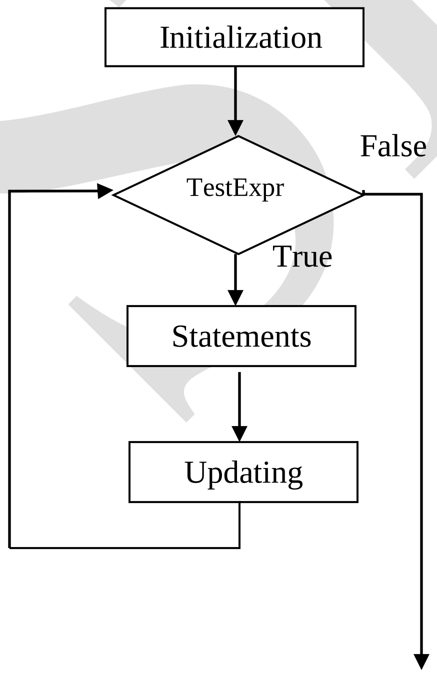


Pre-Test loop **Post-Test loop**

<u>While loop:</u>

While statement is pre-test loop. It uses a test expression to control the loop. Since it is a pre-test loop it evaluates the test-expression before every iteration of loop. Statements will be executed repeatedly until the value of TestExpr becomes false. The initialization of the loop control variable has to be done before start of loop and updating done within loop.

Flow chart



The syntax of while loop is

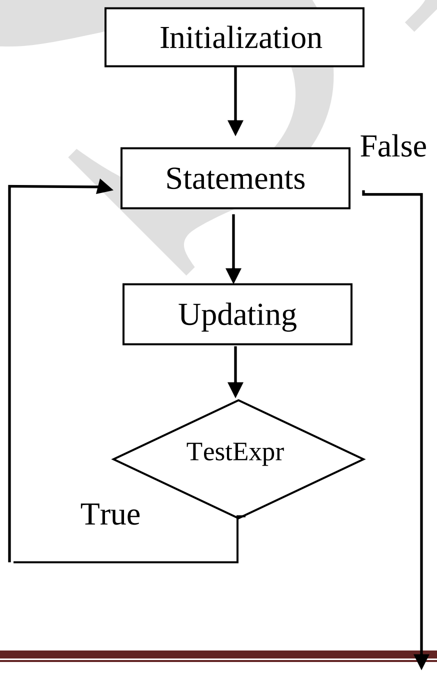
```
while(TestExpr)
{
        Statement1;
        Statement2;

        .
        .
        StatementN;
}
```

Do-While loop:

Do-While statement is post-test loop. It uses a test expression to control the loop. Since it is a post-test loop it evaluates the test-expression after every iteration of loop. Statements will be executed repeatedly until the value of TestExpr becomes false. The initialization of the loop control variable has to be done before start of loop and updating done within loop. The syntax of while loop is:

```
do
{
        Statement1;
        Statement2;
        .
        .
        StatementN;
} while(TestExpr);
```

Flow chart



2. **<u>Bounded iteration/loop</u>:** This iteration is use when we know how many times it will executed.

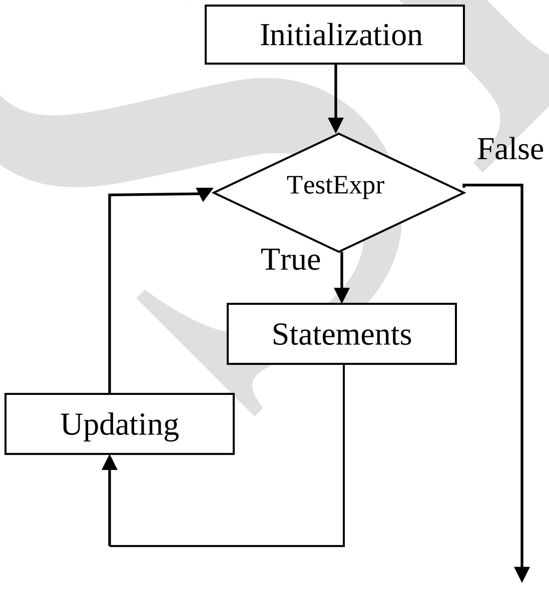
For loop:

A loop formed by using the *for* statement is generally called a definite loop because the programmer knows exactly how many times it will repeat. The syntax of for loop is

```
        For(initialization; TestExpr; updating)
        {
                Statement1;
                Statement2;
                .
                .
                StatementN;
        }
```

- *<u>Initialization</u>*: This part of the loop is first to be executed. The statement of this block executed only once.
- *TestExpr:* TestExpr represents a test expression that must be true for the loop to continue execution.
- *<u>Updating:</u>* The statement contained are executed every time through the loop before loop condition is tested.
- *<u>Statements:</u>* Statements may be single or block of statements.

<u>Flow chart:</u>



Example

for (i=0; i<10; i++)
{

```
    printf("%d",i);
  }
```

Nested Loop

A nested loop refers to as a loop that contain within another loop. In nested loop the inside loop executes completely before the outside loop's next iteration. The inner loop should be enclosed completely in the outer loop. Overlapping loops are not allowed.

```
for(i=0;i<n;i++)
{
  do
   {


}
    }while(j>0);
```

This is not allowed

```
for(i=0;i<n;i++)
{
  do
   {


    }while(j>0);
}
```

This is allowed

Examples:

1]

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int i,j,k,n;
        clrscr();
        printf("Enter any Number ");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
           for(k=0;k<i;k++)
             printf("%c ",65+k);
           printf("\n");
        }
        getch();
        return 0;
}
```

OUTPUT:

```
Enter any Number 5
A
A B
A B C
A B C D
A B C D E
```

2]

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int i,j,k,n;
        clrscr();
        printf("Enter any Number ");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
           for(j=1;j<=n-i;j++)
              printf("   ");
           for(k=i;k>0;k--)
             printf("%2d ",k);
           printf("\n");
        }
        getch();
        return 0;
}
```

OUTPUT:

```
Enter any Number 5
            1
         2  1
      3  2  1
   4  3  2  1
 5  4  3  2  1
```

3]

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int i,j,k,n;
        clrscr();
        printf("Enter any Number");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
           for(j=1;j<=n-i;j++)
             printf("  ");
           for(k=1;k<=i;k++)
             printf("*   ");
           printf("\n");
        }
        for(i=n-1;i>0;i--)
        {
```

```
        for(j=1;j<=n-i;j++)
          printf("  ");
        for(k=1;k<=i;k++)
          printf("*   ");
        printf("\n");
      }

      getch();
      return 0;
}
```

OUTPUT:

```
Enter any Number 5
        *
      *   *
    *   *   *
  *   *   *   *
*   *   *   *   *
  *   *   *   *
    *   *   *
      *   *
        *
```