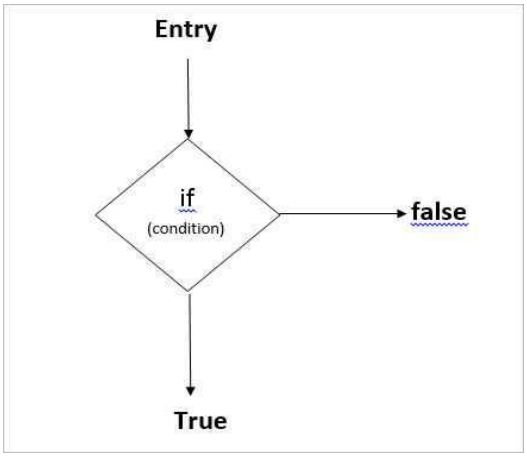
Module-2: Branching and Looping:

Branching: The C language programs follow a sequential form of execution of statements. Many times it is required to alter the flow of sequence of instructions/statements. C language provides statements that can alter the flow of a sequence of instructions. These statements are called as control statements/branching statements. To jump from one part of the program to another, these statements help. The control transfer may be *unconditional* or *conditional*.

Conditional Branching statements: The conditional branching statements help to jump from one part of the program to another depending on whether a particular condition is satisfied or not. Generally they are two types of branching statements.

Two Way Selection:

Depending on the condition result either of one root will be fallowed. If condition results to false then false root will be fallowed, if condition results to true than true root will be fallowed.



Conditional Branching Statements in C:

1. **Simple if statement**
2. **if… else statement**
3. **Nested if…else statement**
4. **else…if ladder**

Simple if statement: It allows the computer to evaluate the expression/condition first and them depending on whether the value of the expression/condition is "true" or "false", it transfer the control to a particular statements. This point of program has two paths to flow, one for the true and the other for the false condition. If condition becomes true than it executes statements written in true block, if condition fails than true block will be skipped.

Syntax:

```
if(test-expression/condition)
{
    True statement-block ;
}
statement-x;
```

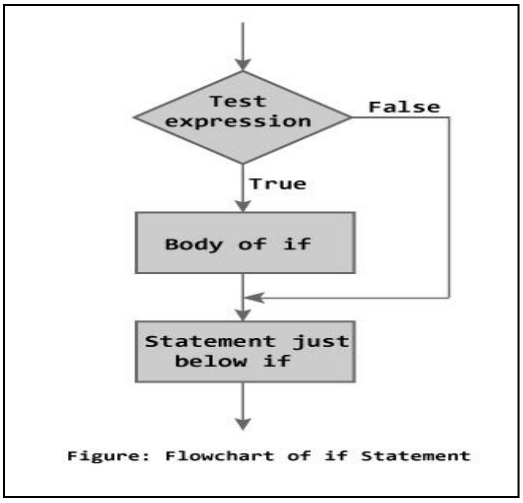


Figure: Flowchart of if Statement

Example: C program calculate the absolute value of an integer using if statement.

```
# include<stdio.h >
void main( )
{
     int numbers;
     clrscr();
     printf (“Type a number:”);
     scanf (“%d”, & number);
     if (number < 0)
        {
         number = – number;
        }
     printf (“The absolute value is % d \n”, number);
     getch();
}
```

Example: C Program to check equivalence of two numbers using if statement

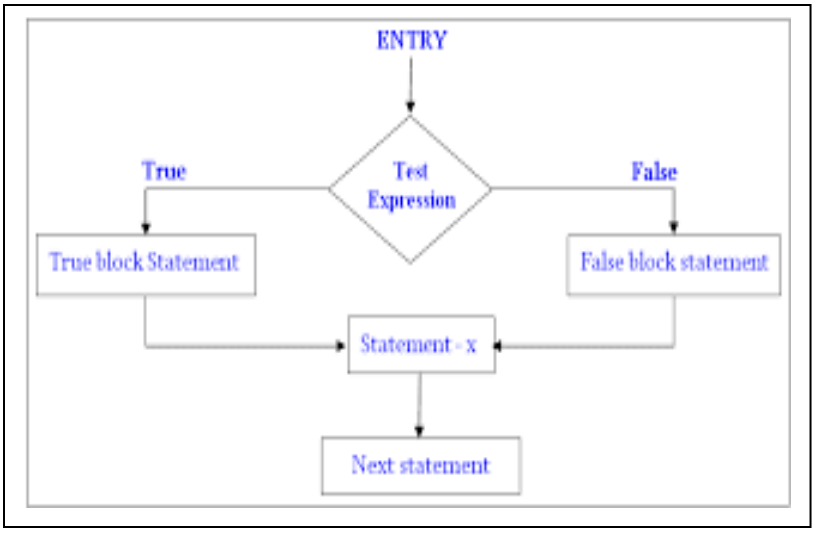
```
#include<conio.h>
#include<stdio.h>
void main()
{
    int m,n;
    clrscr();
    printf(" \n enter two numbers:");
    scanf(" %d %d", &m, &n);
    if(m-n= = 0)
       printf(" \n two numbers are equal");
    getch();
}
```

Note: if statement will execute one statement bellow to it by default, if we want to execute more than one statement, then those all statements we have to group in open and close Curly brackets { and }.

The if-else statement: The if-else statement is an extension of the simple if statement. If the test-expression/condition is true, then true-block statements immediately following if statement are executed otherwise the false-block statements are executed. In other case, either true-block or false-block will be executed, not both.

Syntax:

```
 if(test-expression/condition)
 {
        true-block statements;
 }
else
 {
        false-block statements;
 }
statement-x
```



Example: C program to read any number as input through the keyboard and find out whether it is Odd Number or Even Number.

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int n;
   clrscr();
   printf("Enter the Number");
   scanf("%d",&n);
   if(n%2==0)
   {
      printf("This is Even Number");
   }
   else
   {
      printf("This is Odd Number");
   }
   getch();
}
```

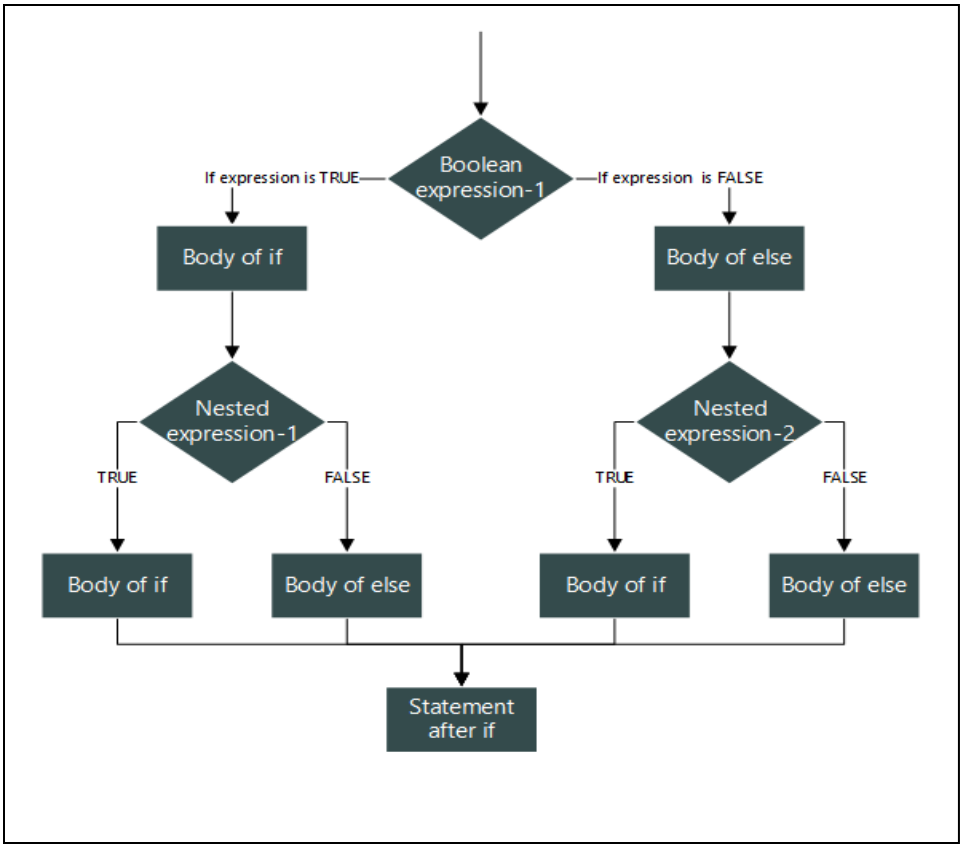
Example: C program to find biggest among two numbers using if else.

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a,b;
   clrscr();
   printf("Enter the two Number");
   scanf("%d%d",&a,&b);
   if(a>b)
   {
      printf("The number a=%d is bigger", a);
   }
   else
   {
      printf("The number b=%d is bigger",b);
   }
   getch();
}
```

Nested if….else statement: C language supports if-else statements to test additional conditions apart from the initial test expression. The if-else construct works in the same way as normal if statement nested if construct is also know as if-else-if construct. When an if statement occurs within another if statement, then such type of is called nested if statement.

Syntax:

```
if(test-condition-1)
   {
     if(test-condition-2)
       {
         statement-1;
       }
      else
       {
         statement-2;
       }
     }
 else
   {
     if(test-condition-3)
       {
        statement-3;
       }
      else
       {
        statement-4
       }
     }
statement-x
```



Example: C program if the ages of Ram, sham and Ajay are input through the keyboard, write a program to determine the youngest of the three

```
#include< stdio.h >
#include< conio.h >
void main()
{
    int ram,sham,ajay;
    clrscr();
    printf("Enter the Three Ages of Ram,Sham and Ajay\n");
    scanf("%d%d%d",&ram,&sham,&ajay);
    if(ram < sham)
      {
       if(ram < ajay)
         {
           printf("Ram is Youngest");
         }
       else
        {
          printf("Ajay is Youngest");
        }
      }
    else
       {
        if(sham < ajay)
          {
            printf("Sham is Youngest");
          }
        else
          {
            printf("Ajay is Youngest");
          }
       }
     getch();
}
```

Example: C program to check whether person is eligible for work or not.

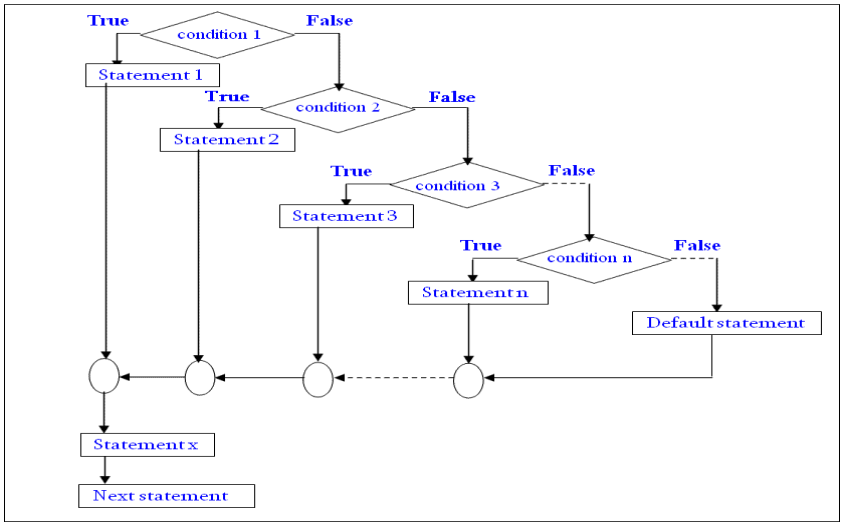
```
#include <stdio.h>
 void main()
 {
        int age;
        printf("Please Enter Your Age Here:\n");
        scanf("%d",&age);
        if ( age < 18 )
        {
              printf("You are Minor.\n");
             printf("Not Eligible to Work");
        }
        else
        {
              if (age >= 18 && age <= 60 )
               {
                    printf("You are Eligible to Work \n");
                    printf("Please fill in your details and apply\n");
               }
              else
               {
                   printf("You are too old to work as per the Government rules\n");
                   printf("Please Collect your pension! \n");
              }
        }
 getch();
}
```

else… if Ladder: There is another way 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 and last else if's else part contain only else.

The conditions are evaluated from the top to bottom. As soon as true condition is found, the statement associated with it is executed and the control is transferred to statement-x(skipping the rest of ladder) when all the conditions become false, then the final else containing the default statement will be executed.

Syntax:

```
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;
```



Example: Program to find maximum of three numbers using else-if ladder.

```
#include<conio.h>
#include<stdio.h>
viod main()
{
   int a,b,c,max;
   clrscr();
   printf("Enter values of a ,b,c");
   scanf("%d%d%d",&a,&b,&c);
   if(a>b && a>c)
      max=a;
   else if(b>c)
      max=b;
   else
     max=c;
  printf("Maximum is %d",max);
  getch();
}
```

Example: C Program to print grade of a student using If Else Ladder Statement.

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int marks;
   printf("Enter your marks between 0-100\n");
   scanf("%d", &marks);
   if(marks >= 90)
    {
       printf("YOUR GRADE : A\n");
    }
  else if (marks >= 70 && marks < 90)
   {
     printf("YOUR GRADE : B\n");
   }
 else if (marks >= 50 && marks < 70)
  {
    printf("YOUR GRADE : C\n");
   }
 else
  {
    printf("YOUR GRADE : Failed\n");
   }
getch();
}
```

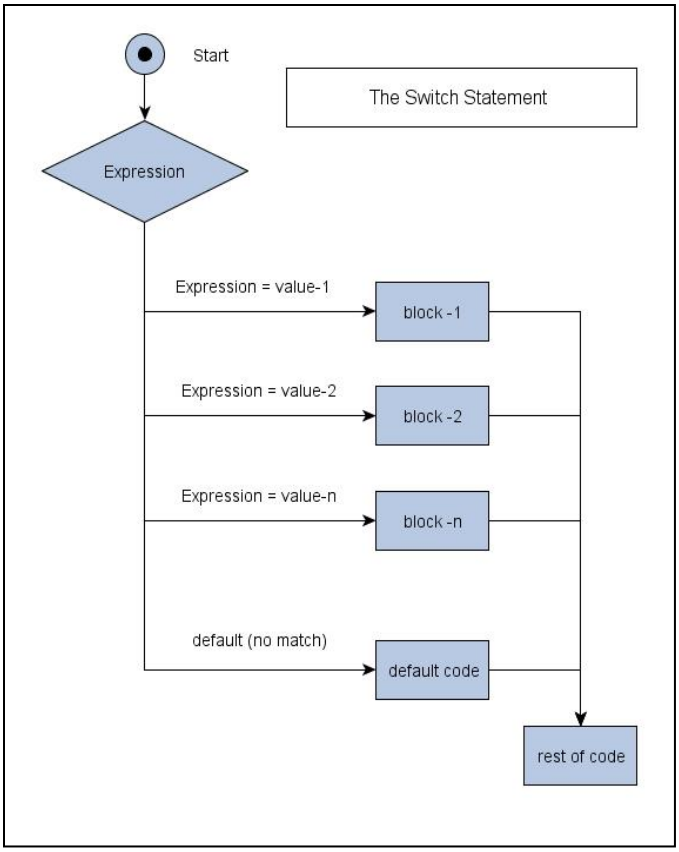
Switch statement: A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case**.

The following rules apply to a **switch** statement −

1. The **expression** used in a **switch** statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
2. You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon (:).
3. The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
4. When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.
5. When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
6. Not every case needs to contain a **break**. If no **break** appears, the flow of control will *follow through* to subsequent cases until a break is reached.
7. A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No **break** is needed in the default case.

Syntax:

```
switch(expression)
{
  case constant-expression-1  :  statement(s);
                                 break;
  case constant-expression-2  : statement(s);
                                 break;
              .
              .
              .
 case constant-expression-n  : statement(s);
                      default :   statement(s);
}
```



Example: C Program to create a simple calculator performs addition, subtraction, multiplication or division depending the input from user.

```
#include<conio.h>
# include <stdio.h>
void main()
 {
   char operator;
   int firstOperand, secondOperand;
   printf("Enter an operator (+, -, *, /): ");
   scanf("%c", &operator);
   printf("Enter two operands: ");
   scanf("%d%d",&firstOperand, &secondOperand);
   switch(operator)
   {
      case '+':  printf("%d+ %d= %d",firstOperand, secondOperand, firstOperand+secondOperand);
                 break;
      case '-': printf("%d - %d = %d", firstOperand, secondOperand, firstOperand-secondOperand);
                 break;
      case '*': printf("%d* %d = %d", firstOperand, secondOperand, firstOperand*secondOperand);
                 break;
      case '/': if(secondOperand==0)
                   printf("Divide by Zero Error");
                else
                   printf("%d / %d = %d", firstOperand, secondOperand, firstOperand/secondOperand);
                 break;

      default: printf("Error! operator is not correct");
   }
getch();
}
```

Example: C program to Check Character is Vowel or not using Switch Case if it is vowel which vowel.

```
#include <stdio.h>
void main()
{
        char ch;
        printf(" Enter any character: ");
        scanf("%c", &ch);
        switch (ch)
        {
                case 'a':
                case 'A': printf(" %c is a vowel", ch);
                          break;
                case 'e':
                case 'E': printf(" %c is a vowel", ch);
                          break;
                case 'i':
                case 'I': printf(" %c is a vowel", ch);
                          break;
                case 'o':
                case 'O': printf(" %c is a vowel", ch);
                          break;
                case 'u':
                case 'U': printf(" %c is a vowel", ch);
                          break;
                default:printf(" %c is not a vowel", ch);
                }
        getch();
        }
```

Example: C Program to find the roots of a quadratic equation using switch statement

```
#include<stdio.h>
#include<math.h>
void main()
{
   int flag;
   float x, x1, x2;
   float a, b, c, d;
   float rpart, ipart;
   clrscr();
   printf("\n Enter 3 numbers: ");
   scanf("%f %f %f", &a, &b, &c);
   if(a==0)
  {
     x=-b/c;
     printf("\n Only root x : %f", x);
     exit();
  }
  d=b*b-4*a*c;
  if(d>0)
     flag=1;
  else if(d==0)
     flag=2;
  else
     flag=3;
  switch(flag)
  {
     case 1:printf("\n Real & Distinct roots are: ");
               x1=(-b+sqrt(d))/(2*a);
               x2=(-b-sqrt(d))/(2*a);
               printf("\n x1=%f \n x2=%f", x1, x2);
               break;
     case 2:printf("\n Repeated roots are: ");
               x1=-b/(2*a);
               x2=x1;
               printf("\n x1 & x1 : %f", x1);
               break;
     case 3: d=sqrt(abs(d));
               rpart=-b/(2*a);
               ipart=d/(2*a);
               printf("\n Complex Roots are: ");
               printf("\n x1=%f+i%f", rpart, ipart);
               printf("\n x2=%f-i%f", rpart, ipart);
   }
   getch();
}
```

UNCONDITIONAL CONTROL STATEMENTS: Unconditional branching is when the programmer forces the execution of a program to jump to another part of the program. Theoretically, this can be done using a good combination of loops and if statements. In fact, as a programmer, you should always try to avoid such unconditional branching and use this technique only when it is very difficult to use a loop.

C Supports These unconditional control statements:

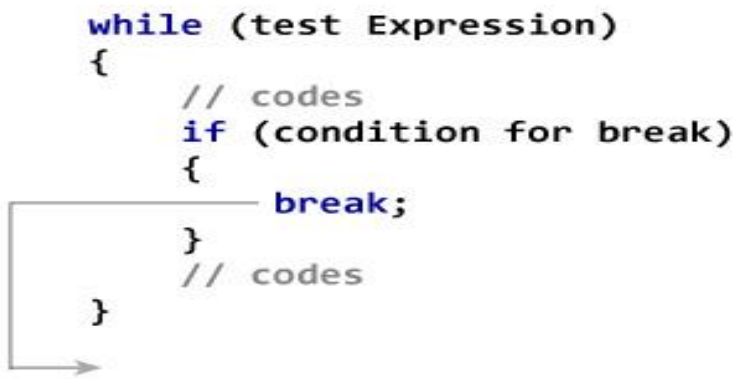
1. BREAK
2. CONTINUE
3. RETURN
4. GOTO

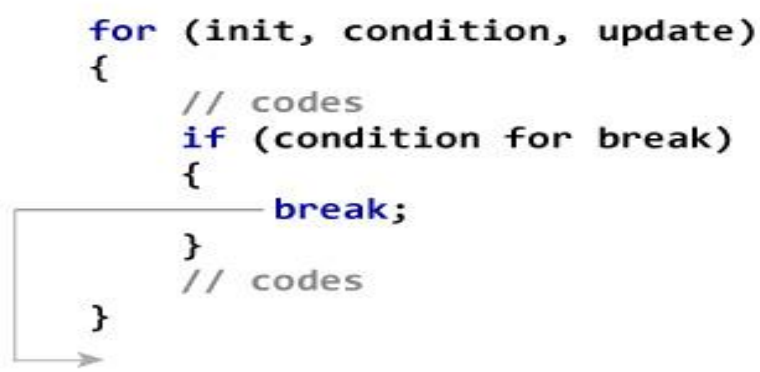
Break: It is used to terminate a switch statement. BREAK is a keyword that allows us to jump out of a loop instantly, without waiting to get back to the conditional test. The break statement terminates the loop (for, while and do...while loop) immediately when it is encountered. The break statement is used with decision making statement such as if...else.

Syntax of break statement

```
break;
```

How break statement works in loops?





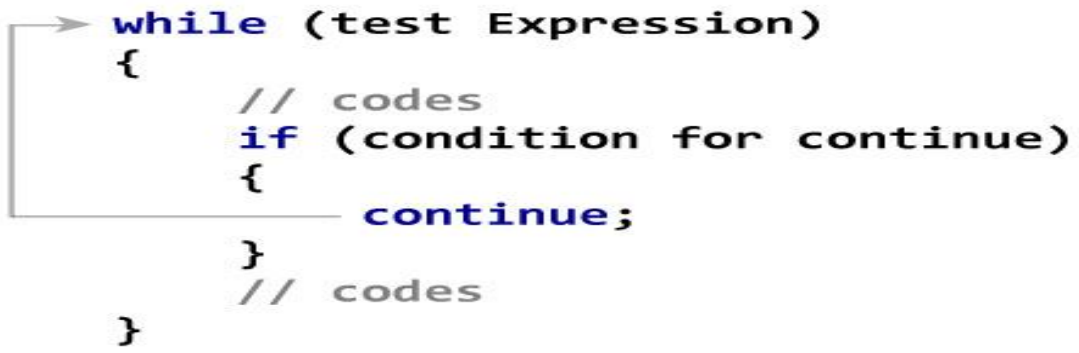
Example of Break in switch we have seen previous topic.

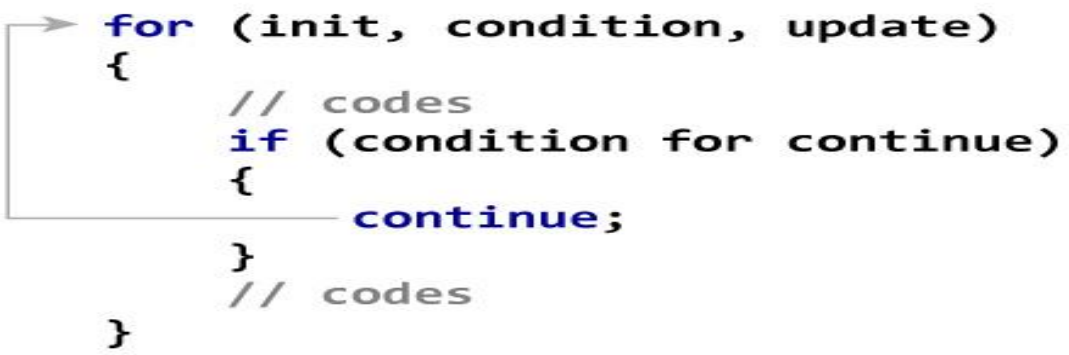
Continue: The continue statement skips some statements inside the loop. The continue statement is used with decision making statement such as if...else.

Syntax of continue Statement

continue;

How continue statement works?





Goto: A goto statement in C programming provides an unconditional jump from the 'goto' to a labeled statement in the same function.

NOTE − Use of goto statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten to avoid them.

Syntax

The syntax for a goto statement in C is as follows −

goto label;

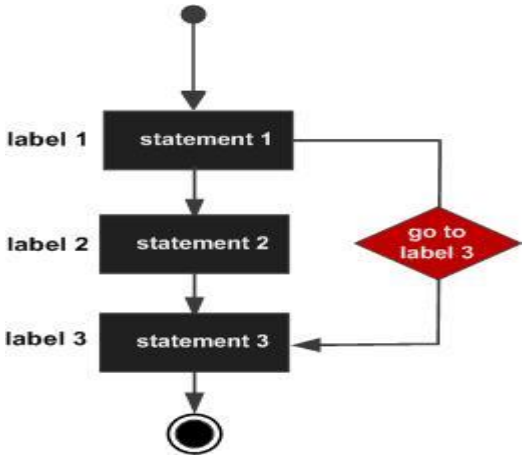
..

..

label: statement;

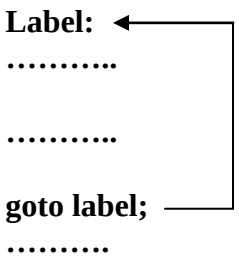
Here label can be any plain text except C keyword and it can be set anywhere in the C program above or below to goto statement.

Flow Diagram



Forward Goto:

Backward Goto



Examples: Forward goto.

```
#include<stdio.h>
#include <conio.h>
int main()
 {
  printf (“Hello Worldn”);
  goto Label;
  printf(“How are you?”);
  printf(“Are you Okey?”);
  Label:
  printf(“Hope you are fine”);
  getch();
 }
```

Example: Backward goto.

```
 #include<stdio.h>
#include <conio.h>
#include<math.h>
int main()
 {
       int num;
       int answer;
  Label:
      printf (“Enter a number:”);
      scanf(“%d”,&num);
      if(num>0)
         answer = sqrt(num);
      printf(“Square root of %d is %dn”,num,answer);
   goto Label;
  getch();
}
```

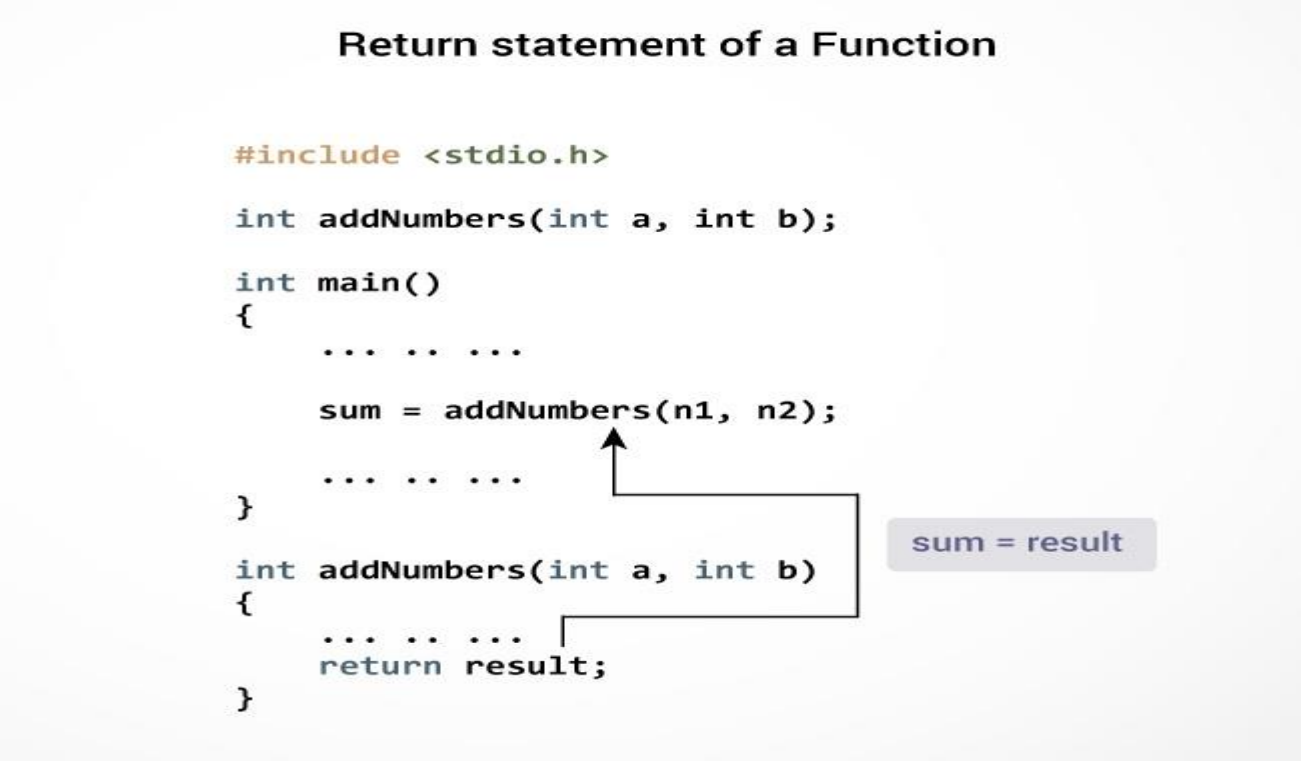
Return: The return statement terminates the execution of a function and returns control to the calling function. Execution resumes in the calling function at the point immediately following the call. A return statement can also return a value to the calling function.

Syntax:

return;

or

return(value);



C – Loop control statements: You may encounter situations, when a block of code needs to be executed several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

A loop statement allows us to execute a statement or group of statements multiple times.

In C programing loops are basically categorized into two categories.

1. Entry control loop
2. Exit control loop

Entry Control loop: An entry control loop checks the condition at the time of entry and if condition or expression becomes true then control transfers into the body of the loop. Such type of loop controls entry to the loop that's why it is called entry control loop.

Flowchart:

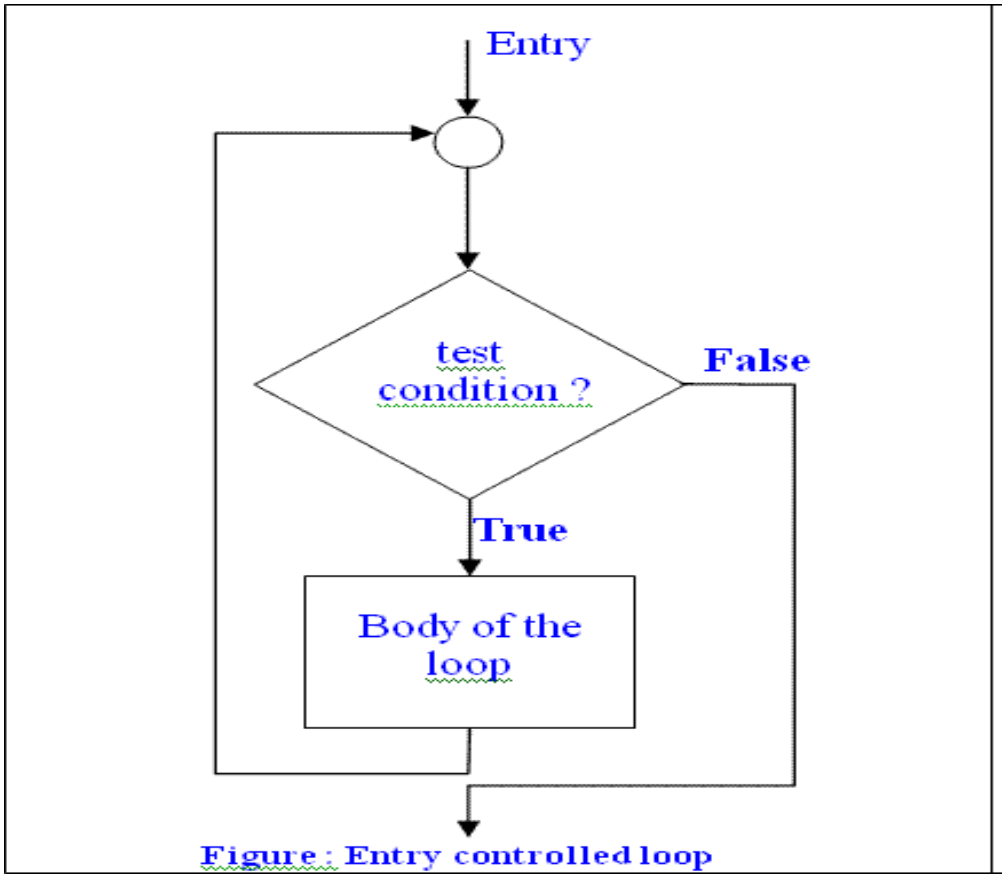


Figure : Entry controlled loop

Note: if condition fails very first time the body of the loop will not be executed.

Exit Control Loop: An Exit Control Loop checks the condition for exit and if given condition for exit evaluate to true, control will exit from the loop body else control will enter again into the loop. Such type of loop controls exit of the loop that's why it is called exit control loop.

Flowchart:

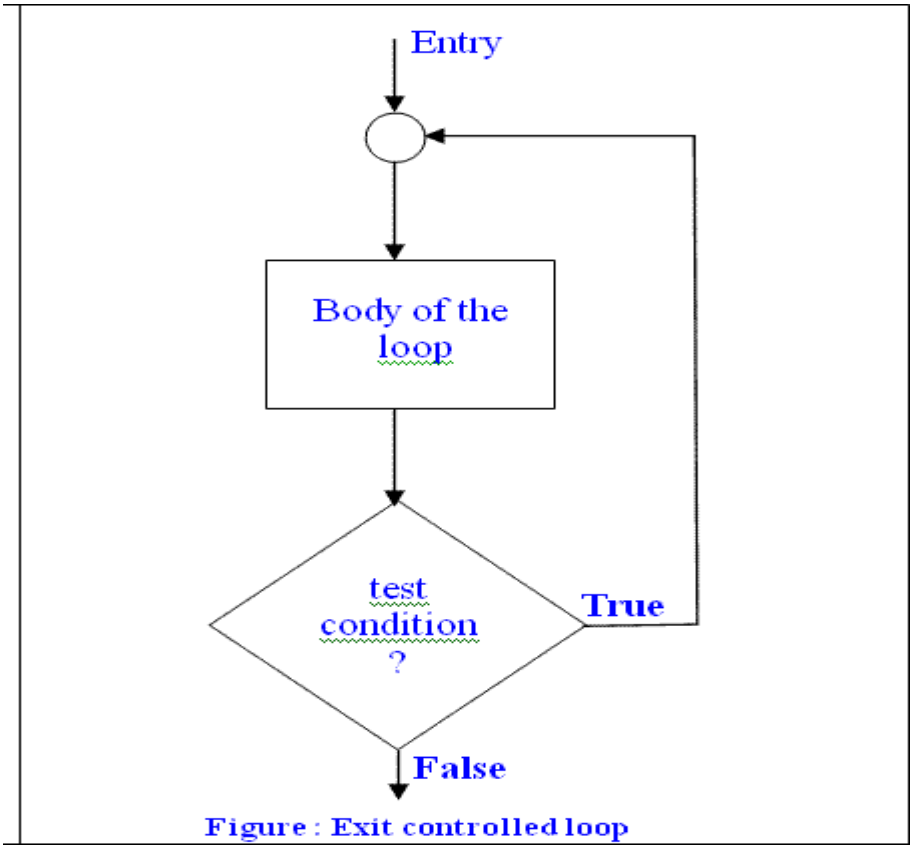


Figure : Exit controlled loop

Note:even if condition fails first time but body of loop will exicute minimum one time.

There are three loops in C programming:

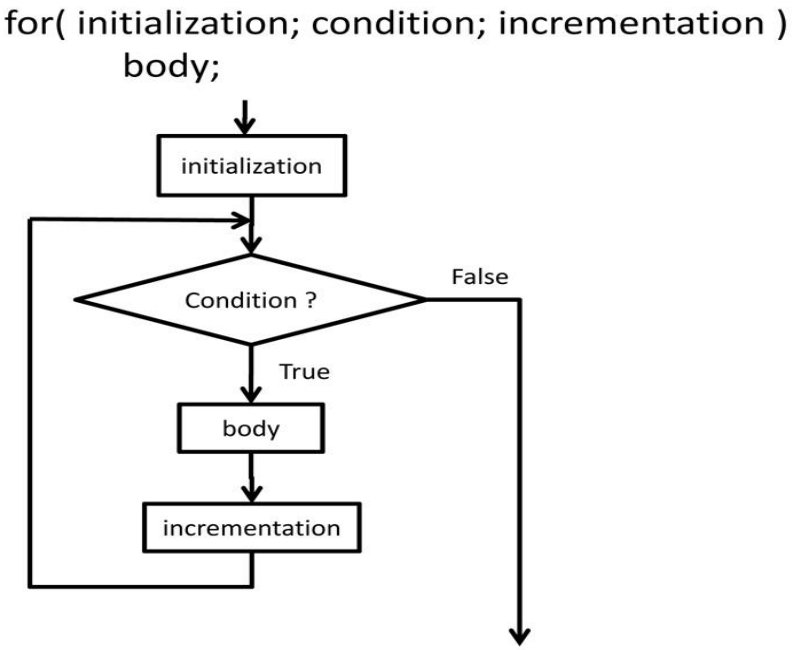
1. for loop
2. while loop
3. do...while loop

for loop: This is one of the most frequently used loop in C programming. This is an entry control loop.

Syntax of for loop:

```
for (initialization; condition test; increment or decrement)
{
    //Statements to be executed repeatedly
}
```

Flowchart of for loop:



Working:

Step 1: First initialization happens and the counter variable gets initialized.

Step 2: In the second step the condition is checked, where the counter variable is tested for the given condition, if the condition returns true then the C statements inside the body of for loop gets executed, if the condition returns false then the for loop gets terminated and the control comes out of the loop.

Step 3: After successful execution of statements inside the body of loop, the counter variable is incremented or decremented, depending on the operation (++ or –).

Example: C Program to calculate the sum of first n natural numbers.

```
#include <stdio.h>
#include<conio.h>
void main()
{
   int num, count, sum = 0;
   printf("Enter a positive integer: ");
   scanf("%d", &num);
   // for loop terminates when n is less than count
   for(count = 1; count <= num; ++count)
   {
      sum += count;
   }
   printf("Sum = %d", sum);
   getch();
}
```

Example: C Program to find Factorial of a Number.

```
#include <stdio.h>
#include<conio.h>
void  main()
{
   int n, i;
   unsigned long long factorial = 1;
   printf("Enter an integer: ");
   scanf("%d",&n);
   // show error if the user enters a negative integer
   if (n < 0)
      printf("Error! Factorial of a negative number doesn't exist.");
   else
   {
      for(i=1; i<=n; ++i)
      {
         factorial *= i;              // factorial = factorial*i;
      }
      printf("Factorial of %d = %llu", n, factorial);
   }
    getch();
}
```

Example: Fibonacci series program in C language.

The Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21

```
#include<stdio.h>
 #include<conio.h>
void main()
{
  int n, first = 0, second = 1, next, c;
   printf("Enter the number of terms\n");
  scanf("%d",&n);
   printf("First %d terms of Fibonacci series are :-\n",n);
   for ( c = 0 ; c < n ; c++ )
  {
     if ( c <= 1 )
        next = c;
     else
     {
        next = first + second;
        first = second;
        second = next;
     }
     printf("%d\n",next);
  }
   getch();
}
```

Various forms of for loop in C:

1) Initialization part can be skipped from loop as shown below, the counter variable is declared before the loop.

```
int num=10;
for (;num<20;num++)
```

Note: Even though we can skip initialization part but semicolon (;) before condition is must, without which you will get compilation error.

2) Like initialization, you can also skip the increment part as we did below. In this case semicolon (;) is must after condition logic. In this case the increment or decrement part is done inside the loop.

```
for (num=10; num<20; )
{
    //Statements
    num++;
}
```

3) This is also possible. The counter variable is initialized before the loop and incremented inside the loop.

```
int num=10;
for (;num<20;)
{
    //Statements
    num++;
}
```

4) As mentioned above, the counter variable can be decremented as well. In the below example the variable gets decremented each time the loop runs until the condition num>10 returns false.

```
for(num=20; num>10; num--)
```

Nested For Loop in C:

```
#include <stdio.h>
void main()
{
  for (int i=0; i<2; i++)
  {
    for (int j=0; j<4; j++)
    {
      printf("%d, %d\n",i ,j);
    }
  }
  getch();
}
```

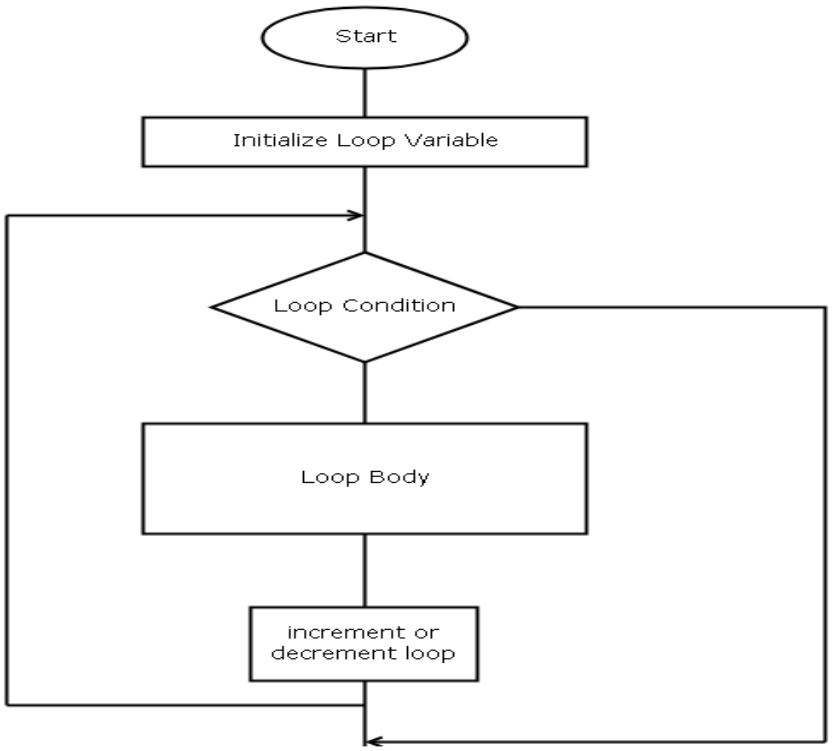
In the above example we have a for loop inside another for loop, this is called nesting of loops. One of the example where we use nested for loop is Two dimensional array.

While loop: while loop is a most basic loop in C programming. while loop has one control condition, and executes as long the condition is true. The condition of the loop is tested before the body of the loop is executed; hence it is called an entry-controlled loop.

Syntax:

```
Initialization of loop variable;
while (condition)
{
  //body of loop
  statement(s);
  updation of loop variable;
}
```

Flowchart:



Working:

step1: The loop variable is initialized with some value and then it has been tested for the condition.

step2: If the condition returns true then the statements inside the body of while loop are executed else control comes out of the loop.

step3: The value of loop variable is incremented/decremented then it has been tested again for the loop condition.

Example: C program to generate Fibonacci Sequence Up to a Certain Number.

```
#include <stdio.h>
void main()
{
   int t1 = 0, t2 = 1, nextTerm = 0, n;
   printf("Enter a positive number: ");
   scanf("%d", &n);
   // displays the first two terms which is always 0 and 1
   printf("Fibonacci Series: %d, %d, ", t1, t2);
   nextTerm = t1 + t2;
   while(nextTerm <= n)
   {
      printf("%d, ",nextTerm);
      t1 = t2;
      t2 = nextTerm;
      nextTerm = t1 + t2;
   }
   getch();
}
```

Example: C program to Calculate sum of digits using while loop.

```
#include<stdio.h>
void main()
{
   int a, s;
   printf("Enter value of a: ");
   scanf("%d",&a);
   s = 0;
   while(a > 0)
   {
      s = s + (a%10);
      a = a / 10;
   }
   printf("Sum of digits: %d",s);
   getch();
}
```

Example: C Program to display the prime numbers using while loop

```
#include<stdio.h>
void main()
{
   int k,remark,first,last;
   printf("Give the First Number for the Range : ");
   scanf("%d",&first);
   printf("\n Give the Last Number for the Range : ");
   scanf("%d",&last);
  while(first< last)
  {
   remark=0;
   k=2;
```

```
 while(k<=first/2)
 {
    if((first % k) == 0)
    {
       remark=1;
       break;
    }
 k++;
  }
 if(remark==0)
   printf("\n    %d ",numbr);
++first;
 }
 getch();
}
```

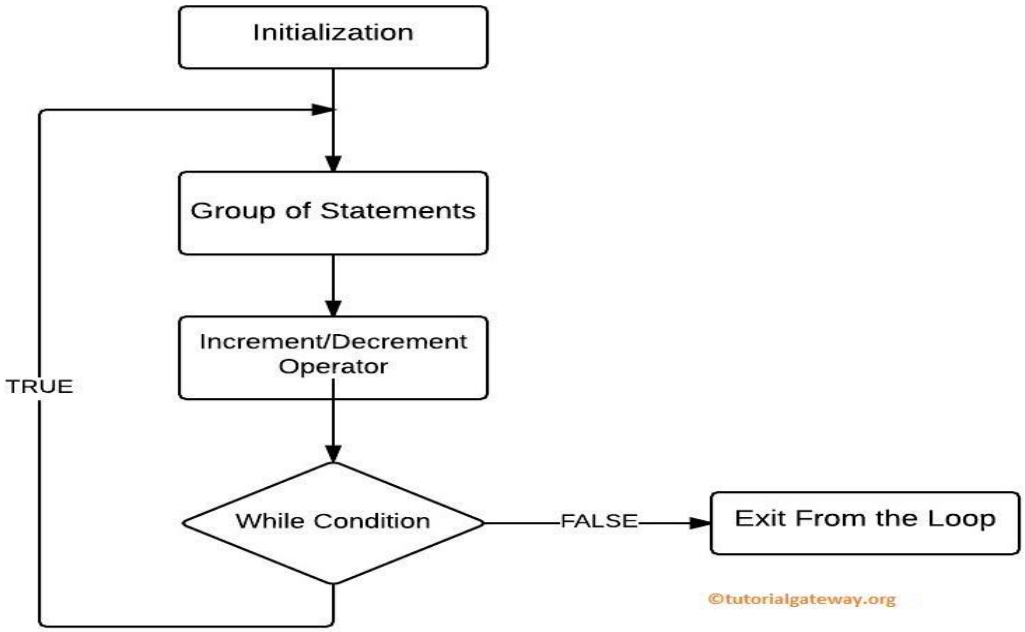
do-while loop: is an exit controlled loop i.e. the condition is checked at the end of loop. It means the statements inside do-while loop are executed at least once even if the condition is false. Do-while loop is an variant of while loop.

Syntax:

```
Initialization of loop variable;
do
{
  statement 1;
  statement 2;
    ……….
  statement n;
updation of loop variable;
}while (condition);
```

NOTE: We have to place semi-colon after the While condition.

Flowchart:



Working:

1. First we initialize our variables, next it will enter into the Do While loop.
2. It will execute the group of statements inside the loop.
3. Next we have to use Increment and Decrement Operator inside the loop to increment or decrements the value.
4. Now it will check for the condition. If the condition is True, then the statements inside the do while loop will be executed again. It will continue the process as long as the condition is True.
5. If the condition is False then it will exit from the loop.

Example: C program to calculate factorial value using do while.

```
#include<stdio.h>
#include<conio.h>
void main()
{
  long int i,n,fact=1; /*variable declaration */
  clrscr();
  printf("Enter the value of n \n");
  scanf("%ld", &n);
  i=1;
  do
  {
    fact*=i;
    i++;
  }
  while(i<=n);
   printf("Factorial = %ld\n",fact);
  getch();
  }
```

Example: Write a C program to print the sum of all even and odd numbers up to n.

```
#include<stdio.h>
void main()
{
    int n,s1=0,s2=0,i;
    printf("Enter Number : ");
    scanf("%d",&n);
    i=1;
    do
    {
        if(i%2==0)
            s1=s1+i;
        else
            s2=s2+i;
     i++;
    }while(i<=n);
    printf("\nSum of Even Numbers :  %d\n",s1);
    printf("\nSum of Odd Numbers  :  %d\n",s2);
    getch();
}
```

Example: C Program to sum of even numbers up to n using continue statement.

```
#include <stdio.h>
#include <conio.h>
 void main()
{
   int N, counter, sum=0;
   printf("Enter a positive number\n");
   scanf("%d", &N);
   for(counter=1; counter <= N; counter++)
   {
      if(counter%2 == 1)
      {
         continue;
      }
      sum+= counter;
   }
   printf("Sum of all even numbers between 1 to %d = %d", N, sum);
   getch();
 }
```

Difference Between break and continue:

break	continue
A `break` can appear in both `switch` and loop (`for`, `while`, `do`) statements.	A `continue` can appear only in loop (`for`, `while`, `do`) statements.
A `break` causes the `switch` or loop statements to terminate the moment it is executed. Loop or `switch` ends abruptly when break is encountered.	A `continue` doesn't terminate the loop, it causes the loop to go to the next iteration. All iterations of the loop are executed even if `continue` is encountered. The `continue` statement is used to skip statements in the loop that appear after the `continue`.
When a `break` statement is encountered, it terminates the block and gets the control out of the `switch` or loop.	When a `continue` statement is encountered, it gets the control to the next iteration of the loop.
A `break` causes the innermost enclosing loop or `switch` to be exited immediately.	A `continue` inside a loop nested within a `switch` causes the next loop iteration.

Difference between while and do while loops:

While	Do While
The loop which continues until the statement holds true and repeats constantly.	The loop which holds true for specific instructions.
Only one statement for all the package to work	Requires separate statement for all the while conditions.

While (condition)
{
statement;
}.
	Do
	{
	statements;
	} while (condition);

Entry control.	Exit control.
Takes less time to execute but and the code is shorter.	Takes more time to execute and code becomes longer.