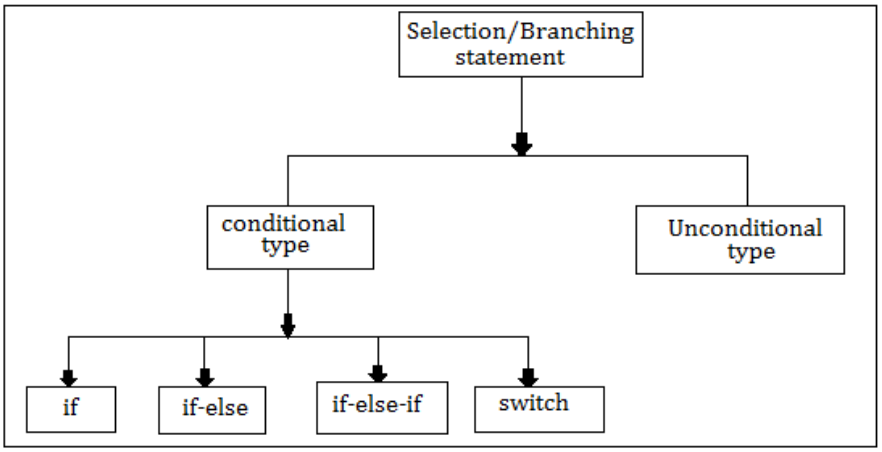
Introduction to C programming
MODULE 2
CHAPTER 2: Decision Making and Branching

1. INTRODUCTION

C supports two types of decision control statements that can alter the flow of a sequence of instructions. These include conditional type branching and unconditional type branching.



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. Also known as control statements.

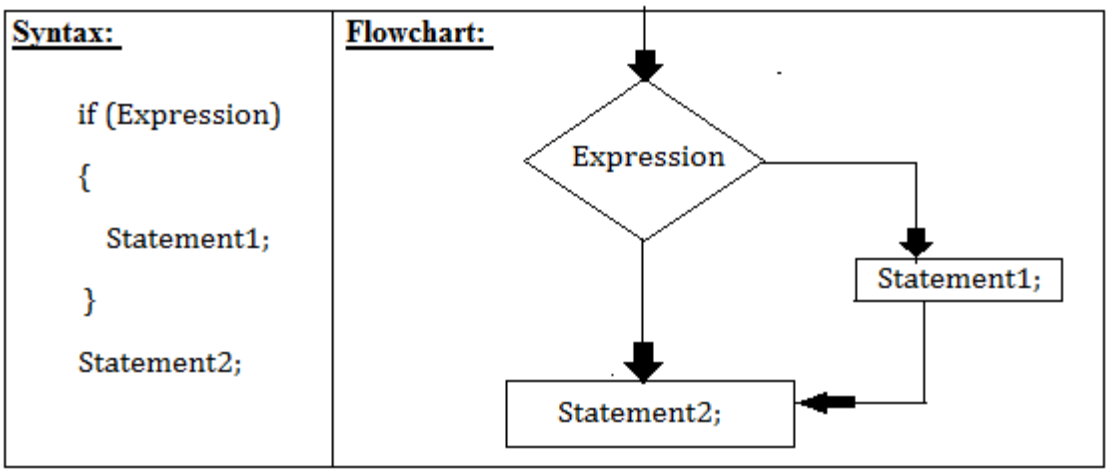
2. DECISION MAKING (Two-Way Selection Statements)

- The basic decision statement in the computer is the two way selection.
- The decision is described to the computer as conditional statement that can be answered TRUE or FALSE.
- If the answer is TRUE, one or more action statements are executed.
- If answer is FALSE, the different action or set of actions are executed.
- Regardless of which set of actions is executed, the program continues with next statement.

C language provides following two-way selection statements:

1. if statement
2. if – else statement
3. Nested if else statement
4. Cascaded if else (also called else-if ladder)

1. if statement: The general form of simple if statements is shown below.



- The Expression is evaluated first, if the value of Expression is true (or non zero) then Statement1 will be executed; otherwise if it is false (or zero), then Statement1 will be skipped and the execution will jump to the Statement2.
- Remember when condition is true, both the Statement1 and Statement2 are executed in sequence. This is illustrated in Figure1.

Note: Statement1 can be single statement or group of statements.

Example:

```
#include<stdio.h>
 void main( )
{
int a=20, b=11;
if (a >b)
{
printf("a is bigger and value is= %d\n", a);
}
printf("job over\n");
}
```

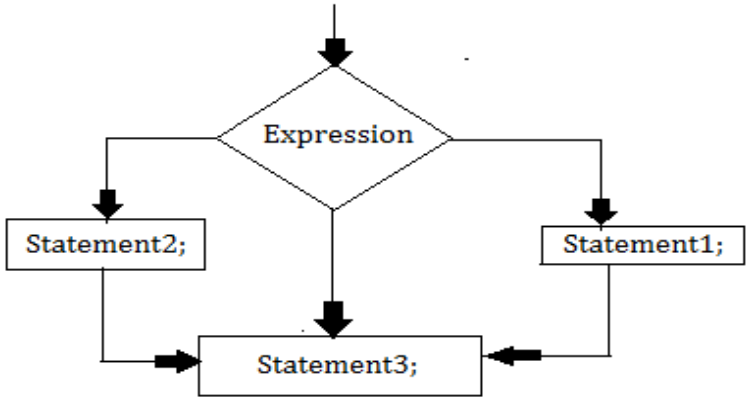
Output: a is bigger and value is=20 job over

2. if..else statement: The if..else statement is an extension of simple if statement. General syntax:

Syntax:

```
if (Expression)
{
  (true-block;)
   Statement1;
}
else
{
  (false-block;)
   Statement2;
}
 Statement3;
```

Flowchart:



- If the Expression is true (or non-zero) then Statement1 will be executed; otherwise if it is false (or zero), then Statement2 will be executed.
- In this case either true block or false block will be executed, but not both.
- This is illustrated in Figure. In both the cases, the control is transferred subsequently to the Statement3.

Example:

```
void main( )
{
   int a=10, b=11;
   if (a >b)
  {
       printf("a is bigger and value is= %d", a);
  }
 else
 {
     printf("b is bigger and value is= %d", b);
  }
}
```

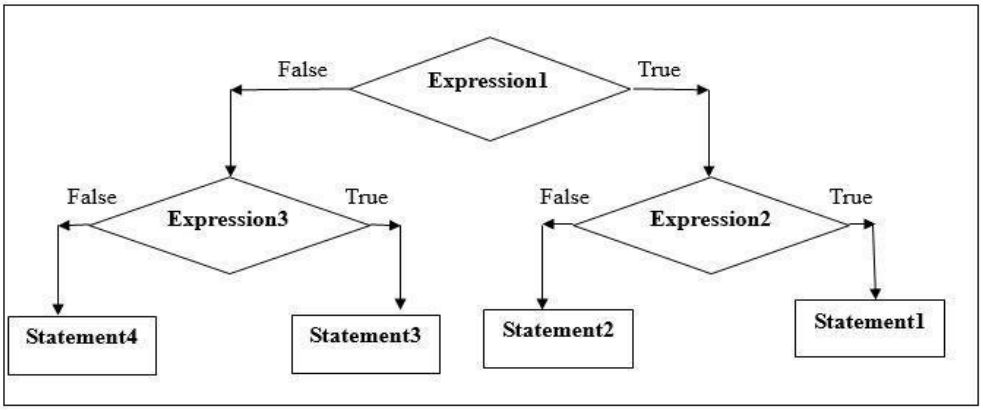
Output: b is bigger and value is=11

3. if –else-if (Nested) statement: When a series of decisions are involved, we have to use more than one if..else statement in nested form as shown below in the general syntax.

Syntax:

```
if (Expression1)
{
 if(Expression2)
        {
        Statement1;
        }
 else
        {
        Statement2;
        }
}
else
{
 if(Expression3)
  {
  Statement3;
  }
 else
  {
   Statement4;
  }
 }
```

Flowchart:



- If Expression1 is true, check for Expression2, if it is also true then Statement1 is executed.
- If Expression1 is true, check for Expression2, if it is false then Statement2 is executed.
- If Expression1 is false, then Statement3 is executed.
- Once we start nesting if .. else statements, we may encounter a classic problem known as dangling else.
- This problem is created when no matching else for every if.
- C solution to this problem is a simple rule "always pair an else to most recent unpaired if in the current block".
- Solution to the dangling else problem, a compound statement.
- In compound statement, we simply enclose true actions in braces to make the second if a compound statement.

Example:

```
void main()
{
   int A=20,B=15,C=3;
   if ( A>B)
    {
       if (A>C)
        {
            printf("largest=%d", A);
```

```
        }
       else
        {
           printf("largest=%d", C);
        }
   }
  else
   {
     if(B>C)
      {
           printf("largest=%d", B);
      }
     else
     {
           printf("largest=%d", C);
      }
   }
}
```

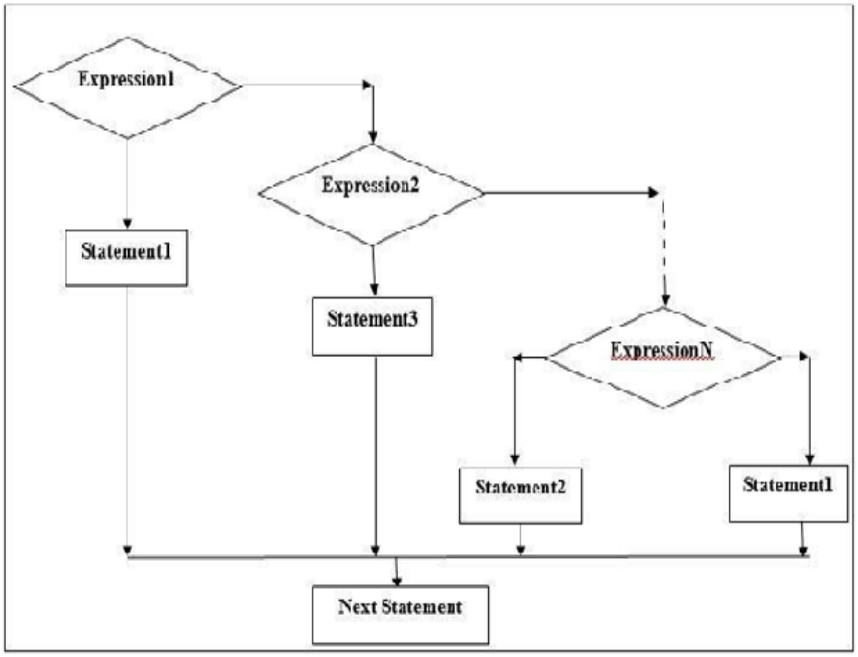
Output: largest=20

4. else-if ladder or cascaded if else: There is another way of putting ifs together when multipath decisions are involved. A multi path decision is a chain of ifs in which the statement associated with each else is an if. It takes the following form. This construct is known as the else if ladder.

Syntax:

```
if (Expression1)
{
      Statement1;
}
     else if(Expression2)
      {
           Statement2;
      }
            else if(Expression3)
            {
                  Statement3;
            }
                  else
                  {
                        Statement4;
                  }
Next Statement;
```

Flowchart:



- The conditions are evaluated from the top (of the ladder), downwards. As soon as true condition is found, the statement associated with it is executed and control transferred to the Next statement skipping the rest of the ladder.
- When all conditions are false then the final else containing the default Statement4 will be executed.

Example:

```
#include<stdio.h>
void main()
{
int choice, a=100, b=5;
printf("enter 1 for addition 2 for subtraction 3 for multiply 4 for modulus\n");
scanf("%d", &choice);
   if(choice==1)
      printf("sum=%d", a+b);
    else if(choice==2)
       printf("difference=%d", a-b);
    else if(choice==3)
         printf("product=%d", a*b);
      else if (choice==4)
          printf("rem=%d", a%b);
        else
           printf("invalid choice");
}
```

Dangling else problem

With nesting of if-else statements, we often encounter a problem known as dangling else problem. This problem is created when there is no matching else for every if statement. In such cases, C always pairs an else statement to the most recent unpaired if statement in current block.

```
if(a>b)
  if(a>c)
     printf("\n a is greater than b and c");
  else
    printf("\n a is greater than b and c");
```

The problem is that both the outer if statement and the inner if statement might conceivably own the else clause. So the programmer must always see that every if statement is paired with an appropriate else statement.

3. SWITCH STATEMENT

C language provides a multi-way decision statement/menu-driven statement so that complex else-if statements can be easily replaced by it. C language's multi-way decision statement is called switch.

General syntax of switch statement is as follows:

```
switch(choice)
{
        case label1: block1;
                     break;
        case label2: block2;
                     break;
        case label3: block-3;
                     break;
        default:     default-block;
                     break;
}
```

- Here switch, case, break and default are built-in C language words.
- If the choice matches to label1 then block1 will be executed else if it evaluates to label2 then block2 will be executed and so on.
- If choice does not matches with any case labels, then default block will be executed.

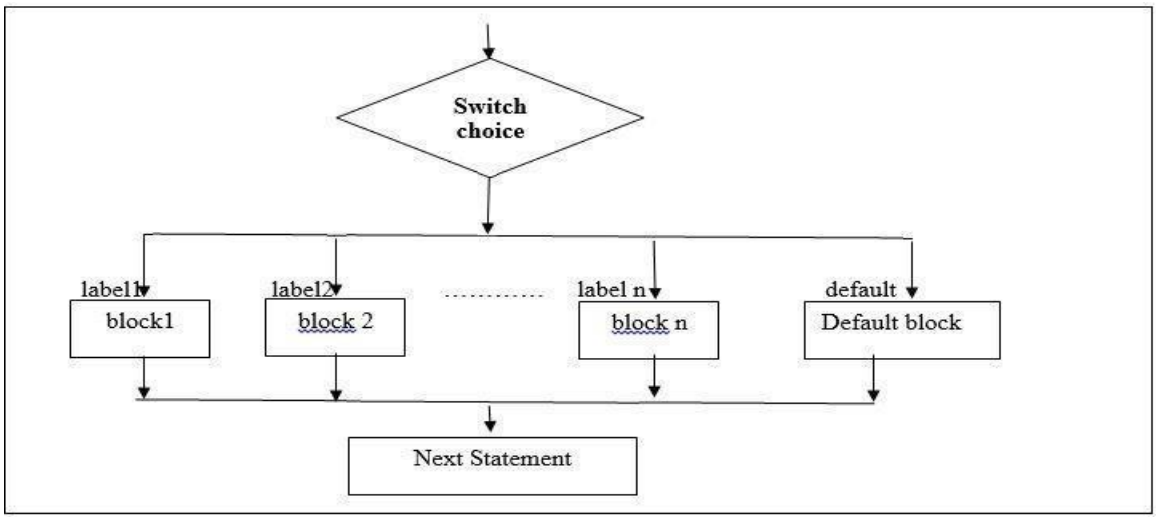


Figure 5: Flow chart of switch case statement

- The choice is an integer expression or characters.
- The label1, label2, label3,.... are constants or constant expression evaluate to integer constants.
- Each of these case labels should be unique within the switch statement. block1, block2, block3, … are statement lists and may contain zero or more statements.
- There is no need to put braces around these blocks. Note that case labels end with colon(:).
- Break statement at the end of each block signals end of a particular case and causes an exit from switch statement.
- The default is an optional case when present, it will execute if the value of the choice does not match with any of the case labels.

Example:

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
  int ch,a,b;
  float res;

  printf("Enter two numbers:\n");
  scanf("%d%d",&a,&b);
  printf("1=Add, 2=Sub, 3=Mul, 4=Div,5=Rem\n");
  printf("Enter your choice:\n"); scanf("%d",&ch);
  switch(ch)
  {
   case 1: res=a+b;
         break;
   case 2: res=a-b;
         break;
   case 3: res=a*b;
         break;
   case 4: res=(float)a/b;
         break;
   case 5: res=a%b;
         break;
   default: printf("Entered Wrong choice\n");
  }
  printf("Result=%d\n",res);
}
```

```
#include<stdlib.h>
void main()
{
  int ch,a,b;
  float res;

  printf("Enter two numbers:\n");
  scanf("%d%d",&a,&b);
  printf("+ for Add, - for Sub, * for mul, / for Div, % for Rem\n");
  printf("Enter your choice:\n");
  scanf("%c",&ch);
  switch(ch)
  {
    case '+': res=a+b;
          break;
    case '-': res=a-b;
          break;
    case '*': res=a*b;
          break;
    case '/': res=(float)a/b;
          break;
    case '%': res=a%b;
          break;
    default: printf("Entered Wrong choice\n");
  }
  printf("Result=%d\n",res);
}
```

In this program if ch=1 case '1' gets executed and if ch=2, case '2' gets executed and so on.

4.Ternary operator or conditional operator (?:)

- C Language has an unusual operator, useful for making two way decisions.
- This operator is combination of two tokens ? and : and takes three operands.
- This operator is known as ternary operator or conditional operator.

Syntax:

```
Expression1 ? Expression2 : Expression3
```

Where,
Expression1 is Condition
Expression2 is Statement Followed if Condition is True Expression3 is Statement Followed if Condition is False

Meaning of Syntax:
1. Expression1 is nothing but Boolean Condition i.e. it results into either TRUE or FALSE
2. If result of expression1 is TRUE then expression2 is executed
3. Expression1 is said to be TRUE if its result is NON-ZERO
4. If result of expression1 is FALSE then expression3 is executed
5. Expression1 is said to be FALSE if its result is ZERO.

Example: Check whether Number is Odd or Even

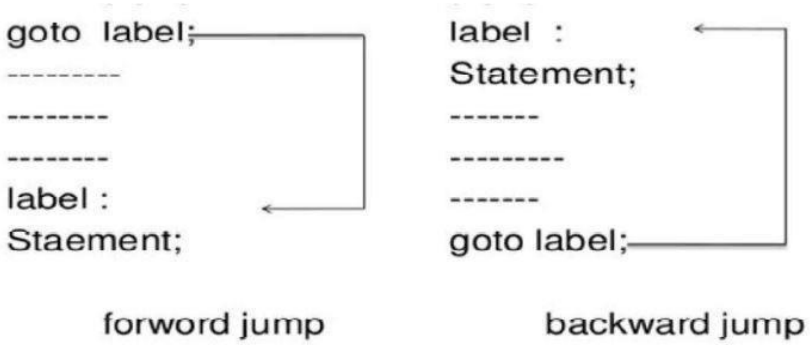
```
#include<stdio.h>
void main()
{
   int  num,flag;
   printf("Enter the Number : ");
   scanf("%d",&num);
  flag = ((num%2==0)?1:0);
     if(flag==1)
        printf("\nEven");
     else
        printf("\nOdd");
}
```

5. Goto statement

goto is an unconditional branching statement.

The syntax of goto is as follows:

```
goto label;
    statement1;
    statement2;
label:
```



- A label is a valid variable name.
- Label need not be declared and must be followed by colon.
- Label should be used along with a statement to which control is transferred.
- Label can be anywhere in the program either before or after the goto label.
- Here control jumps to label skipping statement1 and statement2 without verifying any condition that is the reason we call it unconditional jumping statement.

Example:

```
void main( )
{
int a=5, b=7;
goto end;
a=a+1;
b=b+1;
end:
printf("a=%d b=%d", a,b);
}
```

CHAPTER 3:
DECISION MAKING AND LOOPING

1. INTRODUCTION

Definition of Loop: It is a programming structure used to repeatedly carry out a particular instruction/statement until a condition is true. Each single repetition of the loop is known as an iteration of the loop.

Three important components of any loop are:

1. Initialization (example: ctr=1, i=0 etc)
2. Test Condition (example: ctr<=500, i != 0 etc)
3. Updating loop control values (example: ctr=ctr+1, i =i-1)

Pre-test and Post-test loops

- Loops can be classified into two types based on the placement of test-condition.
- If the test-condition is given in the beginning such loops are called pre-test loops (also known as entry-controlled loops).
- Otherwise if test condition is given at the end of the loop such loops are termed as post-test loops (or exit controlled loops).

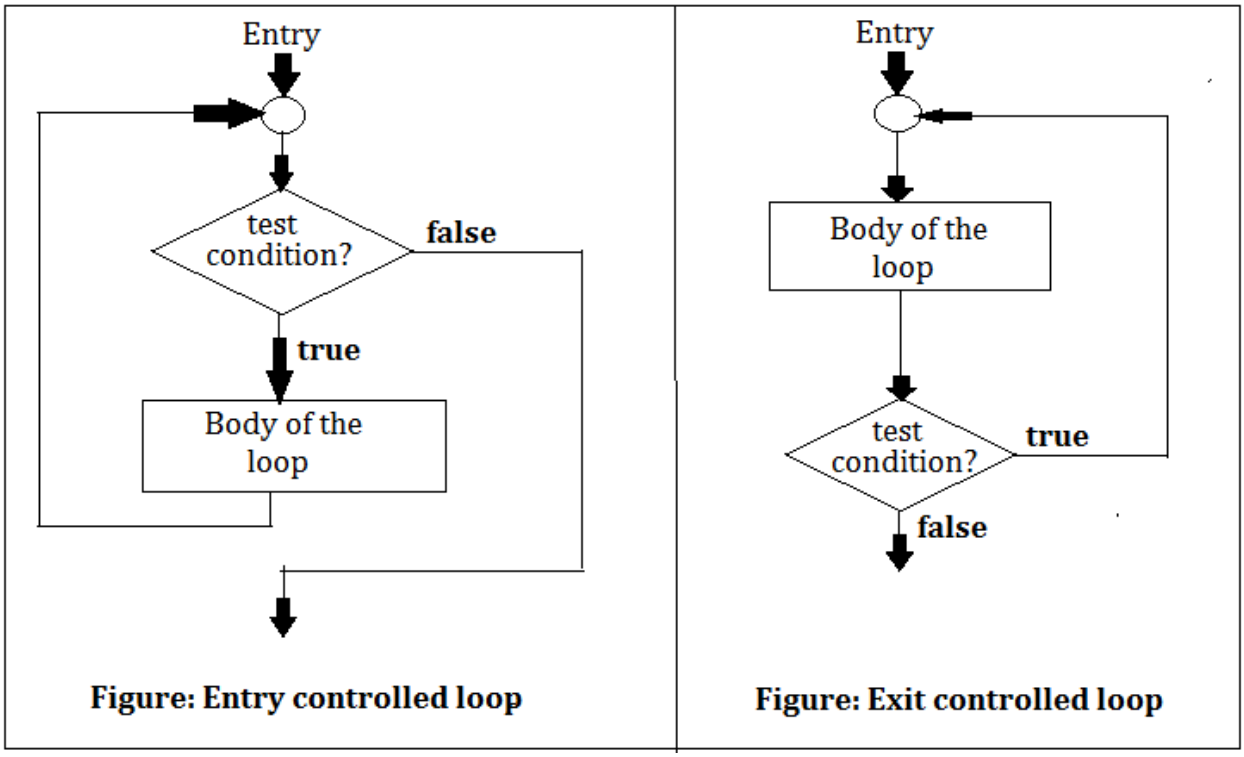


Figure: Entry controlled loop **Figure: Exit controlled loop**

Note Figure 1:
1. Here condition is at the beginning of loop. That is why it is called pre-test loop
2. It is also called as entry controlled loop because condition is tested before entering into the loop.
3. while is a keyword which can be used here.

Note Figure 2:
1. Here condition is at the end of the loop. That is why it is called post-test loop.
2. It is also called as exit controlled loop because condition is tested after body of the loop is executed at least once.
3. do and while are keywords which can be used here.

2. LOOPS IN C:

C language provides 3 looping structures namely:
1. while loop
2. do….while loop
3. for loop

The loops may be classified into two general types. Namely:
1. Counter-controlled loop
2. Sentinel-controlled loop

- When we know in advance exactly how many times the loop will be executed, we use a counter-controlled loop. A counter controlled loop is sometimes called definite repetition loop.
- In a sentinel-controlled loop, a special value called a sentinel value is used to change the loop control expression from true to false. A sentinel-controlled loop is often called indefinite repetition loop.

i. while loop:

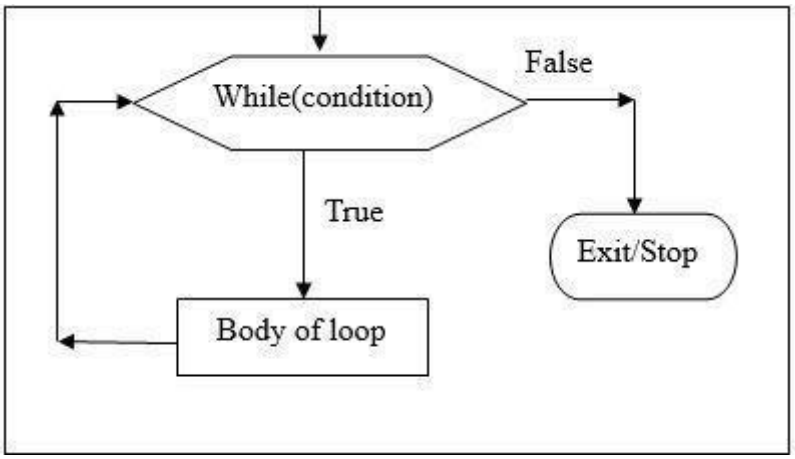
It is a pre-test loop (also known as entry controlled loop). This loop has following

General Syntax:

```
while (condition)
{
    statement-block;
}
```

- In the syntax given above 'while' is a key word and condition is at beginning of the loop.
- If the test condition is true the body of while loop will be executed.
- After execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again.
- This process is repeated until condition finally becomes false and control comes out of the body of the loop.

Flowchart:



Here is an example program using while loop for finding sum of 1 to 10.

Example:

```
void main()
{
int N=10; int i=1;
int sum=0;
while (i<=N)
{
sum=sum+i; i=i+1;
}
printf("%d", sum);
}
```

Before loop execution:
N=10, sum=0, i=1

First Round:
N=10, sum=1, i=2

Second Round:
N=10, sum=3, i=3

ii. do…. while loop:

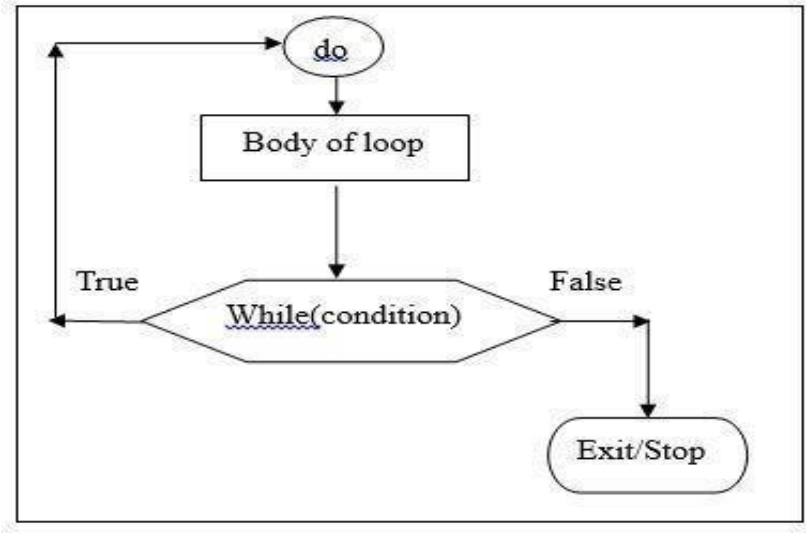
It is a post-test loop (also called exit controlled loop) it has two keywords do and while. The

General syntax:

```
do
{
  statement-block;
} while (condition);
```

- In this loop the body of the loop is executed first and then test condition is evaluated.
- If the condition is true, then the body of loop will be executed once again. This process continues as long as condition is true.
- When condition becomes false, the loop will be terminated and control comes out of the loop.

Flowchart:



Example: The example already discussed with while loop to compute sum of 1 to 10 numbers is given here with do …while loop:

```
void main( )
{
        int N=10;
        int i=1;
        int sum=0;
                do
                {
                 sum=sum+i;
                i=i+1;
                }
                while (i<=N);
        printf("%d", sum);
}
```

Before loop execution:

N=10, sum=0, i=1

First Round:

N=10, sum=1, i=2

Second Round:

N=10, sum=3, i=3

iii. for loop:

It is another pre-test loop (also called entry controlled loop) that provides concise loop control structure. It has one keyword for.

- One important aspect of for loop is all the three components of a loop (viz. initializing, testing condition and updation (increment/decrement)) is given in the head of for loop.

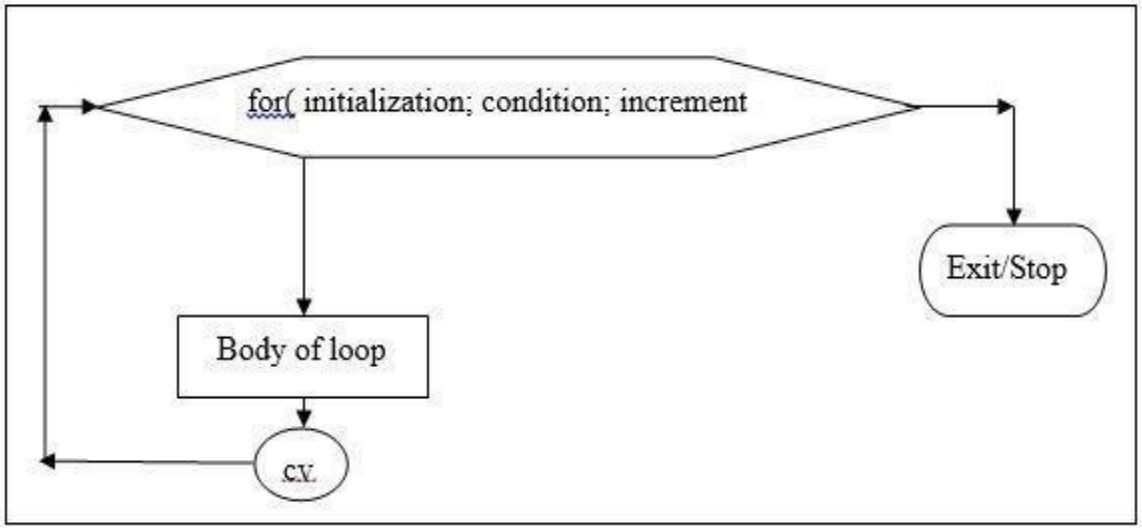
General syntax:

```
for( initialization; test-condition; updation)
{
        Body-of loop;
}
```

The execution of for statement is as follows.

1. In this loop first initialization of control variable is done first, using assignments such as i=1, count=0. The variable i count are called loop control variable.
2. The value of control variable is tested using the test condition. The test condition is evaluated. If the condition is true, the body of the loop is executed; otherwise loop will be terminated.
3. When the body of the loop is executed, the control transferred back to the for statement to update the loop variable. And then condition is checked once again. If condition is true once again body will be executed once again. This process continues till the value of the control variable fails to satisfy the test condition.

Flowchart:



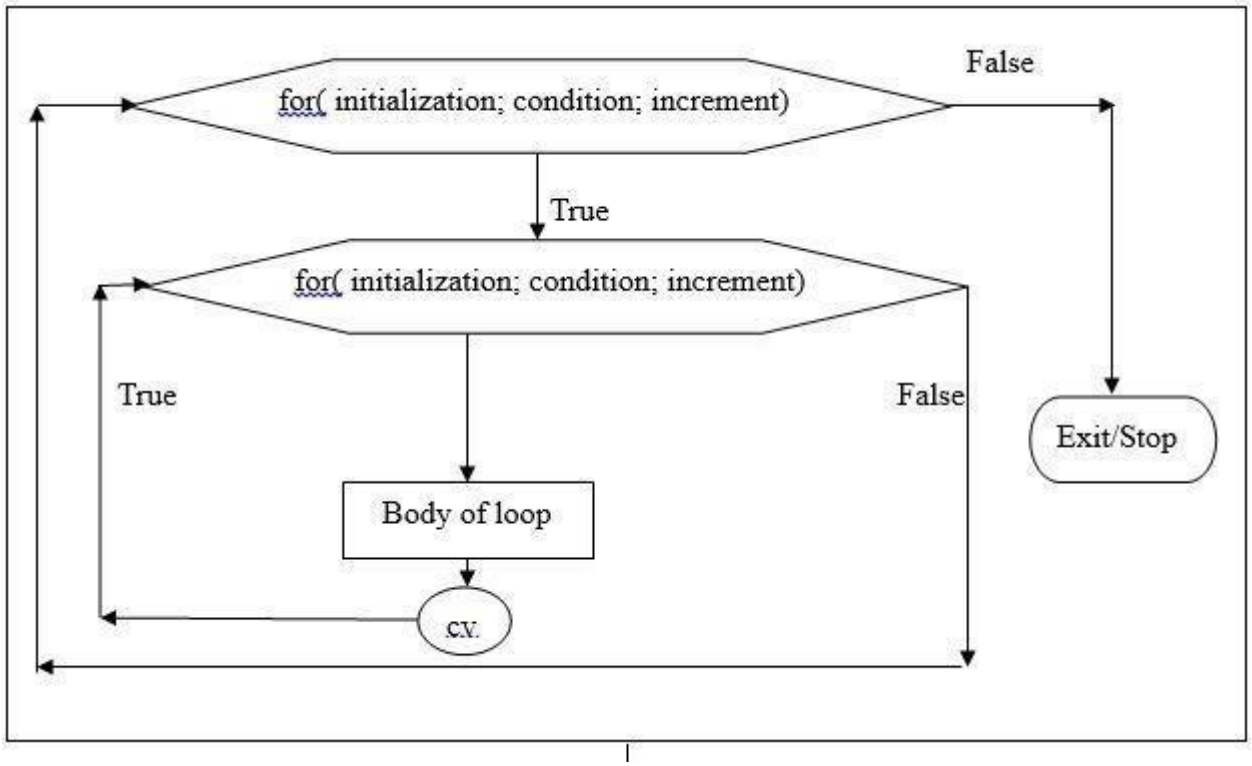
Example: Same program to find some of 1 to 10 numbers can be written using for loop as follows:

```
void main( )
{
int i, N=10, sum=0;
for (i=1; i<=N; i++)
  {
      sum=sum+i;
   }
printf("Total =%d", sum);
}
```

Note: In for loops whether both i++ or ++i operations will be treated as pre-increment only.

Nested for loops:

- In some programming situations within one for loop we have to write another for loop. Such embedded for loops are called nested for loops.
- Following diagram depicts a nested for loop flowchart.



Example:
Here is an example program segment with sample outputs that illustrates the working of a nested for loop:

```
for(rows=1; rows<=4; rows++)

 for(cols=1;cols<=3; cols++)

        {

        printf("%d %d", rows, cols);

        }
```

Working:

1. Initially value of rows=1 and 1 <=4 is true so control enters toinner loop
2. In inner loop cols=1 and 1<=3 is also true so body of the loop (i.e. printf gets executed)

Following are the outputs obtained in every repetition of inner/outer for loops:

Output: 1st round:	Output: 2nd round:	Output: 3rd round:	Output: 4th round:
rows=1,1,1 cols=1,2,3	rows=2,2,2 cols=1,2,3	rows=3,3,3 cols=1,2,3	rows=4,4,4 cols=1,2,3

Initializing, Test condition and Updating: Three Components of a Loop
Normally all loops should have three components:

1. Initialization (example: i=0; j=1 etc)
2. Test condition (example: ctr<=10, i<=20 etc)
3. Updating (example: ctr=ctr+1, i=i+2 etc)

If any of these three components are missing program behaves indifferently. For instance if updating statement is missing loop goes into infinite mode as shown in following example:

```
void main( )
{
    int ctr=1, N=10;
    while(ctr<=N)
      {
          printf("Hello World\n");
          printf("Hi Galaxy\n");
      }
}
```

Output: it prints,
Hello World
Hello Galaxy
infinite number of times as ctr value remains 1 forever as

Let us go through same example with initialization statement missing:
Example:

```
void main( )
{
  int ctr, N=10;
    while(ctr<=N)
       {
          printf("Hello World\n");
          printf("Hi Galaxy\n");
         ctr=ctr+1;
}
```

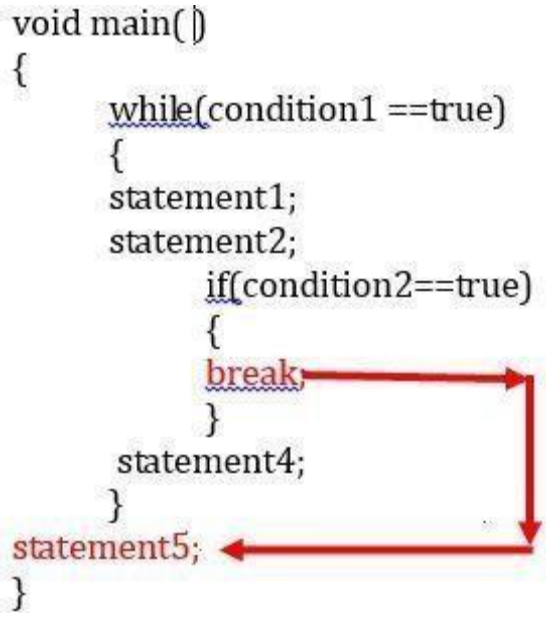
Output: Loop will not at all get executed as initial value of ctr is unknown!
So program prints nothing.

3. JUMPS IN LOOPS

Break and continue: break and continue are unconditional control construct.

i. break

This statement is useful to terminate a loop and transfer control out of loop under special situations.
break statement works with while, do….while, for and switch statements.
Following program syntax diagrammatically represents the working mechanism of break statement.



Note:

Working of break statement:

When condition2 is true the loop gets terminated and control is transferred to statement5 (which is outside the loop). Here we can observe that even though

Note: If break statement is used in the nested loops, the control will come out from the inner loop; still the outer loop is active.

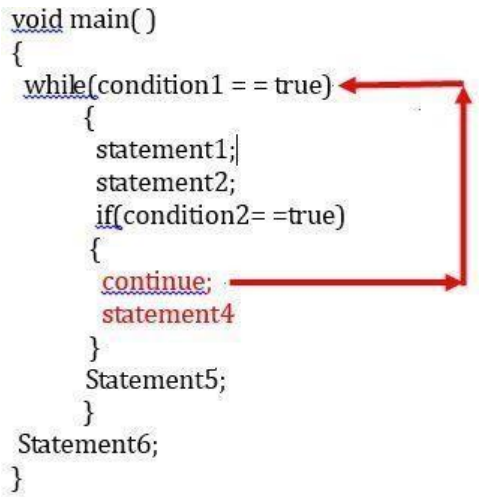
Example program for break statement:

```
void main()
{
int  i;
  for(i=1;i<=5;i++)
   {
      if(i==3)
      break;
      printf("%d ",i);
   }
}
```

Output: 1 2

ii. Continue

- Statement is helpful to skip particular set of statements in loop body and to continue execution from the beginning of loop.
- Following syntax clearly depicts how control shifts to beginning of a loop on finding continue statement.



Note:

Working of continue statement:

When condition2 is true continue statement is executed which results in transfer of control tobeginning of while loop skipping statement4 and statement5.

Example program for continue statement:

```
void main()
{
int i;
  for(i=1;i<=5;i++)
   {
     if(i==3)
     continue;
     printf("%d ",i);
   }
}
```

Output: 1 2 4 5

Practice program

1. Write a C program to determine whether a person is eligible for vote.

```
#include<stdio.h>
int main()
{
        int a ;
        printf("Enter the age of the person: ");
        scanf("%d",&a);

        if (a>=18)              //check voting eligibility
                printf("Eligible for voting");
        return 0;
}
```

Output:
enter the age: 28
Eligible for voting

2. Write a program to find whether a given year is a leap year or not.

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

  if (((year % 4 == 0) && ((year % 100 != 0))|| (year % 400 == 0)) {
    printf("%d is a leap year.", year);

  else
    printf("%d is not a leap year.", year);

  return 0;
}
```

Output 1
Enter a year: 1900
1900 is not a leap year.

Output 2
Enter a year: 2012
2012 is a leap year.

3. Write a C program to test number entered is positive, negative or zero

```
#include<stdio.h>
int main()
{
   int num;
   printf("\n enter the number:")
   scanf("%d",&num);

   if(num == 0)
      printf("Neither positive nor negative");
   else if(num < 0)
      printf("Negative");
   else
      printf("Positive");

   return 0;
}
```

4. Write a program to check greatest of three numbers.

```
#include <stdio.h>
int main() {
  double n1, n2, n3;
  printf("Enter three numbers: ");
  scanf("%lf %lf %lf", &n1, &n2, &n3);

  // if n1 is greater than both n2 and n3, n1 is the largest
  if (n1 >= n2 && n1 >= n3)
    printf("%.2lf is the largest number.", n1);

  // if n2 is greater than both n1 and n3, n2 is the largest
  else if (n2 >= n1 && n2 >= n3)
    printf("%.2lf is the largest number.", n2);

  // if both above conditions are false, n3 is the largest
  else
```

```
   printf("%.2lf is the largest number.", n3);

  return 0;
}
```

5. Write a program to Compute Binomial co-efficient

```
#include<stdio.h>
#define MAX 10
main( )
{
int m, x, binom; printf(“ m x”);
for(m = 0; m <=10; ++m) printf(%4d”, m);
printf(“\n    \n”);
m = 0;
do
{
printf(“%2d”, m); x = 0;
binom = 1; while(x<=m)
{
if(m == 0 || x ==0)
printf(“%4d”, binom); else
{
binom = binom * (m-x + 1) / x; printf(“%4d”, binom);
}
x = x + 1;
}
printf(“\n”); m=m+1;
}
while (m<= MAX);
printf(“       \n”);
}
```

OUTPUT:

```
mx    0       1       2       3       4       5       6       7       8       9       10
-------------------------------------------------------------------------------------------

0       1
1       1       1
```

```
2   1   2   1
3   1   3   3   1
4   1   4   6   4   1
5   1   5   10  10  5   1
6   1   6   15  20  15  6   1
7   1   7   21  35  35  21  7   1
8   1   8   28  56  70  56  28  8   1
9   1   9   36  84  126 126 84  36  9   1
10  1   10  45  120 210 252 210 120 45  10  1
```

---

6. Write a program to calculate roots of quadratic equation.

```
#include<stdio.h>
#include<math.h>
void main()
{
float a, b, c, root1, root2, rpart, ipart, disc;
printf("Enter the 3 coefficients:\n”);
scanf(%f%f%f", &a, &b, &c);
if((a*b*c) = = 0)
{
printf("Roots cannot be Determined:\n");
exit(0);
}
disc=(b*b) - (4*a*c);
if(disc == 0)
{
printf("Roots are equal\n");
root1= -b / (2*a);
root2=root1;
printf ("root1 = %f \n", root1);
printf ("root2 = %f \n", root2);
}
else if(disc>0)
{
printf("Roots are real and distinct\n");
root1= (-b + sqrt(disc)) / (2*a);
root2= (-b - sqrt(disc)) / (2*a);
printf ("root1 = %f \n", root1);
```

```
printf ("root2 = %f \n", root2);
}
else
{
printf("Roots are complex\n");
rpart = -b / (2*a);
ipart = (sqrt (-disc)) / (2*a);
printf("root1 = %f + i %f \n", rpart, ipart);
printf("root2 = %f - i %f \n", rpart, ipart);
}
}
```

Output First Run:

Enter the 3 coefficients: 1
2
1
Roots are equal Root1= -1.000000
Root2= -1.000000

Second Run:

Enter the 3 coefficients: 2
3
2
Roots are real and equal Root1=-0.500000 Root2=-2.000000

Third Run:

Enter the 3 coefficients:
2
2
2
Roots are complex
Root1=-0.500000+i 0.866025
Root2=-0.500000- i 0.866025

7. Write a C program to check whether a character is VOWEL or CONSONANT using switch.

```
#include <stdio.h>
int main()
{
   char ch;
   printf("Enter a character: ");
```

```
   scanf("%c",&ch);
   if((ch>='A' && ch<='Z') || (ch>='a' && ch<='z'))
   {
      switch(ch)
      {
         case 'A':
         case 'E':
         case 'I':
         case 'O':
         case 'U':
         case 'a':
         case 'e':
         case 'i':
         case 'o':
         case 'u':
            printf("%c is a VOWEL.\n",ch);
            break;
         default:
            printf("%c is a CONSONANT.\n",ch);
      }
   }
   else
   {
      printf("%c is not an alphabet.\n",ch);
   }

   return 0;
}
```

Output

First Run:

```
   Enter a character: E
   E is a VOWEL.
```

Second Run:

```
   Enter a character: X
   X is a CONSONANT.
```

Third Run:

```
Enter a character: +
+ is not an alphabet.
```

8. Write a c program to print 20 horizontal asterisks.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int i=1;
  while(i<=100)
   {
     printf("*");
     i++;
   }
 return 0;
}
```

Output:

```
********************
```

9. Write a C program to calculate and prints the squares and cubes of the numbers from 0 to 20 using for loop.

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

   /* Print column names */
   printf("Number\tSquare\tCube\n");
   printf("========================\n");

   for(x=0; x<=20; x++)
      printf("%d\t%d\t%d\n", x, x*x, x*x*x);

   return 0;
}
```

Output

```
Number      Square       Cube
========================
0                  0            0
```

```
1       1       1
2       4       8
3       9       27
4       16      64
5       25      125
6       36      216
7       49      343
8       64      512
9       81      729
10      100     1000
11      121     1331
12      144     1728
13      169     2197
14      196     2744
15      225     3375
16      256     4096
17      289     4913
18      324     5832
19      361     6859
20      400     8000
```

10. **Write a C program to Print Pyramids and Patterns**

```
#include <stdio.h>
int main() {
  int i, j, rows;
  printf("Enter the number of rows: ");
  scanf("%d", &rows);
  for (i = 1; i <= rows; ++i) {
    for (j = 1; j <= i; ++j) {
      printf("* ");
    }
    printf("\n");
  }
  return 0;
}
```

Output

```
Half Pyramid of *
*
```

```
* *
* * *
* * * *
* * * * *
```

11. Write a C program to print Pascal's Triangle

```
#include <stdio.h>
int main() {
  int rows, coef = 1, space, i, j;
  printf("Enter the number of rows: ");
  scanf("%d", &rows);
  for (i = 0; i < rows; i++) {
    for (space = 1; space <= rows - i; space++)
      printf("  ");
    for (j = 0; j <= i; j++) {
      if (j == 0 || i == 0)
        coef = 1;
      else
        coef = coef * (i - j + 1) / j;
      printf("%4d", coef);
    }
    printf("\n");
  }
  return 0;
}
```

Output

```
         1
       1   1
     1   2   1
   1   3   3   1
 1  4    6   4   1
1  5   10   10  5   1
```

12. Write a C Program to Check Prime Number

```
#include <stdio.h>
int main()
{
 int n, i, flag = 0;
 printf("Enter a positive integer: ");
```

```
  scanf("%d", &n);
  if (n == 0 || n == 1)
    flag = 1;
  for (i = 2; i <= n / 2; ++i) {
    if (n % i == 0)
   {
      flag = 1;
      break;
    }
  }
    if (flag == 0)
    printf("%d is a prime number.", n);
  else
    printf("%d is not a prime number.", n);
  return 0;
}
```

Output
Enter a positive integer: 29
29 is a prime number.

13. Write a C Program to Check Palindrome.

```
#include <stdio.h>
int main()
 {
  int n, reversed = 0, remainder, original;
    printf("Enter an integer: ");
    scanf("%d", &n);
    original = n;
    while (n != 0)
       {
        remainder = n % 10;
        reversed = reversed * 10 + remainder;
        n /= 10;
        }

    if (original == reversed)
        printf("%d is a palindrome.", original);
    else
```

```
      printf("%d is not a palindrome.", original);

   return 0;
}
```

Output
Enter an integer: 1001
1001 is a palindrome.

Enter an integer: 11010
01011 is a palindrome.

14. Write a C Program to Check Armstrong Number of n digits.

```
#include <math.h>
#include <stdio.h>
int main()
{
  int num, originalNum, remainder, n = 0;
  float result = 0.0;

  printf("Enter an integer: ");
  scanf("%d", &num);

  originalNum = num;

  for (originalNum = num; originalNum != 0; ++n)
  {
    originalNum /= 10;
  }

  for (originalNum = num; originalNum != 0; originalNum /= 10)
  {
    remainder = originalNum % 10;
    result += pow(remainder, n);
   }

  if ((int)result == num)
   printf("%d is an Armstrong number.", num);
  else
```

```
   printf("%d is not an Armstrong number.", num);
  return 0;
}
```

Output
Enter an integer: 1634
1634 is an Armstrong number.

Enter an integer: 154
154 is not an Armstrong number.

15. Example c program for break statement.

```
#include<stdio.h>
#include<stdlib.h>
void main ()
{
   int i;
   for(i = 0; i<10; i++)
   {
      printf("%d ",i);
      if(i == 5)
      break;
   }
   printf("came outside of loop i = %d",i);

}
```

Output
0 1 2 3 4 5 came outside of loop i = 5