CHAPTER-6
Looping Statements

This Chapter Includes:

- For loop
- While loop
- do while loop
- Difference between while and do while loop
- Nested loops
- Unconditional branching statements(Jumps in loops)
- Additional Programs

Looping statements:

- Looping statements are used to execute one or more statement repeatedly several number of times.
- The purpose of the loop is to repeat the same code a number of times.
- Looping statements are also known as iterative or repetitive statement.
- In C programming language there are three types of loops; while, for and do-while.

Use of Loop:

When you need to execute a block of code several number of times then you need to use looping concept in C language.

Advantage with looping statement:

- Reduce length of Code
- Take less memory space.
- Burden on the developer is reducing.
- Time consuming process to execute the program is reduced.

Difference between conditional and looping statements:

Conditional statement executes only once in the program where as looping statements executes repeatedly several number of time.

Types of Loops:

There are three type of Loops available in 'C' programming language.

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

1. while loop:

- A **while** loop in C programming repeatedly executes a target statement as long as a given condition is true.
- In While Loop First check the condition if condition is true then control goes inside the loop body otherwise goes outside the body. **while loop** will be repeats in clock wise direction.

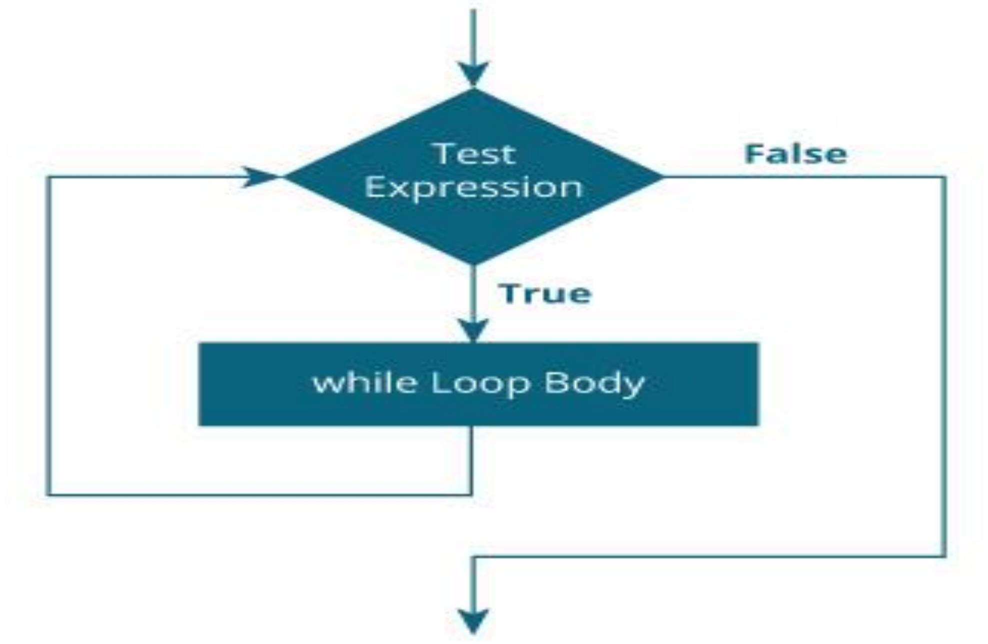
Syntax:

```
initialization;
while(condition)
{
  statements;
  increment/decrement;
}
```

How while loop works?

- The while loop evaluates the test expression (condition) inside the parenthesis ().
- If the test expression is true, statements inside the body of while loop are executed. Then, the test expression is evaluated again.
- The process goes on until the test expression is evaluated to false.
- If the test expression is false, the loop terminates (ends).

Flowchart of while loop:



Program to display your name 5 times using while loop:

```
 #include<stdio.h>
 int main()
 {
 int i=1;
 while(i<=5)
 {
 printf(" NEHA \n");
 i++;
 }
 return 0;
}
```

Output:

```
NEHA
NEHA
NEHA
```

```
NEHA
NEHA
```

Program to display 1 to 10 using while loop

```
#include<stdio.h>
int main()
{
int1 i=1;
while(i<=10)
{
printf("%d \n",i);
i++;
}
return 0;
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Program to display 10 to 1 using while loop

```
#include<stdio.h>
int main()
{
int1 i=10 ;
while(i>=1)
{
printf("%d \n",i);
i--;
}
return 0;
}
```

Output:

```
10
9
8
```

```
7
6
5
4
3
2
1
```

Program to print table for the given number using while loop in C

```
#include<stdio.h>
int main()
{
int i=1,number,b=9;
printf("Enter a number: ");
scanf("%d",&number);
while(i<=10)
{
printf("%d \n",(number*i));
i++;
}
return 0;
}
```

Output:

```
Enter a number: 5
5
10
15
20
25
30
35
40
45
50
```

Example of while loop

```
#include<stdio.h>
int main()
{
  int count=1;
  while (count <= 4)
  {
       printf("%d ", count);
       count++;
  }
  return 0;}
```

Output:

```
1 2 3 4
```

Explanation of above code:

- The variable count is initialized with value 1 and then it has been tested for the condition.
- If the condition returns true then the statements inside the body of while loop are executed else control comes out of the loop.
- The value of count is incremented using ++ operator then it has been tested again for the loop condition.

Infinite while loop:

A loop is said to be infinite when it executes repeatedly and never stops. This usually happens by mistake, when you set the condition in loop in such a way that it never returns false, then it becomes infinite loop.

Programs which can cause infinite iteration of statements:

```
#include<stdio.h>
int main()
{
  int n=1;
  while(n<10)
  {
     printf("%d ",n);
   }
return 0;
}
```

Output:

Above program will execute infinite times, because condition will never get false.

```
#include<stdio.h>
int main()
{
  int n=1;
  while(1)
  {
     printf("hello ");
   }
return 0;
}
```

Output:

Above program will execute infinite times, because condition will never get false

---

Program to display nothing on screen using loop.

When you set the condition in loop in such a way that the condition gets false for the first time only, then no output will be displayed on screen.

```
#include<stdio.h>
int main()
{
  int n=1;
  while(0)
  {
      printf("hello ");
      n++;
   }
return 0;
}
```

Output:
No Output

```
#include<stdio.h>
int main()
{
  int n=10;
  while(n<=5)
  {
      printf("hello ");
      n++;
   }
return 0;
}
```

Output:
No Output

---

2. For Loop:

- A set of statement may have to be repeatedly executes for a specified number of times.
- If we know well in advance as how many times a set of statements has to be executed repeatedly, then for loop is the best choice.

Syntax:

```
for(initialization ; condition ; increment/decrement)
```

Program to display your name 10 times using for loop

```
#include<stdio.h>
int main()
{
  int n;
  for(n=1;n<=10;n++)
  {
        printf("John \n");
  }
return 0;
}
```

Output:

```
John
John
```

```
John
John
John
John
John
John
John
John
```

Program to display 1 to 10 using for loop

```
#include<stdio.h>
int main()
{
  int n;
  for(n=1;n<=10;n++)
  {
        printf("%d ",n);
  }
return 0;
}
```

Output:

```
1 2 3 4 5 6 7 8 9 10
```

Program to display even number between 1 to 20

```
#include<stdio.h>
int main()
{
  int n;
  for(n=1;n<=20;n++)
  {
             if(n%2==0)
       {
        printf("%d ",n);
       }
  }
return 0;
}
```

Output:

```
2 4 6 8 10 12 14 16 18 20
```

Program to display alphabets A to Z using for loop.

```
#include<stdio.h>
int main()
{
  int n;
  for(n=65;n<=90;n++)
  {
    printf("%c ",n);
  }
return 0;
}
```

Output:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

Infinite for loop:

- A loop becomes an infinite loop if a condition never becomes false.The for loop is traditionally used for this purpose.
- Since none of the three expressions that form the 'for' loop are required, you can make an endless loop by leaving the conditional expression empty.

```
#include<stdio.h>
int main()
{
  for(; ;)
  {
   printf(" HI ");
  }
return 0;
}
```

Output:

Above program will execute infinite times.

```
   ================================================
#include<stdio.h>
int main()
{
  for(; 1 ;)
  {
   printf(" HI ");
  }
return 0;
}
```

Output:

Above program will execute infinite times.

```
   ================================================
#include<stdio.h>
int main()
{
  int n;
  for(n=1; ;)
  {
  printf(" HI ");
  }
return 0;
}
```

Output:

```
Above program will execute infinite times.
```

3. do..while Loop:

- A do...while loop in C is similar to the while loop except that the condition is always executed after the body of a loop. It is also called an exit-controlled loop.
- In the do-while loop, the body of a loop is always executed at least once. After the body is executed, then it checks the condition. If the condition is true, then it will again execute the body of a loop otherwise control is transferred out of the loop.

Syntax:

```
do
{
  statements;
}while (expression);
```

Program to display 1 to 10 using do..while loop.

```
#include<stdio.h>
int main()
{
  int n=1;
  do
  {
     printf("%d ",n);
      n++;
  }while(n<=10);
return 0;
}
```

Output:

```
1 2 3 4 5 6 7 8 9 10
```

Program to execute statement even if the condition is false.

```
#include<stdio.h>
int main()
{
  int n=1;
  do
  {
     printf("Hello ");
     n++;
  }while(n<=1);
return 0;
}
```

Output:

```
Hello
```

Depending upon the position of a control statement in a program, looping in C is classified into two types:

1. Entry controlled loop
2. Exit controlled loop

1. Entry controlled loop:

- In an entry controlled loop, a condition is checked before executing the body of a loop. It is also called as a pre-checking loop.
- while and for loop are entry controlled loop

2. Exit controlled loop:

- In an exit controlled loop, a condition is checked after executing the body of a loop. It is also called as a post-checking loop.
- do while is exit controlled loop

The control conditions must be well defined and specified otherwise the loop will execute an infinite number of times. The loop that does not stop executing and processes the statements number of times is called as an **infinite loop**. An infinite loop is also called as an "**Endless loop**." Following are some characteristics of an infinite loop:

1. No termination condition is specified.
2. The specified conditions never meet.

Unconditional Branching statements:

- In c, there are control statements that do not need any condition to control the program execution flow. These control statements are called as unconditional control statements.
- Unconditional control statements are also known as **Jumping Statements.**
- **Jump Statement** makes the control jump to another section of the program unconditionally when encountered. Jump statements are used to interrupt the normal flow of program.
- C programming language provides the following unconditional control statements.

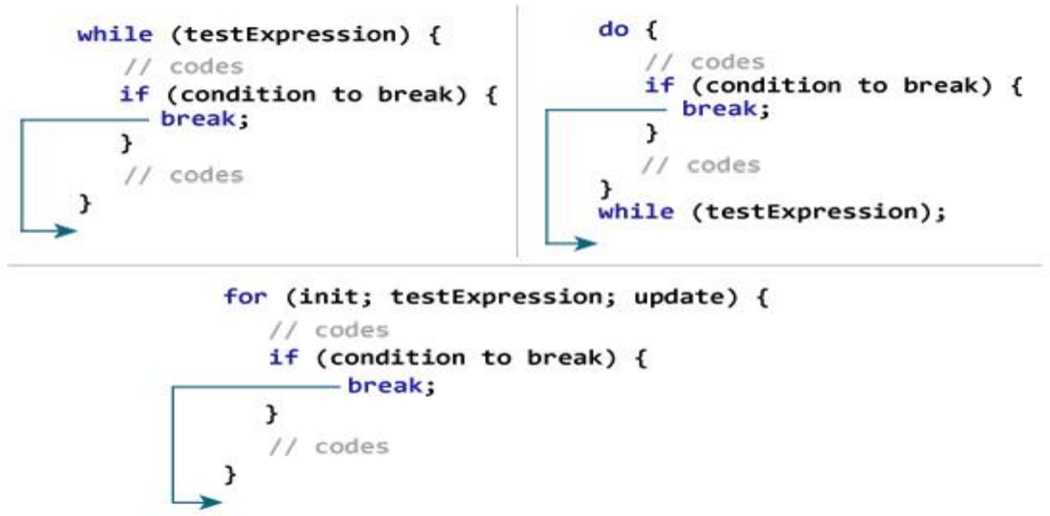
a) break
b) continue
c) goto
d) return

a) break:

In C, the break statement is used to perform the following two things...

- A **break** statement is used to terminate the execution of the rest of the block where it is present and takes the control out of the block.
- It is mostly used in loops and switch-case to bypass the rest of the statement and take the control to the end of the loop.
- When a break statement is encountered inside the switch case statement, the execution control moves out of the switch statement directly.
- When the break statement is encountered inside the looping statement, the execution control moves out of the looping statements.
- Another point to be taken into consideration is that **break** statement when used in nested loops only terminates the inner loop where it is used and not any of the outer loops.

Working of break in loops:



```
while (testExpression) {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
```

```
do {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
```

Program for break statement:

```
#include <stdio.h>
 int main ()
{
   int a = 10;
   while( a < 20 )
 {

      printf("value of a: %d\n", a);
      a++;

      if( a > 15)
      {
         break;
      }
   }
   return 0;
}
```

Output:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
```

Program to take input from user 10 times but if user enters 0 then break the loop.

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

```
    for(n=1;n<=10;n++)
   {
       printf("Enter the value\n");
       scanf("%d",&a);

       if( a==0)
       {
          printf("You have entered zero");
          break;
       }
    }
    return 0;
 }
```

Output:

```
Enter the value
2
Enter the value
3
Enter the value
0
You have entered zero
```

Break statements can also execute infinite loops.

Program:

```
#include<stdio.h>
 int main ()
{
   int n,a;
   for(;;)
   {
       printf("hello");
       break;
    }
   return 0;
}
```

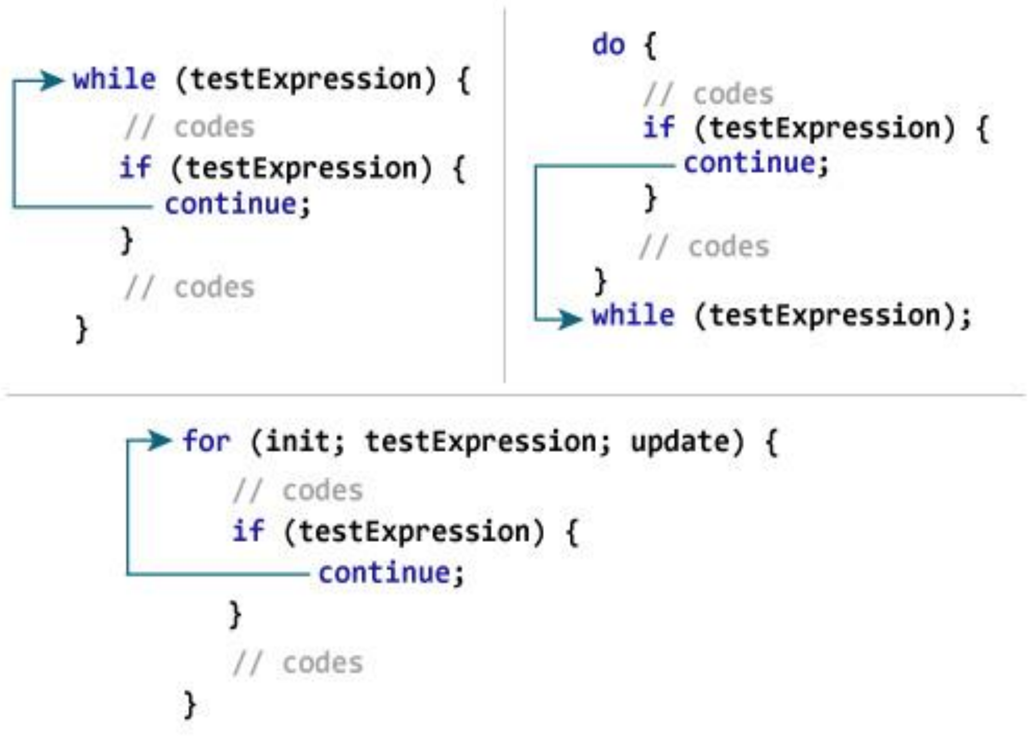
Output:

```
hello
```

b. continue:

- The continue statement skips the current iteration of the loop and continues with the next iteration.
- continue statements are only used in loops.

Working of continue in loops:



```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

Example: continue statement inside for loop

```
#include <stdio.h>
int main()
{
  int j;
   for ( j=0; j<=8; j++)
   {
      if (j==4)
      {
            continue;
       }
       printf("%d ", j);
   }
   return 0;
}
```

Output:

```
0 1 2 3 5 6 7 8
```

Explanation:

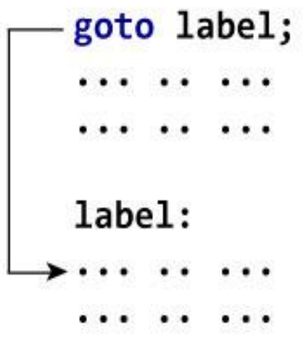
Value 4 is missing in the output, why? When the value of variable j is 4, the program encountered a continue statement, which makes the control to jump at the beginning of the for loop for next iteration, skipping the statements for current iteration (that’s the reason printf didn’t execute when j is equal to 4).

Difference between break and continue

break	continue
When break is executed the statements following break are skipped and causes the loop to be terminated.	When continue statement is executed the statement following continue are skipped and cause the loop to be continued with the next iteration
It can be used in switch and loops	It is only used in loops

E.g.
for(i=1;i<=5;i++)
{
  if(i==2)
   break;
   printf("%d",i);
   return 0;
}
Output:
1 2
	E.g.
	for(i=1;i<=5;i++)
	{
	  if(i==3)
	   continue;
	   printf("%d",i);
	   return 0;
	}
	Output:
	1 2 4 5

c. goto:
- The goto statement allows us to transfer control of the program to the specified label.
- The label is an identifier. When the goto statement is encountered, the control of the program jumps to label: and starts executing the code.



```
goto label;
... .. ...
... .. ...

label:
... .. ...
... .. ...
```

Program for goto:

```
 #include <stdio.h>
 int main ()
{
  start:
         printf("Hello ");
          goto start;
  return 0;
}
```

Output:
Infinite loop

Explanation:
The above program in go infinite loop.

Program to display 5 times hello using goto statement.

```
  #include <stdio.h>
   int main ()
  {
          int n=1;
       start:
               printf("Hello ");
               n++;
                if(n<=5)
                {
                 goto start;
                        }
    return 0;
  }
```

Output:

```
Hello Hello Hello Hello Hello
```

Program to display 1 to 10 using goto statement.

```
  #include <stdio.h>
   int main ()
  {
          int n=1;
     start:
               printf("%d ",n);
               n++;
                if(n<=10)
                {
                 goto start;
                   }
    return 0;
  }
```

Output:

```
1 2 3 4 5 6 7 8 9 10
```

Program to display multiplication table using goto statement.

```
#include <stdio.h>
 int main ()
{
          int n=1,num;
          printf("Enter the number\n");
          scanf("%d",&num);
          printf("\n============\n");
       start:
               printf("%d \n",num*n);
               n++;
                if(n<=10)
                {
```

```
                goto start;
                }
   return 0;
}
```

Output:

```
Enter the number
2
=============
2
4
6
8
10
12
14
16
18
20
```

NESTED LOOPS:

- Loop inside another loop is called nesting of loop.
- Nested loops are basically used when we want to generate output in the form row and column i.e. in tabular form.

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

1. nested for loop:
Syntax:
The syntax for a nested for loop statement in C is as follows −

```
  for(init; condition; increment) ⇒ outer loop
  {
    for(init; condition; increment ) ⇒ inner loop
   {
       statements of inner loop;
     }
       statements of outer loop;
  }
```

Program:

```
#include<stdio.h>
int main()
{
  int i,j;
  int n=1;
  for(i=1;i<=5;i++)
  {
```

```
            for(j=1;j<=4;j++)
            {
                        printf("%d ",i);
            }
            printf("\n");
   }
return 0;
}
```

Output:

```
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4
5 5 5 5
```

Note: outer loop is always used for rows and inner loop is used for columns.

2. nested while loop:

while(condition) $\Longrightarrow$ **outer loop**

{

   while(condition) $\Longrightarrow$ **inner loop**

{

```
                 statements of inner loop;
   }                 statements of outer loop;

  }
```

Program:

```
#include<stdio.h>
int main()
{
  int i=1,j;
  while(i<=4)
  {
            j=1;
            while(j<=3)
            {
               printf("%d ",j);
                        j++;
            }
            i++;
            printf("\n");
     }
           return 0;
    }
```

Output:

```
1 2 3
```

```
1 2 3
1 2 3
1 2 3
```

3. nested do while loop:

Syntax:

```
 do
 {
   statement;
   do
  {
     Statements of inner loop;
   }while( condition );

    statements of outer loop;

  } while(condition);
```

Program:

```
  #include<stdio.h>
  int main()
  {
    int i=1,j;
    do
    {
            j=1;
           do
           {
              printf("%2d ",i*j);
                 j++;
           }while(j<=10);

          i++;
          printf("\n");
    }while(i<=4);
  return 0;
  }
```

Output:

```
 1  2  3  4  5   6    7    8   9    10
 2  4  6  8  10 12 14 16 18  20
 3  6  9 12 15 18 21 24 27  30
 4  8 12 16 20 24 28 32 36 40
```

Note: There is no rule that a loop must be nested inside its own type. In fact, there can be any type of loop nested inside any type and to any level. For e.g. in for loop we can write while loop.

Patterns using loop:

1. **Program to display Half Pyramid of ***

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

```
#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:

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

2. Half Pyramid of Numbers

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
   #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("%d ", j);
       }
```

```
      printf("\n");
    }
    return 0;
}
```

Output:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

4. Inverted half pyramid of *

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

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

Output:

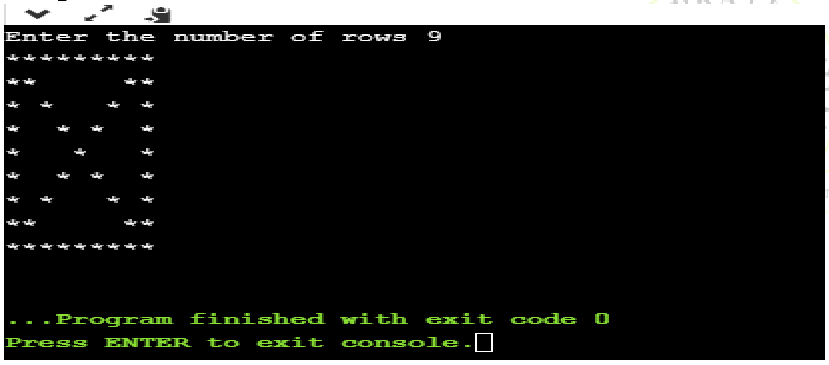
```
* * * * *
* * * *
* * *
* *
*
```

3.Square Pattern with Diagonal

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

```
    printf("Enter the number of rows");
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {
      for(int j=1;j<=n;j++)
      {
        if(i==1 ||i==n||j==1||j==n-i+1||i==j||j==n)
        {
        printf("*");
        }
        else
        {
              printf(" ");
            }
        }
      printf("\n");
    }
    return 0;
  }
```

Output:



```
Enter the number of rows 9
*********
**     **
* *   * *
*  * *  *
*   *   *
*  * *  *
* *   * *
**     **
*********


...Program finished with exit code 0
Press ENTER to exit console.
```

Additional Programs Using Loops and Jumping Statements:

1. Program to calculate sum of natural numbers using while loop

```
  #include <stdio.h>
  int main()
  {
    int n, i, sum = 0;

    printf("Enter a number: ");
    scanf("%d", &n);

    for (i = 1; i <= n; i++)
   {
      sum=sum+i;
    }
    printf("Sum = %d", sum);
    return 0;
}
```

Output:

```
Enter a number: 5
Sum=15
```

2. Program to check whether given number is palindrome or not.

steps to check palindrome:

- Get the number from user
- Hold the number in temporary variable
- Reverse the number
- Compare the temporary number with reversed number
- If both numbers are same, print palindrome number
- Else print not palindrome number

```
#include<stdio.h>
int main()
{
int n,r,sum=0,temp;
printf("Enter the number=");
scanf("%d",&n);
temp=n;
while(n!=0)
{
  r=n%10;
 sum=(sum*10)+r;
 n=n/10;
}
if(temp==sum)
printf("%d is a palindrome number",temp);
else
printf("%d is not a palindrome number",temp);
return 0;  return 0;
}
```

Output 1:

```
Enter the number=453
453 is not a palindrome number
```

Output 2:

```
Enter the number=121
121 is a palindrome number
```

3. Program to display Fibonacci series.

```
 #include <stdio.h>
  int main()
 {
  int i, n, a = 0, b = 1, c;
  printf("Enter the number of terms: ");
  scanf("%d", &n);
  printf("Fibonacci Series: ");

  for (i = 1; i <= n; i++)
 {
      printf("%d ", a);
      c = a + b;
      a = b;
      b = c;
   }
  return 0;
}
```

Output:

```
Enter the number of terms: 6
Fibonacci Series: 0 1 1 2 3 5
```

4. Program to calculate factorial of a given number.

```
  #include<stdio.h>
  int main()
  {
int i,fact=1,number;
printf("Enter a number: ");
 scanf("%d",&number);
   for(i=1;i<=number;i++)
  {
     fact=fact*i;
   }
 printf("Factorial of %d is: %d",number,fact);
return 0;
}
```

Output:

```
Enter a number: 5
```

```
Factorial of 5 is: 120
```

5. Program to check for Armstrong number.

Armstrong number is a number that is equal to the sum of cubes of its digits. For example 0, 1, 153, 370, 371 and 407 are the Armstrong numbers.

Why 153 is an Armstrong number?

153 = (1*1*1)+(5*5*5)+(3*3*3)

where:

(1*1*1)=1

(5*5*5)=125

(3*3*3)=27

So:

1+125+27=153

```
#include<stdio.h>
 int main()
{
int n,r,sum=0,temp;
printf("enter the number=");
scanf("%d",&n);
temp=n;
while(n>0)
{
r=n%10;
sum=sum+(r*r*r);
n=n/10;
}
if(temp==sum)
printf("armstrong  number ");
else
printf("not armstrong number");
return 0;
}
```

Output 1:

```
enter the number=371
armstrong  number
```

Output 2:

```
enter the number=121
not armstrong number
```

6. Program to count number of digits.

```
#include <stdio.h>
```

```
int main()
{
  int n;  // variable declaration
  int count=0;   // variable declaration
  printf("Enter a number");
  scanf("%d",&n);
  while(n!=0)
  {
     n=n/10;
     count++;
  }
  printf("\nThe number of digits in an integer is : %d",count);
  return 0;
}
```

Output:

```
Enter a number
23456
The number of digits in an integer is :5
```

7. Program to count frequency of vowels available in a string.

```
#include <stdio.h>
int main()
{
   char name[]="how are you";
   int i,count=0;

   for(i=0;name[i]!='\0';i++)
   {
         if(name[i]=='a'||name[i]=='e'||name[i]=='i'||name[i]=='o'||name[i]=='u'
          ||name[i]=='A'||name[i]=='E'||name[i]=='I'||name[i]=='O'||name[i]=='U')
         {

            count++;
         }
     }
         printf("No of vowel in \"%s\"  is: %d",name,count);
   return 0;
}
```

Output:

```
No of vowels in "how are you"  is: 5
```

****