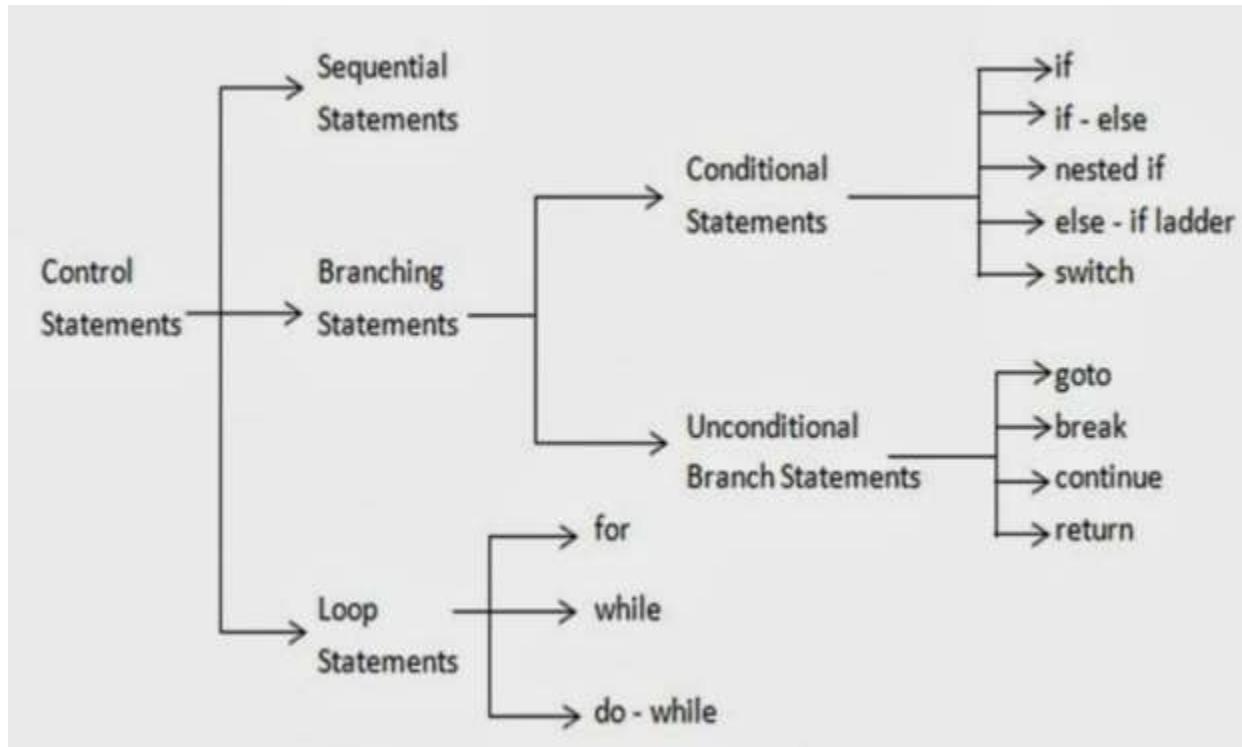


## Module 2

### Branching and Looping

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 Statement:

Conditional Statements in C programming are used to make decisions based on the conditions.

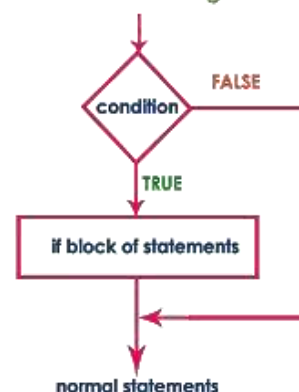
#### Simple-if:

When we need to execute a block of statements only when a given condition is true then we use if statement. It is one-way selection statement.

##### Syntax

```
if ( condition )
{
    ....
    block of statements;
    ....
}
```

##### Execution flow diagram



Example:

```
#include<stdio.h>

void main ()
{
    int num1=1;

    int num2=2;

    if(num1<num2)           //test-condition
    {
        printf ("num1 is smaller than num2");
    }
}
```

**Output:**

num1 is smaller than num2

if-else statement:

The if-else statement in C is used to perform the operations based on some specific condition. The operations specified in if block is executed if and only if the given condition is true. If the condition is false, it will execute else (false) block. It is two-way selection statement.

Syntax:

```
if (test-expression)
{
    True block of statements
}
Else
{
    False block of statements
}
Statements;
```

Flowchart:



Example:

```
#include<stdio.h>

void main()
{
    int num=19;

    if(num<10)
```

```
{  
    printf("The value is less than 10");  
}  
else  
{  
    printf("The value is greater than 10");  
}  
}
```

**Output:**

The value is greater than 10

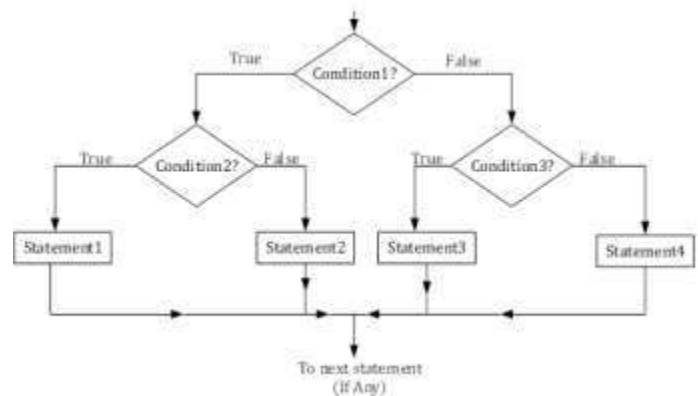
### Nested-if:

Nested if statements mean's an if statement inside another if statement. It is a multi-way selection statement.

Syntax:

```
if(Condition/Expression)  
{  
    if(Condition/Expression)  
    {  
        Statement(s);  
    }  
    else  
    {  
        Statement(s);  
    }  
}  
else  
{  
    if(Condition/Expression)  
    {  
        Statement(s);  
    }  
    else  
    {  
        Statement(s);  
    }  
}
```

Flowchart:



## C Programming for Problem Solving – 21CPS13/23

### Example:

An Example will be good to illustrate the working concept of Nested if statement. Let's take an example and understand. Every person is eligible for working once he or she is above 18 years otherwise not eligible. Moreover, any organization will offer a job if he or she is above 18 years otherwise no job is guaranteed it means the condition then and there becomes false. Therefore, we will make use of another nested if statement to check the retirement age to satisfy with this.

```
#include <stdio.h>
void main()
{
    int a;
    printf(" Enter your current Age Here:\n");
    scanf("%d",&a);
    if ( a < 18 )
    {
        printf("Consider as minor \n");
        printf("Not fit for Working");
    }
    else
    {
        if (a >= 18 && a <= 60 )
        {
            printf("He/She is successfully eligible for Working\n");
            printf("Fill all the details and apply for it\n");
        }
        else
        {
            printf("Age is not satisfactory according to the organization norms\n");
            printf("Ready for retirement and can collect pension \n");
        }
    }
}
```

Output

```
/tmp/dSPubwHGIV.o
Enter your current Age Here:
18
He/She is successfully eligible for Working
Fill all the details and apply for it
```

Output

```
/tmp/dSPubwHGIV.o
Enter your current Age Here:
12
Consider as minor
Not fit for Working
```

Output Clear

```
/tmp/dSPubwHGIV.o
Enter your current Age Here:
63
Age is not satisfactory according to the organization norms
Ready for retirement and can collect pension
```

## C Programming for Problem Solving – 21CPS13/23

Program 2: Write a C program to find the largest of 3 numbers using nested if statement.

```
#include<stdio.h>

Void main ()
{
    int num1, num2, num3;
    printf("Enter three numbers:\n");
    scanf("%d%d%d",&num1, &num2, &num3);
    if(num1>num2)
    {
        if(num1>num3)
        {
            printf("Largest = %d", num1);
        }
    }
    else
    {
        printf("Largest = %d", num3);
    }
}
else
{
    if(num2>num3)
    {
        printf("Largest = %d", num2);
    }
    else
    {
        printf("Largest = %d", num3);
    }
}
}
```

### Output

```
/tmp/d5PmWVG1V.o
Enter three numbers:
1 2 3
Largest = 3
```

### Output

```
/tmp/d5PmWVG1V.o
Enter three numbers:
4 3 2
Largest = 4
```

### Output

```
/tmp/d5PmWVG1V.o
Enter three numbers:
1 5 2
Largest = 5
```

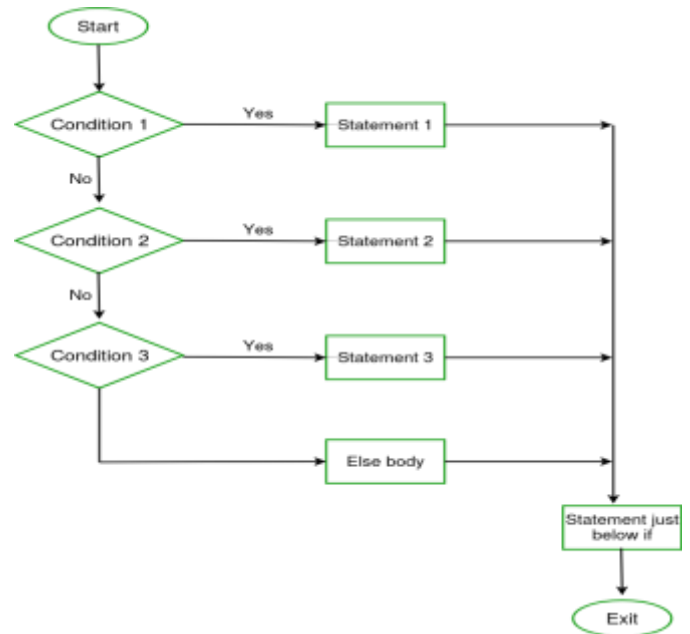
### else-if ladder/cascade-if:

It is another multi-way selection statement. If the condition1 is true, “statement1” is executed, if the condition1 is false the control will move to condition2, if condition2 is true “statement2” is executed, if the condition2 is false then condition3 is checked, and so on until condition is satisfied and statement is executed.

Syntax: if(condition1)

```
{  
    statement1;  
}  
else if (condition2)  
{  
    statement2;  
}  
else  
{  
    statement n;  
}
```

### Flowchart:



Example:

Program 1: C program which displays the grade of student using else if ladder.

```
#include<stdio.h>
void main(){
    int marks;
    printf("Enter the marks of a student:\n");
    scanf("%d",&marks);
    if(marks <=100 && marks >= 90)
        printf("Grade=A");
    else if(marks < 90 && marks>= 80)
        printf("Grade=B");
    else if(marks < 80 && marks >= 70)
        printf("Grade=C");
    else if(marks < 70 && marks >= 60)
        printf("Grade=D");
    else if(marks < 60 && marks > 50)
        printf("Grade=E");
    else if(marks == 50)
        printf("Grade=F");
    else if(marks < 50 && marks >= 0)
        printf("Fail");
    else
        printf("Enter a valid score between 0 and 100");
}
```

#### Output

```
/tmp/NSsr0Pe33W.o
Enter the marks of a student:
98
Grade=A
```

#### Output

```
/tmp/NSsr0Pe33W.o
Enter the marks of a student:
-2
Enter a valid score between 0 and 100
```

#### Output

```
/tmp/NSsr0Pe33W.o
Enter the marks of a student:
65
Grade=D
```

Program 2: C program to print weekday based on given number.

```
#include<stdio.h>
void main()
{
    int day;
    printf("Enter day number: ");
    scanf("%d", &day);
    if(day==1)
    {
        printf("SUNDAY.");
    }
    else if(day==2)
    {
        printf("MONDAY.");
    }
    else if(day==3)
    {
        printf("TUESDAY.");
    }
    else if(day==4)
    {
        printf("WEDNESDAY.");
    }
    else if(day==5)
    {
        printf("THURSDAY.");
    }
    else if(day==6)
    {
        printf("FRIDAY.");
    }
    else if(day==7)
    {
        printf("SATURDAY.");
    }
    else
    {
        printf("INVALID DAY.");
    }
}
```

### Output

/tmp/NSsr0Pe33W.o

Enter day number: 3

TUESDAY.

### Output

/tmp/NSsr0Pe33W.o

Enter day number: 10

INVALID DAY.



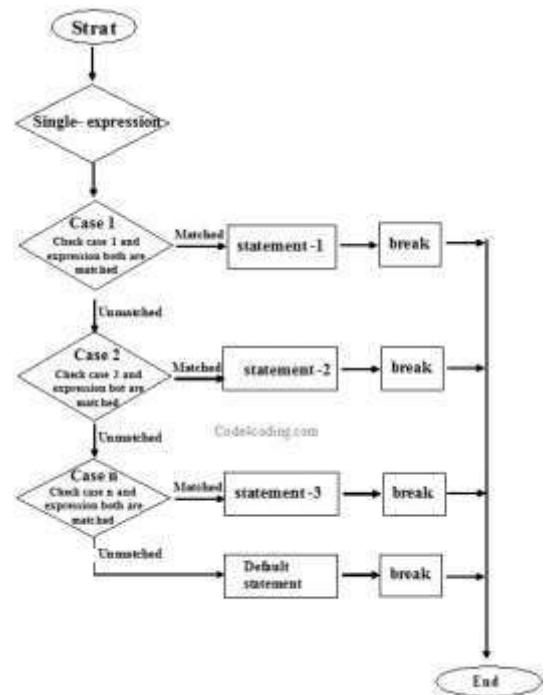
## switch statement:

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.

### Syntax:

```
switch(value/condition)
{
    case value: statement 1;
                break;
    case value: statement 2;
                break;
    case value: statement n;
                break;
    default case: statement:
                break;
}
```

### Flowchart:



Program 1: Develop a program to solve simple computational problems using arithmetic expressions and use of each operator leading to simulation of a commercial calculator.

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

```
void main()
```

```
{
    char Operator;
    float num1, num2, result;
    printf("\n Please Enter an Operator (+, -, *, /) : ");
    scanf("%c", &Operator);
    printf("\n Please Enter the Values for two Operands: num1 and num2 : ");
    scanf("%f%f", &num1, &num2);
    switch(Operator)
    {
```

```
        case '+':    result = num1 + num2;
                    break;

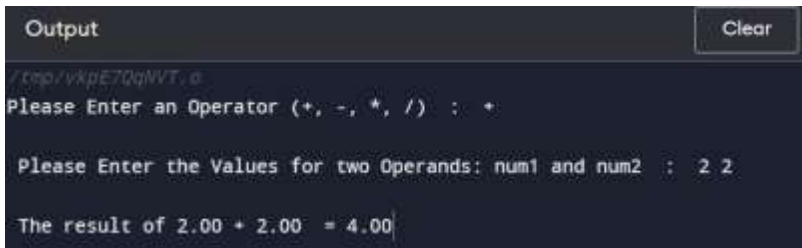
        case '-':    result = num1 - num2;
                    break;

        case '*':    result = num1 * num2;
                    break;

        case '/':    result = num1 / num2;
                    break;

        default: printf("\n You have entered an Invalid Operator ");
    }

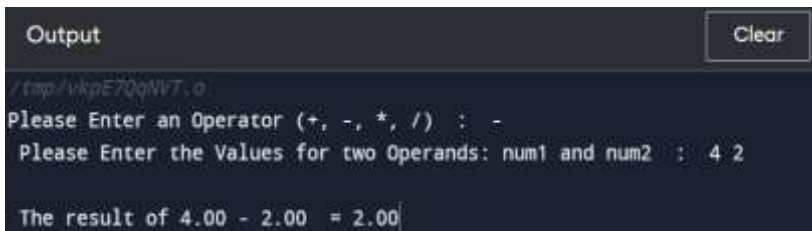
    printf("\n The result of %f %c %f = %f", num1, Operator, num2, result)
}
```



```
Output
/tmp/vkpE7QqNVT.o
Please Enter an Operator (+, -, *, /) : +

Please Enter the Values for two Operands: num1 and num2 : 2 2

The result of 2.00 + 2.00 = 4.00
```



```
Output
/tmp/vkpE7QqNVT.o
Please Enter an Operator (+, -, *, /) : -

Please Enter the Values for two Operands: num1 and num2 : 4 2

The result of 4.00 - 2.00 = 2.00
```

## Unconditional 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.

C programming language provides the following unconditional control statements.

- break
- continue
- goto
- return

## break statement:

A break statement terminates the execution of the loop and the control is transferred to the statement immediately following the loop. i.e., the break statement is used to terminate loops or to exit from a switch.

- It can be used within for, while, do-while, or switch statement.
- The break statement is written simply as break;
- In case of inner loops, it terminates the control of inner loop only.

Syntax:

Jump-statement;

break;

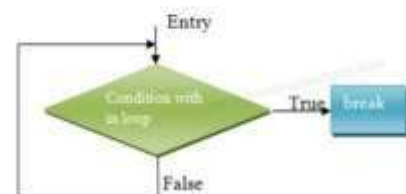
The jump statement in c break syntax can be while loop, do while loop, for loop or switch case.

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

```
void main()
```

```
{  
    int i=1;  
    for(i=1;i<=10;i++)  
    {  
        printf("%d ",i);  
        if(i==5)  
        {  
            break;  
        }  
    }  
}
```

Flowchart :



## Continue Statement:

- The continue statement is used to bypass the remainder of the current pass through a loop.
- The loop does not terminate when a continue statement is encountered. Instead, the remaining loop statements are skipped and the computation proceeds directly to the next pass through the loop.
- The continue statement can be included within a while, do-while, for statement.
- It is simply written as “continue”.

## C Programming for Problem Solving – 21CPS13/23

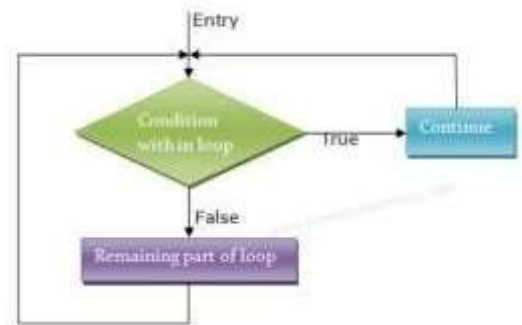
- The continue statement tells the compiler “Skip the following Statements and continue with the next Iteration”.
- In “while” and “do” loops continue causes the control to go directly to the test –condition and then to continue the iteration process.
- In the case of “for” loop, the updating section of the loop is executed before test-condition, is evaluated.

Syntax:

Jump-statement;

continue;

Flowchart :



```
#include<stdio.h>

void main( )
{
    int i=1, num, sum =0;
    for(i=0; i < 5; i ++ )
    {
        printf(“ Enter an integer:”);
        scanf( “%d”, &num);
        if(num < 0)
        {
            printf(“\n you have entered a negative number”);
            continue ;
        }
        sum += num ;
    }
    printf(“The sum of the Positive Integers Entered = % d \ n”,
sum);
}
```

Output

```
/tmp/9Dqv5PXy2W.o
Enter an integer:2
Enter an integer:3
Enter an integer:4
Enter an integer:-5

you have entered a negative number
Enter an integer:6
The sum of the Positive Integers Entered = 15
```

goto statement:

- C supports the “goto” statement to branch unconditionally from one point to another in the program.
- The goto requires a label in order to identify the place where the branch is to be made.
- A label is any valid variable name and must be followed by a colon (:).
- The label is placed immediately before the statement where the control is to be transferred.
- The label can be anywhere in the program either before or after the goto label statement.

Syntax :

```
goto    label;
.....
.....
.....
label:
statement;
```

In this syntax, label is an identifier. When, the control of program reaches to goto statement, the control of the program will jump to the label: and executes the code below it.

```
goto    label;
.....
.....
.....
label:
statement;
Forward Jump
```

```
label:
Statement;
.....
.....
.....
goto    label;
Backward Jump
```

## Looping Statements:

Loops are control structures used to repeat a given section of code a certain number of times or until a particular condition is met.

Three types of looping statements:

- while loop
- do-while loop
- for loop

**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.

The basic format of while loop statement is:

## C Programming for Problem Solving – 21CPS13/23

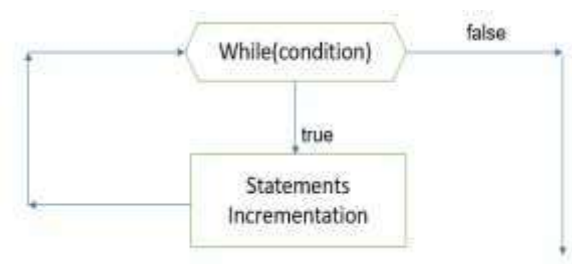
while (condition)

{

statement(s);

Incrementation;

}



Examples:

Program 1: Develop a program to demonstrate while loop.

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

```
void main ()
```

```
{
    int n = 1,times=5;
    while( n <= times )
    {
        printf("C while loops: %d\n", n);
        n++;
    }
}
```

Output

```
/tmp/SB0LZ9bc1a.o
C while loops: 1
C while loops: 2
C while loops: 3
C while loops: 4
C while loops: 5
```

Program 2: Develop a program to display numbers between 10 and 20

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

```
void main ()
```

```
{
    int a = 10;
    while( a < 20 ) {
        printf("value of a: %d\n", a);
        a++;
    }
}
```

Output

```
/tmp/SB0LZ9bc1a.o
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```


Program 3: develop a program to display numbers parallelly

Mr. Suhas A Bhyratae, Dept. of CSE, SCEM, Mangalore

## C Programming for Problem Solving – 21CPS13/23

```
#include <stdio.h>

void main()
{
    int i=1, j=1;
    while (i <= 4 || j <= 3)
    {
        printf("%d %d\n", i, j);
        i++;
        j++;
    }
}
```



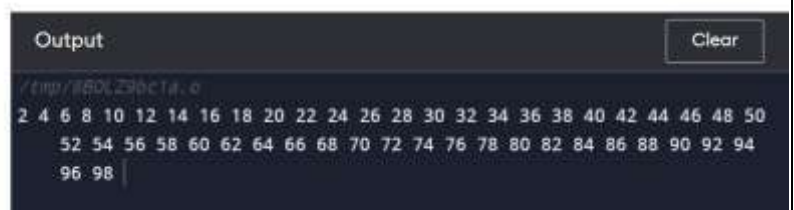
Output

```
/tmp/880LZ9bc1a.o
1 1
2 2
3 3
4 4
```

Program 4: The following program uses while loop to prints all even numbers between 1 to 100:

```
#include<stdio.h>

void main()
{
    int i = 1;
    while(i < 100)
    {
        if(i % 2 == 0)
        {
            printf("%d ", i);
        }
        i++;
    }
}
```



Output

```
/tmp/880LZ9bc1a.o
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50
52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92 94
96 98
```

Clear

### do-while loop:

Unlike while loops, which test the loop condition at the top of the loop, the do...while loop in C programming checks its condition at the bottom of the loop.

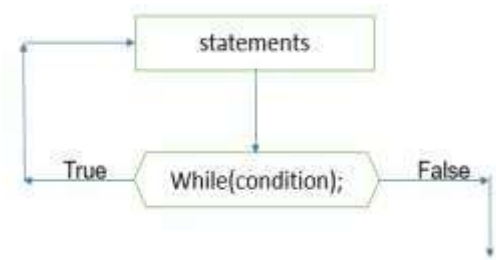
A do...while loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

do...while loop is called as exit-controlled statement.

Syntax:

```
do {  
    statement(s);  
} while (condition);
```

Flowchart:



Examples:

Program 1: Program to display number between 10 and 20 using do-while loop

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

```
void main ()
```

```
{
```

```
int a = 10;
```

```
do
```

```
{
```

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

```
    a = a + 1;
```

```
}while( a < 20 );
```

```
}
```

```
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  
value of a: 16  
value of a: 17  
value of a: 18  
value of a: 19
```

Difference between while and do-while

While loop	Do-while loop
1.Condition is checked first then statement(s) is executed	1.Statement(s) is executed at least once, thereafter condition is checked.
2.It might occur statement(s) is executed zero times, if condition is false.	2.At least once the statement(s) is executed.
3. No semicolon at the end of while. while(condition)	3. Semicolon at the end of while. while(condition);
4. Variable in condition is initialized before the execution of loop	4. variable may be initialized before or within the loop.
5. while loop is entry-controlled loop.	5. do-while loop is exit controlled loop.
6. while(condition) { statement(s); }	6. do { statement(s); } while(condition);



While loop:

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

```
void main()
```

```
{
```

```
    int i=0;
```

```
    while(i==1)
```

```
    {
```

```
        printf("while vs do-while\n");
```

```
    }
```

```
    printf("Out of loop");
```

```
}
```

Output

```
/tmp/h3Kjeuc09W.o
```

```
Out of loop
```

Do-while loop:

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

```
void main()
```

```
{
```

```
    int i=0;
```

```
    do
```

```
    {
```

```
        printf("while vs do-while\n");
```

```
    }while(i==1);
```

```
    printf("Out of loop");
```

```
}
```

Output

```
/tmp/h3Kjeuc09W.o
```

```
while vs do-while
```

```
Out of loop
```

for loop:

A for loop is a repetition control structure which allows us to write a loop that is executed a specific number of times.

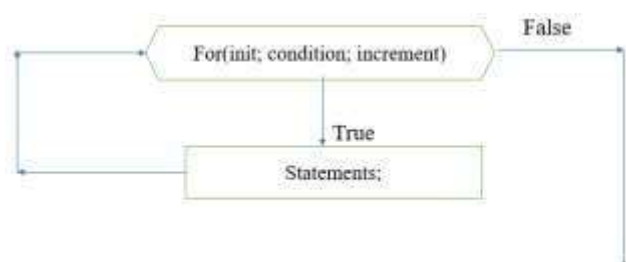
The syntax of a for loop in C programming language is:

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

```
{
```

```
    statement(s);
```

```
}
```



Here is the flow of control in a 'for' loop –

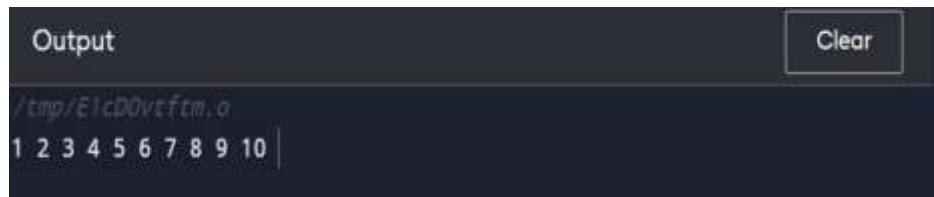
- The init step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
- Next, the condition is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.
- After the body of the 'for' loop executes, the flow of control jumps back up to the increment statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

Examples:

Program 1: Print numbers from 1 to 10

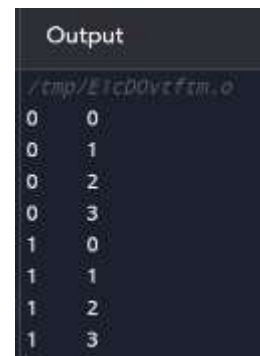
```
#include <stdio.h>

void main()
{
    int i;
    for (i = 1; i < 11; ++i)
    {
        printf("%d ", i);
    }
}
```



Program 2: Nesting of loop is also possible. Let's take an example to understand this:

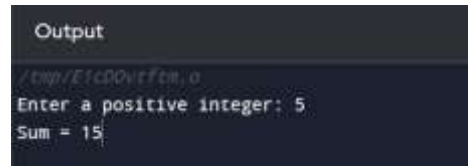
```
#include <stdio.h>
void main()
{
    for (int i=0; i<2; i++)
    {
        for (int j=0; j<4; j++)
        {
            printf("%d\t %d\n", i, j);
        }
    }
}
```



## C Programming for Problem Solving – 21CPS13/23

Program 3: Program to calculate the sum of first n natural numbers

```
#include <stdio.h>
void main()
{
    int num, count, sum = 0;
    printf("Enter a positive integer: ");
    scanf("%d", &num);
    for(count = 1; count <= num; ++count)
    {
        sum += count;
    }
    printf("Sum = %d", sum);
}
```

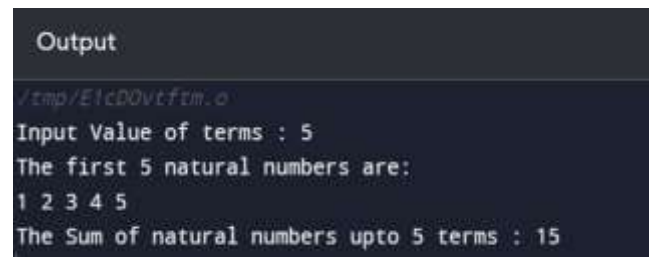


Output

```
/tmp/E1cDOvtftm.o
Enter a positive integer: 5
Sum = 15
```

Program 3: Write a program in C to display n terms of natural number and their sum.

```
#include <stdio.h>
void main()
{
    int i,n,sum=0;
    printf("Input Value of terms : ");
    scanf("%d",&n);
    printf("\nThe first %d natural numbers are:\n",n);
    for(i=1;i<=n;i++)
    {
        printf("%d ",i);
        sum+=i;
    }
    printf("\nThe Sum of natural numbers upto %d terms : %d \n",n,sum);
}
```

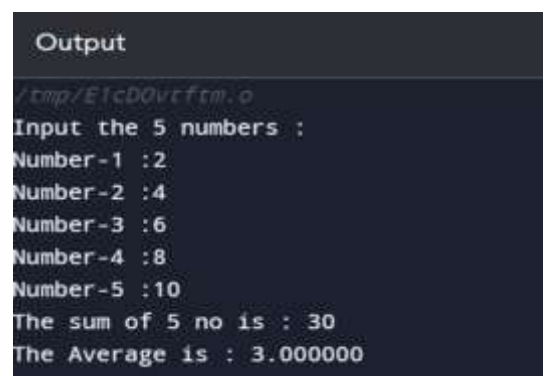


Output

```
/tmp/E1cDOvtftm.o
Input Value of terms : 5
The first 5 natural numbers are:
1 2 3 4 5
The Sum of natural numbers upto 5 terms : 15
```

Program 4: Write a program in C to read 5 numbers from keyboard and find their sum and average.

```
#include <stdio.h>
void main()
{
    int i, n, sum=0;
    float avg;
    printf("Input the 5 numbers : \n");
    for (i=1;i<=5;i++)
    {
        printf("Number-%d :",i);
        scanf("%d",&n);
        sum += n;
    }
    avg=sum/5.0;
    printf("The sum of 5 no is : %d\nThe Average is : %f\n",sum,avg);
}
```



Output

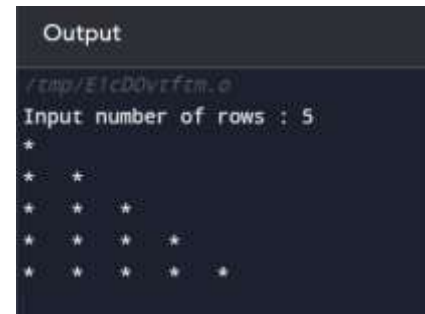
```
/tmp/E1cDOvtftm.o
Input the 5 numbers :
Number-1 :2
Number-2 :4
Number-3 :6
Number-4 :8
Number-5 :10
The sum of 5 no is : 30
The Average is : 3.000000
```

Program 5: Write a program in C to display the pattern like right angle triangle using an asterisk.

The pattern like:

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

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



## Sample Programs:

Develop a program to compute the roots of a quadratic equation by accepting the co-efficient. Print appropriate messages

```
#include<stdio.h>
#include<math.h>
void main()
{
    float a,b,c,disc,root1,root2,real,imag;
    printf("Enter values for a, b and c \n");
    scanf("%f%f%f",&a,&b,&c);
    if((a==0)&&(b==0))
    {
        printf("Roots cannot be determined\n");
    }
    else if(a==0)
    {
        printf("linear equation is\n");
        root1=-c/b;
        printf("root1=%.3f \n",root1);
    }
    else
    {
        disc=b*b-4*a*c;
```

```
        if(disc==0)
        {
            root1=root2=-b/(2*a);
            printf("The real and equal roots are :-\n");
            printf("root1=root2=%.3f \n",root1);
        }
        else if(disc>0)
        {
            root1=(-b+sqrt(disc))/(2*a);
            root2=(-b-sqrt(disc))/(2*a);
            printf("The real and distinct roots are :-\n");
            printf("root1=%.3f\troot2=%.3f \n",root1,root2);
        }
        else
        {
            real=-b/(2*a);
            imag=sqrt(fabs(disc))/(2*a);
            printf("The complex roots are :-\n");
            printf("root1=%.3f+i%.3f\t", real, imag);
            printf("root2=%.3f-i%.3f \n", real, imag);
        }
    }
}
```

Develop a program to implement Pascal's Tringle

```
#include<stdio.h>
void main()
{
    int rows, coef = 1, space, i, j;
    printf("\nEnter the number of rows :");
    scanf("%d",&rows);
    printf("\n");

    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\n");  
    }  
}
```

## C-Type Casting:

Converting one datatype into another is known as type-conversion or type casting.

Types of type conversion:

- Implicit Type conversion: Also known as automatic type conversion. Done by the compiler on its own, without any external trigger from the user

```
#include<stdio.h>  
  
void main()  
{  
  
    int x=10;  
    char y = 'a';  
    float z;  
    x = x+y;  
    z= x+1.0;  
    printf(" x= %d and z= %f", x, z);  
  
}
```

- Explicit Type conversion: This process is also called type casting/Explicit type conversion and it is user defined. Here the user can type cast the result to make it of a particular data type.

The syntax in C:

**(datatype) expression;**

datatype indicated the data type to which the final result is converted.

```
#include<stdio.h>  
  
void main()  
{  
  
    float x=1.2;  
    int sum;  
    sum = (int) x + 14;  
    printf("sum=%d", sum);  
  
}
```