# Structured Programming Approach

Anuradha
BHATIA

## Table of Contents

## Disclaimer

# 1. Basic of C

## CONTENTS

## 1.1 History of C

1. C is a general purpose structured programming language that is powerful, efficient and compact.
2. C is an offspring of the “basic combined programming language” called B.
3. This was modified by Dennis Ritchie and was implemented in the Bell laboratories in 1972.
4. C is a structured programming language. C combines the features of a high –level language with the elements of the assembler.
5. This flexibility allows C to be used for system programming.
6. A number of implementation of C, are available like Microsoft C, Quick C, Turbo C, Lattice C, Aztec C.
7. C has large number of operators and relatively small instruction set.
8. It allows user to write library functions of its own.

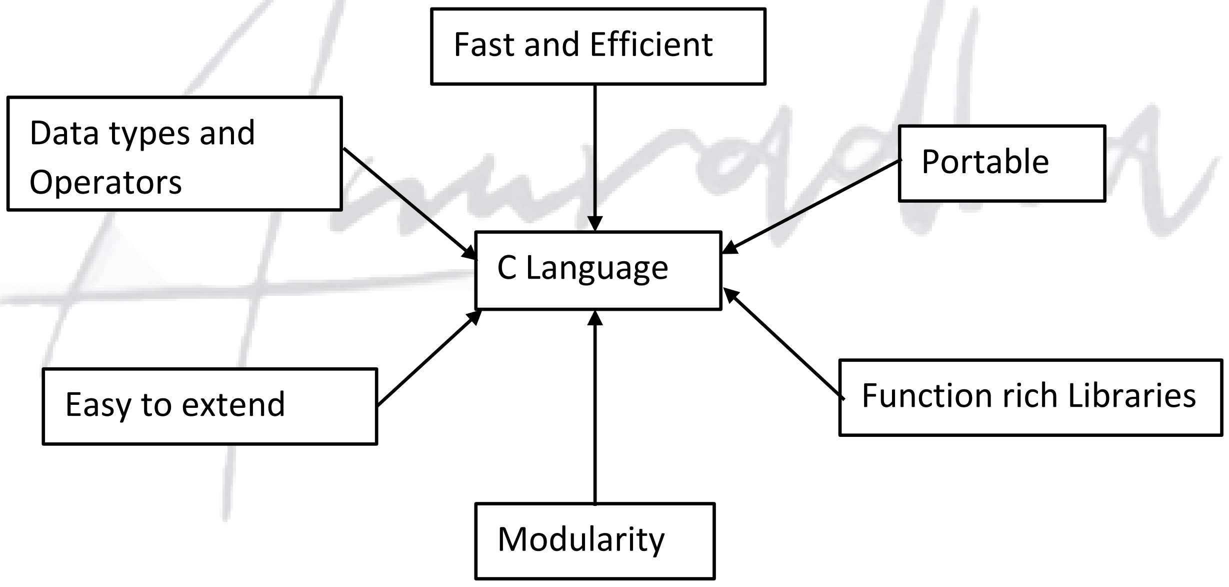


Figure 1.1: C Language Accessibility

## 1.2 C character set, tokens, constants, variables, keywords , identifiers

1. **C program** is a collection of one or more functions.
2. **Function** is a module that contains one or more C statements and performs one or more tasks.
3. **C-tokens** are identifiers, constants, variables, keywords, strings, operators, special symbol.

4. **Alphabets,** numbers and special characters are used to construct constants, variables and keywords.
5. **Identifiers** are name of variables, functions and arrays. They are combination of alphabets and an underscore, no spaces are allowed.
6. **Constants** is a quantity that does not change. Constant can be stored at a location in the memory of the computer. E.g. 15.5, -6.7
7. **Variable** is a name given to the location in the memory where constant is stored. E.g. num, sum.
8. **Keywords** are the words whose meaning has already been explained to the compiler. E.g. For, while, if

## 1.3 C operators- arithmetic, Logical, assignment, relational, increment and decrement, conditional, bit wise, special, operator precedence, C expressions data types

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators

1. **Arithmetic Operators**

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20 then −

| Operator | Description | Example |
|---|---|---|
| + | Adds two operands. | A + B = 30 |
| − | Subtracts second operand from the first. | A − B = 10 |
| * | Multiplies both operands. | A * B = 200 |
| / | Divides numerator by de-numerator. | B / A = 2 |
| % | Modulus Operator and remainder of after an integer division. | B % A = 0 |
| ++ | Increment operator increases the integer value by one. | A++ = 11 |
| -- | Decrement operator decreases the integer value by one. | A-- = 9 |

Table 1.1: Arithmetic Operator

2. **Relational Operators**

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then −

| Operator | Description | Example |
|---|---|---|
| == | Checks if the values of two operands are equal or not. If yes, then the condition becomes true. | (A == B) is not true. |
| != | Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true. | (A!= B) is true. |
| > | Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true. | (A > B) is not true. |
| < | Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true. | (A < B) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true. | (A >= B) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true. | (A <= B) is true. |

Table 1.2: Relational Operator

3. **Logical Operators**

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then −

| Operator | Description | Example |
|---|---|---|
| && | Called Logical AND operator. If both the operands are non-zero, then the condition becomes true. | (A && B) is false. |
| \|\| | Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true. | (A \|\| B) is true. |
| ! | Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false. | !(A && B) is true. |

Table 1.3: Logical Operator

4. **Bitwise Operators**

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows

| p | q | p & q | p \| q | p ^ q |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |

Table 1.4: Bitwise Operator Truth Table

Assume A = 60 and B = 13 in binary format, they will be as follows −

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then

| Operator | Description | Example |
|---|---|---|
| & | Binary AND Operator copies a bit to the result if it exists in both operands. | (A & B) = 12, i.e., 0000 1100 |
| \| | Binary OR Operator copies a bit if it exists in either operand. | (A \| B) = 61, i.e., 0011 1101 |
| ^ | Binary XOR Operator copies the bit if it is set in one operand but not both. | (A ^ B) = 49, i.e., 0011 0001 |
| ~ | Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. | (~A ) = -61, i.e,. 1100 0011 in 2's complement form. |
| << | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. | A << 2 = 240 i.e., 1111 0000 |
| >> | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | A >> 2 = 15 i.e., 0000 1111 |

Table 1.5: Bitwise Operator

5. **Assignment Operators**

The following table lists the assignment operators supported by the C language

| Operator | Description | Example |
|---|---|---|
| = | Simple assignment operator. Assigns values from right side operands to left side operand | C = A + B will assign the value of A + B to C |
| += | Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand. | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand. | C -= A is equivalent to C = C - A |
| *= | Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand. | C *= A is equivalent to C = C * A |
| /= | Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand. | C /= A is equivalent to C = C / A |

| | | |
|---|---|---|
| %= | Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand. | C %= A is equivalent to C = C % A |
| <<= | Left shift AND assignment operator. | C <<= 2 is same as C = C << 2 |
| >>= | Right shift AND assignment operator. | C >>= 2 is same as C = C >> 2 |
| &= | Bitwise AND assignment operator. | C &= 2 is same as C = C & 2 |
| ^= | Bitwise exclusive OR and assignment operator. | C ^= 2 is same as C = C ^ 2 |
| \|= | Bitwise inclusive OR and assignment operator. | C \|= 2 is same as C = C \| 2 |

Table 1.6: Assignment Operator

6. **Miscellaneous Operators**

Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :** supported by the C Language.

| **Operator** | **Description** | **Example** |
|---|---|---|
| sizeof() | Returns the size of a variable. | sizeof(a), where a is integer, will return 4. |
| & | Returns the address of a variable. | &a; returns the actual address of the variable. |
| * | Pointer to a variable. | *a; |
| ? : | Conditional Expression. | If Condition is true ? then value X : otherwise value Y |

Table 1.7: Miscellaneous Operator

7. **Operators Precedence in C**

i. Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated.
ii. Operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.
iii. For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.
iv. Operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

| **Category** | **Operator** | **Associativity** |
|---|---|---|
| Postfix | () [] -> . ++ - - | Left to right |
| Unary | + - ! ~ ++ - - (type)* & sizeof | Right to left |
| Multiplicative | * / % | Left to right |

| | | |
|---|---|---|
| Additive | + - | Left to right |
| Shift | << >> | Left to right |
| Relational | < <= > >= | Left to right |
| Equality | == != | Left to right |
| Bitwise AND | & | Left to right |
| Bitwise XOR | ^ | Left to right |
| Bitwise OR | \| | Left to right |
| Logical AND | && | Left to right |
| Logical OR | \|\| | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= *= /= %=>>= <<= &= ^= \|= | Right to left |
| Comma | , | Left to right |

Table 1.8: Operator Precedence

## 1.4 Problem solving techniques: flowchart and algorithm

1. **Data types** available in C are integer declared as **int**, character declared as **char**, float or real declared as **float** (max up to six decimal places), long integer as **long int**, and string as **char [20]** where 20 is the length of a string.
2. **Program** is a set of instructions written in a sequence.
3. **Flowchart** is pictorial representation of the program.

The following are the symbols used in flowchart.

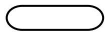
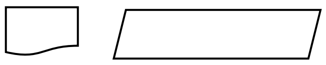
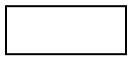
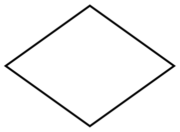
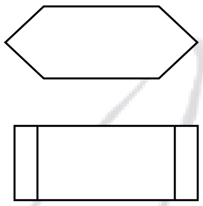
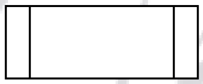
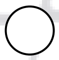
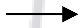
| Symbol | Description |
|---|---|
|  | *START /STOP* of the program. |
|  | *INPUT /OUTPUT BOX* |
|  | *ASSIGNMENT BOX* |
|  | *DECISION (IF LOOP, WHILE LOOP, SWITCH CASE)    BOX* |
|  | *FOR NEXT LOOP* |
|  | *FUNCTION CALL* |
|  | *CONNECTOR* |
|  | *FLOW ARROW* |

Table 1.9: Flowchart Symbols

***(Question: Define algorithm. 2 Marks)***

4. **Algorithm** is a step by step logic of a program written in simple English. It is independent of any programming language.

## 1.5 Features and Formatted input, formatted output instructions.

1. **Features of C programming**
   i. It is a high-level language.
   ii. It follows top down approach.
   iii. It uses structures and pointers.
   iv. Unique feature of C is the preprocessor.

2. **Simple structure of a C program**

```
#include<stdio.h>
main( )
{
    _________
    _________
    body of C program
    _________
    _________
 }
```

**Explanation**

i. **#** is a **preprocessor directive** which follows simple syntax rules.
ii. Preprocessor as the name implies is a program that processes the source code before it passes through the compiler.
iii. It operates under the preprocessor command lines or directives.
iv. They all begin with the symbol **#** in the first column and does not end with a semicolon.
v. The commonly used directives are **# include** and **# define**.

| Directive | Function |
|---|---|
| #include | Specifies the file to be included. |
| #define | Defines a macro substitution. |
| #undef | Undefines a macro. |
| #ifdef | Tests for a macro definition. |
| #endif | Specifies the end of #if |
| #ifndef | Tests if a macro is not defined. |
| #if | Tests a compile time condition. |

| | |
|---|---|
| #else | Specifies alternatives when #if test fails. |

- **stdio** : standard input output file
- **conio** : console input out file
- **.h** : header file.
- The **stdio.h** is an abbreviation for standard input –output header file.
- **#include<stdio.h>** tells the compiler to search for a file name stdio.h and place its content at the particular point in the program.
- The contents of the header file become part of the source code when it is compiled.
- The basic two functions are **printf** and **scanf**.

**3. printf scanf in C**

The printf() and scanf() functions are used for input and output in C language. Both functions are inbuilt library functions, defined in stdio.h (header file).

**i. printf() function**

- The printf() function is used for output. It prints the given statement to the console.
- The syntax of printf() function is given below:

```
printf("format string",argument_list);
```

- The format string can be %d (integer), %c (character), %s (string), %f (float) etc.

**ii. scanf() function**

- The scanf() function is used for input. It reads the input data from the console.

```
scanf("format string",argument_list);
```

## 4. Getting started with c programming

**PROGRAM 1**

```
#include<stdio.h>
#include<conio.h>
main( )
{
  clrscr( );
  printf(“Welcome to C programming”);
}
```

**Explanation**

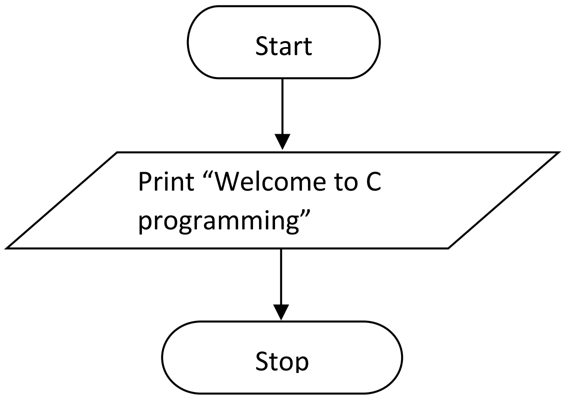
i. Above written is a simple C program to print Welcome to C programming.
ii. Stdio.h is necessary as it has the functions **printf** in it.
iii. All the statements of C should end with a semicolon except #, main( ), { }, and all the loops.
iv. The **clrscr( )** is used to clear the screen before printing anything on it.
v. Whatever is written inside the **printf** statement is printed as it is on to the output screen.

To save the file → **F2**
To compile the program → **Alt + F9**
To compile and run the program → **Control + F9**
To see the output → **Alt + F5**

The **output** of the above program is

```
Welcome to C programming
```

The flowchart is:



**PROGRAM 2**

Write a C program to find the sum of two numbers.

```
#include<stdio.h>
#include<conio.h>
main( )
{
  int a = 10 , b = 20 , sum ;     //assignment statement
  sum = a + b ;
  printf(" the sum of two numbers is ➔ %d " , sum);
  getch( );
}
```

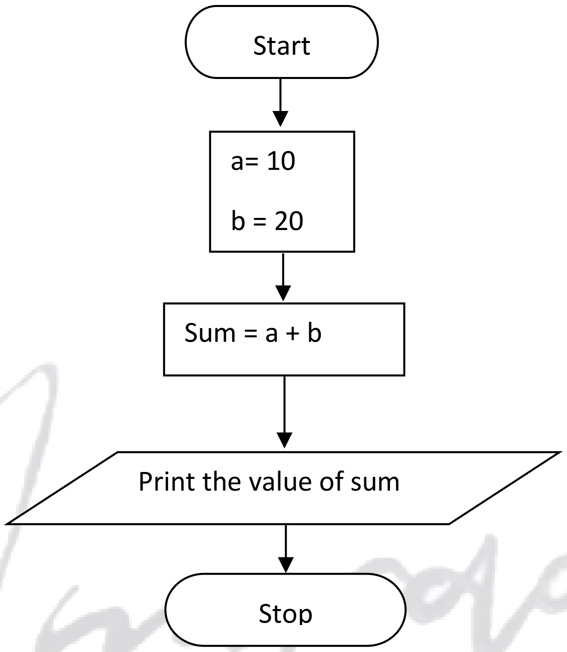
**Explanation**

i. In the above program a is assigned the value 10 and b is assigned the value 20.
ii. The sum is declared as an integer variable to hold integer value.
iii. Then we calculate sum by adding the two values.
iv. To print the value of sum we introduce a new term called **format specifiers or data qualifiers** as **%d** for integer, **%f** for float, **%c** for character, **%s** for string, **%ld** for long integer.
v. In this where %d is written the value of sum will be printed.
vi. **getch( )** is used to accept a single character from the keyboard i.e. after printing the output on the screen it will wait for the user to enter the value.

The **output** is

The sum of two numbers is ➔ **30**

The flowchart for the above program is:



**<u>Note</u>**: In the above program we have introduced assignment statement, **getch( )** and comment statements i.e. anything which is written between the // or /*…..*/ is an comment.

The above program can be modified from the assignment statement to the user's choice program, i.e. the two numbers can be entered from the keyboard.

**PROGRAM 3**

WAP to find the sum of two numbers.

```
#include<stdio.h>
#include<conio.h>
main( )
{
        int a ,b, sum;
        printf("Enter the first number➔");
        scanf("%d",&a);
        printf("\n Enter the second number ➔");
        scanf("%d",&b);
        sum = a + b;
        printf("\n The sum of two numbers %d and %d is ➔ %d " ,a ,b, sum);
        getch( );
}
```

**Explanation**

i. The general syntax of the **scanf** statement whish is known as an input statement is
ii. scanf("control string ", &variable 1, variable2);
iii. The control string contains the format of data being received.
iv. The symbol '**&**'known as an **ampersand** before each variable specifies the variable name's address.
v. We must used this before an input variable otherwise unexpected result occur.
vi. The other new symbol introduced is **'\n'** which is a new line character. It allows what ever written after it to move to the next new line.
vii. In the third **printf** statement the %d signs are replaced by their respective values.
viii. **Imp**: Integer value has a limitation that is –32768 to +32767.

The **output** is:

```
Enter the first number➔ 30
Enter the second number ➔ 49
The sum of two numbers 30 and 49 is ➔ 79
```

**Question:** Draw the flowchart for the above program.

**PROGRAM 4**

Write a program (WAP) to print cube of given number

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int number;
        clrscr();
        printf("Enter a number:");
        scanf("%d",&number);
        printf("Cube of number is:%d ",number*number*number);
        getch();
}
```

The Output is

```
Enter a number: 5
Cube of number is: 125
```

**Explanation**

i. The scanf("%d",&number) statement reads integer number from the console and stores the given value in number variable.
ii. The printf("cube of number is:%d ",number*number*number) statement prints the cube of number on the console.

**Practical problems**

Also draw the flowchart.

1. Write a program to print your name, father's name, and address, telephone number each on separate line using simple **printf** statement.
2. Write a program to calculate simple interest.

$$\text{Interest} = \frac{\text{principal * no. of years * rate of interest}}{100}$$

3. Write a program to convert the Fahrenheit temperature to Celsius.

$$\text{Celsius} = \frac{(\text{Fahrenheit} - 32.0)}{1.8}$$

4. Write a program to find sum and average of three numbers.

# 2. Decision Making

## 2.1 Decision making and branching if-statement – if, if-else, else-if ladder, nested if else, switch case statement, break statement

1. Programs are normally executed sequentially in the order in which they appear.
2. This is true if no repetition or decision making is necessary.
3. But number of times we need to change the sequence of program depending upon the requirement, or we need to repeat some statements number of times.
4. C language allows the user for decision making and change the sequence of the program using control or decision making statements like:
    i. **if** statement .
    ii. **switch** satetement .
    iii. **break** ..
    iv. **conditional** operators
    v. **goto** statement.

**i. if statement**

➢ if statement is used where we need to choose between the two things depending upon a particular value . It holds the value either true or false or yes or no . This is fur divided into

✓ **simple if** statement

The *syntax* of is

```
if ( condition )
{
statements ;
}
```

**PROGRAM 5**

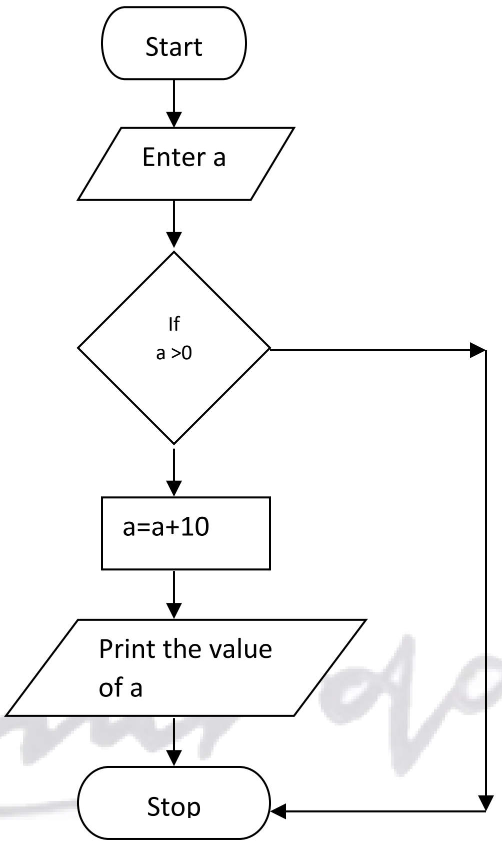
Write a program to add 10 to a number if its value is more than 0 and print it.

```
#include <stdio.h>
#include<conio.h>
main( )
{
        int a ;
        clrscr( );
        printf("Enter a number → ") ;
        scanf ("%d",&a);
        if (a > 0 )
        {       a = a + 10 ;
                printf (" the value of  a is ➔ %d ", a);
        }
        getch( );
}
```

The **output** is :

```
Enter a number → 45
the value of  a is ➔ 55
```

The flowchart for the above program is



**Explanation**

i. In the above program we enter a number and store in the variable a, then check whether a is greater than 0 , if the answer is yes than we add 10 to it and print the result .

ii. But if it does'nt satisfy the condition than it terminates the program and come out.

✓ **if ….else** statement

```
if (condition)
        {
            true block statement;
        }
        else
        {
            false block statement ;
        }
```

**PROGRAM 6**

Write a program to find the largest of two numbers.

```
#include<stdio.h>
#include<conio.h>
main( )
{
        int a , b ;
clrscr( );
        printf(" Enter the two numbers ➔");
        scanf(" %d  %d ",&a ,&b);
        if ( a > b )
           printf(" \n a is large %d ",a);
        else
           printf("\n b is large %d  " , b);
        getch( );
}
```

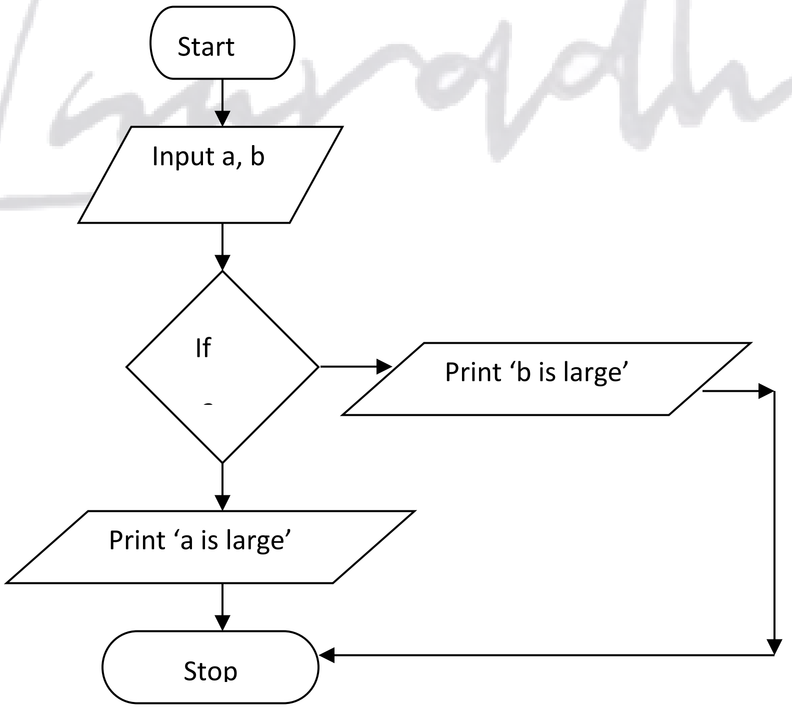
**Explanation**

i. In the above program we want to find the largest of two numbers.
ii. We input two numbers and store it in the variables a and b.
iii. Then with the, if loop we check that whether a > b if the answer is yes or true than we print that a is large and if this statement is false than we print that the other number in this case b is large.

The **output** is:

```
Enter the two numbers ➔ 34 56
b is large 56
```

The **flowchart** for the above program is:



✓ **nested if …else** statement

➢ If we need to make multiple choices then the nested if statement is used. If the above program is modified for the largest of three numbers than the program becomes as given below and the syntax of the nested is

```
if ( condition )
{
        if(condition)
        {
          statement;
        }
        else
        {
          statement ;
        }
}
else
{
        if(condition)
        {
          statement ;
        }
        else
        {
          statement ;
        }
}
```

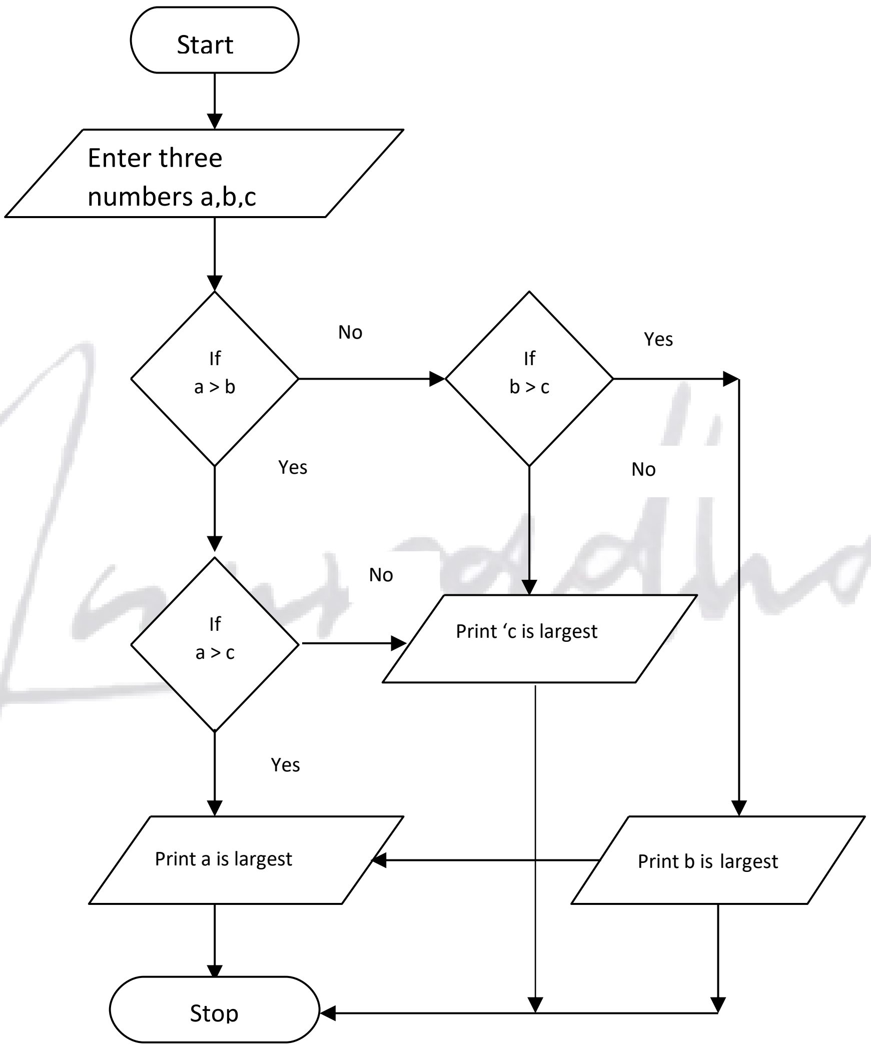
**PROGRAM 7**

Write a program to find the largest of three numbers using if else.

```
#include<stdio.h>
#include<conio.h>
main( )
{
        int a,b,c;
       clrscr( );
        printf( "\n Enter three numbers �");
        scanf("%d  %d  %d" ,&a,&b,&c);
        if ( a < b)
        {
                if( a > c)
{
printf("\na is largest of three numbers %d %d  %d d",a,b,c,a);
                }
                else
                {
printf("\nc is largest of three numbers %d %d %d %d",a,b,c,c);
}
        else
        {
                if (b > c)
                {
printf("\nb is largest of three numbers %d %d %d d",a,b,c,b);
                }
                else
printf("\nc is largest of three numbers %d %d %d %d",a,b,c,c);
        }
```

getch( )

}The **flowchart** is:



**Explanation**:

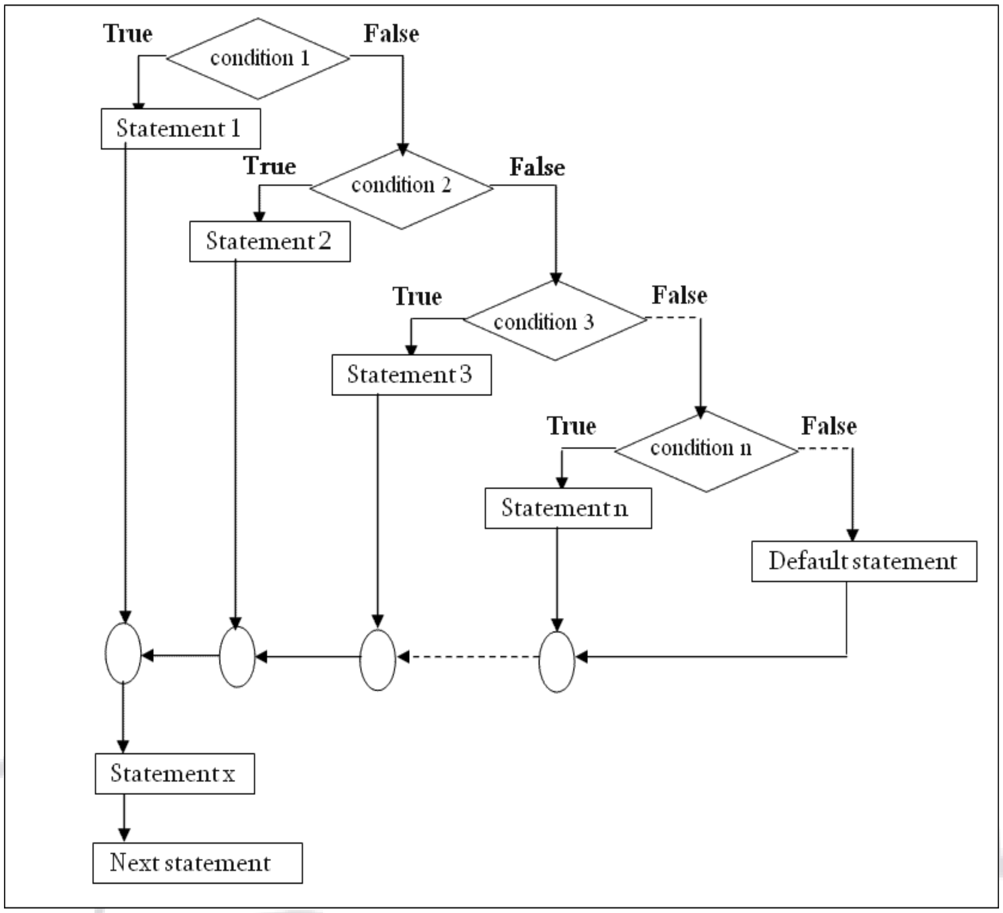
i. The above program finds the largest of three numbers. as we input three numbers a , b, c and check whether a is larger than b
ii. if the answer is yes or true than if check whether a is larger than c than we print that a is largest of three numbers
iii. otherwise the number which is stored in the variable c is largest of three numbers
iv. But if the first condition fails than we check that if b is larger than c if yes than we print that b is largest of the three numbers else c is the largest.

The **output** of the above program is:

```
Enter three numbers 15 34 23
b is largest of three numbers 15 34 23  34
```

✓ **else if ladder** statement

```
if ( condition )
{
        statement;
}
else if (condition )
    {
        Statement;
    }
    else if(condition )
         {
             Statement;
          }
```



**PROGRAM 8**

Write a program to print that if a student scores above 90 than he gets A++ grade and if he gets between 80 to 89 than he gets A+ grade and if he gets between 70 to 79 than he gets A grade and if he gets between 60 to 69 than he gets B+ grade and if he gets between 50 to 59 than he gets B grade else print C grade. (Program uses logical &&) Also draw the flow chart for the same.

**PROGRAM 9**

Write a program to print the roots of a quadratic equation i.e –b+sqrt($b^2$-4ac)/2a

## 2. SWITCH CASE STATEMENT

i. The use of if else becomes tedious if there are lot many comparisons, hence a new statement called switch case came into picture.

ii. The **syntax** is

```
switch (expression)
{
   case  option –1 :
                      statements;
                      break;

   case  option –2 :
                      statements;
                      break;
   default :
                      statements;
                      break;
}
```

The programs written using switch case are known as **menu driven programs**.

**Program 10**

Write a program to add, multiply, subtract and divides a number using a switch case statement.

```
#include<stdio.h>
#include<conio.h>
main( )
{
```

```
int num1,num2,n=0;
float ans;
int choice ;
printf(“Enter two numbers  “);
scanf(“%d %d”,&num1,&num2);
printf(“1:Add”);
printf(“2:Sub”);
printf(“3:Mul”);
printf(“4:Div”);
printf(“Enter your choice  “);
scanf(“%d”,&choice);
switch (choice)
{
        case 1:   ans = num1+num2;
                  n=1;
                  break;
case 2:   ans = num1-num2;
                  n=1;
                break;
case 3:   ans = num1*num2;
                  n=1;
                break;
case 4:   ans = num1/num2;
                  n=1;
                break;
        default: printf(“u have entered wrong choice”)
                 break;
}
if ( n==1)
```

```
                printf(“The result is %f”,ans);
            getch( );
            }
```

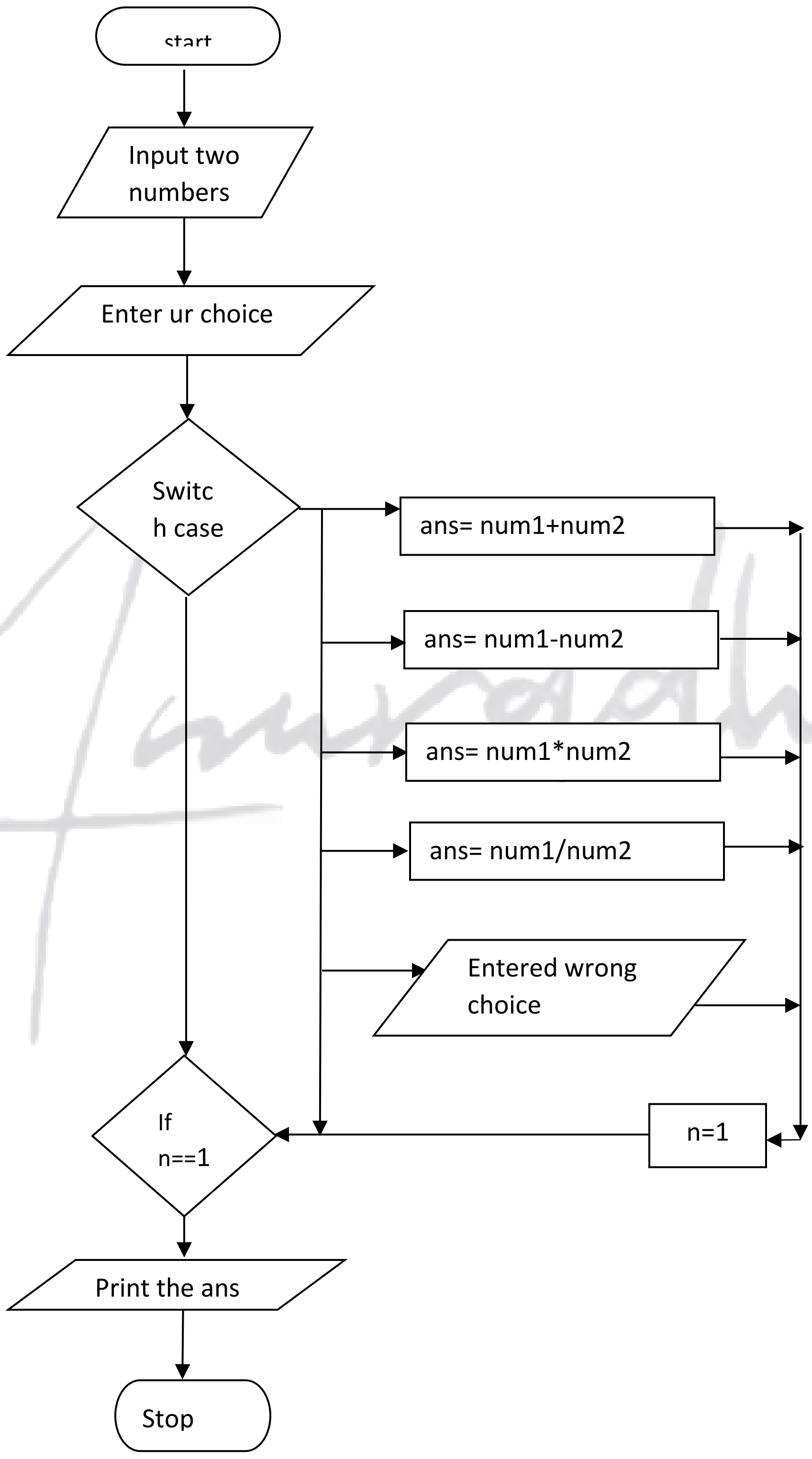
**Explanation:**

i. In the above program instead of using if.else for the choices we use switch case statement which reduces the ambiguity of looping continuously.

The **output** is:

```
Enter two numbers 45 67
1: Add
2: Sub
3: Mul
4: Div
Enter your choice 1
The result is 112
```

The **flowchart** is:



**PROGRAM 11**

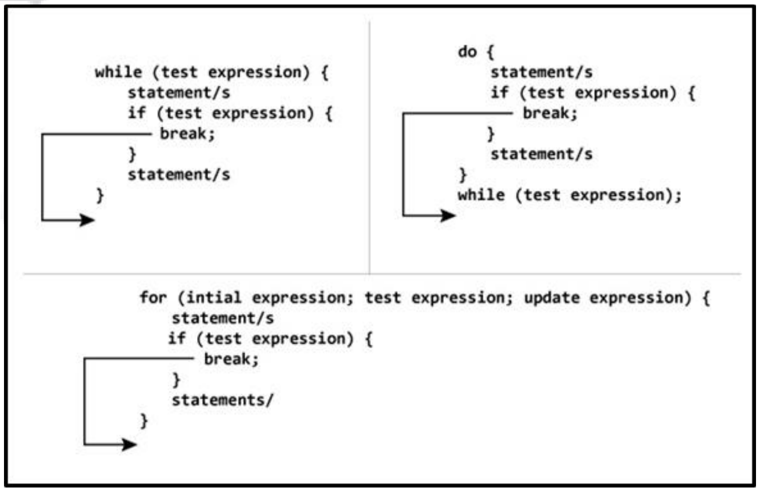
Write a program using switch case to print whether the given number is even or odd, to print the factorial of a number and prime number.

```
#include<stdio.h>
main()
{
int choice, factorial, temp, number, i;
while (1)
{
    printf("\n1. Factorial");
    printf("\n2. Prime");
    printf("\n3. Odd/ Even");
    printf("\n4. Exit");
    printf("\n\nYour Choice? : ");
    scanf ("%d", &choice);
switch (choice)
{
case 1:
    printf("\nEnter the Number:");
    scanf ("%d", &number);
    for (temp=1,factorial=1; temp<=number;temp++)
    {
        factorial=factorial*temp;
        continue;
    }
    printf("Factorial=%d\n\n",factorial);
    break;
case 2:
    printf("Enter the Number:");
    scanf ("%d", &number);
    if (number==1)
    {
        printf("The Number 1 is a Unique Number.\n\n ");
    }
    for (i=2; i<=number-1; i++)
    {
    if (number%i==0)
    {
        printf("Number is not prime\n\n");
        break;
    }
    if (number==i || number%i !=0)
    {
        printf("Number is Prime\n\n");
        break;
    }
    }
    break; //break out of this case.
case 3:
    printf("Enter the Number:");
    scanf ("%d", &number);
    if (number%2==0)
    {
        printf("The number is Even\n\n");
    }
    else
    {
          printf("The number is Odd\n\n");
      }
      break;
case 4:
      exit();
default:
printf("You have entered the wrong choice. Please try again. \n");
      break;
}
}
}
```

**3. BREAK statement**

i. The break statement is used anywhere in the program to break the flow of the program and to come out of the particular loop.

ii. The break statement in C programming has the following two usages



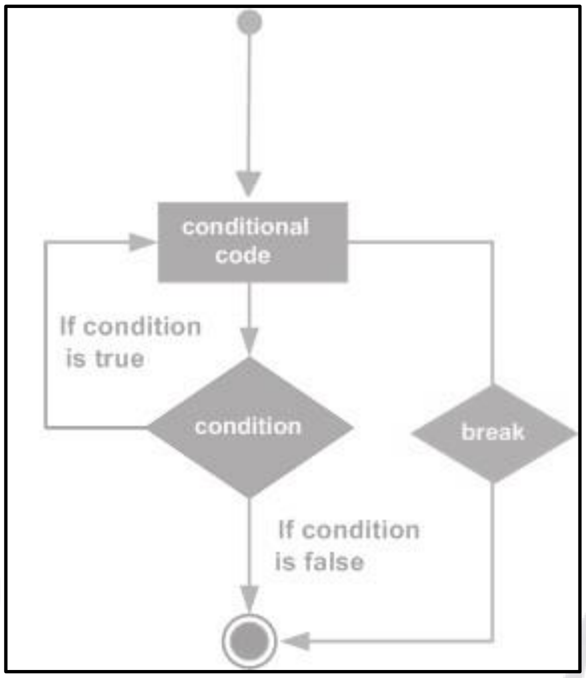
```
while (test expression) {
    statement/s
    if (test expression) {
        break;
    }
    statement/s
}
```

```
do {
    statement/s
    if (test expression) {
        break;
    }
    statement/s
}
while (test expression);
```

```
for (intial expression; test expression; update expression) {
    statement/s
    if (test expression) {
        break;
    }
    statements/
}
```

iii. When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

iv. It can be used to terminate a case in the switch statement (covered in the next chapter).



**Example**

```
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;
}
```

The **output** is

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

**ii. The? : operator**

- This operator is popularly known as **conditional operator**. This is useful for making two way decision.
- The general **format** is

  **Conditional expression ? expression-1 :expression-2;**
- In this conditional expression is evaluated first and if the result is true than the expression-1 is returned as the value of the conditional expression otherwise expression-2 is returned as the value of conditional operator.

Example: to check the largest of two numbers

```
large = (a > b) ? a : b ;
```

- In the above C statement it checks if the value of a is greater than b , if the answer is true than the variable large gets the value of a otherwise it gets the value of b.
- This statement is a shortcut method of if..else statement .

**6. goto Statement:**

i. This is an unconditional branch statement.

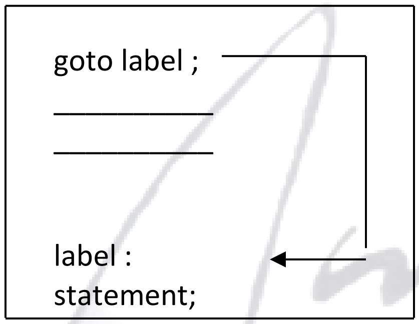
ii. This is used to jump in the program from one statement to another without any condition.

iii. The goto requires a label to identify the place it wants to branch to.

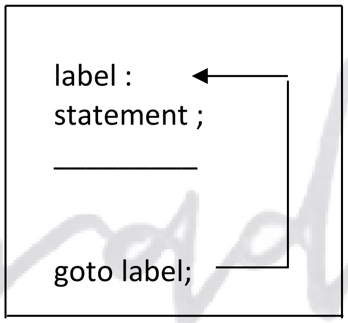
iv. A label is any valid variable name and must be followed by a colon ':' .

v. The label is placed immediately before the statement where the control is to be transferred.

vi. The goto breaks the normal sequence of the program and is very rarely used



**FORWARD JUMP**



**BACKWARD JUMP**

## 2.2 Decision making and looping - while, do, do- while statement, for loop, continue statement

1. The looping process should include the following steps.
   i. Setting and initializing of the counters.
   ii. Test for the specified condition for the execution of the loop.
   iii. Execution of statements in the loop.
   iv. Incrementing the counter.

2. The C language supports three loop statements.
   i. **while** loop statement.
   ii. **do** loop statement.
   iii. **for** loop statement.

**i. WHILE LOOP STATEMENT**

- This is the simplest of all the loops in C language. The general format for the while loop is .

```
while ( test condition )
{
        body of the loop ;
}
```

- The while loop is an entry controlled loop , which means that if the while loop condition is true than only the control be transferred to the loop inside , otherwise it will terminate the loop or come out of the loop.
- If the condition is true than the control is transferred inside the body of the loop, and after it is executed it once again check for the condition, this process continues till the condition is not satisfied.

**PROGRAM 12**

Write a program to find the sum of first 10 numbers.

```
#include<stdio.h>
#include<conio.h>
main( )
{       int num = 1, sum = 0;
        clrscr( );
        while ( num < = 10)
        {
                sum = sum + num ;
                num = num + 1;
        }
printf (“ the sum of 10 numbers is ➔ %d “, sum );
getch( );
}
```

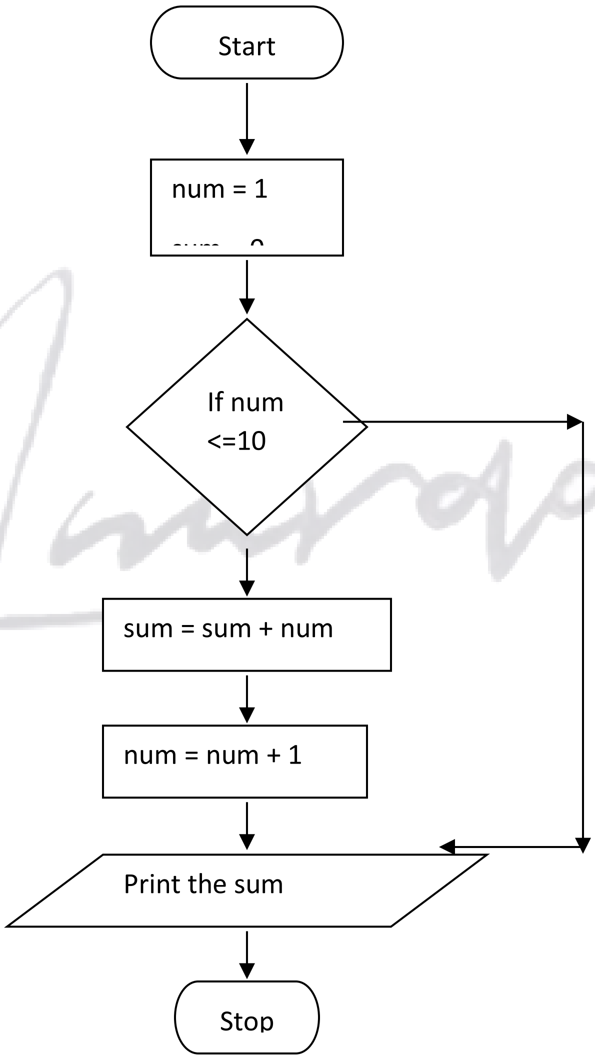
**Explanation:**

i. The above program is used to find the sum of first 10 numbers.
ii. We first assign a variable called num the initial value of 1, we have to find the sum from numbers 1 to 10.
iii. Another variable sum is initialized to 0 to store the sum of all the numbers.
iv. We than check the while loop that the statements inside the loop should be executed till the value of num is more than 10.
v. As initially the value of num is 1 it enters the loop and adds up the num to the variable sum and the value of num is incremented by one.
vi. Once again the value of num is checked in the while loop and as num is less than 10 it again enters the loop , and this procedure continued till the value of num is more than 10 .
vii. When the value of num is 11, it terminates the loop and print value of sum.

The **output** is

```
The sum of 10 numbers is ➔ 55
```

The **flowchart** is



**Algorithm**

```
1. Initialize the value of num to 1 and sum = 0.
2. Check while ( num <= 10 ) do the following
        sum = sum + num
        num = num + 1
3. Print the sum
```

**ii. DO LOOP STATEMENT**

- The while loop we studied first checks the condition and if the condition is true or if the condition is satisfied than only executes the statement in the loop else comes out of it.
- But sometimes it is required that the set of statements to be executed before the check condition, in such cases the do while loop is used.
- The syntax for it is as given below.

```
do
{
        body of loop;
}while(test condition) ;
```

**PROGRAM 13**

Write a program to check whether the number is even or odd, the program should continue till user choses to quit.

```
#include<stdio.h>
#include<conio.h>
main( )
{       int num ;
        char ans;
do
{       printf("Enter the number to be checked ➔");
        scanf("%d",&num);
        if (num % 2 == 0)
                printf (" \nthe number %d is even",num);
        else
                printf(" \nThe number %d is odd ",num);
        printf(" \nDo u want to continue➔ ");
        scanf ("%c ", &ans);
}while (ans =='y');
getch( );
}
```

The **output** is

```
Enter the number to be checked ➔56
The number 56 is even
Do u want to continue➔y
Enter the number to be checked ➔73
The number 73 is odd
Do u want to continue ➔n
```

**Explanation:**

i. In the above program we want to find whether the number is even or odd , and we want to check for the numbers till the user wants to quit the program for this we use a do while loop .
ii. We enter the number to be checked and in the if loop we check if the number divided by two the remainder is zero than the number is an even number else it is an odd number .
iii. Then the user is asked whether the user wants to check for more numbers if the answer is 'y', then the program continues, otherwise it comes out of the loop and program ends.

Draw the flowchart for the same.

***IMP NOTE:*** The difference between the while loop and do..while loop is that in the while loop it checks for the condition initially and if it is true than only the body of the loop is executed whereas in the do.. while loop the body of the loop i.e the statements are executed at least once and then the condition is checked and if the condition is true than it re-enters the loop else comes out of it.

**iii. FOR LOOP STATEMENT**

- This is also an entry controlled loop. The general syntax for the for loop is

```
for(initial cond ; final condition ; increment operator)
{
        body of loop;
}
```

**PROGRAM 14**

Write a program to find the sum of first 10 numbers.

```
#include<st dio.h>
#include<conio.h>
main( )
{
        int i , sum = 0 ;
        clrscr( );
        for( i = 1 ; i < = 10 ; i++)
        {
            sum = sum + i ;
        }
        printf("\nthe sum of first 10 numbers is ➔%d ",sum);
        getch( );
}
```

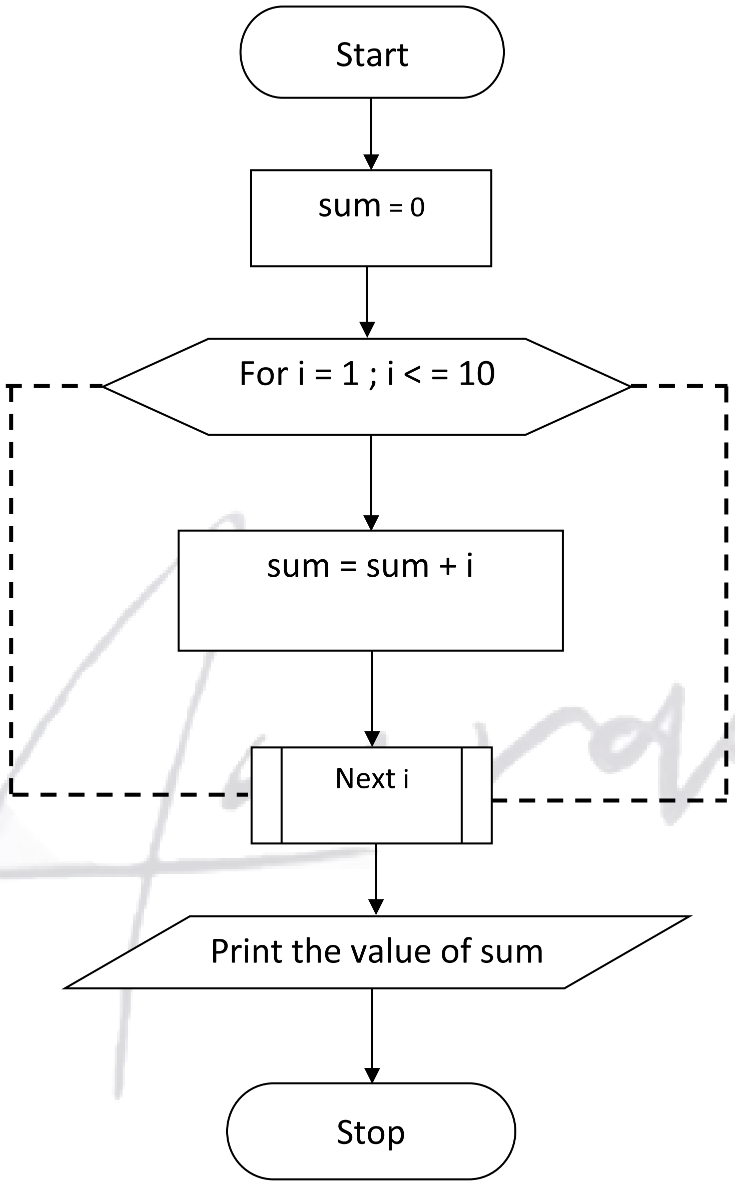
The **output** is

```
The sum of first 10 numbers is ➔55
```

**Explanation**

i. As we want to find the sum of first 10 numbers, so we know that our starting number is one and the last number is 10, and the number increase gradually with one.
ii. Sum is initialized to zero as it may take the garbage value. So the initial condition of for loop is 1 and the termination condition or the final condition of the for loop will be 10 and the loop variable is incremented by 1.
iii. The value of sum is added to the loop variable and saved back to sum.

The **flow chart** is:



**PROGRAM 15**

Write a program to input the10 numbers from the user and find its sum and average.

**PROGRAM 16**

WAP to enter 10 numbers and find the largest of the number.

**Using a nested for loop**

The nested for loop is used where two or more loops are used to perform certain tasks. In ANSI C maximum of 15 levels can be used.

The syntax looks like.

```
for ( i = 0  ; i < 5 ; i ++)
{
        for (j = i ; i < = 5 ; j++)
        {
                _________
                _________
        }
}
```

**PROGRAM 17**

Write a program to print the following pattern

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

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

```
#include<conio.h>
main( )
{
        int i , j ;
        for( i = 1 ; i < = 5 ; i++)
        {
                for ( j = 1 ; j <= i ; j ++)
                 {    printf("%d\t ",j);   }
                printf("\n");
        }
        getch( );
}
```

Draw the flowchart for the above program.

**IMPORTANT FEATURE OF FOR LOOP**

1. The most important feature of for loop is that more than one variable can be initialized in the for loop at a time.

    ```
    a = 1;
    for ( b = 0 ; b < 5 ; b ++)
    ```

    can be written as

    ```
    for ( a = 1 ,b = 0 ; b < 5 ; b ++)
    ```

    - Like the initial condition all other loop variables can also have more than one variable and all of them will be separated by a **comma.**

.

2. The test condition may also have a compound statement in it like

    ```
    sum = 0;
    for(a = 0 ; a < 10 && sum <100 ; a++)
            sum = sum + a ;
    ```

3. It is also permissible to use expressions in the initialization and increment part of the loop

```
for ( a = (c+d) /2 ; a> 30 ; a = a/2)
```

4. A unique part of for loop is one or more sections can be omitted if needed

```
a= 5;
for( ; a != 100 ; )
{
printf("%d",a) ;
a = a+ 2 ;
}
```

- The initial and the increment conditions are omitted, but the conditions are given separately.
- If the conditions are not given then it becomes an infinite loop and can be terminated using a break or a goto statement.

5. A for loop can also be written to give a time delay and such kind of loops are called as null loop.

```
for( a = 1000 ; a > 0 ; a --);
```

- As this loop ends with a semicolon and does not have any statements in it, it will be executed for 1000 times.

# 3. Arrays and String

3.1 Arrays Declaration and initialization of one dimensional, two Dimensional and character arrays, accessing array elements.

3.2 Declaration and initialization of string variables, string handling functions from standard library – strlen(), strcpy(), strcat(), strcmp()

## 3.1 Arrays Declaration and initialization of one dimensional, two Dimensional and character arrays, accessing array elements.

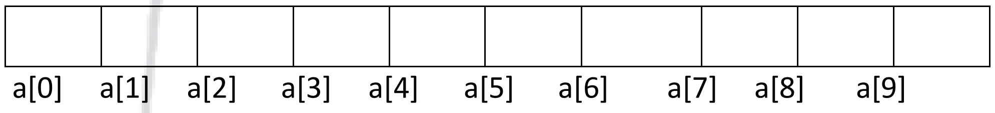
1. An **array** is a group of data items belong to same type and same storage class that share a common name.
2. An individual value in a group is called an **element**. The number of **subscripts/index** determines the dimensionality i.e. the number of elements in an array.
3. An array with one subscript is called as **single – dimension array**.
4. Each subscript must be a non-negative integer.
5. If a programmer wants to find the sum of 10 numbers then it is very tedious to declare 10 variables and then add them.
6. For this we declare an array which can store 10 numbers. The general syntax is :

   **datatype variable-name [size];**

   And for the above example it can be written as

   **int** a[10];

   For the above declaration the computer reserves 10 storage locations as shown



7. In the memory it is allocated from 0 to 9. An integer variable takes two bytes in the memory and a character variable takes one byte in the memory.

   Arrays can be initialized as

   Int a[5] = {1,2,3,4,5};

**PROGRAM 18**

Write a program to **find the sum of 10 numbers**.

```
#include<stdio.h>
#include<conio.h>
main( )
{
        int  i ,a[10] , sum = 0;
        clrscr( );
        printf(“Enter 10 numbers ➔”);
        for( i = 0 ; i < 10 ; i++)
                scanf (“%d”,&a[i]);
        for( i = 0 ; i < 10 ; i++)
                sum = sum + a[i] ;
        for( i = 0 ; i < 10 ; i++)
                printf (“\nThe 10 numbers are %d “, a[i]);
        printf(“\n the sum of 10 numbers is ➔ %d “,sum);
        getch( );
}
```

**Explanation**

i. We use the first for loop to enter 10 numbers and store in an array.

ii. The second for loop is used to find the sum of 10 numbers which was entered and stored in the array a[ ]. And the third for loop is used to printf the numbers.

The **output** of the program is :

```
Enter 10 numbers ➔ 23
56
34
22
12
58
11
89
20
88
The 10 numbers are  23 56 34 22 12 58 11 89 20 88
the sum of 10 numbers is ➔ 413
```

**PROGRAM 19**

Write a program to sort the numbers in ascending order.

```
#include<stdio.h>
#include<conio.h>
main( )
{
    int i ,j ,num[5] ;
            for( i = 0 ; i <5 ; i ++)
        a[i] = 0;
    printf(“Enter the elements of the list ➔”);
    for( i = 0 ; i <5 ; i ++)
        scanf(“%d”,&a[i]);
    for( i = 0 ; i <4 ; i ++)
```

```
              {
                    for(j =i ; j <5 ; j ++)
                    {
                          if(a[i] > a[j])
                          {
                                temp = a[i] ;
                                a[i] = a[j];
                                a[j] = temp;
                          }
                    }
              }
        printf("\nThe sorted list is ➔");
        for( i = 0 ; i <5 ; i ++)
                    printf("%d ",a[i])
        getch( );
        }
```

The **output** is:

```
Enter the elements of the list ➔21
56
22
8
34
The sorted list is ➔8 21 22 34 56
```

**NOTE:** in the above program as we know the maximum size of the array, so rather than writing it again and again in the loop we can put it in a single statement using #define as

```
#define max 5
int a[max];
```

- So in the loop wherever we used five we can replace it with max .
- The major advantage of it is that if we want to change the size of my array than we have to just change it in the **#define**.

**TWO DIMENSIONAL ARRAY**

i. A two dimensional array is like an matrix where you have number of rows and columns. It is declared as

```
int a[2][2] ;
```

The values can be initialized as

```
int a[2][2] = {1,2,3,4};
        or
int a[2][2] = {{1,2},{3,4}};
        or
int a[2][2] = {
                {1,2},
                {3,4}
              };
```

If the values are missing they will be set to zero.

The number of elements it can take is 2*2 i.e 4

ii. The array with more than two dimensions is known as multi-dimension array.

```
Type array_name[ ][ ][ ]….[ ];
int a[2][3][4];
```

iii. This is an array which can store 2*3*4 = 24 integer type element.

**PROGRAM 20**

WAP to print the **reverse of an array**.

```
#include<stdio.h>
#include<conio.h>
main( )
{
```

```
                int i , a[20],num;
                printf("Total numbers in an  array are ➔");
                scanf("%d", &num);
printf("\nEnter the numbers in an array ➔");
                for(i = 0 ;i <num;i++)
                        scanf("%d",&a[i]);
                printf("\nThe reverse of an array is ➔ ");
                for(i = num-1 ; i > 0 ; i--)
                        printf("%d  ",a[i]);
                getch( );
    }
```

The **output** is:

```
Total numbers in an array are ➔5
Enter the numbers in an array ➔25
67
34
89
12
The reverse of an array is ➔12 89 34 67 25
```

**PROGRAM 21**

WAP to enter elements in a **two dimension array** and print it.

```
#include<stdio.h>
#include<conio.h>
main( )
{
        int i ,j , a[20], num;
        printf("Total numbers in an  matrix are ➔");
        scanf("%d", &num);
        printf("\nEnter the numbers in an matrix ➔");
        for(i = 0 ;i <num;i++)
        {
                for(j = 0 ;j <num;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }
        printf("\nThe matrix is ➔ ");
        for(i = 0 ;i <num;i++)
        {
                for(j = 0 ;j <num;j++)
                {
                        printf("%d  ",a[i][j]);
                }
        }
        getch( );
        }
```

**PROGRAM 22**

WAP to the product of the multiplication of two matrices.

**PROGRAM 23**

WAP to print the lower triangle of a matrix.

**PROGRAM 24**

WAP to print the diagonal of a matrix.

## 3.2 Declaration and initialization of string variables, string handling functions from standard library – strlen(), strcpy(), strcat(), strcmp()

1. A string is an array of characters. Any characters marked between the double quotes are printed as string.
   “welcome to C programming”
2. We have seen that %c is used to enter a single character into the variable.
3. If we want to enter someone’s name or the details of a company name or address than we declare it as string.
   char string[size];

```
#include<stdio.h>
main( )
{
        char name[12];
        printf(“Enter your name ➔ ”);
        scanf(“%s”,name);
        printf(“\nThe name is ➔”, name);
        getch( );
}
```

The **output** is:

```
Enter your name ➔Anuradha Bhatia
The name is ➔Anuradha Bhatia
```

**Explanation**

i. In the above program name is declared as an array of 12 characters, which means that it can hold a string whose length is 12.
ii. When we enter the name as Harry Potter, we observe that the output shows only the first name as Harry and discards whatever is written after the space.
iii. The **disadvantage** of **scanf** is that it terminates the input after the first white space. White spaces are blanks, carriage return, enter, form feed, newline.
iv. But if the name is separated with a special character like **"-"** or **"_"** than it accepts it as a single string.

**NOTE:** Whatever is the size of the string available it takes one character less as the last character is the null character i.e '\0'.

```
char jest[5] = { 'G','O','O','D','\0'}
```

**Reading a line of text** from the keyboard we use **getchar( )**.

1. We can use this function to read successive single characters from the input and place them into a character array.
2. So an entire line of text can be read and stored in an array.
3. The reading is terminated when the new line character '\n' is entered and the null character is inserted at the end of the string.

**PROGRAM 25**

WAP to **input a line of text** from the keyboard.

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

```
main( )
{
        char text[81] , ch;
        int num = 0;
printf("Enter ur text ➔");
do
{
        text[num] = getchar( );
        num++;
}while(ch !='\n');
num = num-1;
text[num] = '\0';
printf("\nThe output string is ➔%s\n",text);
}
```

The **output** is:

```
Enter your text ➔ string handling in C
The output string is ➔ string handling in C
```

**Explanation**

i. In the above program a character array text with size 81 is declared which means that it can take maximum up to 80 characters and 81 is the null character.
ii. **ch** is the variable which accepts and stores the single character from the getchar( ) function.
iii. This is than stored in the array text and each time the index gets incremented, till a return is pressed.
iv. Here num – 1 is given so the last character should be null.

**Writing a line of text**: <u>Printf</u> is used to print the line of text with an %s format, it uses an array to print the line which is ended with a null character.

- We can also specify the precision for the %s.

```
char state[15] = “Andaman Nicobar”
1. printf(“%15s”,state”);
```

```
Andaman Nicobar
```

```
2. printf(“%8s”,state”);
```

```
Andaman Nicobar
```

```
3. printf(“%15.5s”,state”);
```

```
          Andam
```

**PROGRAM 26**

WAP to **print the** following **pattern**

```
P
PR
PRO
PROG
PROGR
PROGRA
PROGRAM
```

```
#include<stdio.h>
#include<conio.h>
        main( )
        {
        int loop1,loop2;
        char  str[] = “PROGRAM”;
        for(loop1 = 0 ; loop1<=6;loop1++)
        {
                loop2 = loop1 + 1;
                printf(“%-7.*s\n “, loop2 , str);
        }
        getch( );
        }
```

**Important**

1. To print the **ASCII value of a character** we write it as

   a =’b’;

   printf(“%d”,a);

   The above statement will print the number 98 on to the screen.
2. It also performs arithematic operations on the character constants and variables like

   a = ‘z’ – 1;

   In ASCII the value of ‘z’ is 122 and therefore the statement will assign the value 121 to a
3. We may also use character constants in relational expressions like

   if ( ch >=’A’ && ch <=’H’)
4. C library also supports a function which converts string constant to its integer value.

   x = **atoi**(string);

   e.g  x= atoi(“1988”);

**COMPARISON OF TWO STRINGS**

1. To compare two strings we have to use a while loop till we encounter a null character in both the strings.
2. Each string has to be compared character by character to to equal to each other the simple part of the code explains the same.

```
i = 0;
while(str1[i] == str2[i] && str1[i] != ‘\0’ && str2[1] != ‘\0’)
        i = i + 1;
if (str1[i] == ‘\0’ && str2[i] == ‘\0’)
```

```
                    printf("strings are equal");
            else
                    printf("strings are not equal ");
```

**STRING HANDLING FUNCTIONS OF C**

1. C library supports large number of string functions and some of the commonly used functions are:
   i. **strcat( )** ➔ **concatenates** two strings.
   ii. **strcmp( )** ➔ **compares** two strings.
   iii. **strcpy( )** ➔ **copies** one string over another .
   iv. **strlen( )** ➔ **finds the length** of the string .

i. **strcat( ) Function**

➢ This function is used to join two strings together.

strcat (string1 , string 2);

➢ It copies the content of string2 to string1 by removing the null character from the end of string1 but the contents of string2 remains unchanged.

string1:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| P | R | O | G | R | A | M | | \0 |

string2:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| W | R | I | T | I | N | G | \O |

After the strcat( ) function

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P | R | O | G | R | A | M | | W | R | I | T | I | N | G | \0 |

And string2 remains the same

ii. **strcmp( ) Function**

➢ This function compares two strings identified by the argument and return the value 0 with they are equal otherwise it returns the numeric difference between the first nonmatching character in the string.

strcmp(string1 , string2);

- String1 and string2 may be string variables or string constant loke :

  strcmp(name1,name2) ;

  strcmp(name1,"face") ;

  strcmp("reel","real") ;

- The value of mismatch is really important eg.

  strcmp("reel",real");

- The ASCII value of e is 101 and the ASCII value of a is 97, and thus it will give the value 4, as the answer is nonzero the strings are not equal.

iii. **strcpy( ) Function**

- This function as the name tells copies one string to another.

  strcpy( string1,string2);

- Copies the content of string2 to string1. String2 can be a character array variable or a constant.

iv. **strlen( ) Function:**

- This function counts and returns the length of the string.

  len = strlen(string);

  Where len is an integer variable.

**PROGRAM 27**

WAP to **find the length** of a **string**.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main( )
{
        int len ;
```

```
            char a[10];
            printf(“Enter the string ➔”);
            scanf(“%s “, a);
            len = strlen(a);
            printf(“\n The length of the string is ➔ %d ”,len);
            getch( );
        }
```

The **output** of:

```
Enter the string ➔second
The length of the string is ➔6
```

**PROGRAM 28**

WAP to **count the number of vowels** in a string.

**PROGRAM 29**

WAP to check whether the given **string is a palindrome or not** (madam is a palindrome)

1. Enter the string.
2. Find the length of the string.
3. Copy the string in the reverse order to another string array.
4. Check whether both are same.
5. If yes print it as palindrome else print not a palindrome.

# 4 Functions & Structures

4.1 Functions: - Need of functions, scope and lifetime of variables, defining functions, function call, call by value, call by reference, return values, storage classes.

4.2 Category of function - No argument No return value, No argument with return value, argument with return value, recursion, command line arguments

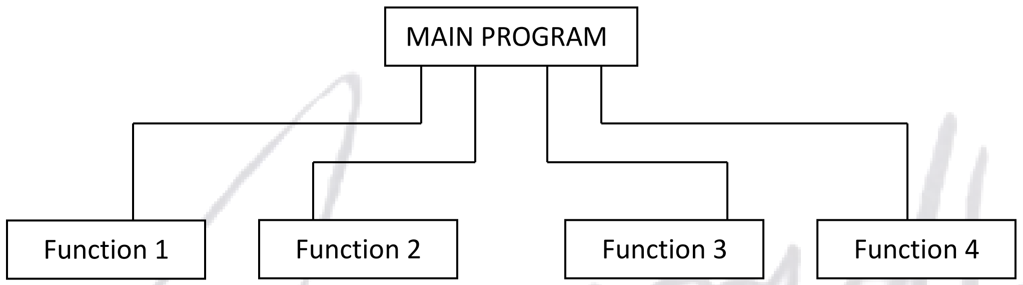
4.3 Structures: - Defining structure, declaring and accessing structure members, initialization of structure, arrays of structure.

## 4.1 Functions: - Need of functions, scope and lifetime of variables, defining functions, function call, call by value, call by reference, return values, storage classes.

1. Functions are small programs with a specific task to be performed.
2. These are small subprograms.
3. Functions are part of the main ( ) program itself.
4. When a function is called only the control is transferred to the called function, the required processing is done and then the control is transferred back to the main ( ) function for further processing.
5. The important function of C is main ( ).
6. A function definition in C programming consists of a function header and a function body. Here are all the parts of a function −
    i. **Return Type** − A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.
    ii. **Function Name** − This is the actual name of the function. The function name and the parameter list together constitute the function signature.
    iii. **Parameters** − A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
    iv. **Function Body** − The function body contains a collection of statements that define what the function does.

**The advantages of functions are**:

1. It facilitates top-down modular programming approach i.e. the main () function is solved first and then the subprograms are executed.
2. The length of the source program can be reduced by using functions, as the repeated code can be written in a function and called whenever required.
3. It is easy to debug the program.
4. It can be used by many programs.



**TOP DOWN MODULAR PROGRAMMING**

The syntax of the function is:

```
function-name(argument list)
argument declaration ;
{
        local variable declaration ;
        body of the function ;
        _________________;
        _________________;
        return(expression);
}
```

1. Function name is compulsory, the argument list may be present or absent depending on the need of the function.

```
int max(int num1, int num2) {
  int result;
   if (num1 > num2)
     result = num1;
  else
     result = num2;
   return result;
}
```

2. Local variables are variables whose life is only till that function and cannot be used after that. Body of the function consists of the executable statements.
3. The return statement returns the expression or the value needed.
4. If not returns indicates that no value is being returned to the calling function.
5. The function are called in two ways as Call by Value and Call by Reference.

## 4.2 Category of function - No argument No return value, No argument with return value, argument with return value, recursion, command line arguments

1. **Call by value**: The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function.
   i. This method copies the actual value of an argument into the formal parameter of the function.
   ii. Changes made to the parameter inside the function have no effect on the argument.
   iii. The parameter inside the function have no effect on the argument.
   iv. The function by passing the values to the function.
   v. As Sum(10,20) . Calling the function Sum by passing the values 10 and 20 to the variables in the function defined.

```
# include<stdio.h>
void sum(int, int);
void sum(int x , int y)
    {
            int temp;
            temp = x+y;
            printf(“the sum is ➔%d”temp);
        }
void main( )
{
    sum(10,20);
}
```

2. **Call by reference:** this is used to call functions by calling them using pointer variables .It is discussed in pointers.

   i. This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call.
   ii. This means that changes made to the parameter affect the argument.The call by reference method of passing arguments to a function copies the address of an argument into the formal parameter.
   iii. Inside the function, the address is used to access the actual argument used in the call. It means the changes made to the parameter affect the passed argument.
   iv. To pass a value by reference, argument pointers are passed to the functions just like any other value.
   v. To declare the function parameters as pointer types as in the following function swap(), which exchanges the values of the two integer variables pointed to, by their arguments.

```
void swap(int *x, int *y) {
  int temp;
```

```
    temp = *x;    /* save the value at address x */
    *x = *y;      /* put y into x */
    *y = temp;    /* put temp into y */
    return;
  }
```

3. **Recursion**

   i. Recursion is the process of repeating items in a self-similar way.

   ii. When a program allows you to call a function inside the same function, then it is called a recursive call of the function.

   iii. The structure of recursion is

```
void recursion() {
   recursion(); /* function calls itself */
}

int main() {
   recursion();
}
```

   iv. Recursion is more elegant and requires few variables which make program clean.

   v. Recursion can be used to replace complex nesting code by dividing the problem into same problem of its sub-type.

   vi. It is hard to think the logic of a recursive function. It is also difficult to debug the code containing recursion

## 4.3 Structures: - Defining structure, declaring and accessing structure members, initialization of structure, arrays of structure.

1. Arrays allow to define type of variables that can hold several data items of the same kind.
2. Structure is another user defined data type available in C that allows to combine data items of different kinds.
3. Structures are used to represent a record.
4. To keep track of your books in a library.

   Title

   Author

   Subject

   Book ID

**I. Defining a Structure**

1. To define a structure, you must use the struct statement.
2. The struct statement defines a new data type, with more than one member.
3. The format of the struct statement is as follows −

```
struct [structure tag]
{
  member definition;
  member definition;
  ...
  member definition;
} [one or more structure variables];
```

4. The structure tag is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition.
5. At the end of the structure's definition, before the final semicolon, specify one or more structure variables but it is optional.
6. Here is the way you would declare the Book structure −

```
struct Books
```

```
{
  char  title[50];
  char  author[50];
  char  subject[100];
  int   book_id;
} book;
```

**II. Accessing Structure Members**

1. To access any member of a structure, we use the member access operator (.).
2. The member access operator is coded as a period between the structure variable name and the structure member that we wish to access.
3. The keyword struct to define variables of structure type.
4. The following example shows how to use a structure in a program −

```
#include <stdio.h>
#include <string.h>
struct Books {
  char  title[50];
  char  author[50];
  char  subject[100];
  int   book_id;
};
void printBook( struct Books *book );
int main( )
{
  struct Books Book1;
  struct Books Book2;
  strcpy( Book1.title, "SPA");
  strcpy( Book1.author, "Anuradha Bhatia");
  strcpy( Book1.subject, "C Programming ");
  Book1.book_id = #120100;

  strcpy( Book2.title, " Object Oriented Programming ");
  strcpy( Book2.author, "Yaswant Kanitkar");
  strcpy( Book2.subject, "C++");
  Book2.book_id = #121100;
  printBook( &Book1 );
  printBook( &Book2 );
  return 0;
}
void printBook( struct Books *book ) {

  printf( "Book title : %s\n", book->title);
  printf( "Book author : %s\n", book->author);
  printf( "Book subject : %s\n", book->subject);
  printf( "Book book_id : %d\n", book->book_id);
}
```

The Output is

```
   Book title : SPA
   Book author : Anuradha Bhatia
   Book subject : C programming
   Book book_id : #120100
   Book title : Object Oriented Programming
   Book author : Yaswant Kanitkar
   Book subject : C++
   Book book_id : #121100
```

**III. Structures as Function Arguments**

1. Structure as a function argument in the same way as you pass any other variable or pointer.

```
#include <stdio.h>
#include <string.h>
struct Books
{
        char  title[50];
        char  author[50];
        char  subject[100];
        int   book_id;
};
void printBook( struct Books book );
int main( )
{
  struct Books Book1;       /* Declare Book1 of type Book */
  struct Books Book2;       /* Declare Book2 of type Book */
   /* book 1 specification */
    strcpy( Book1.title, "SPA");
        strcpy( Book1.author, "Anuradha Bhatia");
        strcpy( Book1.subject, "C Programming ");
        Book1.book_id = #120100;
        strcpy( Book2.title, " Object Oriented Programming ");
        strcpy( Book2.author, "Yaswant Kanitkar");
        strcpy( Book2.subject, "C++");
        Book2.book_id = #121100;
        /* print Book1 info */
        printBook( Book1 );
        /* Print Book2 info */
        printBook( Book2 );
return 0;
}
void printBook( struct Books book ) {
  printf( "Book title : %s\n", book.title);
  printf( "Book author : %s\n", book.author);
  printf( "Book subject : %s\n", book.subject);
  printf( "Book book_id : %d\n", book.book_id);
}
```

The output is:

```
    The Output is
    Book title : SPA
    Book author : Anuradha Bhatia
    Book subject : C programming
    Book book_id : #120100
    Book title : Object Oriented Programming
    Book author : Yaswant Kanitkar
    Book subject : C++
    Book book_id : #121100
```

# 5 Pointers & Files

5.1 Understanding pointers, declaring pointer variable, initialization of pointer variable, accessing address of a variable, pointer expressions, Pointers arithmetic

5.2 Working with Files

## 5.1 Understanding pointers, declaring pointer variable, initialization of pointer variable, accessing address of a variable, pointer expressions, Pointers arithmetic

1. A pointer is a variable that contains the address of a variable.
2. Pointers are much used in partly because they are sometimes the only way to express a computation, and partly because they usually lead to more compact and efficient code than can be obtained in other ways.
3. Pointers and arrays are closely related; this chapter also explores this relationship and shows how to exploit it.
4. Pointers have been lumped with the goto statement as a marvelous way to create impossible to- understand programs.
5. This is certainly true when they are used carelessly, and it is easy to create pointers that point somewhere unexpected. With discipline, however, pointers can also be used to achieve clarity and simplicity.
6. This is the aspect that we will try to illustrate.
7. The main change in ANSI C is to make explicit the rules about how pointers can be manipulated, in effect mandating what good programmers already practice and good compilers already enforce. In addition, the type void * (pointer to void) replaces char * as the proper type for a generic pointer.

### I. Pointers and Addresses

1. A typical machine has an array of consecutively numbered or addressed memory cells that may be manipulated individually or in contiguous groups.
2. One common situation is that any byte can be a char, a pair of one-byte cells can be treated as a short integer, and four adjacent bytes form a long.
3. A pointer is a group of cells (often two or four) that can hold an address.

4. So if c is a char and p is a pointer that points to it, we could represent the situation this way:
5. The unary operator & gives the address of an object, so the statement p = &c; assigns the address of c to the variable p, and p is said to ``point to'' c.
6. The & operator only applies to objects in memory: variables and array elements.
7. It cannot be applied to expressions, constants, or register variables.
8. The unary operator * is the *indirection* or *dereferencing* operator; when applied to a pointer, it accesses the object the pointer points to. Suppose that x and y are integers and ip is a pointer to int.
9. This artificial sequence shows how to declare a pointer and how to use & and *:

```
int x = 1, y = 2, z[10];
int *ip;                    /* ip is a pointer to int */
ip = &x;                    /* ip now points to x */
y = *ip;                    /* y is now 1 */
*ip = 0;                    /* x is now 0 */
ip = &z[0];                 /* ip now points to z[0] */
```

10 The declaration of x, y, and z are what we've seen all along. The declaration of the pointer ip, int *ip; is intended as a mnemonic; it says that the expression *ip is an int.

11 The syntax of the declaration for a variable the syntax of expressions in which the variable might appear.

12. This reasoning applies to function declarations as well.
13. For example,
    double *dp, atof(char *);
14. In an expression *dp and atof(s) have values of double, and that the argument of atof is a pointer to char.

15. If ip points to the integer x, then *ip can occur in any context where x could, so

```
*ip = *ip + 10;
```

increments *ip by 10.

16. The unary operators * and & bind more tightly than arithmetic operators, so the assignment

```
y = *ip + 1
```

takes whatever ip points at, adds 1, and assigns the result to y, while

```
*ip += 1
```

increments what ip points to, as do

```
++*ip
```

And (*ip) ++

17. The parentheses are necessary in this last example; without them, the expression would increment ip instead of what it points to, because unary operators like * and ++ associate right to left.
18. Finally, since pointers are variables, they can be used without dereferencing. For example, if iq is another pointer to int, iq = ip copies the contents of ip into iq, thus making iq point to whatever ip pointed to.

## II. Pointers and Function Arguments

1. Since C passes arguments to functions by value, there is no direct way for the called function to alter a variable in the calling function.
2. For instance, a sorting routine might exchange two out-of-order arguments with a function called swap. It is not enough to write

```
swap(a, b);
void swap(int x, int y) /* WRONG */
{
int temp;
temp = x;
x = y;
y = temp;
}
```

3. Because of call by value, swap can't affect the arguments a and b in the routine that called it. The function above swaps *copies* of a and b.
4. The way to obtain the desired effect is for the calling program to pass *pointers* to the values to be changed: swap(&a, &b);
5. Since the operator & produces the address of a variable, &a is a pointer to a. In swap itself, the parameters are declared as pointers, and the operands are accessed indirectly through them.

```
void swap(int *px, int *py) /* interchange *px and *py */
{
int temp;
temp = *px;
*px = *py;
*py = temp;
}
```

pointer argument to store the converted integer back in the calling function.

**III. Pointers and Arrays**

1. Any operation that can be achieved by array subscripting can also be done with pointers.
2. The pointer version will in general be faster but, atleast to the uninitiated, somewhat harder to understand.
3. The declaration int a[10]; defines an array of size 10, that is, a block of 10 consecutive objects named a[0], a[1],...,a[9].
4. The notation a[i] refers to the i-th element of the array. If pa is a pointer to an integer, declared as int *pa; then the assignment pa = &a[0];

5. Sets pa to point to element zero of a; that is, pa contains the address of a[0]. Now the assignment x = *pa; will copy the contents of a[0] into x.
6. If pa points to a particular element of an array, then by definition pa+1 points to the next element, pa+i points i elements after pa, and pa-i points i elements before.
7. Thus, if pa points to a[0], *(pa+1) refers to the contents of a[1], pa+i is the address of a[i], and *(pa+i) is the contents of a[i].
8. These remarks are true regardless of the type or size of the variables in the array a.
9. The meaning of ``adding 1 to a pointer,'' and by extension, all pointer arithmetic, is that pa+1 points to the next object, and pa+i points to the i-th object beyond pa.
10. The correspondence between indexing and pointer arithmetic is very close. By definition, the value of a variable or expression of type array is the address of element zero of the array.
11. Thus after the assignment pa = &a[0]; pa and a have identical values. Since the name of an array is a synonym for the location of the initial element, the assignment pa=&a[0] can also be written as pa = a;

## IV. Address Arithmetic

1. If p is a pointer to some element of an array, then p++ increments p to point to the next element, and p+=i increments it to point i elements beyond where it currently does.
2. These and similar constructions are the simples forms of pointer or address arithmetic.
3. C is consistent and regular in its approach to address arithmetic; its integration of pointers, arrays, and address arithmetic is one of the strengths of the language. Let us illustrate by

**V. Pointers to Structures**

1. Pointers to structures is declared in the same way as you define pointer to any other variable −

   struct Books *struct_pointer;

2. The address of a structure variable is now stored in the above defined pointer variable.
3. To find the address of a structure variable, place the '&'; operator before the structure's name as follows −

   struct_pointer = &Book1;

4. To access the members of a structure using a pointer to that structure, you must use the → operator as follows −

   struct_pointer->title;

**Example:**

```
#include <stdio.h>
#include <string.h>
struct Books {
  char  title[50];
  char  author[50];
  char  subject[100];
  int   book_id;
};

void printBook( struct Books *book );
int main( ) {

  struct Books Book1;
  struct Books Book2;
  strcpy( Book1.title, "SPA");
  strcpy( Book1.author, "Anuradha Bhatia");
```

```
    strcpy( Book1.subject, "C Programming ");
    Book1.book_id = #120100;

    /* book 2 specification */
    strcpy( Book2.title, " Object Oriented Programming ");
    strcpy( Book2.author, "Yaswant Kanetkar");
    strcpy( Book2.subject, "C++");
    Book2.book_id = #121100;
    printBook( &Book1 );
    printBook( &Book2 );
    return 0;
}

void printBook( struct Books *book )
{
    printf( "Book title : %s\n", book->title);
    printf( "Book author : %s\n", book->author);
    printf( "Book subject : %s\n", book->subject);
    printf( "Book book_id : %d\n", book->book_id);
}
```

The **Output** is

```
Book title : SPA
Book author : Anuradha Bhatia
Book subject : C programming
Book book_id : #120100
Book title : Object Oriented Programming
Book author : Yashwant kanetkar
Book subject : C++
```

Book book_id : #121100

**Program 30**

WAP to copy one string to another using pointers

```
#include<stdio.h>
#include<conio.h>
void main()
{
        char  str1[10],str2[10],*ptr1,*ptr2;
        clrscr();
        printf("enter a string➔ ");
        scanf("%s",str1);
        ptr1=str1;
        ptr2=str2;
        while(*ptr1!='\0')
        {
                *ptr2=*ptr1;
                ptr2++;
                ptr1++;
        }
*ptr2='\0';
cout<<"string copied:"<<str2;
getch();
}
```

**Program 31**

WAP to find the number of vowels in a string

```
#include<iostream.h>
#include<conio.h>
void main()
{
        char  str[10],*ptr;
        int count=0,i;
        printf("enter a string➔ ");
        scanf("%s",str);
        for(ptr=str;*ptr!='\0';ptr++)
        {
        if(*ptr=='a'||*ptr=='e'||*ptr=='i'||*ptr=='o'||*ptr=='u')
        count++;
        }
        cout<<"mumber of vowels are:"<<count;
        getch();
}
```

**Program 32**

WAP to find the length of the string

```
#include<iostream.h>
#include<conio.h>
void main()
{
char  str[10],*ptr;
        int i,count=0;
        printf("enter a string➔ ";
        scanf("%s" ,str) ;
        for(ptr=str;*ptr!='\0';ptr++)
        {
                count++;
        }
        while(count>=0)
        {
                cout<<*ptr;
                ptr--;
                count--;
        }
        getch();
}
```

**Program 33**

WAP to find a character in a string

```
#include<iostream.h>
#include<conio.h>
void main()
{
        char  str1[10],ch,*p;
        clrscr();
        cout<<"enter a string: ";
        cin>>str1;
        p=str1;
        cout<<"enter character to be searched: ";
        cin>>ch;
        while(*p!='\0')
        {
                if(*p==ch)
                {
                        cout<<"character found";
                        break;
                }
                p++;
        }
        if(*p=='\0')
                cout<<"not found";
        getch();
}
```

## 5.2 Working with Files

1. When the program is terminated, the entire data is lost in C programming.
2. To keep large volume of data, it is time consuming to enter the entire data.
3. File is created, these information can be accessed using few commands.
4. There are large numbers of functions to handle file I/O in C language. In this tutorial, you will learn to handle standard I/O(High level file I/O functions) in C.

High level file I/O functions can be categorized as:

1. Text file
2. Binary file

File Operations

1. Creating a new file
2. Opening an existing file
3. Reading from and writing information to a file
4. Closing a file

**I. Working with file**

i. While working with file, you need to declare a pointer of type file. This declaration is needed for communication between file and program.

```
FILE *ptr;
```

**II. Opening a file**

i. Opening a file is performed using library function fopen(). The syntax for opening a file in standard I/O is:

```
ptr=fopen("fileopen","mode")
```

For Example:

```
fopen("E:\\cprogram\program.txt","w");
```

E:\\cprogram\program.txt is the location to create file.

"w" represents the mode for writing.

Here, the program.txt file is opened for writing mode.

| Opening Modes in Standard I/O | | |
|---|---|---|
| **File Mode** | **Meaning of Mode** | **During Inexistence of file** |
| r | Open for reading. | If the file does not exist, fopen() returns NULL. |
| w | Open for writing. | If the file exists, its contents are overwritten. If the file does not exist, it will be created. |
| a | Open for append. i.e, Data is added to end of file. | If the file does not exists, it will be created. |
| r+ | Open for both reading and writing. | If the file does not exist, fopen() returns NULL. |
| w+ | Open for both reading and writing. | If the file exists, its contents are overwritten. If the file does not exist, it will be created. |
| a+ | Open for both reading and appending. | If the file does not exists, it will be created. |

**III. Closing a File**

i. The file should be closed after reading/writing of a file. Closing a file is performed using library function fclose().

```
fclose(ptr);
```

**IV. The Functions fprintf() and fscanf() functions.**

i. The functions fprintf() and fscanf() are the file version of printf() and fscanf(). The only difference while using fprintf() and fscanf() is that, the first argument is a pointer to the structure FILE.

**PROGRAM 34**

WAP to print the content in to a file.

```
#include <stdio.h>
int main()
{
  int n;
  FILE *fptr;
  fptr=fopen("C:\\program.txt","w");
  if(fptr==NULL){
    printf("Error!");
    exit(1);
  }
  printf("Enter n: ");
  scanf("%d",&n);
  fprintf(fptr,"%d",n);
  fclose(fptr);
  return 0;
}
```

This program takes the number from user and stores in file. After you compile and run this program, you can see a text file program.txt created in C drive of your computer. When you open that file, you can see the integer you entered. Similarly, fscanf() can be used to read data from file.

**PROGRAM 35**

WAP to Read from file

```
#include <stdio.h>
int main()
{
  int n;
  FILE *fptr;
  if ((fptr=fopen("C:\\program.txt","r"))==NULL){
      printf("Error! opening file");
      exit(1);        /* Program exits if file pointer returns NULL. */
  }
  fscanf(fptr,"%d",&n);
  printf("Value of n=%d",n);
  fclose(fptr);
  return 0;
}
```

## 2. Binary Files

i. Depending upon the way file is opened for processing, a file is classified into text file and binary file.

ii. If a large amount of numerical data it to be stored, text mode will be insufficient. In such case binary file is used.

iii. Working of binary files is similar to text files with few differences in opening modes, reading from file and writing to file.

**I. Opening modes of binary files**

i. Opening modes of binary files are rb, rb+, wb, wb+,ab and ab+.

ii. The only difference between opening modes of text and binary files is that, b is appended to indicate that, it is binary file.

**II. Reading and writing of a binary file.**

i. Functions fread() and fwrite() are used for reading from and writing to a file on the disk respectively in case of binary files.

ii. Function fwrite() takes four arguments, address of data to be written in disk, size of data to be written in disk, number of such type of data and pointer to the file where you want to write.

fwrite(address_data,size_data,numbers_data,pointer_to_file);

iii. Function fread() also take 4 arguments similar to fwrite() function as above.

**PROGRAM 36**

WAP to open a file and to print it contents on screen.

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

int main()
{
   char ch, file_name[25];
```

```
    FILE *fp;

    printf("Enter the name of file you wish to see\n");
    gets(file_name);

    fp = fopen(file_name,"r"); // read mode

    if( fp == NULL )
    {
      perror("Error while opening the file.\n");
      exit(EXIT_FAILURE);
    }

    printf("The contents of %s file are :\n", file_name);

    while( ( ch = fgetc(fp) ) != EOF )
      printf("%c",ch);

    fclose(fp);
    return 0;
  }
```

**PROGRAM 37**

WAP to read name and marks of n number of students from user and store them in a file

```
  #include <stdio.h>
  int main(){
    char name[50];
    int marks,i,n;
    printf("Enter number of students: ");
```

```
        scanf("%d",&n);
        FILE *fptr;
        fptr=(fopen("C:\\student.txt","w"));
        if(fptr==NULL){
           printf("Error!");
           exit(1);
        }
        for(i=0;i<n;++i)
        {
          printf("For student%d\nEnter name: ",i+1);
          scanf("%s",name);
          printf("Enter marks: ");
          scanf("%d",&marks);
          fprintf(fptr,"\nName: %s \nMarks=%d \n",name,marks);
        }
        fclose(fptr);
        return 0;
      }
```

**PROGRAM 38**

WAP to read name and marks of n number of students from user and store them in a file. If the file previously exits, add the information of n students.

```
#include <stdio.h>
int main(){
  char name[50];
  int marks,i,n;
  printf("Enter number of students: ");
  scanf("%d",&n);
  FILE *fptr;
```

```
    fptr=(fopen("C:\\student.txt","a"));
    if(fptr==NULL){
        printf("Error!");
        exit(1);
    }
    for(i=0;i<n;++i)
    {
      printf("For student%d\nEnter name: ",i+1);
      scanf("%s",name);
      printf("Enter marks: ");
      scanf("%d",&marks);
      fprintf(fptr,"\nName: %s \nMarks=%d \n",name,marks);
    }
    fclose(fptr);
    return 0;
  }
```

**PROGRAM 39**

WAP to write all the members of an array of structures to a file using fwrite(). Read the array from the file and display on the screen.

```
#include <stdio.h>
struct s
{
char name[50];
int height;
};
int main(){
    struct s a[5],b[5];
    FILE *fptr;
    int i;
    fptr=fopen("file.txt","wb");
    for(i=0;i<5;++i)
    {
        fflush(stdin);
        printf("Enter name: ");
        gets(a[i].name);
        printf("Enter height: ");
        scanf("%d",&a[i].height);
    }
    fwrite(a,sizeof(a),1,fptr);
    fclose(fptr);
    fptr=fopen("file.txt","rb");
    fread(b,sizeof(b),1,fptr);
    for(i=0;i<5;++i)
    {
        printf("Name: %s\nHeight: %d",b[i].name,b[i].height);
    }
    fclose(fptr);
}
```

# 6 Programs

**PROGRAM 40**

WAP to reverse a number.

```
#include <stdio.h>
 int main()
{
   int n, reverse = 0;
   printf("Enter a number to reverse\n");
   scanf("%d", &n);
   while (n != 0)
   {
      reverse = reverse * 10;
      reverse = reverse + n%10;
      n      = n/10;
   }
   printf("Reverse of entered number is = %d\n", reverse);
   return 0;
}
```

**PROGRAM 41**

WAP to check whether a given is a palindrome or not.

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

**PROGRAM 42**

WAP to print the diamond pattern in c

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

```
#include <stdio.h>
 int main()
{
  int n, c, k, space = 1;
   printf("Enter number of rows\n");
  scanf("%d", &n);
   space = n - 1;
   for (k = 1; k <= n; k++)
  {
    for (c = 1; c <= space; c++)
      printf(" ");
     space--;
     for (c = 1; c <= 2*k-1; c++)
      printf("*");
     printf("\n");
  }
   space = 1;
   for (k = 1; k <= n - 1; k++)
  {
    for (c = 1; c <= space; c++)
      printf(" ");
     space++;
     for (c = 1 ; c <= 2*(n-k)-1; c++)
      printf("*");
     printf("\n");
  }
  return 0;
}
```

**PROGRAM 43**

WAP to check whether a number is prime or not.

```
#include<stdio.h>
 int main()
{
   int n, i = 3, count, c;
    printf("Enter the number of prime numbers required\n");
   scanf("%d",&n);
    if ( n >= 1 )
   {
      printf("First %d prime numbers are :\n",n);
      printf("2\n");
   }
    for ( count = 2 ; count <= n ;  )
   {
      for ( c = 2 ; c <= i - 1 ; c++ )
      {
         if ( i%c == 0 )
            break;
      }
      if ( c == i )
      {
         printf("%d\n",i);
         count++;
      }
      i++;
   }
    return 0;
}
```

**PROGRAM 44**

WAP to check whether a number is an armstrong number.

Examples:

7 = 7^1

371 = 3^3 + 7^3 + 1^3 (27 + 343 +1)

8208 = 8^4 + 2^4 +0^4 + 8^4 (4096 + 16 + 0 + 4096).

```
#include <stdio.h>
 int power(int, int);
 int main()
{
   int n, sum = 0, temp, remainder, digits = 0;
    printf("Input an integer\n");
   scanf("%d", &n);
    temp = n;
   // Count number of digits
   while (temp != 0) {
      digits++;
      temp = temp/10;
   }
    temp = n;
    while (temp != 0) {
      remainder = temp%10;
      sum = sum + power(remainder, digits);
      temp = temp/10;
   }
    if (n == sum)
      printf("%d is an Armstrong number.\n", n);
   else
      printf("%d is not an Armstrong number.\n", n);
    return 0;
}
 int power(int n, int r) {
   int c, p = 1;
    for (c = 1; c <= r; c++)
      p = p*n;
    return p;
}
```

**PROGRAM 45**

WAP to print Fibonacci Series.

```
#include<stdio.h>
 int 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);
   }
    return 0;
}
```

**PROGRAM 46**

WAP to print floyd’s triangle.

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

```
#include <stdio.h>
 int main()
{
  int n, i,  c, a = 1;
   printf("Enter the number of rows of Floyd's triangle to print\n");
  scanf("%d", &n);
   for (i = 1; i <= n; i++)
  {
    for (c = 1; c <= i; c++)
    {
      printf("%d ",a);
      a++;
    }
    printf("\n");
  }
   return 0;
}
```

**PROGRAM 47**

WAP to print pascal triangle.

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

```
#include <stdio.h>
 long factorial(int);
 int main()
{
   int i, n, c;
    printf("Enter the number of rows you wish to see in pascal triangle\n");
   scanf("%d",&n);
    for (i = 0; i < n; i++)
   {
      for (c = 0; c <= (n - i - 2); c++)
         printf(" ");
       for (c = 0 ; c <= i; c++)
         printf("%ld ",factorial(i)/(factorial(c)*factorial(i-c)));
       printf("\n");
   }
    return 0;
}
 long factorial(int n)
{
   int c;
   long result = 1;
    for (c = 1; c <= n; c++)
         result = result*c;
```

```
        return result;
    }
```

**PROGRAM 48**

WAP to add two numbers using pointers.

```
#include <stdio.h>
 int main()
{
   int first, second, *p, *q, sum;
    printf("Enter two integers to add\n");
   scanf("%d%d", &first, &second);
    p = &first;
   q = &second;
    sum = *p + *q;
    printf("Sum of entered numbers = %d\n",sum);
    return 0;
}
```

**PROGRAM 49**

WAP to add two numbers using call by reference.

```
#include <stdio.h>
 long add(long *, long *);
 int main()
{
   long first, second, *p, *q, sum;
    printf("Input two integers to add\n");
   scanf("%ld%ld", &first, &second);
    sum = add(&first, &second);
    printf("(%ld) + (%ld) = (%ld)\n", first, second, sum);
    return 0;
}
 long add(long *x, long *y) {
   long sum;
    sum = *x + *y;
    return sum;
}
```

**PROGRAM 50**

WAP to print maximum number of element in the array.

```
#include <stdio.h>
 int main()
{
  int array[100], maximum, size, c, location = 1;
   printf("Enter the number of elements in array\n");
  scanf("%d", &size);
   printf("Enter %d integers\n", size);
   for (c = 0; c < size; c++)
    scanf("%d", &array[c]);
   maximum = array[0];
   for (c = 1; c < size; c++)
  {
    if (array[c] > maximum)
    {
       maximum  = array[c];
       location = c+1;
    }
  }
   printf("Maximum element is present at location %d and it's value is %d.\n",
location, maximum);
  return 0;
}
```

**PROGRAM 51**

WAP to print minimum number of element in the array.

```
#include <stdio.h>
 int main()
{
   int array[100], minimum, size, c, location = 1;
    printf("Enter the number of elements in array\n");
   scanf("%d",&size);
    printf("Enter %d integers\n", size);
    for ( c = 0 ; c < size ; c++ )
      scanf("%d", &array[c]);
    minimum = array[0];
    for ( c = 1 ; c < size ; c++ )
   {
      if ( array[c] < minimum )
      {
         minimum = array[c];
         location = c+1;
      }
   }
    printf("Minimum element is present at location %d and it's value is %d.\n",
location, minimum);
   return 0;
}
```

**PROGRAM 52**

WAP to add two matrices.

```
First Matrix:-
1 2
3 4
Second matrix:-
4 5
-1 5
Sum of First and Second matrix
5 7
2 9
```

```
#include <stdio.h>
int main()
{
  int m, n, c, d, first[10][10], second[10][10], sum[10][10];
   printf("Enter the number of rows and columns of matrix\n");
  scanf("%d%d", &m, &n);
  printf("Enter the elements of first matrix\n");
   for (c = 0; c < m; c++)
     for (d = 0; d < n; d++)
        scanf("%d", &first[c][d]);
   printf("Enter the elements of second matrix\n");
   for (c = 0; c < m; c++)
     for (d = 0 ; d < n; d++)
          scanf("%d", &second[c][d]);
   printf("Sum of entered matrices:-\n");
   for (c = 0; c < m; c++) {
     for (d = 0 ; d < n; d++) {
        sum[c][d] = first[c][d] + second[c][d];
```

```
                printf("%d\t", sum[c][d]);
              }
              printf("\n");
            }
            return 0;
          }
```

**PROGRAM 53**

WAP to subtract two matrices.

```
#include <stdio.h>
 int main()
{
   int m, n, c, d, first[10][10], second[10][10], difference[10][10];
    printf("Enter the number of rows and columns of matrix\n");
   scanf("%d%d", &m, &n);
   printf("Enter the elements of first matrix\n");
    for (c = 0; c < m; c++)
     for (d = 0 ; d < n; d++)
       scanf("%d", &first[c][d]);
    printf("Enter the elements of second matrix\n");
    for (c = 0; c < m; c++)
     for (d = 0; d < n; d++)
        scanf("%d", &second[c][d]);
    printf("Difference of entered matrices:-\n");
    for (c = 0; c < m; c++) {
     for (d = 0; d < n; d++) {
       difference[c][d] = first[c][d] - second[c][d];
       printf("%d\t",difference[c][d]);
     }
     printf("\n");
    }
    return 0;
  }
```

**PROGRAM 54**

WAP to find transpose of a matix.

```
1 2
 3 4
 5 6
 then transpose of above matrix will be
 1 3 5
 2 4 6
#include <stdio.h>
 int main()
{
  int m, n, c, d, matrix[10][10], transpose[10][10];

  printf("Enter the number of rows and columns of matrix\n");
  scanf("%d%d", &m, &n);
   printf("Enter the elements of matrix\n");
   for (c = 0; c < m; c++)
     for(d = 0; d < n; d++)
        scanf("%d",&matrix[c][d]);
   for (c = 0; c < m; c++)
     for( d = 0 ; d < n ; d++ )
        transpose[d][c] = matrix[c][d];
   printf("Transpose of entered matrix :-\n");
   for (c = 0; c < n; c++) {
     for (d = 0; d < m; d++)
        printf("%d\t",transpose[c][d]);
     printf("\n");
  }
   return 0;
}
```

**PROGRAM 55**

WAP to multiply two matrix

```
#include <stdio.h>
int main()
{
  int m, n, p, q, c, d, k, sum = 0;
  int first[10][10], second[10][10], multiply[10][10];
   printf("Enter the number of rows and columns of first matrix\n");
  scanf("%d%d", &m, &n);
  printf("Enter the elements of first matrix\n");

  for (c = 0; c < m; c++)
    for (d = 0; d < n; d++)
      scanf("%d", &first[c][d]);

  printf("Enter the number of rows and columns of second matrix\n");
  scanf("%d%d", &p, &q);

  if (n != p)
    printf("Matrices with entered orders can't be multiplied with each other.\n");
  else
  {
    printf("Enter the elements of second matrix\n");

    for (c = 0; c < p; c++)
      for (d = 0; d < q; d++)
        scanf("%d", &second[c][d]);
  for (c = 0; c < m; c++) {
   for (d = 0; d < q; d++) {
    for (k = 0; k < p; k++) {
     sum = sum + first[c][k]*second[k][d];
    }

    multiply[c][d] = sum;
    sum = 0;
   }
  }

  printf("Product of entered matrices:-\n");
  for (c = 0; c < m; c++) {
   for (d = 0; d < q; d++)
    printf("%d\t", multiply[c][d]);

   printf("\n");
  }
 }

 return 0;
}
```

**PROGRAM 56**

WAP to swap two values using Call by Value

```
#include <stdio.h>
/* function declaration goes here.*/
void swap( int p1, int p2 );
int main()
{
  int a = 10;
  int b = 20;
  printf("Before: Value of a = %d and value of b = %d\n", a, b );
  swap( a, b );
  printf("After: Value of a = %d and value of b = %d\n", a, b );
}
void swap( int p1, int p2 )
{
   int t;

   t = p2;
   p2 = p1;
   p1 = t;
  printf("Value of a (p1) = %d and value of b(p2) = %d\n", p1, p2 );
}
```

**PROGRAM 57**

WAP to swap two values using Call by Reference

```
#include <stdio.h>

/* function declaration goes here.*/
void swap( int *p1, int *p2 );

int main()
{
   int a = 10;
   int b = 20;

   printf("Before: Value of a = %d and value of b = %d\n", a, b );
   swap( &a, &b );
   printf("After: Value of a = %d and value of b = %d\n", a, b );
}

void swap( int *p1, int *p2 )
{
    int t;

    t = *p2;
    *p2 = *p1;
    *p1 = t;
   printf("Value of a (p1) = %d and value of b(p2) = %d\n", *p1, *p2 );
}
```

**PROGRAM 58**

WAP to print the series of prime numbers

```
#include<stdio.h>
int check_prime(int num);
int main(){
  int n1,n2,i,flag;
  printf("Enter two numbers(intervals): ");
  scanf("%d %d",&n1, &n2);
  printf("Prime numbers between %d and %d are: ", n1, n2);
  for(i=n1+1;i<n2;++i)
  {
    flag=check_prime(i);
    if(flag==0)
      printf("%d ",i);
  }
  return 0;
}
int check_prime(int num) /* User-defined function to check prime number*/
{
  int j,flag=0;
  for(j=2;j<=num/2;++j){
     if(num%j==0){
        flag=1;
        break;
     }
  }
  return flag;
}
```

**PROGRAM 59**

WAP to find sum of natural numbers using recursion.

```
#include<stdio.h>
int add(int n);
int main()
{
   int n;
   printf("Enter an positive integer: ");
   scanf("%d",&n);
   printf("Sum = %d",add(n));
   return 0;
}
int add(int n)
{
   if(n!=0)
    return n+add(n-1);  /* recursive call */
}
```

**PROGRAM 60**

WAP to find factorial of a number using recursion

```
#include<stdio.h>
int factorial(int n);
int main()
{
   int n;
   printf("Enter an positive integer: ");
   scanf("%d",&n);
   printf("Factorial of %d = %ld", n, factorial(n));
   return 0;
}
int factorial(int n)
{
   if(n!=1)
    return n*factorial(n-1);
}
```

**PROGRAM 61**

WAP to find HCF using recursion

```
#include <stdio.h>
int hcf(int n1, int n2);
int main()
{
  int n1, n2;
  printf("Enter two positive integers: ");
  scanf("%d%d", &n1, &n2);
  printf("H.C.F of %d and %d = %d", n1, n2, hcf(n1,n2));
  return 0;
}
int hcf(int n1, int n2)
{
   if (n2!=0)
      return hcf(n2, n1%n2);
   else
      return n1;
}
```

**PROGRAM 62**

WAP to reverse a sentence using recursion

```
#include <stdio.h>
void Reverse();
int main()
{
   printf("Enter a sentence: ");
   Reverse();
   return 0;
}
void Reverse()
{
   char c;
   scanf("%c",&c);
   if( c != '\n')
   {
      Reverse();
      printf("%c",c);
   }
}
```

**PROGRAM 63**

WAP to convert binary to decimal or decimal to binary

```
#include <stdio.h>
#include <math.h>
int binary_decimal(int n);
int decimal_binary(int n);
int main()
{
  int n;
  char c;
  printf("Instructions:\n");
  printf("1. Enter alphabet 'd' to convert binary to decimal.\n");
  printf("2. Enter alphabet 'b' to convert decimal to binary.\n");
  scanf("%c",&c);
  if (c =='d' || c == 'D')
  {
     printf("Enter a binary number: ");
     scanf("%d", &n);
     printf("%d in binary = %d in decimal", n, binary_decimal(n));
  }
  if (c =='b' || c == 'B')
  {
     printf("Enter a decimal number: ");
     scanf("%d", &n);
     printf("%d in decimal = %d in binary", n, decimal_binary(n));
  }
  return 0;
}
```

```
int decimal_binary(int n)  /* Function to convert decimal to binary.*/
{
   int rem, i=1, binary=0;
   while (n!=0)
   {
      rem=n%2;
      n/=2;
      binary+=rem*i;
      i*=10;
   }
   return binary;
}

int binary_decimal(int n) /* Function to convert binary to decimal.*/

{
   int decimal=0, i=0, rem;
   while (n!=0)
   {
      rem = n%10;
      n/=10;
      decimal += rem*pow(2,i);
      ++i;
   }
   return decimal;
}
```

**PROGRAM 64**

WAP to rpint Fibonacci series using recursion

```
#include <stdio.h>
int fibonaci(int i) {
  if(i == 0) {
    return 0;
  }
  if(i == 1) {
    return 1;
  }
  return fibonaci(i-1) + fibonaci(i-2);
}
int  main() {
  int i;

  for (i = 0; i < 10; i++) {
    printf("%d\t\n", fibonaci(i));
  }
  return 0;
}
```