BASIC CPP

C++:

C++ is a cross-platform language that can be used to create high-performance applications.

C++ was developed by Bjarne Stroustrup, as an extension to the C language.

C++ gives programmers a high level of control over system resources and memory.

COMMENTS:

Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

#include directory:

#include <iostream> is a header file library that lets us work with input and output objects, such as coutHeader files add functionality to C++ programs.

Using namespace std:

using namespace std means that we can use names for objects and variables from the standard library.

main():

Another thing that always appear in a C++ program, is int main(). This is called a function. Any code inside its curly brackets {} will be executed.

Every C++ statement ends with a semicolon ;

return 0 ends the main function.

PRINT STATEMENT:

The cout object, together with the << operator, is used to output values/print text.To insert a new line, you can use the \n character.Another way to insert a new line, is with the endl manipulator

PROGRAM 1:

```
// Header file for input output functions

#include<iostream>

using namespace std;

/* main function -

 where the execution of program begins*/

int main()

{

        // prints hello world

        cout<<"Hello World";

        return 0;

}
```

DATA TYPES:

Data type specifies the size and type of variable values, and it has no additional methods.

VARIABLES:

Variables are containers for storing data values.

USER INPUT:

cin is a predefined variable that reads data from the keyboard with the extraction operator (>>).

PROGRAM 2:

```
#include <iostream>

using namespace std;

int main()

{

        int age;

        cout << "Enter your age:";

        cin >> age;

        cout << "\nYour age is: " << age;



        return 0;

}
```

PROGRAM 3:

```
#include <iostream>

using namespace std;

int main()

 {

  cout << "Hello World! \n";

  cout << "\tI am learning C++";

  return 0;

}
```

PROGRAM 4:

```
#include<iostream>

using namespace std;

int main()

{

        cout << "Size of char : " << sizeof(char) << " byte" << endl;

        cout << "Size of int : " << sizeof(int)       << " bytes" << endl;

        cout << "Size of wchar_t : " << sizeof(bool)      << " bytes" <<endl;

        cout << "Size of float : " << sizeof(float)   << " bytes" <<endl;

        cout << "Size of double : " << sizeof(double) << " bytes" << endl;

        cout << "Size of long : " << sizeof(long) << " bytes" << endl;

        cout << "Size of wchar_t : " << sizeof(wchar_t)  << " bytes" <<endl;

        return 0;

}
```

PROGRAM 5:

```
#include <iostream>

using namespace std;

int main()

{

  int myNum = 5;

  double myFloatNum = 5.99;
  char myLetter = 'D';

  string myText = "Hello";

  bool myBoolean = true;

  cout << myNum;

  cout << myFloatNum;

  cout << myLetter;

  cout << myText;

  cout << myBoolean;

  return 0;

}
```

CONSTANTS:

To override existing variable values, use the const keyword (this will declare the variable as "constant", which means unchangeable and read-only)

PROGRAM 6:

```
#include <iostream>

using namespace std;

int main()

{

  const int minutesPerHour = 60;

  const float PI = 3.14;

  cout << minutesPerHour << "\n";

  cout << PI;
  return 0;

}
```

OPERATORS:

Operators are used to perform operations on variables and values.

Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.

PROGRAM 7:

```
#include <iostream>

using namespace std;

int main()

{

  int x = 5;

  int y = 3;

  cout <<"\n"<< x + y;

  cout <<"\n"<< x - y;

  cout <<"\n"<< x * y;

  cout <<"\n"<< x / y;

  cout <<"\n"<< x % y;

  cout <<"\n"<< ++x;

  cout <<"\n"<< --y;

  return 0;
}
```

ASSIGNMENT OPERATOR:

Assignment operators are used to assign values to variables.

PROGRAM 8:

```
#include <iostream>

using namespace std;

int main()

{

  int x = 5;

  cout << x;

  x += 3;

  cout << x;

  x -= 3;

  cout << x;

  x *= 3;

  cout << x;

  x /= 3;

  cout << x;

  x %= 3;

  cout << x;

  return 0;
}
```

RELATIONAL OPERATORS:

Comparison operators are used to compare two values.

The return value of a comparison is either true (1) or false (0).

PROGRAM 9:

```
#include <iostream>

using namespace std;

int main()

{

  int x = 5;

  int y = 3;

  cout << (x > y);

  cout << (x < y);

  cout << (x >= y);

  cout << (x <= y);

  cout << (x == y);

  cout << (x != y);

  return 0;

}
```

LOGICAL OPERATORS:

Logical operators are used to determine the logic between variables or values:

PROGRAM 10:

```
#include <iostream>

using namespace std;

int main()

{

  int x = 5;

  int y = 3;

  cout << (x > 3 && x < 10);

  cout << (x > 3 || x < 10);

  cout << (!(x > 3 && x < 10));

  return 0;

}
```

DECISION MAKING STATEMENTS

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

IF CONDITION:

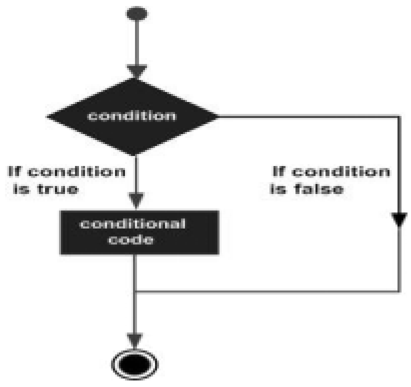
if statement is the most simple decision making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

SIMPLE IF:

SYNTAX:

```
if(condition)

{

        /*Statements to execute if

        condition is true*/

}
```

FLOW CHART:



PROGRAM 11:

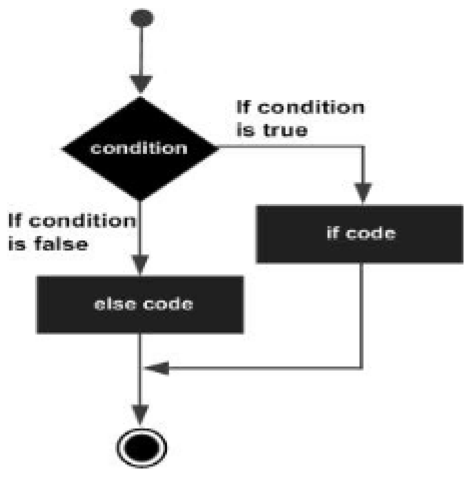
```
#include <iostream>

int main()

 {

        int i = 10;

        if (i > 15)

        {

        cout<<"10 is less than 15";

        }
cout<<"I am Not in if";

}
```

IF ELSE CONDITION:

SYNTAX:

```
if(condition)

{

        /*Executes this block if

        condition is true */

}

else

{

        /*Executes this block if

        condition is false*/

}
```

FLOW CHART:



PROGRAM 12:

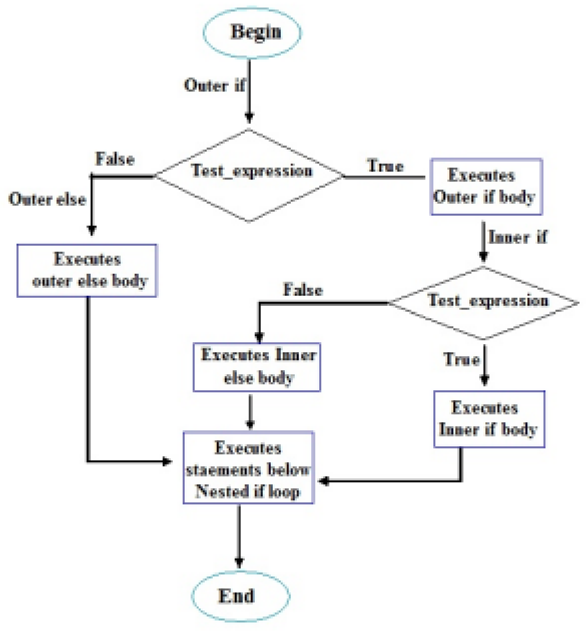
```
#include <iostream>

int main()

{

        int i = 20;

        if (i < 15)

        {

                cout<<"i is smaller than 15";

        }

        else

        {

                cout<<"i is greater than 15";

        }

        return 0;

}
```

NESTED IF CONDITION:

SYNTAX:

```
if(condition1)

{

        //executes if the condition 1 is true

        if(condition2)
        {
                //Executes if the condition 2 is true
        }
}
```

FLOW CHART:



PROGRAM 14:

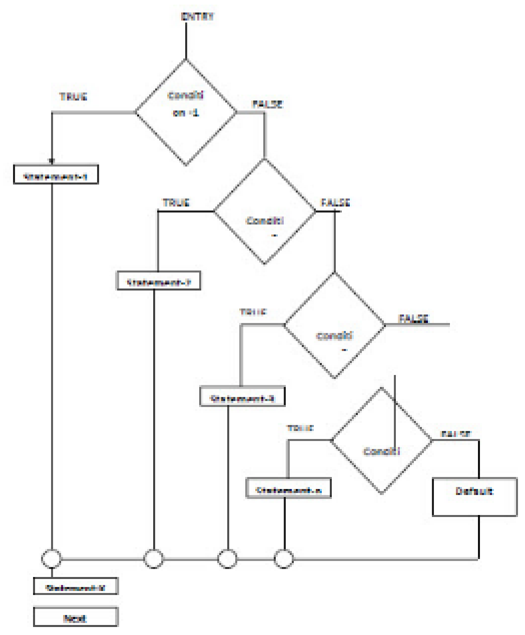
```
#include <iostream>

int main()

{

        int i = 10;

        if (i == 10)

        {

                if (i < 15)
                cout<<"i is smaller than 15\n";

                if (i < 12)

                        cout<<"i is smaller than 12 too\n";

                else

                        cout<<"i is greater than 15";

        }

        return 0;

}
```

ELSE IF LADDER:

SYNTAX:

```
if (condition)

        statement;

else if(condition)

        statements;

.

.

.

else

        statement;
```

FLOW CHART:



PROGRAM 15:

```
#include <iostream>

int main()

{

        int i = 20;

        if (i == 10)

                cout<<"i is 10";

        else if (i == 15)

                cout<<"i is 15";

        else if (i == 20)

                cout<<"i is 20";

        else

                cout<<"i is not present";
}
```

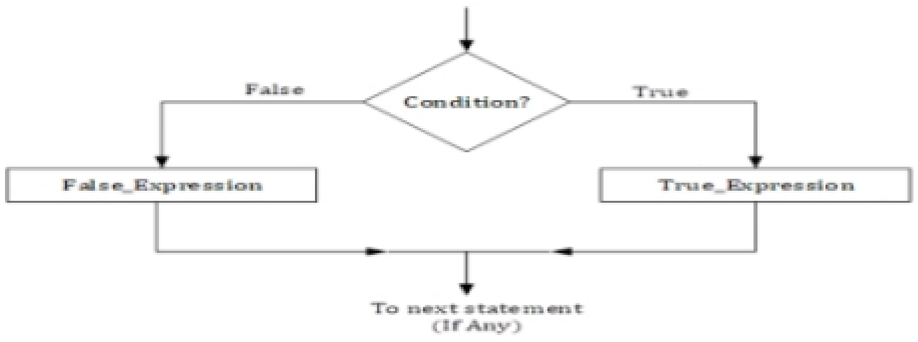
TERNARY OPERATOR:

The **ternary operator** consists of three operands. It can be used to replace multiple lines of code with a single line. It is often used to replace simple if else statements:

SYNTAX:

Expression1? expression2: expression3;

FLOW CHART:



PROGRAM 16:

```
#include <iostream>

using namespace std;

int main ()

{

  int x, y = 10;

  x = (y < 10) ? 30 : 40;

  cout << "value of x: " << x << endl;

  return 0;

}
```

}

PROGRAM 17:

```
#include <iostream>

using namespace std;

int main()

{

  int a = 5;

  int b = 20;

  int c ;

  if((a>1) && (b<=20))

  {

    cout << "AND operator-true statement"<< endl ;

  }

  else

  {

    cout << "AND operator-false statement"<< endl ;

  }

  if((a>1) || (b<=20))

  {

    cout << "OR operator-true statement"<< endl ;

  }

  else
  {
     cout << "OR operator-false statement"<< endl ;
  }
  if(!(a>b))
  {
     cout << "Not operator - reverse answer"<< endl ;
  }
  return 0;
}
```

LOOPING STATEMENTS

Loops can execute a block of code as long as a specified condition is reached.

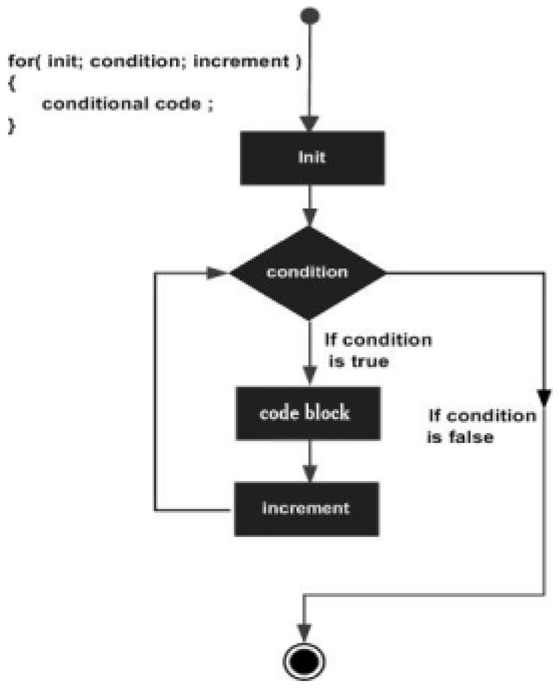
Loops are handy because they save time, reduce errors, and they make code more readable.

FOR LOOPS:

SYNTAX:

```
for(initialization,condition,increment)
{
        //statements;
}
```

FLOW CHART:



PROGRAM 1:

```
#include <iostream>

using namespace std;

int main()

{

  for (int i = 0; i < 5; i++)

{

    cout << i << "\n";

  }

  return 0;

}
```

PROGRAM 2:

```
#include <iostream>

using namespace std;

int main()

{

   int num, sum;

   sum = 0;

   cout << "Enter a positive integer: ";

   cin >> num;

   for (int i = 1; i <= num; ++i)

  {

      sum += i;

   }

   cout << "Sum = " << sum << endl;

   return 0;

}
```

PROGRAM 3:

```
#include <iostream>

using namespace std;

int main()

{

   int num_array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

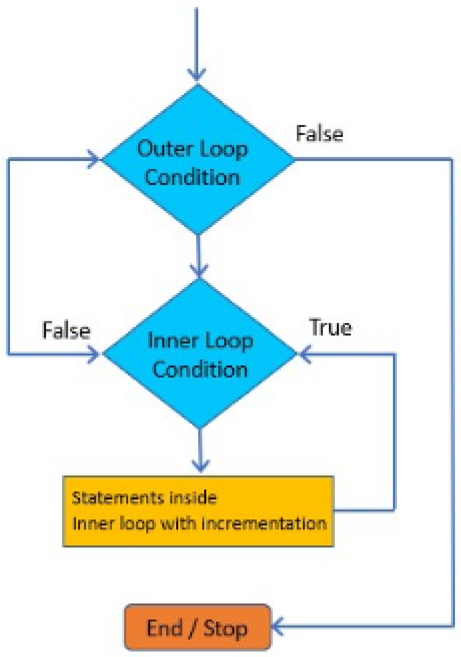
```
    for (int n : num_array)
    {
        cout << n << " ";
    }
      return 0;
}
```

NESTED LOOPS:

SYNTAX:

```
do
{
   while(condition)
{
      for ( initialization; condition; increment )
{
              // statement of inside for loop
      }
      // statement of inside while loop
   }
   // statement of outer do-while loop
}while(condition);
```

FLOW CHART:



PROGRAM 4:

```
#include <iostream>

using namespace std;

int main ()

{

        for(int i=1;i<=3;i++)

       {

             for(int j=1;j<=3;j++)

            {

            cout<<i<<" "<<j<<"\n";

            }

        }

    }
```

PROGRAM 5:

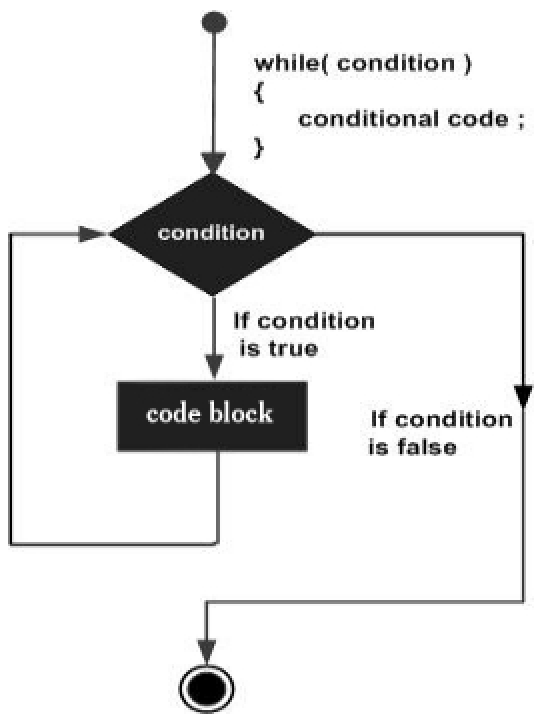
```
#include <iostream>

using namespace std;

int main ()

{

        for (; ;)

          {

                  cout<<"Infinitive For Loop";

          }

    }
```

WHILE LOOP:

SYNTAX:

```
initialization;

while (condition)

{

   // statements

   Increment/decrement;

}
```

FLOW CHART:



PROGRAM 6:

```
#include <iostream>

using namespace std;

int main()

{

  int i = 0;

  while (i < 5)

 {

    cout << i << "\n";

    i++;

  }

  return 0;
}
```

PROGRAM 7:

```
#include <iostream>
using namespace std;
int main ()
{
        int i=1;
        while(i<=3)
        {
              int j = 1;
              while (j <= 3)
             {
              cout<<i<<" "<<j<<"\n";
              j++;
             }
           i++;
         }
    }
```

PROGRAM 8:

```
#include <iostream>
```

```
using namespace std;
int main ()
{
        while(true)
          {
                    cout<<"Infinitive While Loop";
          }
    }
```

DO WHILE LOOP:

SYNTAX:

initialization;

do

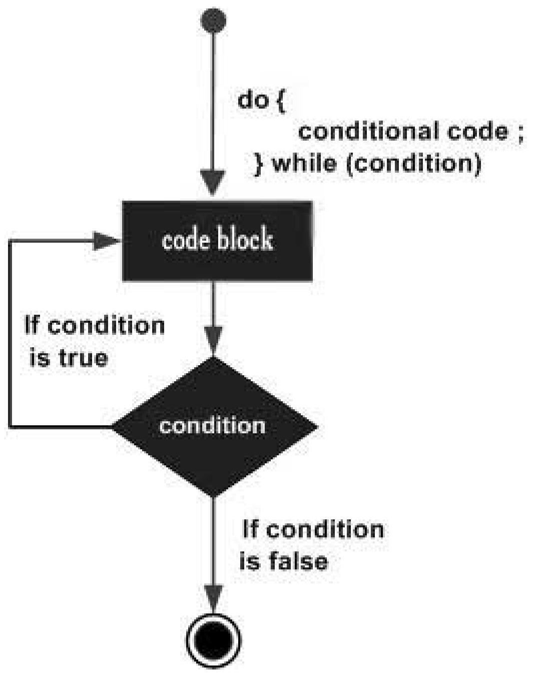
{

   // statements

   Increment/decrement;

} while (**condition**);

FLOW CHART:



PROGRAM 9:

```
#include <iostream>

using namespace std;

int main()

{

  int i = 0;

  do

 {

    cout << i << "\n";

    i++;

  }

  while (i < 5);

  return 0;
}
```

PROGRAM 10:

```
#include <iostream>

using namespace std;

int main()

{

    int i = 1;

        do

       {

            int j = 1;

            do

           {

              cout<<i<<"\n";

                j++;

            } while (j <= 3) ;

            i++;

        } while (i <= 3) ;

}
```

PROGRAM 11:

```
#include <iostream>

using namespace std;
```

```
int main()

{

      do

     {

              cout<<"Infinitive do-while Loop";

      } while(true);

}
```

EXERCISE:

1. Write a program in C++ to find the first 10 natural numbers using for loop
2. Write a program in C++ to find the sum of first 10 natural numbers using while loop
3. Write a program in C++ to check whether a number is prime or not using do while loop.
4. Write a program in C++ to find the factorial of a number.
5. Write a program in C++ to find the sum of digits of a given number.

SOLUTION:

PROGRAM 1:

```
#include <iostream>

using namespace std;

int main()

{

    int i;
```

```
   cout << "\n\n Find the first 10 natural numbers:\n";
   cout << " The natural numbers are: \n";
   for (i = 1; i <= 10; i++)
   {
      cout << i << " ";
   }
   cout << endl;
}
```

PROGRAM 2:

```
#include <iostream>
using namespace std;
int main()
{
   int i=1,sum=0;
   cout << "\n\n Find the first 10 natural numbers:\n";
   cout << " The natural numbers are: \n";
   while(i <= 10)
   {
      cout << i << " ";
                 sum=sum+i;
      i++;
   }
    cout << "\n The sum of first 10 natural numbers: "<<sum << endl;
}
```

PROGRAM 3:

```
#include <iostream>
using namespace std;
int main()
{
   int a=1,num1, ctr = 0;
   cout << "\n\n Check whether a number is prime or not:\n";
        cout << " Input a number to check prime or not: ";
        cin>> num1;
   do
   {
      if (num1 % a == 0)
      {
         ctr++;
      }
      a++;
   }while(a<=num1);
   if (ctr == 2)
```

```
    {
        cout << " The entered number is a prime number. \n";
    }
    else {
        cout << " The number you entered is not a prime number. \n";
    }
}
```

PROGRAM 4:

```
#include <iostream>
using namespace std;
int main()
{
    int num1,factorial=1;
    cout << "\n\n Find the factorial of a number:\n";
           cout << " Input a number to find the factorial: ";
           cin>> num1;
    for(int a=1;a<=num1;a++)
    {
        factorial=factorial*a;
    }
           cout<<" The factorial of the given number is: "<<factorial<<endl;
```

```
   return 0;
}
```

PROGRAM 5:

```
#include <iostream>
using namespace std;
int main()
{
   int num1,factorial=1;
   cout << "\n\n Find the factorial of a number:\n";
        cout << " Input a number to find the factorial: ";
        cin>> num1;
   for(int a=1;a<=num1;a++)
   {
      factorial=factorial*a;
   }
        cout<<" The factorial of the given number is: "<<factorial<<endl;
   return 0;
}
```

PROGRAM 6:

```
#include <iostream>
using namespace std;
```

```
int main()

{

    int num1, num2, temp, sum=0;

    cout << "\n\n Find the sum of digits of a given number:\n";

    cout << " Input a number: ";

    cin >> num1;

    num2 = num1;

    while (num1 > 0)

    {

        temp = num1 % 10;

        num1 = num1 / 10;

        sum = sum + temp;

    }

    cout << " The sum of digits of " << num2 << " is: " << sum << endl;

}
```

EXERCISE:

1. Create a c++ program to perform Arithmetic operations without variable
2. Create a c++ program to perform Arithmetic operations ,relational operations and logical operations with variables of same data type
3. Create a c++ program to perform Arithmetic operations,relational operations and logical with variables of different data types(Modulo not possible with different data types)
4. Create a c++ program to perform Arithmetic operations,relational operations and logical operations based on the user input

5. Create a C++ program to swap two values
6. Create a C++ program to calculate area of rectangle,triangle,circle and square.(Area of rectangle:length * breadth,Area of triangle:½*base*height,Area of Circle:PI*r*r,Area of square:side*side)
7. Create a C++ program to find the quotient and remainder of user given values
8. Create a C++ program to find the total and averages of 5 default values without variables and 5 values given by user
9. Create a C++ program to calculate multiplication table upto 10
10. Create a C++ program to find the greatest and smallest among 3 numbers
11. Create a C++ program which accepts the user's first and last name and print them in reverse order with a space between them.

SOLUTION:

PROGRAM 1:

```
#include <iostream>
using namespace std;
int main()
{
   cout<<"\nSum \t"<<5+6;
   cout<<"\nSubtraction \t"<<5-6;
   cout<<"\nMultiplication \t"<<5*6;
   cout<<"\nDivision\t"<<5/6;
   cout<<"\nModulo\t"<<5%6;
   return 0;
}
```

PROGRAM 2:

```
#include <iostream>
using namespace std;
int main()
{
 int x=10;
 int y=20;
 cout<<"\nSum \t"<<x+y<<"\nSubtract \t"<<x-y<<"\nMultiplication
\t"<<x*y<<"\nDivision \t"<<x/y<<"\nModulo \t"<<x%y<<"\nGreater \t"<< (x>y)
<<"\nLesser\t"<<(x<y)<<"\nGreater than or equal\t"<<(x>=y)<<"\nLesser than or
equal \t"<<(x<=y)<<"\nEquality \t"<<(x==y)<<"\nNot equal\t"<<(x!=y);
 cout<<"\nAND Operator\t"<<(x<15 && y<=20)<<"\n OR OPerator\t"<<(x<15 ||
y<=20)<<"\nNot OPerator\t"<<(!(x<15 || y<=20));
 return 0;
}
```

PROGRAM 3:

```
#include <iostream>
using namespace std;
int main()
{
 int x=10;
 float y=2.99;
 cout<<"\nSum \t"<<x+y<<"\nSubtract \t"<<x-y<<"\nMultiplication
\t"<<x*y<<"\nDivision \t"<<x/y<<"\nGreater \t"<< (x>y)
<<"\nLesser\t"<<(x<y)<<"\nGreater than or equal\t"<<(x>=y)<<"\nLesser than or
equal \t"<<(x<=y)<<"\nEquality \t"<<(x==y)<<"\nNot equal\t"<<(x!=y);
 cout<<"\nAND Operator\t"<<(x<15 && y<=20)<<"\n OR Operator\t"<<(x<15 ||
y<=20)<<"\nNot OPerator\t"<<(!(x<15 || y<=20));
 return 0;
}
```

PROGRAM 4:

```
#include <iostream>
using namespace std;
int main()
{
 int x,y;
 cout<<"\nEnter two numbers";
 cin>>x>>y;
 cout<<"\nSum \t"<<x+y<<"\nSubtract \t"<<x-y<<"\nMultiplication
\t"<<x*y<<"\nDivision \t"<<x/y<<"\nGreater \t"<< (x>y)
<<"\nLesser\t"<<(x<y)<<"\nGreater than or equal\t"<<(x>=y)<<"\nLesser than or
equal \t"<<(x<=y)<<"\nEquality \t"<<(x==y)<<"\nNot equal\t"<<(x!=y);
 cout<<"\nAND Operator\t"<<(x<15 && y<=20)<<"\n OR OPerator\t"<<(x<15 ||
y<=20)<<"\nNot OPerator\t"<<(!(x<15 || y<=20));
 return 0;
}
```

PROGRAM 5:

```
#include <iostream>
using namespace std;
int main()
{
 int x,y,z;
 cout<<"\nEnter two numbers";
 cin>>x>>y;
 z=x;
 x=y;
 y=z;
 cout<<"\nThe swapped values are\n"<<x<<"\n"<<y;
 return 0;
}
```

PROGRAM 6:

```
#include <iostream>
using namespace std;
int main()
{
 int length,breadth,base,height,area,radius,side;
 cout<<"\nEnter length  and breadth";
 cin>>length>>breadth;
 area=length*breadth;
 cout<<"\nThe area of rectangle is\t"<<area;
 cout<<"\nEnter base  and height";
 cin>>base>>height;
 area=0.5*base*height;
 cout<<"\nThe area of triangle is\t"<<area;
```

```
  cout<<"\nEnter radius";
 cin>>radius;
 area=3.14*radius*radius;
 cout<<"\nThe area of circle is\t"<<area;
 cout<<"\nEnter side";
 cin>>side;
 area=side*side;
 cout<<"\nThe area of Square is\t"<<area;
 return 0;
}
```

PROGRAM 7:

```
#include <iostream>
using namespace std;
int main()
{
 int a=12;
 int b=15;
 cout<<"\nThe quotient is\t"<<a/b;
 cout<<"\nThe remainder is\t"<<a%b;
 return 0;
}
```

PROGRAM 8:(With variables)

```
#include <iostream>
using namespace std;
int main()
{
 int a,b,c,d,e;
 cout<<"\nEnter five values\t";
 cin>>a>>b>>c>>d>>e;
 a=a+b+c+d+e;
 b=a/5;
 cout<<"\nThe total is\t"<<a;
 cout<<"\nThe average is\t"<<b;
 return 0;
}
```

WITHOUT VARIABLES:

```
#include <iostream>
using namespace std;
int main()
{
 cout<<"\nThe total is\t"<<(5+10+15+14+20)/2;
 return 0;
}
```

PROGRAM 9:

```
#include<iostream>
using namespace std;
int main()
{
    int num;
    cout<<"Enter Number To Find Multiplication table ";
    cin>>num;
    for(int a=1;a<=10;a++)
    {
        cout<<num<<" * "<<a<<" = "<<num*a<<endl;
    }
  return 0;
}
```

PROGRAM 10:

```
#include <iostream>
using namespace std;
int main()
{
    int x=10,y=15,z=20;
    if(x>y && x>z)
        cout<<"\nX is the greatest number \t"<<x;
    else if(y>x && y>z)
        cout<<"\nY is the greatest value\t"<<y;
    else
```

```
        cout<<"\nZ is the greatest value \t" << z;
    if(x<y && x<z)
        cout<<"\nX is the greatest number \t"<<x;
    else if(y<x && y<z)
        cout<<"\nY is the greatest value\t"<<y;
    else
        cout<<"\nZ is the greatest value \t" << z;
    return 0;
}
```

PROGRAM 11:

```
# include <iostream>
# include <string>
using namespace std;
int main()
{
char fname[30], lname [30];
 cout << "\n\n Print the name in reverse where last name comes first:\n";
 cout << " Input First Name: ";
 cin >> fname;
 cout << " Input Last Name: ";
 cin >> lname;
 cout << " Name in reverse is: "<< lname << " "<< fname <<endl;
 cout << endl;
return 0;
}
```

SWITCH CASE PROGRAMS:

SIMPLE SWITCH:

Use the switch statement to select one of many code blocks to be executed.

· The switch expression is evaluated once.

· The value of the expression is compared with the values of each case.

· If there is a match, the associated block of code is executed.

· The break and default keywords are optional

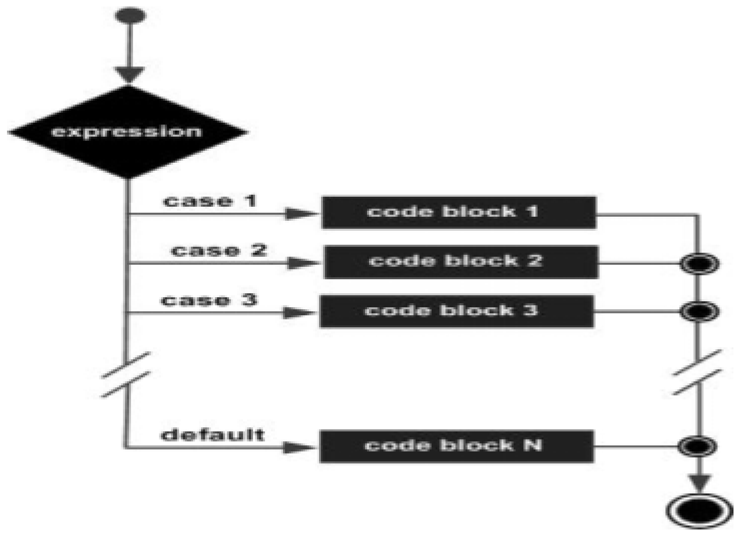
· When Java reaches a break keyword, it breaks out of the switch block.

· The default keyword specifies some code to run if there is no case match:

SYNTAX:

```
switch(expression)
{
        case constant-expression:
                statement1;
                break; //optional
        case constant-expression:
                statement2;
                break; //optional
                        //you can have any number of case statements
        default: //optional
                statement n;
}
```

FLOW CHART:



PROGRAM 1:

```
#include <iostream>

using namespace std;

int main()

{

int x = 2;

switch (x)

{

        case 1:

                cout<<"Choice is 1";

                break;

        case 2:

                cout<<"Choice is 2";

                break;
        case 3:
                cout<<"Choice is 3";
                break;
        default:
                cout<<"Invalid Input";
                break;
}
return 0;
}
```

NESTED SWITCH:

SYNTAX:

```
switch(condition)
{
        case constant-expression:
                statement1;
                break; //optional
        switch(condition)
        {
                case constant-expression:
                        statement1;
                        break; //optional
                case constant-expression:
                        statement1;
                        break; //optional
        }
        case constant-expression:
                statement1;
                break; //optional
}
```

PROGRAM 2:

```
#include<iostream>
using namespace std;
int main()
{
        int x = 1, y = 2;
        cout<<"Enter your choice\n";
        cin>>x;
        switch(x)
   {
   case 1:
            cout<<"Choice is 1\n";
```

```
          switch (y)
{
          case 2:
                     cout<<"Choice is 2\n";
                  break;
          case 3:
                     cout<<"Choice is 3\n";
                     break;
          }
          break;
          case 4:
                cout<<"Choice is 4\n";
                break;
          case 5:
                cout<<"Choice is 5\n";
                break;
     default:
                cout<<"Choice is other than 1, 2 3, 4, or 5\n";
                break;
  }
  return 0;
```

```
}
```

EXERCISE:

1. Create a C++ program to Calculate the percentage of marks and show grade using the switch statements.
   Percentage >= 90% : Grade A
   Percentage >= 80% : Grade B
   Percentage >= 70% : Grade C
   Percentage >= 60% : Grade D
   Percentage >= 40% : Grade E
   Percentage < 40% : Grade F
2. Create a C++ program to perform the arithmetic operations using Switch statements
3. Create a c++ program to find area of rectangle,circle,triangle and square using Switch statements
4. Create a C++ program to find whether a given letter is vowel or consonant Switch statement
5. Create a C++ program to display week days and month using switch statements.

SOLUTION:

PROGRAM 1:

```
# include <iostream>
using namespace std;
int main()
{
int s1,s2,s3,s4,s5;
float per;
char grade;
cout<<"enter marks of five subjects"<<endl;
cin>>s1>>s2>>s3>>s4>>s5;
per=((s1+s2+s3+s4+s5)*100)/500;
cout<<per<<endl;
switch (per>=90&&per<100)
```

```
{
case 1:
cout<<"your grade is A";
break;
case 0:
switch(per>=80 &&per<90)
{
case 1:
cout<<"grade is B";
break;
case 0:
switch(per>=70&&per<80)
{
case 1:
cout<<"your grade is C";
break;
case 0:
switch (per>=60&&per<70)
{
case 1:
cout<<"grade is D";
break;
case 0:
switch (per>=50&&per<60)
{
case 1:
cout<<"grade is E";
break;
case 0:
cout<<"grade is F";
break;
}
break;
}
break;
}
break;
}
break;
}
}
```

PROGRAM 2:

```
#include <iostream>

using namespace std;

int main()

{

        int a,b,c,choice;

        cout<<"\n Enter the 1st value";

   cin>>a;

   cout<<"\nEnter the 2nd value";

   cin>>b;

   cout<<"\nEnter your choice \n 1.Addition \n 2.Subtraction \n 3.Multiplication \n
4.Division \n 5.Remainder";

   cin>>choice;

        switch(choice)

        {

        case 1:

         cout<<"\nAddition";
```

```
                    c=a+b;
 cout<<"\nThe sum of \t"<<a<<"\tand\t"<<b<<"\tis\t"<<c;
              break;
case 2:
 cout<<"\nSubtraction";
                    c=a-b;
 cout<<"\nThe subtraction of\t "<<a<<"\tand\t"<<b<<"\tis\t"<<c;
              break;
case 3:
 cout<<"\nMultiplication";
                    c=a*b;
 cout<<"\nThe product of\t"<<a<<"\tand\t"<<b<<"\tis\t"<<c;
          break;
case 4:
 cout<<"\nDivision";
                    c=a/b;
 cout<<"\nThe division of \t"<<a<<"\tand\t"<<b<<"\tis\t"<<c;
              break;
case 5:
 cout<<"\nRemainder";
                    c=a%b;
```

```
         cout<<"\nThe remainder of \t"<<a<<"\tand\t"<<b<<"\tis\t"<<c;
                    break;
        default:
                    cout<<"Invalid input";
                    break;
   }
return 0;
}
```

PROGRAM 3:

```
 #include<iostream>
using namespace std;
int main()
{
        int figcode;
        float side,base,height,length,breadth,area,radius;
        cout<<"\n 1…Circle";
   cout<<"\n 2…Rectangle";
   cout<<"\n 3…Triangle";
   cout<<"\n 4…Square";
   cout<<"\nEnter the figure code";
   cout<<"\n********";
```

```
cin>>figcode;

     switch(figcode)

     {

     case 1:

   cout<<"Enter the radius\n";

   cin>>radius;

             area=3.14*radius*radius;

   cout<<"\n The area of a circle\n"<<area;

             break;

     case 2:

   cout<<"Enter the breadth and length\n";

   cin>>breadth>>length;

             area=length*breadth;

   cout<<"\n The area of a rectangle\n"<<area;

             break;

     case 3:

   cout<<"Enter the base and height\n";

   cin>>base>>height;

             area=0.5*base*height;

   cout<<"\n The area of a circle\n"<<area;

             break;
```

```
        case 4:
      cout<<"Enter the side\n";
      cin>>side;
                area=side*side;
      cout<<"\n The area of a square\n"<<area;
                break;
        default:
                cout<<"Error in figure code";
                break;
}
return 0;
}
```

PROGRAM 4:

```
#include <iostream>
using namespace std;
int main()
{
   char ch;
   cout<<"Enter any alphabet: ";
   cin>>ch;
   switch(ch)
```

```
{
   case 'a': cout<<"vowel";
      break;
   case 'e': cout<<"vowel";
      break;
   case 'i': cout<<"vowel";
      break;
   case 'o': cout<<"vowel";
      break;
   case 'u': cout<<"vowel";
      break;
   case 'A': cout<<"vowel";
      break;
   case 'E': cout<<"vowel";
      break;
   case 'I': cout<<"vowel";
      break;
   case 'O': cout<<"vowel";
      break;
   case 'U': cout<<"vowel";
      break;
```

```
        default: cout<<"consonant";
    }
     return 0;
 }
```

PROGRAM 5:

```
#include <iostream>
using namespace std;
int main()
{
    int weeknumber,month;
    cout<<"Enter week number(1-7): ";
    cin>>weeknumber>>month;
    switch(weeknumber)
    {
        case 1: cout<<"Monday";
            break;
        case 2: cout<<"Tuesday";
            break;
        case 3: cout<<"Wednesday";
            break;
        case 4: cout<<"Thursday";
```

```
            break;

        case 5: cout<<"Friday";

            break;

        case 6: cout<<"Saturday";

            break;

        case 7: cout<<"Sunday";

            break;

        default: cout<<"Invalid inpput! Please enter week no. between 1-7 for week
days";

    }

    switch(month)

    {

        case 1: cout<<"January";

            break;

        case 2: cout<<"February";

            break;

        case 3: cout<<"March";

            break;

        case 4: cout<<"April";

            break;

        case 5: cout<<"May";

            break;
        case 6: cout<<"June";
           break;
        case 7: cout<<"July";
           break;
        case 8: cout<<"August";
           break;
        case 9: cout<<"September";
           break;
        case 10: cout<<"October";
           break;
        case 11: cout<<"November";
           break;
        case 12: cout<<"December";
           break;
        default: cout<<"Invalid inpput! Please enter month no. between 1-12 for MOnth
names";
   }


   return 0;
 }
```

FUNCTIONS IN CPP

A **functions** is a block of code which only runs when it is called.

You can pass data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as **functions**.

The main advantage of methods is reusability of code.

A method must be declared within a class. It is defined with the name of the method, followed by parentheses **()**.

PROGRAM 1:

```
#include <iostream>

using namespace std;

void function()

{

   cout << "Hello!";

}

int main()

{

   function();

   return 0;

}
```

PROGRAM 2

```
#include <iostream>

using namespace std;

int add(int a, int b)
```

```
{
    return (a + b);
}
int main()
{
    int sum;
    sum = add(100, 78);
    cout << "100 + 78 = " << sum << endl;
    return 0;
}
```

PROGRAM 3:

```
#include<iostream>
using namespace std;
void sum();
void subtract();
int main()
{
    sum();
    subtract();
}
void sum()
```

```
{

    int a=20,b=10;

    cout<<"The sum is %d\n"<<a+b;

}

void subtract()

{

    int a=20,b=10;

    cout<<"The subtraction is %d\n"<<a-b;

}
```

PROGRAM 4:

```
#include <iostream>

using namespace std;

int max(int x,int y);

int main()

{

int a = 10, b = 20;

int m = max(a, b);

cout<<"m is %d\t"<<m;

return 0;

}

int max(int x, int y)
```

```
{
        if (x > y)
                return x;
        else
                return y;
}
```

PROGRAM 5:

```
#include<iostream>
using namespace std;
int square();
int main()
{
   cout<<"\nThe area of the square\n";
   float area = square();
   cout<<"The area of the square:"<<area;
}
int square()
{
float side;
cout<<"Enter the length of the side in meters:\n ";
cin>>side;
return side * side;

}
```

PROGRAM 6:

```
#include <iostream>

#include <cmath>

using namespace std;

int main()

{

    double number, squareRoot;

     number = 25.0;

    squareRoot = sqrt(number);

    cout << "Square root of " << number << " = " << squareRoot;

    return 0;

}
```

EXERCISE:

1. Create a program in C++ to find the sum of first 10 natural numbers using functions
2. Create a c++ program to perform Arithmetic operations, relational operations and logical operations with functions
3. Create a program in C++ to swap two numbers using function
4. Create a program in C++ to check a given number is even or odd using the function.
5. Create a program in C++ to check whether a number is a prime number or not using the function.

SOLUTION:

PROGRAM 1:

```
#include<iostream>
using namespace std;
int main()
{
int i,n,sum=0;
cout<<"How many numbers? ";
cin>>n;
for(i=1;i<=n;++i)
{
sum+=i;
}
cout<<"Sum="<<sum;
return 0;
}
```

PROGRAM 2:

```
#include <iostream>
using namespace std;
void arithmetic(int a, int b)
{
   cout << "100 + 78 = " << a+b << endl;
   cout << "100 - 78 = " << a-b << endl;
   cout << "100 * 78 = " << a*b << endl;
   cout << "100 / 78 = " << a/b << endl;
   cout << "100 % 78 = " << a%b << endl;
}
void relational(int a, int b)
{
   cout << "100 < 78 = " << (a<b) << endl;
   cout << "100 > 78 = " << (a>b) << endl;
   cout << "100 <= 78 = " << (a<=b) << endl;
   cout << "100 >= 78 = " << (a>=b) << endl;
   cout << "100 == 78 = " << (a==b) << endl;
   cout << "100 != 78 = " << (a!=b) << endl;
}
void logical(int a, int b)
{
   cout << (a > 3 && b < 10)<<endl;
   cout << (a > 3 || b < 10)<<endl;
   cout << (!(a > 3 && b < 10))<<endl;

}

int main()
{
   arithmetic(100,78);
   relational(100,120);
   logical(10,20);
   return 0;
}
```

PROGRAM 3:

```
#include <iostream>
using namespace std;
void swap() {
        cout<<"this is my swap";
}
int main() {
        int firstNum , secondNum;
        cout<<"Enter value for First Number:   ";
        cin>>firstNum;
        cout<<"Enter value for Second Number:  ";
        cin>>secondNum;
        cout<<"\n\n";
        cout<<"Values BEFORE Calling Built in Swap Function"<<endl<<endl;
        cout<<"\tFirst  Number = "<<firstNum<<endl;
        cout<<"\tSecond Number = "<<secondNum<<endl;
        cout<<"\n\n";
        std::swap( firstNum , secondNum );
        cout<<"Values AFTER Calling Bulit in  Swap Function"<<endl<<endl;
        cout<<"\tFirst Number = "<<firstNum<<endl;
        cout<<"\tSecond Number = "<<secondNum<<endl;
        return 0;
}
```

PROGRAM 4:

```
#include <iostream>
using namespace std;

int isEven(int num)
{
   return (num%2==0);
}
int main()
{
   int num;
   cout<<"Enter any number: ";
   cin>>num;
   if(isEven(num))
   {
      cout<<"The entered number is even.";
   }
   else
   {
      cout<<"The entered number is odd.";
   }
   return 0;
}
```

PROGRAM 5:

```
#include <iostream>
using namespace std;
bool checkPrimeNumber(int);
int main()
```

```
{
    int n;
    cout << "Enter a positive  integer: ";
    cin >> n;
    if (checkPrimeNumber(n))
        cout << n << " is a prime number.";
    else
        cout << n << " is not a prime number.";
    return 0;
}
bool checkPrimeNumber(int n)
{
    bool isPrime = true;
    if (n == 0 || n == 1)
    {
        isPrime = false;
    }
    else
    {
        for (int i = 2; i <= n / 2; ++i)
        {
            if (n % i == 0) {
                isPrime = false;
                break;
            }
        }
    }
    return isPrime;
}
```

OOPS CONCEPT:

OOP stands for **Object-Oriented Programming**.

Object-oriented programming has several advantages over procedural programming:

· OOP is faster and easier to execute

· OOP provides a clear structure for the programs

· OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug

· OOP makes it possible to create full reusable applications with less code and shorter development time

CLASS AND OBJECT:

C++ is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.

A Class is like an object constructor, or a "blueprint" for creating objects.

To create a class, use the keyword class

In Java, an object is created from a class. We have already created the class named Main, so now we can use this to create objects.

To create an object of Main, specify the class name, followed by the object name, and use the keyword new

Variables are usually called as attributes of class and functions are class methods

PROGRAM 1:

```
#include <iostream>

using namespace std;

class Student

{
   public:

      int id;

      string name;

 };

int main()

{

    Student s1;

    s1.id = 001;

    s1.name = "ABC";

    cout<<s1.id<<endl;

    cout<<s1.name<<endl;

    return 0;

}
```

CONSTRUCTORS AND DESTRUCTORS

Constructors are special class functions which performs initialization of every object. The Compiler calls the Constructor whenever an object is created. Constructors initialize values to object members after storage is allocated to the object.

Whereas, Destructor on the other hand is used to destroy the class object.

PROGRAM 2:

```
#include <iostream>

using namespace std;
```

```
class Employee

 {

   public:

        Employee()

        {

            cout<<"Constructor Invoked"<<endl;

        }

        ~Employee()

        {

            cout<<"Destructor Invoked"<<endl;

        }

};

int main(void)

{

    Employee e1;

    Employee e2;

    return 0;

}
```

INHERITANCE:

In Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:

· **subclass** (child) - the class that inherits from another class

· **superclass** (parent) - the class being inherited from

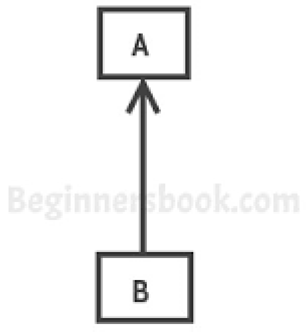
· **Reusability-**As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

To inherit from a class, use the extends keyword.

SINGLE INHERITANCE:

When a class inherits another class, it is known as a *single inheritance*. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

FLOW CHART:



Single Inheritance

PROGRAM 3(Single level)

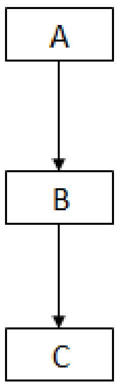
```
#include <iostream>

using namespace std;

 class Account

 {

   public:
```

```
   float salary = 60000;
 };
   class Programmer: public Account
   {
   public:
   float bonus = 5000;
   };
int main(void)
{
     Programmer p1;
     cout<<"Salary: "<<p1.salary<<endl;
     cout<<"Bonus: "<<p1.bonus<<endl;
    return 0;
}
```

MULTILEVEL INHERITANCE:

When there is a chain of inheritance, it is known as *multilevel inheritance*

FLOW CHART:



Multilevel Inheritance

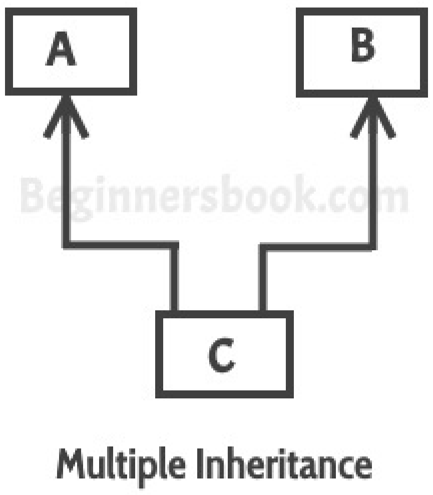
PROGRAM 4(Multilevel)

```
#include <iostream>

using namespace std;

 class Animal

{

   public:

 void eat()

{

    cout<<"Eating..."<<endl;

 }

   };

   class Dog: public Animal

   {

       public:

     void bark()
    {
    cout<<"Barking..."<<endl;
     }
   };
   class BabyDog: public Dog
   {
       public:
     void weep()
    {
       cout<<"Weeping...";
     }
   };
int main()
{
    BabyDog d1;
    d1.eat();
    d1.bark();
     d1.weep();
     return 0;
}
```

MULTIPLE INHERITANCE:

Multiple Inheritance is a feature of object oriented concept, where a class can inherit properties of more than one parent class.

FLOW CHART:



PROGRAM 5(Multiple)

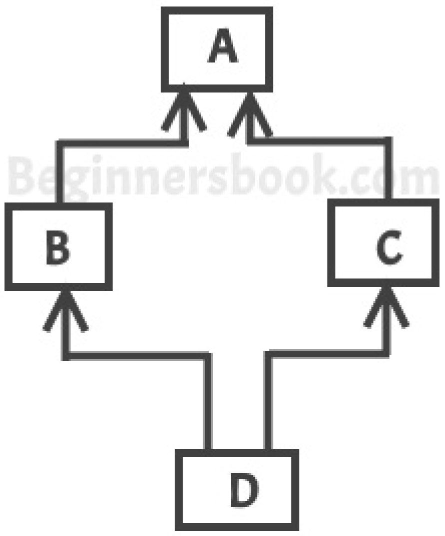
```
#include <iostream>

using namespace std;

class A

{

    protected:

     int a;

    public:

    void get_a(int n)

    {

        a = n;
    }

};

class B

{

    protected:

    int b;

    public:

    void get_b(int n)

    {

        b = n;

    }

};

class C : public A,public B

{

   public:

    void display()

    {

        cout << "The value of a is : " <<a<<endl;

        cout << "The value of b is : " <<b<<endl;

        cout<<"Addition of a and b is : "<<a+b;

    }
```

```
};
int main()
{
   C c;
   c.get_a(10);
   c.get_b(20);
   c.display();
   return 0;
}
```

HYBRID INHERITANCE

Hybrid inheritance is a combination of simple, multiple inheritance and hierarchical inheritance.

FLOW CHART:



Hybrid Inheritance

PROGRAM 6(Hybrid)

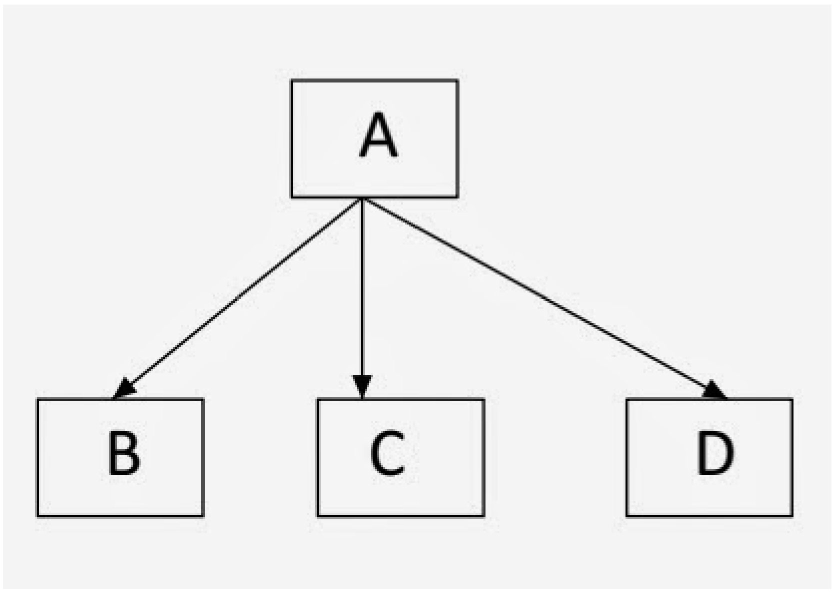
```
#include <iostream>

using namespace std;

class A

{

   protected:

   int a;

   public:

   void get_a()

   {

      cout << "Enter the value of 'a' : " <<endl;

      cin>>a;

   }

};


class B : public A

{

   protected:

   int b;

   public:

   void get_b()

   {
       cout << "Enter the value of 'b' : " <<endl;

       cin>>b;

   }

};

class C

{

   protected:

   int c;

   public:

   void get_c()

   {

       cout<< "Enter the value of c is : " <<endl;

       cin>>c;

   }

};

 class D : public B, public C

{

   protected:

   int d;

   public:

   void mul()
```

```
   {
       get_a();
       get_b();
       get_c();
       cout << "Multiplication of a,b,c is : " <<a*b*c<<endl;
   }
};
int main()
{
   D d;
   d.mul();
   return 0;
}
```

HIERARCHICAL INHERITANCE:

When two or more classes inherits a single class, it is known as *hierarchical inheritance*.

FLOW CHART:



PROGRAM 7(Hierarchical):

```
#include <iostream>

using namespace std;

class Shape                 // Declaration of base class.

{

   public:

   int a;

   int b;

   void get_data(int n,int m)

   {

       a= n;

       b = m;

   }

};
```

```
class Rectangle : public Shape  // inheriting Shape class

{

   public:

   int rect_area()

   {

      int result = a*b;

      return result;

   }

};

class Triangle : public Shape    // inheriting Shape class

{

   public:

   int triangle_area()

   {

      float result = 0.5*a*b;

      return result;

   }

};

int main()

{

   Rectangle r;
```

```
    Triangle t;

    int length,breadth,base,height;

    cout << "Enter the length and breadth of a rectangle: " << endl;

    cin>>length>>breadth;

    r.get_data(length,breadth);

    int m = r.rect_area();

    cout << "Area of the rectangle is : " <<m<< endl;

    cout << "Enter the base and height of the triangle: " << endl;

    cin>>base>>height;

    t.get_data(base,height);

    float n = t.triangle_area();

    cout <<"Area of the triangle is : "  << n<<endl;

    return 0;

}
```

POLYMORPHISM

POLYMORPHISM:

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

METHOD OVERLOADING:

Method overloading is the process of overloading the method that has the same name but different parameters.Method overloading allows users to use the same name to another method, but the parameters passed to the methods should be different. The return type of methods can be the same or different.

PROGRAM 8(Method overloading)

```
#include <iostream>

using namespace std;

class Cal

{

    public:

static int add(int a,int b)

{

        return a + b;

}

static int add(int a, int b, int c)

{

        return a + b + c;

}

};

int main(void)

{

    Cal C;

    cout<<C.add(10, 20)<<endl;

    cout<<C.add(12, 20, 23);

   return 0;
}
```

OPERATOR OVERLOADING:

C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading.

For example, we can overload an operator ‘+’ in a class like String so that we can concatenate two strings by just using +.

PROGRAM 9(Operator overloading):

```
#include <iostream>

using namespace std;

class Complex

{

  private:

   float real;

   float imag;

   public:

   Complex()

   {

      real=0;

      imag=0;

   }

   void input()

   {
        cout << "Enter real and imaginary parts respectively:\n ";

        cin >> real;

        cin >> imag;

    }

    Complex operator + (Complex &obj)

    {

        Complex temp;

        temp.real = real + obj.real;

        temp.imag = imag + obj.imag;

        return temp;

    }

    void output()

    {

        if (imag < 0)

            cout << "\nOutput Complex number: " << real << imag << "i";

        else

            cout << "\nOutput Complex number: " << real << "+" << imag << "i";

    }

};

int main()

{
    Complex complex1, complex2, result;

    cout << "\nEnter first complex number:\n";

    complex1.input();

    cout << "\nEnter second complex number:\n";

    complex2.input();

    result = complex1 + complex2;

    result.output();

    return 0;

}
```

METHOD OVERRIDING:

If derived class defines same function as defined in its base class, it is known as function overriding in C++. It is used to achieve runtime polymorphism. It enables you to provide specific implementation of the function which is already provided by its base class.

PROGRAM 10(Method overriding):

```
#include <iostream>

using namespace std;

class Animal

{

    public:

    void eat()

    {

        cout<<"Eating...";
    }
};
class Dog: public Animal
{
 public:
 void eat()
 {
       cout<<"Eating bread...";
 }
};
int main(void)
{
   Dog d = Dog();
   d.eat();
   return 0;
}
```

DATA ABSTRACTION:

Data **abstraction** is the process of hiding certain details and showing only essential information to the user.

Abstraction can be achieved with either **abstract classes**

};

The abstract keyword is a non-access modifier, used for classes and methods:

- **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

An abstract class can have both abstract and regular methods

PROGRAM 11:

```
#include <iostream>

using namespace std;

 class Sum

{

private: int x, y, z;

public:

void add()

{

cout<<"Enter two numbers: ";

cin>>x>>y;

z= x+y;

cout<<"Sum of two number is: "<<z<<endl;

}

};

int main()
```

```
{
Sum sm;
sm.add();
return 0;
}
```

ENCAPSULATION:

The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

- declare class variables/attributes as private
- provide public **get** and **set** methods to access and update the value of a private variable

PROGRAM 12:

```
 #include<iostream>
using namespace std;
class Encapsulation
{
        private:
                int x;
        public:
        void set(int a)
                {
                        x =a;
                }

                int get()

                {

                        return x;

                }

};

int main()

{

        Encapsulation obj;

        obj.set(5);

        cout<<obj.get();

        return 0;

}
```

EXERCISE 1:

1. Write a **Rectangle class** in CPP language, allowing you to build a rectangle with **length** and **width** attributes.
2. Create a **Area()** method to calculate the area of the rectangle.
3. Create a method **display()** that display the length, width and area of an object created using an instantiation on rectangle class.

EXERCISE 2:

1. Create a CPP class **Person** with attributes: **name** and **age** of type string.
2. Create a **display()** method that displays the name and age of an object created via the Person class.

3. Create a **child class Student** which **inherits** from the Person class and which also has a **section** attribute.
4. Create a **method displayStudent()** that displays the name, age and section of an object created via the Student class.
5. Create a **student object** via an instantiation on the Student class and then test the displayStudent method.

EXERCISE 3:

1. Create a CPP class called **BankAccount** which represents a bank account, having as attributes: **accountNumber** (numeric type), **name** (name of the account owner as string type), balance.
2. Create a **Deposit()** method which manages the deposit actions.
3. Create a **Withdrawal() method** which manages withdrawals actions.
4. Create a bankFees**()** method to apply the bank fees with a percentage of 5% of the balance account.
5. Create a **display()** method to display account details.
6. Give the complete code for the **BankAccount class**.

EXERCISE 4:

1. Create a **Computation class** with a default constructor (without parameters) allowing to perform various calculations on integers numbers.

2. Create a method called **Sum()** allowing to calculate the sum of the first n integers **1 + 2 + 3 + .. + n.** Test this method.

EXERCISE 5:

1. Create a **calculation class**

2. Create a method called **Average()** in the Calculation class to test the average of a given integer. Test this method.

3. Create an object

EXERCISE 6:

1. Define a **Book** class with the following attributes: **Title, Author (Full name), Price**.
2. Define a **constructor** used to initialize the attributes of the method with values entered by the user.
3. Set the View() method to display information for the current book.
4. Write a program to test the Book class.

EXERCISE 7:

Write a CPP function to find the Max of three numbers

SOLUTION PROGRAMS:

PROGRAM 1:

```
#include <iostream>

using namespace std;

class Rectangle

{

private: int length, width,a;

public:

void area()

{

cout<<"\nEnter length and breadth: ";

cin>>length>>width;

a= length*width;

}
void display()
{
    cout<<"\nThe area of rectangle is "<<a<<endl;
}
};
int main()
{
Rectangle r;
r.area();
r.display();
return 0;
}
```

PROGRAM 2:

```
#include <iostream>
using namespace std;
class Person
{
private: string name , age;
public:
void display()
{
    cout<<"Enter your name and age";

    cin>>name>>age;

    cout<<"\nThe name is "<<name<<"\nThe age is "<<age<<endl;

}

};

class Student:public Person

{

  private: string section;

  public:

  void display_student()

 {

     cout<<"\nEnter your section of study";

     cin>>section;

     cout<<"\nSection is "<<section<<endl;

 }

};

int main()

{

Student s;

s.display();

s.display_student();
```

```
return 0;
}
```

PROGRAM 3:

```
 #include <iostream>
using namespace std;
class BankAccount
{
private:
string name;
int account_number,balance,dep,wd,fee;
public:
void deposit()
{
   cout<<"Enter the balance amount ";
   cin>>balance;
   cout<<"Enter the amount to deposit";
   cin>>dep;
   cout<<"The deposited amount is"<<dep;
   balance+=dep;
   cout<<"\nCurrent balance is"<<balance<<endl;
}
```

```
void withdraw()

{

   cout<<"Enter the amount to withdraw";

   cin>>wd;

   cout<<"The withdraw amount is"<<wd;

   balance-=wd;

   cout<<"\nCurrent balance is"<<balance<<endl;

}

void bankfees()

{

   fee=(balance*5)/100;

   cout<<"\nBank fee is"<<fee<<endl;

}

};

int main()

{

BankAccount b;

b.deposit();

b.withdraw();

b.bankfees();

return 0;
```

```
}
```

PROGRAM 4:

```
#include <iostream>

using namespace std;

class Computational

{

private:

int n,res=0;

public:

void sum()

{

   cout<<"Enter values ";

   cin>>n;

   for(int i=1;i<=n;++i)

   {

      res+=i;

   }

   cout<<"\nThe sum is"<<res<<endl;

}

};

int main()
```

```
{
Computational c;
c.sum();
return 0;
}
```

PROGRAM 5:

```
#include <iostream>
using namespace std;
class Calculation
{
private:
int a,b,avg;
public:
void Average()
{
   cout<<"Enter two values ";
   cin>>a>>b;
   a+=b;
   avg=a/2;
   cout<<"\nThe sum is"<<a<<endl;
   cout<<"\nThe average is"<<avg<<endl;
}

};

int main()

{

Calculation c;

c.Average();

return 0;

}
```

PROGRAM 6:

```
#include <iostream>

using namespace std;

class Book

{

private:

string book,author;

int price;

public:

Book(string a,string b,int c)

{

    book=a;

    author=b;
    price=c;
}
void view()
{
    cout<<"Book details";
    cout<<"\n Book name\t"<<book<<"\nAuthor\t"<<author<<"\nPrice of the
book\t"<<price;
}
};
int main()
{
string x,y;
int z;
cout<<"Enter the book name, author and price of the book";
cin>>x>>y>>z;
Book b(x,y,z);
b.view();
return 0;
}
```

PROGRAM 7:

```
#include <iostream>
using namespace std;
```

```
class Max

{

private:

int a,b,c;

public:

void max()

{

    cout<<"Enter three values";

    cin>>a>>b>>c;

    if(a>b && a>c)

        cout<<"\n A is the greatest value";

    else if(b>a && b>c)

        cout<<"\n B is the greatest value";

    else

        cout<<"\n C is the greatest value";

}

};

int main()

{

Max m;

m.max();
return 0;

}
```

FILE HANDLING IN CPP

Files are used to store data in a storage device permanently. File handling provides a mechanism to store the output of a program in a file and to perform various operations on it.

A stream is an abstraction that represents a device on which operations of input and output are performed. A stream can be represented as a source or destination of characters of indefinite length depending on its usage.

In C++ we have a set of file handling methods. These include ifstream, ofstream, and fstream. These classes are derived from fstrembase and from the corresponding iostream class. These classes, designed to manage the disk files, are declared in fstream and therefore we must include fstream and therefore we must include this file in any program that uses files.

fstream Library:

· ofstream – Creates and writes to files

· ifstream - Reads from files

· fstream – A combination of ofstream and ifstream:creates, reads and writes to a file

Example:

```
#include<iostream>

#include<fstream>
```

FILE OPERATIONS IN CPP:

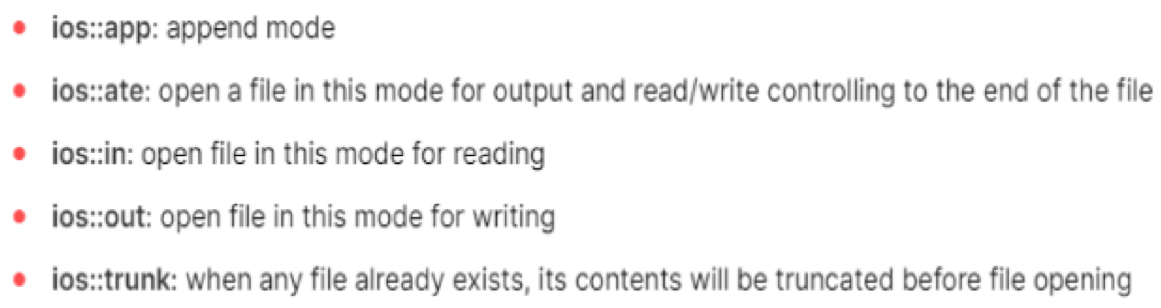
1. open()-Create a file

2. read() – Read the data from the file

3. write() – Write the data to file

4. close() – Close the file

MODES FOR OPENING FILES:



- **ios::app**: append mode
- **ios::ate**: open a file in this mode for output and read/write controlling to the end of the file
- **ios::in**: open file in this mode for reading
- **ios::out**: open file in this mode for writing
- **ios::trunk**: when any file already exists, its contents will be truncated before file opening

PROGRAM 1:

```
#include<iostream>

#include<fstream>

using namespace std;

int main()

{

   fstream file;

   file.open("FileName", ios::out);

   if (!file)

   {

      cout<<"Error while creating the file";

   }

   else
```

```
    {
        cout<<"File created successfully";
        file.close();
    }
    return 0;
}
```

PROGRAM 2:

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  ofstream MyFile("filename.txt");
  MyFile << "Files can be tricky, but it is fun enough!";
  MyFile.close();
}
```

PROGRAM 3:

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
```

```
{
   fstream FileName;
   FileName.open("FileName.txt", ios::out);
   if (!FileName)
   {
      cout<<" Error while creating the file ";
   }
   else
   {
      cout<<"File created and data got written to file";
      FileName<<" Files can be tricky, but it is fun enough!";
      FileName.close();
   }
   return 0;
}
```

PROGRAM 4:

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
```

```
{
  ofstream MyWriteFile("filename.txt");
  MyWriteFile << "Files can be tricky, but it is fun enough!";
  MyWriteFile.close();
  string myText;
  ifstream MyReadFile("filename.txt");
  while (getline (MyReadFile, myText))
  {
    cout << myText;
  }
  MyReadFile.close();
}
```

PROGRAM 5:

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
 ofstream file1("File");
 file1<<"Hello";
 file1.close();
 ifstream file2("File");

 char ch;

 while(file2)

 {

  file2.get(ch);

  cout<<ch;

 }

 file2.close();

}
```

PROGRAM 6:

```
    #include <fstream>

    #include <iostream>

    using namespace std;

    int main ()

    {

       char input[75];

       ofstream file;

       file.open("testout.txt");

       cout <<"\nWriting to a text file:" << endl;

       cout << "\nPlease Enter your name:\n ";

       cin.getline(input, 100);
       file << input << endl;

       cout << "\nPlease Enter your age:\n ";

       cin >> input;

       cin.ignore();

       file << input << endl;

       file.close();

       ifstream f;

       string line;

       f.open("testout.txt");

       cout << "\nReading from a text file:" << endl;

       while (getline (f,line))

       {

       cout << line << endl;

       }

       f.close();

       return 0;

    }
```

ARRAYS

An array is a collection of elements of the same type placed in contiguous memory locations that can be individually referenced by using an index to a unique identifier.

Advantages of an Array in C/C++:

1. Random access of elements using array index.

2. Use of less line of code as it creates a single array of multiple elements.
3. Easy access to all the elements.
4. Traversal through the array becomes easy using a single loop.
5. Sorting becomes easy as it can be accomplished by writing less line of code.

Disadvantages of an Array in C/C++:

1. Allows a fixed number of elements to be entered which is decided at the time of declaration. Unlike a linked list, an array in C is not dynamic.
2. Insertion and deletion of elements can be costly since the elements are needed to be managed in accordance with the new memory allocation.

SINGLE - DIMENsIONAL ARRAY DECLARATION:

SYNTAX:

datatype arrayname[arraysize];

datatype: Type of data to be stored in the array. Here data_type is valid C datatype

arrayname: Name of the array

arraysize: Size of the array

EXAMPLE:

Array declaration by specifying size:

```
// Array declaration by specifying size
int arr1[10];
int n = 10;
int arr2[n];
```

Array declaration by initializing elements:

```
// Array declaration by initializing elements
```

int arr[] = { 10, 20, 30, 40 }

```
/* Compiler creates an array of size 4.
Above is same as "int arr[4] = {10, 20, 30, 40}"*/
```

Array declaration by specifying size and elements:

```
// Array declaration by specifying size and initializing elements
```

int arr[6] = { 10, 20, 30, 40 }

```
/* Compiler creates an array of size 6, initializes first 4 elements as specified
by user        and rest two elements as 0. above is same as "int arr[] = {10, 20,
30,40, 0, 0}"*/
```

PROGRAM1:

```
#include<iostream>
using namespace std;
int main()
{
int i=0;
int marks[5];
marks[0]=80;
marks[1]=60;
marks[2]=70;
marks[3]=85;
marks[4]=75;
for(i=0;i<5;i++)
{
cout<<"\n"<<marks[i];
}
return 0;
}
```

PROGRAM 2:

```
#include<iostream>
using namespace std;
int main()
{
int i=0;
int marks[5]={20,30,40,50,60};
for(i=0;i<5;i++)
{
cout<<"\n"<<marks[i];
}
return 0;
}
```

PROGRAM 3:

```
#include<iostream>
using namespace std;
int main()
{
int mark[5],i,sum=0,avg;
 for(i=0;i<5;++i)
{
	cout<<"Enter marks:";
```

```
            cin>>mark[i];

            sum+=mark[i];

}

 avg=sum/5;

cout<<"Average=\t"<<avg;

 return 0;

}
```

Two - Dimensional array:

SYNTAX

data_type array_name[x][y];

INITIALIZATION:

```
 int x[3][4] = {0, 1 ,2 ,3 ,4 , 5 , 6 , 7 , 8 , 9 , 10 , 11}
```

(OR)

```
int x[3][4] = {{0,1,2,3}, {4,5,6,7}, {8,9,10,11}};
```

PROGRAM 4:

```
#include <iostream>

using namespace std;

int main()

{

   int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};

   int i, j;

   for ( i = 0; i < 5; i++ )
{
      for ( j = 0; j < 2; j++ )
     {
        cout<<"a["<<i<<"]["<<j<<"] ="<<a[i][j]<<"\n";
      }
   }
      return 0;
}
```

MULTI DIMENSIONAL ARRAYS:

SYNTAX:

data_type array_name[size1][size2]....[sizeN];

INITIALIZATION:

```
int x[2][3][4] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
                  20, 21, 22, 23};
```

(OR)

```
int x[2][3][4] =
 {
   { {0,1,2,3}, {4,5,6,7}, {8,9,10,11} },{ {12,13,14,15},{16,17,18,19},{20,21,22,23} }
 };
```

PROGRAM 5:

```
#include<iostream>
using namespace std;
```

```
int main()
{
        int x[2][3][2] ;
        cout<<"Enter 12 values:\n";
    for (int i = 0; i < 2; ++i)
        {
                for (int j = 0; j < 3; ++j)
                {
                        for (int k = 0; k < 2; ++k)
                        {
                                cin>>x[i][j][k];
                        cout<<"X["<<i<<"]["<<j<<"]["<<k<<"]="<<x[i][j][k]<<"\n";
                        }
                }
        }
        return 0;
}
```

STRINGS

Strings are used for storing text.

A string variable contains a collection of characters surrounded by double quotes:

PROGRAM 1:

```
#include <iostream>

#include <string>
```

```
using namespace std;
int main() {
  string greeting = "Hello";
  cout << greeting;
  return 0;
}
```

PROGRAM 2:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
  string firstName = "ABC";
  string lastName = "XYZ";
  string fullName = firstName + " " + lastName;
  cout << fullName;
  return 0;
}
```

PROGRAM 3:

```
#include <iostream>
#include <string>
using namespace std;
int main()
```

```
{
  string firstName = "ABC ";
  string lastName = "XYZ";
  string fullName = firstName.append(lastName);
  cout << fullName;
  return 0;
}
```

PROGRAM 4:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
  string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  cout << "The length of the txt string is: " << txt.length();
  return 0;
}
```

PROGRAM 5:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
  string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

  cout << "The length of the txt string is: " << txt.size();

  return 0;

}
```

PROGRAM 6:

```
#include <iostream>

#include <string>

using namespace std;

int main()

{

  string myString = "Hello";

  cout << myString[0];

  cout << myString[1];

  cout << myString[2];

  cout << myString[3];

  cout << myString[4];

  return 0;

}
```

PROGRAM 7:

```
#include <iostream>

#include <string>

using namespace std;

int main()
```

```
{
  string myString = "Hello";
  myString[0] = 'J';
  cout << myString;
  return 0;
}
```

PROGRAM 8:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
  string fullName;
  cout << "Type your full name: ";
  getline (cin, fullName);
  cout << "Your name is: " << fullName;
  return 0;
}
```