Ex:No: 01	Inline Functions and Default Arguments

Aim:

To Write a Program to implement the concept of Inline Functions and Default Arguments.

Program:

//Program 1A
//Program using Inline Function to findout the square of a number

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

// Inline function
inline int square(int x)
{
	return x * x;
}

int main()
{
	int num = 5;

	// Calling inline function
	int res = square(num);
	cout << res;
	return 0;
}
```

Output:

25

//Program 1B
// Inline function to return maximum of two numbers

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

inline int getMax(int a, int b)
```

```
{
    return (a > b) ? a : b;
}

int main()
{
    int num1, num2;

    cout << "Enter two numbers: ";
    cin >> num1 >> num2;

    cout << "Maximum is: " << getMax(num1, num2) << endl;

    return 0;
}
```

Output:

```
Enter two numbers:
48
65
Maximum is: 65
```

//Program 1C
// Inline function to convert Celsius to Fahrenheit

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

inline float toFahrenheit(float celsius)
{
    return (celsius * 9.0 / 5.0) + 32;
}

int main()
{
    float celsius;

    cout << "Enter temperature in Celsius: ";
    cin >> celsius;

    cout << "Temperature in Fahrenheit: " << toFahrenheit(celsius) <<
endl;
```

```
    return 0;
}
```

Output:

```
Enter temperature in Celsius: 32
Temperature in Fahrenheit: 89.6

Enter temperature in Celsius: 28
Temperature in Fahrenheit: 82.4
```

//Program 1D
// Inline function to check whether the number is even or odd0

```
#include <iostream>
using namespace std;
inline bool isEven(int num)
{
    return (num % 2 == 0);
}
int main()
{
    int number;

    cout << "Enter a number: ";
    cin >> number;

    if (isEven(number))
        cout << number << " is even." << endl;
    else
        cout << number << " is odd." << endl;

    return 0;
}
```

Output:

```
Enter a number: 56
56 is even.


Enter a number: 65
65 is odd.
```

//Program 1E
//Program to define function with default arguments

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

int main()
{
clrscr();
int sum(int a,int b=10,int c=15,int d=20);
int a=2;
nit b=3;
int c=4;
int d=5;

cout<<"Sum = "<<sum(a,b,c,d);
cout<<"Sum = "<<sum(a,b,c);
cout<<"Sum = "<<sum(a,b);
cout<<"Sum = "<<sum(a);
cout<<"Sum = "<<sum(b,c,d);
return 0;
}

sum(int j,int k,int l,int m)
{
return(j+k+l+m);
}
```

Output:

```
sum=14
sum=29
sum=40
sum=47
sum=32
```

//Program 1F
// Function with default arguments for character and count

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

void printLine(char ch = '*', int count = 10)
{
```

```
    for (int i = 0; i < count; ++i)
{
        cout << ch;
    }
    cout << endl;
}

int main()
{
    // Calling with no arguments (uses both default values)
    printLine();

    // Calling with only one argument (default count)
    printLine('#');

    // Calling with both arguments
    printLine('=', 20);

    return 0;
}
```

Output:

```
**********
##########
====================
```

//Program 1G
// Function with default arguments for width and height

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

int volume(int length, int width = 1, int height = 1)
{
    return length * width * height;
}

int main()
{
    // Only length provided (default width = 1, height = 1)
    cout << "Volume (length only): " << volume(10) << endl;
```

```
    // Length and width provided (default height = 1)
    cout << "Volume (length and width): " << volume(10, 5) << endl;

    // All dimensions provided
    cout << "Volume (length, width, height): " << volume(10, 5, 2) << endl;

    return 0;
}
```

Output:

```
Volume (length only): 10
Volume (length and width): 50
Volume (length, width, height): 100
```

Ex:No: 02	Call by value, call by address and call by reference

Aim:

To Write a Program to implement the concept of Call by Value, Call by Reference and Call by Address and display the result.

Program:

//Program 2A

// Write a Program to implement the concept of Call by Value, Call by //Reference and Call by Address

```
#include <iostream.h>
#include <process.h>
using namespace std;
void main()
{
        void funa(int s);
        void funb(int &);
        void func(int *);
        int s=4;
        funa(s);
        cout<<"\nvalue of s="<<s<<"address of s:"<<unsigned(&s);
        funb(s);
        cout<<"\nvalue of s="<<s<<"address of s:"<<unsigned(&s);
        func(&s);
        cout<<"\nvalue of s="<<s<<"address of s:"<<unsigned(&s);
}
void funa(int i)
{
        i++;
}
void funb(int &k)
{
        k++;
}
void func(int *j)
{
        ++*j;
}
```

Output:

value of s=4address of s:65524
value of s=5address of s:65524
value of s=6address of s:65524

//Program 2B
//Call by Value, **Call by Reference**, and **Call by Address – for swapping of two Numbers**

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

// Call by Value
void swapByValue(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
}

// Call by Address (using pointers)
void swapByAddress(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Call by Reference
void swapByReference(int &a, int &b)
{
    int temp = a;
    a = b;
    b = temp;
}

int main()
{
    int x = 10, y = 20;

    cout << "Original values: x = " << x << ", y = " << y << endl;

    swapByValue(x, y);
    cout << "After swapByValue: x = " << x << ", y = " << y << " (No
    change)" << endl;

    swapByAddress(&x, &y);
    cout << "After swapByAddress: x = " << x << ", y = " << y << " (Values
    swapped)" << endl;
```

```
    swapByReference(x, y);
    cout << "After swapByReference: x = " << x << ", y = " << y << "
    (Swapped again)" << endl;

    return 0;
}
```

Output:

Original values: x = 10, y = 20

After swapByValue: x = 10, y = 20 (No change)

After swapByAddress: x = 20, y = 10 (Values swapped)

After swapByReference: x = 10, y = 20 (Swapped again)

Ex:No: 03	Function overloading

Aim:

To Write a Program to implement the concept of Function overloading.

Program:

//Program 3A
//Function overloading for squaring a number

```
#include<iostream>
using namespace std;
int sqr(int);
float sqr(float);
main()
{
        int a=15;float b=2.5;
        cout<<"square="<<sqr(a)<<"\n";
        cout<<"square="<<sqr(b)<<"\n";
        return 0;
}

int sqr(int s)
{
        return(s*s);
}

float sqr(float j)
{
        return(j*j);
}
```

Output:

square = 225

square = 6.25

//Program 3B
//Program to find the area of rectangle,triangle and sphere by overloading function

```
#include <iostream>
using  namespace std;
#define pi 3.14

int calarea(int length,int breadth);
float calarea(double base,double height);
float calarea(double radius);
void main()
{
        int area1;float area2;double area3;
        area1=calarea(10,20);
        area2=calarea(4.5,2.1);
        area3=calarea(3.121);
        cout<<"Area of rectangle is:"<<area1<<endl;
        cout<<"Area of triangle is:"<<area2<<endl;
        cout<<"Area of sphere is:"<<area3<<endl;
}
int calarea(int length,int breadth)
{
        return(length*breadth);
}
float calarea(double base,double height)
{
        return((0.5)*base*height);
}
float calarea(double radius)
{
        return((4/3)*pi*radius*radius*radius);
}
```

Output:

Area of rectangle is: 200

Area of Triangle is :4.725

Area of Sphere is : 95.457697

//Program 3C
//Function Overloading to add two and three integers

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

// Function to add two integers
int add(int a, int b)
{
    return a + b;
}

// Function to add three integers
int add(int a, int b, int c)
{
    return a + b + c;
}

// Function to add two doubles
double add(double a, double b)
{
    return a + b;
}

int main()
{
    cout << "add(5, 10) = " << add(5, 10) << endl;
    cout << "add(5, 10, 15) = " << add(5, 10, 15) << endl;
    cout << "add(5.5, 4.5) = " << add(5.5, 4.5) << endl;
    return 0;
}
```

Output:

```
add(5, 10) = 15

add(5, 10, 15) = 30

add(5.5, 4.5) = 10
```

```
//Program 3D
//Function Overloading for arithmetic operations
#include <iostream>
using namespace std;

// Addition
int calculate(int a, int b)
{
    return a + b;
}

// Subtraction
float calculate(float a, float b)
{
    return a - b;
}

// Multiplication
double calculate(double a, double b, char op)
{
    if (op == '*')
        return a * b;
    else
        return 0;
}

// Division
double calculate(double a, double b, int dummy)
{
    if (b != 0)
        return a / b;
    else {
        cout << "Error: Division by zero!" << endl;
        return 0;
    }
}

int main()
{
    int a = 10, b = 5;
    float x = 20.5f, y = 10.2f;
    double m = 15.0, n = 3.0;


    cout << "Addition (int): " << calculate(a, b) << endl;
    cout << "Subtraction (float): " << calculate(x, y) << endl;
```

```
    cout << "Multiplication (double): " << calculate(m, n, '*') << endl;
    cout << "Division (double): " << calculate(m, n, 0) << endl;


    return 0;

}
```

Output:

Addition (int): 15

Subtraction (float): 10.3

Multiplication (double): 45

Division (double): 5

//Program 3E

```
// Function Overloading for Arithmetic Operations

#include <iostream>
using namespace std;
// Addition for two integers
int add(int a, int b)
{
   return a + b;
}
// Addition for two floating-point numbers
float add(float a, float b)
{
   return a + b;
}
// Subtraction for integers
int subtract(int a, int b)
{
   return a - b;
}
// Subtraction for floating-point numbers
float subtract(float a, float b)
{
   return a - b;
}
// Multiplication for integers
int multiply(int a, int b)
{
   return a * b;
}
// Multiplication for floating-point numbers
float multiply(float a, float b)
{
   return a * b;
}
// Division for integers
int divide(int a, int b)
{
        if (b != 0)
        {
                return a / b;
        }
        else
         {
                 cout << "Error: Division by zero!" << endl;
                return 0;
```

```
    }
}
// Division for floating-point numbers
float divide(float a, float b)
{
    if (b != 0.0f)
    {
        return a / b;
    }
   else
   {
        cout << "Error: Division by zero!" << endl;
        return 0.0f;
    }
}

int main()
{
    int a = 8, b = 4;
    float x = 9.5f, y = 2.5f;
    cout << "Addition (int): " << add(a, b) << endl;
    cout << "Addition (float): " << add(x, y) << endl;
    cout << "Subtraction (int): " << subtract(a, b) << endl;
    cout << "Subtraction (float): " << subtract(x, y) << endl;
    cout << "Multiplication (int): " << multiply(a, b) << endl;
    cout << "Multiplication (float): " << multiply(x, y) << endl;
    cout << "Division (int): " << divide(a, b) << endl;
    cout << "Division (float): " << divide(x, y) << endl;
    return 0;
}
```

Output:

```
Addition (int): 12
Addition (float): 12
Subtraction (int): 4
Subtraction (float): 7
Multiplication (int): 32
Multiplication (float): 23.75
Division (int): 2
Division (float): 3.8
```

//Program 3F

// Function Overloading for Displaying Different Types

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

// Display integer
void display(int a) {
    cout << "Integer: " << a << endl;
}

// Display float
void display(float a) {
    cout << "Float: " << a << endl;
}

// Display string (C-style)
void display(const char* a) {
    cout << "String: " << a << endl;
}

// Display two integers
void display(int a, int b) {
    cout << "Two integers: " << a << " and " << b << endl;
}

int main() {
    display(42);
    display(3.14159f);
    display("Hello, world!");
    display(10, 20);

    return 0;
}
```

Output:

```
Integer: 42
Float: 3.14159
String: Hello, world!
Two integers: 10 and 20
```

//Program 3G

```
// Function Overloading for drawing different types of lines

#include <iostream>
using namespace std;

// Draw dotted line: length times '.'
void drawLine(int length)
{
    for (int i = 0; i < length; i++)
    {
         cout << ". ";
    }
    cout << endl;
}

// Draw dashed line: length times '--'
void drawLine(double length)
{
    for (int i = 0; i < (int)length; i++)
    {
        cout << "-- ";
    }
    cout << endl;
}

// Draw starred line: length times '*'
void drawLine(char ch, int length)
{
    for (int i = 0; i < length; i++)
    {
        cout << ch << " ";
    }
    cout << endl;
}

int main()
{
    int len = 10;
    cout << "Dotted line:" << endl;
    drawLine(len);         // calls drawLine(int)
    cout << "Dashed line:" << endl;
    drawLine(10.0);        // calls drawLine(double)
```

```
    cout << "Starred line:" << endl;
    drawLine('*', len);    // calls drawLine(char, int)
    return 0;
}
```

Output:

Dotted line:

. . . . . .

Dashed line:

-- -- -- -- -- -- -- -- -- --

Starred line:

* * * * * * * * * *

Ex:No: 04	Classes and objects

Aim:

To Write a Program to implement the concept of Classes and objects.

Program:

```
//Program 4A
//Classes and Objects
#include <iostream>
using namespace std;
// Define a class called Car
class Car
{
public:
    // Data members (attributes)
    string brand;
    string model;
    int year;

    // Member function (method)
    void displayInfo()
{
        cout << "Brand: " << brand << endl;
        cout << "Model: " << model << endl;
        cout << "Year: " << year << endl;
    }
};
int main()
{
    // Create an object of Car
    Car car1;
    // Assign values to the object's data members
    car1.brand = "Toyota";
    car1.model = "Innova";
    car1.year = 2022;
    // Call the member function using the object
    car1.displayInfo();
    return 0;
}
```

Output:

Brand: Toyota

Model: Innova

Year: 2022

//Program 4B

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

// Class definition
class Car {
public:
    // Properties (data members)
    string brand;
    string model;
    int year;

    // Method (member function)
    void displayInfo() {
        cout << "Brand: " << brand << ", Model: " << model << ", Year: " << year
<< endl;
    }
};

int main() {
    // Creating objects of the Car class
    Car car1;
    Car car2;

    // Assigning values to object properties
    car1.brand = "Toyota";
    car1.model = "Corolla";
    car1.year = 2015;

    car2.brand = "Ford";
    car2.model = "Mustang";
    car2.year = 2020;

    // Calling method to display car information
    car1.displayInfo();
    car2.displayInfo();

    return 0;
}
```

Output:

Brand: Toyota, Model: Corolla, Year: 2015

Brand: Ford, Model: Mustang, Year: 2020

//Program 4C

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

// Define a class called Student
class Student
{
        public:
                string name;
                  int age;
         void setDetails(string n, int a)
        {
                name = n;
                age = a;
        }

        void displayDetails()
        {
                cout << "Name: " << name << ", Age: " << age << endl;
        }
};

int main()
{
   // Create multiple objects of Student
   Student student1, student2;
   // Set details for each object
   student1.setDetails("Arun", 20);
   student2.setDetails("Priya", 19);
   // Display details of each object
   cout << "Student 1 Details:\n";
   student1.displayDetails();
   cout << "Student 2 Details:\n";
   student2.displayDetails();
   return 0;
}
```

Output:

Student 1 Details:
Name: Arun, Age: 20
Student 2 Details:
Name: Priya, Age: 19

//Program 4D

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

// Define a class called Employee
class Employee
{
        private:
           string name;
           int age;

        public:
        // Member function to set data
           void setData(string empName, int empAge)
        {
                 name = empName;
                 age = empAge;
         }

        // Member function to display data
        void displayData()
        {
                cout << "Employee Name: " << name << endl;
                 cout << "Employee Age: " << age << endl;
        }
};
int main()
{
   // Declare variables to hold input
   string nameInput;
   int ageInput;
   cout << "Enter employee name: ";
   getline(cin, nameInput);
   cout << "Enter employee age: ";
   cin >> ageInput;
   // Create an object of Employee
   Employee emp;
   // Pass data to member function
   emp.setData(nameInput, ageInput);
   // Display the data
   emp.displayData();
   return 0;
}
```

Output:

Enter employee name: Kavitha

Enter employee age: 28

Employee Name: Kavitha

Employee Age: 28

//Program 4E

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

// Class definition
class Employee
{
        public:
                 int empID;
                string name;
                double salary;

    // Member function to set data
    void setData(int id, string empName, double empSalary)
        {
                empID = id;
                name = empName;
                salary = empSalary;
        }

    // Member function to display data
    void displayData()
 {
         cout << "Employee ID: " << empID << ", Name: " << name << ", Salary:
        $" << salary << endl;
    }
};

int main()
{
    // Creating multiple Employee objects
    Employee emp1, emp2, emp3;

    // Assigning data to each object
    emp1.setData(101, "Alice", 50000);
    emp1.displayData();
}
```

Output:

Employee ID: 101, Name: Alice, Salary: $50000

//Program 4E
//Member Function Inside the class

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

// Class definition
class Employee
{
        public:
                int empID;
                string name;
                double salary;

    // Member function to set data
    void setData(int id, string empName, double empSalary)
   {
        empID = id;
        name = empName;
        salary = empSalary;
    }

    // Member function to display data
    void displayData()
{
        cout << "Employee ID : " << empID << ", Name: " << name << ", Salary:
        $" << salary << endl;
    }
};

int main()
{
    // Creating multiple Employee objects
    Employee emp1, emp2, emp3;
    int eid;
    string ename;
    double esal;
        // Assigning data to each object
    cout<<"Enter Employee ID:";
    cin>>eid;
    cout<<"Enter Employee Name:";
    cin>>ename;
    cout<<"Enter Employee Salary:";
    cin>>esal;
```

```
    emp1.setData(eid,ename,esal);
    emp1.displayData();
}
```

Output:

Enter Employee ID:88

Enter Employee Name:Shakthi

Enter Employee Salary:85000

Employee ID : 88, Name: Shakthi, Salary: $85000

//Program 4F
//Program to access member function outside the class

```
#include <iostream>
using namespace std;
class stud
{
public:
int rno,m1,m2,m3,m4,m5,tot;
char name[10],br[5],res[10];
float avg;
void read();
void write();
};
void stud::read()
{
        cout<<"\nEnter Name:";
        cin>>name;
        cout<<"\nEnter Roll No:";
        cin>>rno;
        cout<<"\nEnter Branch:";
        cin>>br;
        cout<<"\nEnter mark1:";
        cin>>m1;
        cout<<"\nEnter mark2:";
        cin>>m2;
        cout<<"\nEnter mark3:";
        cin>>m3;
        cout<<"\nEnter mark4:";
        cin>>m4;
        cout<<"\nEnter mark5:";
        cin>>m5;
}

void stud::write()
{
        tot=m1+m2+m3+m4+m5;
        avg=tot/5;
        cout<<"\nThe Student Name is: "<<name;
        cout<<"\nThe Rollno is: "<<rno;
        cout<<"\nThe Branch is: "<<br;
        cout<<"\nMark1 :"<<m1;
        cout<<"\nMark2 :"<<m2;
        cout<<"\nMark3 :"<<m3;
        cout<<"\nMark4 :"<<m4;
        cout<<"\nMark5 :"<<m5;
```

```
        cout<<"\nThe Total is :"<<tot;
        cout<<"\nThe average is :"<<avg;
        if ((m1>=40)&&(m2>=40)&&(m3>=40)&&(m4>=40)&&(m5>=40))
                cout<<"\nThe Result is Pass";
        else
                cout<<"\nThe Result is Fail";
}
int main()
{
        stud s;
        s.read();
        s.write();
        return 0;
}
```

Output:

```
Enter Name:Vijay
Enter Roll No:14
Enter Branch:IT
Enter mark1:85
Enter mark2:96
Enter mark3:54
Enter mark4:87
Enter mark5:82

The Student Name is: Vijay
The Rollno is: 14
The Branch is: IT
Mark1 :85
Mark2 :96
Mark3 :54
Mark4 :87
Mark5 :82
The Total is :404
The average is :80
The Result is Pass
```

Ex:No: 05	Constructors and destructors

Aim:

To Write a Program to implement the concept of Constructors and destructors.

Program:

//Program 6A

//**Program that demonstrates the use of constructors and destructors**

```
#include <iostream>

using namespace std;

class Sample
{
        public:
        // Constructor
        Sample()
        {
                cout << "Constructor called: Object created." << endl;
        }
        // Member function
        void showMessage()
        {
                cout << "Hello from the member function!" << endl;
        }
        // Destructor
         ~Sample()
        {
                 cout << "Destructor called: Object destroyed." << endl;
        }
};
int main()
{
   cout << "Entering main function." << endl;
   Sample obj; // Constructor will be called here
   obj.showMessage();
   cout << "Exiting main function." << endl;
   // Destructor will be called automatically when obj goes out of scope
   return 0;
}
```

Output:

Entering main function.

Constructor called: Object created.

Hello from the member function!

Exiting main function.

Destructor called: Object destroyed.

//Program 6B

//program with multiple constructors for the single class

```
#include <iostream>

using namespace std;

class num
{
        private:
        int a;
        float b;
        char c;
        public:
        num(int m,float j, char k);
        num(int m,float j);
        num();
        void show()
        {
                cout<<"\n\ta="<<a<<"b="<<b<<"c="<<c;
        }
        };

num::num(int m,float j,char k)
{
        cout<<"\nconstructor with three arguments";
        a=m;
        b=j;
        c=k;
}

num::num(int m,float j)
{
        cout<<"\nconstructor with 2 arg";
        a=m;
        b=j;
        c=' ';
}

num::num()
{
        cout<<"\nconstructor without arg";
        a=b=c=NULL;
}
```

```
main()
{
        class num x(4,5.6,'s');
        x.show();
        class num y(1,3.4);
        y.show();
        class num z;
        z.show();
        return 0;
}
```

Output:

constructor with three arguments

a=4b=5.6c=s

constructor with 2 arg

a=1b=3.4c=

constructor without arg

a=0b=0c=

//Program 6C

//Parameterized Constructor

//To write a program to create constructor with arguments and

// pass the arguments to the constructor*/

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

class num
{
        private:
        int a,b,c;
        public:
        num(int m,int j,int k);
        void show()
        {
                cout<<"\na="<<a<<"b="<<b<<"c="<<c;
        }
};

num::num(int m,int j,int k)
{
        a=m;
        b=j;
        c=k;
}
main()
{
        num x=num(4,5,7);
        num y(1,2,3);
        x.show();
        y.show();
        return 0;
}
```

Output:

```
a=4 b=5 c=7
a=1 b=2 c=3
```

//Program 6D

```
#include <iostream>
using namespace std;
class Book
{
        private:
                string title,author;
                float price;
        public:
           // Constructor
           Book(string t1, string t2, float t3)
           {
              title = t1;
              author = t2;
              price = t3;
              cout << "Constructor called. Book \"" << title << "\" is
              created.\n";
           }

        // Destructor
        ~Book()
        {
                        cout << "Destructor called. Book \"" << title << "\" is
                        destroyed.\n";
         }

        void display()
        {
                cout << "Book Title: " << title << endl;
                cout<< "Author Name: "<< author<<endl;
                cout<<"Book Price"<< price<<endl;
         }
};
int main()
{
   Book b1("C++ Fundamentals", "Richard",485.50 );  // Constructor is automatically called here
   b1.display();
   return 0;  // Destructor will be called automatically when b1 goes out of scope
}
```

Output:

```
onstructor called. Book "C++ Fundamentals" is created.
Book Title: C++ Fundamentals
Author Name: Richard
Book Price485.5
Destructor called. Book "C++ Fundamentals" is destroyed.
```

//Program 6E

```
#include <iostream>

using namespace std;

class Student
{
        private:
           string name;
           int age;

        public:
        // Constructor
        Student()
        {
                name = "Shakthi";
                age = 20;
                cout << "Constructor called!" << endl;
        }

        // Member function to display values
         void display()
        {
                 cout << "Name: " << name << ", Age: " << age << endl;
        }
};

int main()
{
   // Object is created; constructor is automatically called
   Student s1;
   s1.display();

   return 0;
}
```

Output:

```
Constructor called!

Name: Shakthi, Age: 20
```

//Program 6E

//Program1 for Copy Constructor

```
// Parameterized Constructor
   Book(string t, int p)
{
       title = t;
       pages = p;
       cout<<"Parameterized Constructor Called"<<endl;
       cout << "Title: " << title << ", Pages: " << pages << endl;
 }

   // Copy Constructor
   Book(const Book &b)
   {
       title = b.title;
       pages = b.pages;
       cout << "Copy constructor called!" << endl;
   }

   // Display function
   void display()
   {
       cout << "Title: " << title << ", Pages: " << pages << endl;
   }
};
int main()
{
        Book book1("C++ Programming", 450); // Calls parameterized
        constructor
        Book book2 = book1;                 // Calls copy constructor
        cout << "Book 1 details: ";
        book1.display();
        cout << "Book 2 details: ";
        book2.display();
        return 0;
}
```

Output:

Parameterized Constructor Called
Title: C++ Programming, Pages: 450
Copy constructor called!
Book 1 details: Title: C++ Programming, Pages: 450
Book 2 details: Title: C++ Programming, Pages: 450

//Program 6E

//Program2 for Copy Constructor

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

class Employee
{
        private:
                int id;
                string name;

        public:
                // Parameterized Constructor
                Employee(int empId, string empName)
                {
                        id = empId;
                        name = empName;
                        cout << "Parameterized constructor called.\n";
                }

                // Copy Constructor
                Employee(const Employee &emp)
                {
                        id = emp.id;
                        name = emp.name;
                        cout << "Copy constructor called.\n";
                }

                // Display method
                void display()
                {
                          cout << "Employee ID: " << id << "\n";
                         cout << "Employee Name: " << name << "\n";
                }
};

int main()
{
   Employee e1(101, "Veesha");   // Calls parameterized constructor
   cout << "Original Employee:\n";
   e1.display();

   Employee e2 = e1;  // Calls copy constructor
   cout << "\nCopied Employee:\n";
```

```
    e2.display();

    return 0;
}
```

Output:

```
Parameterized constructor called.
Original Employee:
Employee ID: 101
Employee Name: Veesha
Copy constructor called.

Copied Employee:
Employee ID: 101
Employee Name: Veesha
```

Ex:No: 06	Friend Function & Friend Class

Aim:

To Write a Program to understand friend function & friend class

Program:

// Friend Function – 6A
//Program for adding 2 numbers using friend function

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

class first;
class second
{
   int s;
   public:
      void getvalue()
      {
         cout<<"\n enter a no for class second:";
         cin>>s;
      }
      friend void sum (second,first);
};

class first
{
   int f;
   public:
      void getvalue()
      {
         cout<<"\n enter a no for class first:";
         cin>>f;
      }
      friend void sum (second,first);
};

void sum (second d,first t)
{
   cout<<"\nThe sum of two numbers is : "<<t.f+d.s;
}
```

```
int main()
{
        first a;
        second b;
        a.getvalue();
        b.getvalue();
        sum(b,a);
        return 0;
}
```

Output:

```
enter a no for class first:55
 enter a no for class second:89
The sum of two numbers is : 144
```

// Friend Function – 6B
//Program for adding 2 numbers using friend function

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

class ClassB; // Forward declaration

class ClassA
{
    private:
     int numA;

    public:
        ClassA(int a)
        {
            numA = a;
        }
        // Declare friend function
    friend void add(ClassA, ClassB);
};

class ClassB
{
    private:
     int numB;

    public:
        ClassB(int b)
    {
        numB = b;
    };

    // Same friend function
    friend void add(ClassA, ClassB);
};

// Friend function definition
void add(ClassA objA, ClassB objB)
{
    cout << "Sum = " << objA.numA + objB.numB << endl;
}
```

```
int main()
{
    ClassA a(100);
    ClassB b(20);
    add(a, b);
    return 0;
}
```

Output:

```
Sum = 120
```

//Friend Class Example – Student & Result – 6C

```
#include <iostream>
using namespace std;
class Student
{
    private:
        string name;
         int marks;

    public:
        Student(string n, int m)
        {
            name = n;
            marks = m;
        }
    // Make Result a friend class
    friend class Result;
};

class Result
{
    public:
        void display(Student s)
        {
            cout << "Student: " << s.name << endl;
            cout << "Marks: " << s.marks << endl;
            if (s.marks >= 50)
                cout << "Result: PASS" << endl;
            else
                cout << "Result: FAIL" << endl;
        }
};
int main()
{
    Student s1("Veesha", 92);
    Result r1;
    r1.display(s1);
    return 0;
}
```

Output:

```
Student: Veesha
Marks: 92
Result: PASS
```

Ex:No: 07	Unary operator overloading.

Aim:

To Write a Program to implement the concept of Unary operator overloading.

Program:

// Unary operator overloading – 7A
//Write a program to increment member variables of objet.
//overload unary ++ operator

```
#include <iostream>
using namespace std;
class num
{
        private:
        int a,b,c,d;
        public:
        num(int j,int k, int m,int l)
        {
                a=j;
                b=k;
                c=m;
                d=l;
        }
void show(void);
void operator ++();
};

void num::show()
{
        cout<<"a="<<a<<endl;
        cout<<"b="<<b<<endl;
        cout<<"c="<<c<<endl;
        cout<<"d="<<d<<endl;
}

void num::operator ++()
{
        ++a;
        ++b;
        ++c;
        ++d;
}
```

```
int main()
{
        num x(33,200,507,777);
        cout<<"\nBefore Increment of Object X:"<<endl;
        x.show();
        ++x;
        cout<<"\nAfter Increment of Object X:"<<endl;
        x.show();
        return 0;
}
```

Output:

Before Increment of X:a=3b=2c=5d=7

After Increment of X:a=4b=3c=6d=8

//Unary operator overloading program in C++ with unary minus(-), increment(++)and Decrement(- -) Operator – 7B

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

class Number
{
   private:
      int value;

   public:
      // Constructor
      Number(int v = 0) : value(v)
      {

      }

   // Display function
   void display()
   {
      cout << "Value = " << value << endl;
   }

   // Overloading unary minus (-)
   Number operator-()
   {
      return Number(-value);   // Return a new object
   }

   // Overloading prefix increment (++n)
   Number operator++()
   {
      value = value + 1;
      return Number(value);    // Return new object
   }

   // Overloading postfix increment (n++)
   Number operator++(int)
   {
      Number temp(value);      // Store old value
      value = value + 1;       // Increase current object
      return temp;             // Return old value object
   }
};
```

```
int main()
{
    Number n1(1665);

    cout << "Original: ";
    n1.display();

    Number n2 = -n1;   // Unary minus
    cout << "After Unary - : ";
    n2.display();

    Number n3 = ++n1;  // Prefix increment
    cout << "After Prefix ++ : ";
    n3.display();

    Number n4 = n1++;  // Postfix increment
    cout << "After Postfix ++ : ";
    n4.display();

    cout << "Final value of n1: ";
    n1.display();

    return 0;
}
```

Output:

```
Original: Value = 1665

After Unary - : Value = -1665

After Prefix ++ : Value = 1666

After Postfix ++ : Value = 1666

Final value of n1: Value = 1667
```

// **Unary operator overloading** in C++ with **unary minus (-) – 7C**

```
#include <iostream>
using namespace std;
class Number
{
    private:
        int value;

    public:
        // Constructor
        Number(int v = 0) : value(v)
        {

        }
    // Display function
    void display()
    {
        cout << "Value = " << value << endl;
    }
    // Overloading unary minus (-)
    Number operator-()
    {
        return Number(-value);   // Return new object with negative value
    }
};
int main()
{
    Number n1(121);
    cout << "Original: ";
    n1.display();
    Number n2 = -n1;   // Applying overloaded unary minus
    cout << "After Unary - : ";
    n2.display();
    return 0;
}
```

Output:

Original: Value = 121

After Unary - : Value = -121

Ex:No: 08	Binary Operator Overloading

Aim:

To Write a Program to implement the concept of Binary operator overloading.

Program:

//Binary operator overloading – 8A
//Write a program to overload + binary operator

```
#include <iostream>
using namespace std;
class num
{
        private:
                int a,b,c,d;
        public:
                 void input(void);
                void show(void);
                num operator+(num);
};
void num::input()
{
   cout<<"\nEnter values for a,b,c and d:";
   cin>>a;
   cin>>b;
   cin>>c;
   cin>>d;
}
void num::show()
{
   cout<<"\na="<<a;
   cout<<"\nb="<<b;
   cout<<"\nc="<<c;
   cout<<"\nd="<<d;
}
num num::operator +(num t)
{
   num tmp;
   tmp.a=a+t.a;
   tmp.b=b+t.b;
   tmp.c=c+t.c;
   tmp.d=d+t.d;
```

```
    return(tmp);
}
int main()
{
    num x,y,z;
    cout<<"\nIn object x";
    x.input();
    cout<<"\nIn object y";
    y.input();
    z=x+y;
    cout<<"\nObject X Data Members are:";
    x.show();
    cout<<"\nObject Y Data Members are:";
    y.show();
    cout<<"\nAddition of Object X and Object Y are stored in Object Z
are:";
    z.show();
    return 0;
}
```

Output:

```
In object x
Enter values for a,b,c and d:8 9 4 23

In object y
Enter values for a,b,c and d:7 54 21 38

Object X Data Members are:
a=8
b=9
c=4
d=23

Object Y Data Members are:
a=7
b=54
c=21
d=38

Addition of Object X and Object Y are stored in Object Z are:
a=15
b=63
c=25
d=61
```

// Binary operator overloading using multiplication operator (*) – 8B

```
#include <iostream>
using namespace std;
class Multiply
{
        private:
                int num;
        public:
            // Constructor
            Multiply(int n = 0)
            {
                num = n;
            }
            // Display function
            void display()
            {
                cout << "Value = " << num << endl;
            }
    // Overloading binary * operator
    Multiply operator*(Multiply m)
    {
        return Multiply(num * m.num);
    }
};
int main()
{
    Multiply m1(44), m2(52);
    cout << "First Number: ";
    m1.display();
    cout << "Second Number: ";
    m2.display();
    Multiply m3 = m1 * m2;
    cout << "After Multiplication: ";
    m3.display();
    return 0;
}
```

Output:

First Number: Value = 44

Second Number: Value = 52

After Multiplication: Value = 2288

Ex:No: 09	Inheritance

Aim:

To Write a Program to demonstrate the concept of Single inheritance, Multiple inheritance, Multilevel inheritance, Hierarchical inheritance and Hybrid inheritance.

Program:

//Single inheritance – 9.1A

```
#include<iostream.h>
using namespace std;
class ABC
{
        protected:
        char name[15];
        int age;
};

class abc:public ABC
{
        float ht;
        float wt;
        public:

void getdata()
{
        cout<<"\nEnter the name:";
        cin>>name;
        cout<<"\nenter the age:";
        cin>>age;
        cout<<"\nenter height:";
        cin>>ht;
        cout<<"\nenter the weight:";
        cin>>wt;
}

void show()
{
        cout<<"\nThe name is:"<<name;
        cout<<"\nThe age is:"<<age;
        cout<<"\nThe Height:"<<ht;
        cout<<"\nThe weight:"<<wt;
```

```
}
};

void main()
{
        abc x;
        x.getdata();
        x.show();
        getch();
}
```

Output:

Enter the name:Dhanush

enter the age:5

enter height:100

enter the weight:15

The name is:Dhanush

The age is:5

The Height:100

The weight:15

//Single Inheritance – 9.1B

```
#include <iostream>
using namespace std;
class base
{
    protected:
        int rollno;
        string name,fclr;
 };
class der:public base
{
    protected:
        float salary;
        string fcomp;
    public:
    void input();
    void display();
};
void der::input()
{
    cout<<"Enter Name: ";
    cin>>name;
    cout<<"Enter Rollno:  ";
    cin>>rollno;
    cout<<"Enter Favourite Colour: ";
    cin>>fclr;
    cout<<"Enter Salary: ";
    cin>>salary;
    cout<<"Enter Favourite Company: ";
    cin>>fcomp;
}
void der::display()
{
    cout<<"Name is:"<<name<<endl;
    cout<<"Roll Number is:  "<<rollno<<endl;
    cout<<"Favourite Colour is: "<<fclr<<endl;
    cout<<"Salary is: "<<salary<<endl;
    cout<<"Favourite Company is: "<<fcomp<<endl;
}

int main()
{
    der d;
    d.input();
```

```
    d.display();
    return 0;

}
```

Output:

Enter Name: Nila

Enter Rollno: 121

Enter Favourite Colour: Black

Enter Salary: 85000

Enter Favourite Company: Accenture

Name is:Nila

Roll Number is: 121

Favourite Colour is: Black

Salary is: 85000

Favourite Company is: Accenture

//Single Inheritance – 9.1C

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

void Base::displayX() {
    cout << "Value of x (from Base class): " << x << endl;
}
class Derived : public Base {
private:
    int y;

public:
    void setY(int b);
    void displayY();
};

// Member function definitions outside the class
void Derived::setY(int b) {
    y = b;
}

void Derived::displayY() {
    cout << "Value of y (from Derived class): " << y << endl;
}

int main() {
    Derived d;

    d.setX(10);
    d.setY(20);

    d.displayX();
    d.displayY();
```

```
    return 0;
}
```

Output:

Value of x (from Base class): 10

Value of y (from Derived class): 20

//Multiple Inheritance – 9.2A

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

class A
{
    protected:
        int a;
    public:
        void p()
        {
         cout<<"HI"<<endl;
        }
};
class B
{
        protected:
                int b;
};
class C
{
        protected:
                int c;
};
class D
{
        protected:
                nt d;
};
class E:public A,B,C,D
{
        int e;
        public:
        void getdata()
        {
                cout<<"\nenter values of a,b,c,d & e:";
                cin>>a>>b>>c>>d>>e;
        }
        void showdata()
        {
                cout<<"\na="<<a;
                cout<<"\nb="<<b;
                cout<<"\nc="<<c;
                cout<<"\nd="<<d;
                cout<<"\ne="<<e<<endl;
```

```
        }
};
int main()
{
        E x;
        x.getdata();
        x.p();
        x.showdata();
        x.p();
        return 0;
}
```

Output:

```
enter values of a,b,c,d & e:11 22 33 44 55

HI

a=11
b=22
c=33
d=44
e=55

HI
```

//Multiple Inheritance – 9.2 B

```
#include <iostream>
using namespace std;
class tenth
{
    public:
        int tentotal;
};

class twelth
{
    public:
        int twtotal;
};

class derived:public tenth, public twelth
{
    public:
        string collname, stname, stbr;
    public:
        void get()
        {
            cout<<"Enter Student Name: ";
            cin>>stname;
            cout<<"Enter College Name: ";
            cin>>collname;
            cout<<"Enter Tenth Total: ";
            cin>>tentotal;
            cout<<"Enter Twelth Total: ";
            cin>>twtotal;
            cout<<"Enter Student Branch: ";
            cin>>stbr;
            cout<<endl;
        }
        void put()
        {
            cout<<"Student Name is: "<<stname<<endl;
            cout<<"College Name is: "<<collname<<endl;
            cout<<"Tenth Mark is  : "<<tentotal<<endl;
            cout<<"Twelth Mark is : "<<twtotal<<endl;
            cout<<"Student Branch is:"<<stbr<<endl;
        }
};
```

```
int main()
{
    derived d;
    d.get();
    d.put();
    return 0;
}
```

Output:

Enter Student Name: Nila

Enter College Name: RPSIT

Enter Tenth Total: 452

Enter Twelth Total: 485

Enter Student Branch: IT

Student Name is: Nila

College Name is: RPSIT

Tenth Mark is : 452

Twelth Mark is : 485

Student Branch is:IT

//Multiple Inheritance – 9.2 C

```
#include <iostream>
using namespace std;
class Person
{
public:
    void showPerson()
    {
        cout << "This is a Person Class." << endl;
    }
};


class Employee
{
public:
    void showEmployee()
    {
        cout << "This is an Employee Class." << endl;
    }
};

class Manager : public Person, public Employee
{
public:
    void showManager()
    {
        cout << "This is a Manager Class." << endl;
    }
};

int main()
{
    Manager m;
    m.showPerson();
    m.showEmployee();
    m.showManager();
    return 0;
}
```

Output:

```
This is a Person Class.
This is an Employee Class.
This is a Manager Class.
```

//Multilevel Inheritance – 9.3 A

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

class A1
{
        protected:
                char name [15];
                int age;
};
class A2:public A1
{
        protected:
                float ht;
                float wt;
};
class A3:public A2
{
        protected:
                char sex;
                public:
        void get()
        {
                cout<<"name=";
                cin>>name;
                cout<<"age=";
                cin>>age;
                cout<<"sex=";
                cin>>sex;
                cout<<"height=";
                cin>>ht;
                cout<<"weight=";
                cin>>wt;
        }
        void show()
        {
                cout<<"\n name="<<name;
                cout<<"\n age="<<age<<"year";
                cout<<"\n sex="<<sex;
                cout<<"\n height="<<ht<<"feet";
                cout<<"\n weight="<<wt<<"kg";
        }
};
```

```
int main()
{
        A3 x;
        x.get();
        x.show();
        return 0;
}
```

Output:

name=Sathya

age=32

sex=F

height=155

weight=68

name=Sathya

age=32year

sex=F

height=155feet

weight=68kg

//Multilevel Inheritance – 9.3B

```
#include <iostream>
using namespace std;
class Base
{
    protected:
        int x;

    public:
        void setX(int a);
        void displayX();
};

void Base::setX(int a)
{
    x = a;
}
void Base::displayX()
{
    cout << "Value of x (from Base): " << x << endl;
}

class Derived1 : public Base
{
    protected:
        int y;

    public:
        void setY(int b);
        void displayY();
};

void Derived1::setY(int b)
{
    y = b;
}
void Derived1::displayY()
{
    cout << "Value of y (from Derived1): " << y << endl;
}

class Derived2 : public Derived1
{
    private:
        int sum;
```

```
    public:
        void calculateSum();
        void displaySum();
};

void Derived2::calculateSum()
{
    sum = x + y;
}
void Derived2::displaySum()
{
    cout << "Sum (x + y) = " << sum << endl;
}

int main()
{
    Derived2 d;

    d.setX(10);
    d.setY(20);
    d.displayX();
    d.displayY();
    d.calculateSum();
    d.displaySum();

    return 0;
}
```

Output:

```
Value of x (from Base): 10

Value of y (from Derived1): 20

Sum (x + y) = 30
```

//Multilevel Inheritance – 9.3C

```
#include <iostream>
using namespace std;
class Base
{
    protected:
        int x;

    public:
        void inX();
        void outX();
};

void Base::inX()
{
   cout<<"Enter X Value: ";
   cin>>x;
}
void Base::outX()
{
    cout << "Value of X is: " << x << endl;
}

class Derived1 : public Base
{
    protected:
        int y;

    public:
        void inY();
        void outY();
};

void Derived1::inY()
{
    cout<<"Enter Y Value: ";
    cin>>y;
}
void Derived1::outY()
{
    cout << "Value of Y is: " << y << endl;
}

class Derived2 : public Derived1
{
```

```
    private:
        int sum;

    public:
        void calculateSum();
        void displaySum();
};

void Derived2::calculateSum()
{
    sum = x + y;
}
void Derived2::displaySum()
{
    cout << "Sum (x + y) = " << sum << endl;
}

int main()
{
    Derived2 d;

    d.inX();
    d.inY();
    d.outX();
    d.outY();
    d.calculateSum();
    d.displaySum();

    return 0;
}
```

Output:

```
Enter X Value: 450

Enter Y Value: 422

Value of X is: 450

Value of Y is: 422

Sum (x + y) = 872
```

//Hierarchical Inheritance – 9.4

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

class Person
{
   protected:
      string name;
      int age;

   public:
      void getData();
      void displayData();
};

void Person::getData()
{
   cout << "Enter name: ";
   cin >> name;
   cout << "Enter age: ";
   cin >> age;
}

void Person::displayData()
{
   cout << "Name: " << name << "\n";
   cout << "Age: " << age << "\n";
}

class Teacher : public Person
{
   char subject[20];
   public:
      void getTeacherData();
      void displayTeacherData();
};

void Teacher::getTeacherData()
{
   getData();   // from Person
   cout << "Enter subject taught: ";
   cin >> subject;
}

void Teacher::displayTeacherData()
```

```
{
    displayData();  // from Person
    cout << "Subject: " << subject << "\n";
}

class Engineer : public Person
{
        char specialization[20];
        public:
                void getEngineerData();
                void displayEngineerData();
};

void Engineer::getEngineerData()
{
    getData();   // from Person
    cout << "Enter specialization: ";
    cin >> specialization;
}

void Engineer::displayEngineerData()
{
    displayData();  // from Person
    cout << "Specialization: " << specialization << "\n";
}

// Derived Class 3
class Doctor : public Person
{
        char hospital[20];
        public:
                void getDoctorData();
                void displayDoctorData();
};

void Doctor::getDoctorData()
{
    getData();   // from Person
    cout << "Enter hospital name: ";
    cin >> hospital;
}

void Doctor::displayDoctorData()
{
    displayData();  // from Person
    cout << "Hospital: " << hospital << "\n";
```

```
}

// Main Function
int main()
{
    Teacher t;
    Engineer e;
    Doctor d;

    cout << "\n--- Enter Teacher Details ---\n";
    t.getTeacherData();

    cout << "\n--- Enter Engineer Details ---\n";
    e.getEngineerData();

    cout << "\n--- Enter Doctor Details ---\n";
    d.getDoctorData();

    cout << "\n--- Teacher Details ---\n";
    t.displayTeacherData();

    cout << "\n--- Engineer Details ---\n";
    e.displayEngineerData();

    cout << "\n--- Doctor Details ---\n";
    d.displayDoctorData();

    return 0;
}
```

Output:

```
--- Enter Teacher Details ---
Enter name: Veejai
Enter age: 38
Enter subject taught: Networks

--- Enter Engineer Details ---
Enter name: Prabu
Enter age: 34
Enter specialization: Animation

--- Enter Doctor Details ---
Enter name: Sathya
Enter age: 32
```

Enter hospital name: KMC

--- Teacher Details ---
Name: Veejai
Age: 38
Subject: Networks

--- Engineer Details ---
Name: Prabu
Age: 34
Specialization: Animation

--- Doctor Details ---
Name: Sathya
Age: 32
Hospital: KMC

//Hybrid Inheritance - 9.5

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

class Person
{
public:
    void showPerson()
    {
        cout << "I am a person." << endl;
    }
};

class Player : public Person
{
    public:
        void showPlayer()
        {
            cout << "I am a player." << endl;
        }
};

class CricketPlayer : public Player
{
    public:
        void showCricketSkills()
        {
         cout << "I play cricket." << endl;
        }
};

class FootballPlayer : public Player
{
public:
    void showFootballSkills()
    {
        cout << "I play football." << endl;
    }
};

class AllRounder : public CricketPlayer, public FootballPlayer
{
public:
    void showAllSkills()
    {
```

```
        cout << "I am an all-rounder in sports!" << endl;
    }
};

int main()
{
    AllRounder ar;

    // Ambiguity arises because Person and Player are inherited twice
    // So we need to specify which path to use
    ar.CricketPlayer::showPerson();
    ar.CricketPlayer::showPlayer();
    ar.showCricketSkills();
    ar.showFootballSkills();
    ar.showAllSkills();

    return 0;
}
```

Output:

I am a person.

I am a player.

I play cricket.

I play football.

I am an all-rounder in sports!

Ex:No: 10	Function templates and Class templates

Aim:

To Write a Program to define the Function templates and Class templates.

Program:

// Function Template Example – 10A

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

// Function Template to find maximum
template <typename T>
T findMax(T a, T b)
{
    return (a > b) ? a : b;
}

int main()
{
    cout << "Max of 10 and 20: " << findMax(10, 20) << endl;
    cout << "Max of 5.5 and 3.2: " << findMax(5.5, 3.2) << endl;
    cout << "Max of 'A' and 'Z': " << findMax('A', 'Z') << endl;
    return 0;
}
```

Output:

```
Max of 10 and 20: 20
Max of 5.5 and 3.2: 5.5
Max of 'A' and 'Z': Z
```

// Class Template Example – 10B

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

// Class Template

template <class T>
class Box
{
        T value;
        public:
                Box(T v) { value = v; }
                void display() {
                cout << "Value: " << value << endl;
   }
};

int main()
{
   Box<int> intBox(100);
   Box<double> doubleBox(99.99);
   Box<string> stringBox("Hello Templates");

   intBox.display();
   doubleBox.display();
   stringBox.display();
   return 0;
}
```

Output:

```
Value: 100
Value: 99.99
Value: Hello Templates
```

//Write a program to define members of template type – 10C

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

template <class T>
class data
{
        T x;
        public:
        data (T u)
        {
                x=u;
        }
        void show (T y)
        {
                cout<<"X="<<x;
                cout<<"\nY="<<y<<"\n";
        }
}

int main()
{
        data <char> c('B');
        data <int> i(100);
        data <double> d(48.25);
        c.show('A');
        i.show(65);
        d.show(68.25);
        return 0;
}
```

Output:

X=B

Y=A

X=100

Y=65

X=48.25

Y=68.25

//Write a program to create square() function using templates – 10 D

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

template <class S>
class sqr
{
        public:
        sqr (S c)
        {
                cout<<"\n"<<"C="<<c*c;
        }
};

int main()
{
        sqr <int> i(25);
        sqr <float> f(15.2);
        return 0;
}
```

Output:

C=625

C=231.039993

Ex:No: 11	Virtual Function

Aim:

To Write a Program for illustrating the concept of Virtual function.

Program:

//Virtual Base Class

```
#include <iostream.h>
using namespace std;
class A1
{
        protected:
        int a1;
};
class A2:public virtual A1   //virtual declararion
{
        protected:
        int a2;
};
class A3:public virtual A1   //virtual declaration
{
        protected:
        int a3;
};
class A4:public A2,A3        //virtual declaration
{
        int a4;
        public:
        void get()
        {
```

```
            cout<<"\nenter values for a1,a2,a3and a4:";
            cin>>a1>>a2>>a3>>a4;
        }
        void put()
        {
            cout<<"\na1="<<a1;
            cout<<"\na2="<<a2;
            cout<<"\na3="<<a3;
            cout<<"\na4="<<a4;
        }
};
void main()
{
        A4 a;
        a.get();
        a.put();
        getch();
}
```

Output:

```
enter values for a1,a2,a3and a4:
4
5
6
7

a1=4
a2=5
a3=6
a4=7
```

//Virtual function – 11B

//write a program to declare virtual function

//and execute the same function defined in base and derived classes

```
#include<iostream.h>
using namespace std;
class first
{
        int b;
        public:
        first(){b=10;}
        virtual void display()
        {
        cout<<"\nb="<<b;
        }
};
class second:public first
{
        int d;
        public:
        second(){d=20;}
        void display()
        {
                cout<<"\nd="<<d;
        }
};
int main()
{
        first f, *p;
        second s;
        p=&f;
```

```
        p->display();
        p=&s;
        p->display();
        getch();
        return 0;
}
```

Output:

b=10

d=20

Virtual Function – 11C

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

class Shape
{
	public:
	virtual void draw()
	{
		cout << "Drawing a generic shape" << endl;
	 }
};

class Circle : public Shape
{
	public:
	void draw() override
	{
		cout << "Drawing a Circle" << endl;
	 }
};

class Rectangle : public Shape
{
	public:
	void draw() override
	{
		 cout << "Drawing a Rectangle" << endl;
	 }
};
```

```
int main()
{
    Shape* s;   // Base class pointer

    Circle c;
    Rectangle r;

    s = &c;
    s->draw();   // Calls Circle's version (runtime polymorphism)

    s = &r;
    s->draw();   // Calls Rectangle's version

    return 0;
}
```

Output:

```
Drawing a Circle
Circle
Drawing a Rectangle
Rectangle
```

Ex:No:12	Abstract class

Aim:

To Write a Program to implement the concept of Abstract class

Program:

//Programs using Abstract class – 12 A

//Simple Abstract Class

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

// Abstract class
class Shape
{
        public:
           virtual void draw() = 0;  // Pure virtual function
};

// Derived class
class Circle : public Shape
{
        public:
                 void draw() override
        {
                cout << "Drawing Circle" << endl;
         }
};

// Derived class
class Square : public Shape
{
        public:
           void draw() override
        {
       cout << "Drawing Square" << endl;
   }
};

int main()
{
   Shape* s1 = new Circle();
```

```
    Shape* s2 = new Square();

    s1->draw();
    s2->draw();

    delete s1;
    delete s2;

    return 0;
}
```

Output:

```
Drawing Circle
Drawing Square
```

// Abstract Class with Normal Function – 12 B

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

class Animal
{
        public:
        virtual void sound() = 0;  // Pure virtual function
          // Normal function
        void sleep()
        {
                cout << "Sleeping..." << endl;
        }
};

class Dog : public Animal
{
        public:
                void sound() override
        {
                cout << "Dog barks: Woof! Woof!" << endl;
         }
};

int main()
{
    Animal* a = new Dog();
    a->sound();
    a->sleep();

    delete a;
    return 0;
}
```

Output:

```
Dog barks: Woof! Woof!
Sleeping...
```

// Abstract Class - Multiple Derived Classes - 12 C

```
#include <iostream>
using namespace std;
class Vehicle
{
        public:
          virtual void start() = 0; // pure virtual
};

class Car : public Vehicle
{
        public:
                void start() override
        {
                cout << "Car starts with key" << endl;
        }
};
class Bike : public Vehicle
{
        public:
                    void start() override
                {
                        cout << "Bike starts with kick" << endl;
                 }
};
int main()
{
   Vehicle* v1 = new Car();
   Vehicle* v2 = new Bike();

   v1->start();
   v2->start();

   delete v1;
   delete v2;

   return 0;
}
```

Output:

```
Car starts with key
Bike starts with kick
```

Ex:No: 13	String Manipulation

Aim:

To Write a Program to implement the concept of String Manipulation.

//Manipulating Strings in C++ - 13 A

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

int main()
{
   char str1[50] = "Hello";
   char str2[50] = "World";

   cout << "Length of str1: " << strlen(str1) << endl;

   strcat(str1, str2);  // Concatenate
   cout << "Concatenated String: " << str1 << endl;

   strcpy(str2, str1);  // Copy
   cout << "Copied String: " << str2 << endl;

   cout << "Comparison result: " << strcmp(str1, str2) << endl;

   return 0;
}
```

Output:

Length of str1: 5

Concatenated String: HelloWorld

Copied String: HelloWorld

Comparison result: 0

// Manipulating Strings in C++ Using string Class - 13B

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

int main()
{
    string str1 = "Hello";
    string str2 = "World";

    cout << "Length of str1: " << str1.length() << endl;

    str1 += " " + str2;  // Concatenate
    cout << "Concatenated String: " << str1 << endl;

    string sub = str1.substr(0, 5);  // Extract substring
    cout << "Substring: " << sub << endl;

    str1.replace(6, 5, "C++");  // Replace "World" with "C++"
    cout << "After Replace: " << str1 << endl;

    cout << "Find position of 'C++': " << str1.find("C++") << endl;

    return 0;
}
```

Output:

Length of str1: 5

Concatenated String: Hello World

Substring: Hello

After Replace: Hello C++

Find position of 'C++': 6

```
//Iterating through a String  - 13C
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string str = "Programming";

    cout << "Characters in string: ";
    for (char c : str)
   {
        cout << c << " ";
   }
    cout << endl;

    return 0;
}
```

Output:

```
Characters in string: P r o g r a m m i n g
```