C++ PROGRAMMING LAB

S.No	Details	No. of Hours
1	Write a C++ program to demonstrate function overloading, Default arguments and Inline function.	
2	Write a C++program to demonstrate Class and Objects	
3	WriteaC++programtodemonstratetheconceptofPassingObjectstoFunctions	
4	Write a C++ program to demonstrate the Friend Functions.	
5	Write a C++ program to demonstrate the concept to Passing Objects to Functions	
6	Write a C++ program to demonstrate Constructor and Destructor	
7	Write a C++ program to demonstrate Unary Operator Overloading	
8	Write a C++ program to demonstrate Binary Operator Overloading	
9	Write a C++ program to demonstrate: • Single Inheritance • Multilevel Inheritance • Multiple Inheritance • Hierarchical Inheritance • Hybrid Inheritance	
10	Write a C++ program to demonstrate Virtual Functions.	
11	Write a C++ program to manipulate a Text File.	
12	Write a C++ program to perform Sequential I/O Operations on a file.	
13	Write a C++ program to find the Biggest Number using Command Line Arguments	

14	Write a C++ program to demonstrate Class Template	
15	Write a C++ program to demonstrate Function Template.	
16	Write a C++ program to demonstrate Exception Handling.	

1. **Write a C++ program to demonstrate function overloading, Default Arguments and Inline function.**

```
#include <iostream>
using namespace std;
int sum(float a,int b){
  cout<<"using functions with 2 arguments"<<endl;
  return a+b;
}
int sum(int a,int b,int c){
  cout<<"using unctions with 3 arguments"<<endl;
  return a+b+c;
}
int volume(double r,int h){
  return(3.14*r*r*h);
}
int volume(int a){
  return(a*a*a);
}
int volume(int l,int b,int h){
  return(l*b*h);
}
int main(){
  cout<<"the sm of 1 and 2 is"<<sum(1,2)<<endl;
  cout<<"the sum of 1,2 and 5 is"<<sum(1,2,5)<<endl;
  cout<<"the volume of cuboid of 1,2 and 5 is"<<volume(1,2,5)<<endl;
  cout<<"the volume of clinder of radius 1 and height 5 is"<<volume(1,5)<<endl;
  cout<<"the volume o cube o side 1 is"<<volume(1)<<endl;
  return 0;
}
```

Output:

```
the sm of 1 and 2 isusing functions with 2 arguments
3
```

the sum of 1,2 and 5 isusing unctions with 3 arguments

8

the volume of cuboid of 1,2 and 5 is10

the volume of clinder of radius 1 and height 5 is15

the volume o cube o side 1 is1

2. **Write a C++ program to demonstrate Class and Objects**

```
/ Program to illustrate the working of
// objects and class in C++ Programming

#include <iostream>
using namespace std;

// create a class
class Room {

  public:
    double length;
    double breadth;
    double height;

    double calculate_area() {
        return length * breadth;
    }

    double calculate_volume() {
        return length * breadth * height;
    }
};

int main() {
```

```
    // create object of Room class
    Room room1;

    // assign values to data members
    room1.length = 42.5;
    room1.breadth = 30.8;
    room1.height = 19.2;

    // calculate and display the area and volume of the room
    cout << "Area of Room =  " << room1.calculate_area() << endl;
    cout << "Volume of Room =  " << room1.calculate_volume() << endl;

    return 0;
}
```

Output

Area of Room = 1309

Volume of Room = 25132.8

3. **Write a C++ program to demonstrate the concept to Passing Objects to Functions**

```
// C++ program to calculate the average marks of two students

#include <iostream>
using namespace std;

class Student {

  public:
   double marks;

   // constructor to initialize marks
   Student(double m)
```

```
    : marks{m} {
  }
};

// function that has objects as parameters
double average_marks(Student s1, Student s2) {

   // return the average of marks of s1 and s2
   return (s1.marks + s2.marks)/2 ;
}

int main() {
   Student student1(88.0), student2(56.0);

 // pass the objects as arguments
  cout << "Average Marks = " << average_marks(student1, student2) << "\n";

   return 0;
}
```

Output

Average Marks = 72

4. Write a C++ program to demonstrate the Friend Functions.

```
// C++ program to demonstrate the working of friend function

#include <iostream>
using namespace std;

class Distance {
   private:
```

```
    int meter;

    // friend function
    friend int addFive(Distance);

  public:
    Distance() : meter(0) {}

};

// friend function definition
int addFive(Distance d) {

  //accessing private members from the friend function
  d.meter += 5;
  return d.meter;
}

int main() {
  Distance D;
  cout << "Distance: " << addFive(D);
  return 0;
}
```

Output

Distance: 5

5. **Write a C++ program to demonstrate the concept to Passing Objects to Functions**

```
// C++ program to calculate the average marks of two students
```

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

class Student {

  public:
   double marks;

   // constructor to initialize marks
   Student(double m)
    : marks{m} {
   }
};

// function that has objects as parameters
double average_marks(Student s1, Student s2) {

   // return the average of marks of s1 and s2
   return (s1.marks + s2.marks)/2 ;
}

int main() {
   Student student1(88.0), student2(56.0);

  // pass the objects as arguments
   cout << "Average Marks = " << average_marks(student1, student2) << "\n";

   return 0;
}
```

Output

Average Marks = 72

6. **Write a C++ program to demonstrate Constructor and Destructor**

```
#include <iostream>

using namespace std;

//Rectangle class to demonstrate the working of Constructor and Destructor in
CPP
class Rectangle {
    public:
        float length, breadth;

    //Declaration of the default Constructor of the Rectangle Class
    public:
        Rectangle() {
            cout << "\n\n****** Inside the Constructor ******* \n\n";
            length = 2;
            breadth = 4;
        }

    //Declaration of the Destructor of the Rectangle Class
    public:
        ~Rectangle() {
            cout << "\n\n****** Inside the Destructor ******* \n\n";
        }
};

//Defining the main method to access the members of the class
int main() {
```

```
    cout << "\n\nWelcome to Studytonight :-)\n\n\n";
    cout << " =====  Program to demonstrate the concept of Constructor and Destructor in CPP  ===== \n\n";

    cout << "\nCalling the default Constructor of the Rectangle class to initialize the object.\n\n";

    //Declaring the Class object to access the class members
    Rectangle rect;

    cout << "\nThe Length of the Rectangle set by the Constructor is = " << rect.length << "\n\n";

    cout << "\nThe Breadth of the Rectangle set by the Constructor is = " << rect.breadth << "\n\n";

    return 0;
}
```

Output:

```
Welcome to Studytonight :-)

 =====  Program to demonstrate the concept of Constructor and Destructor in CPP  =====

Calling the default Constructor of the Rectangle class to initialize the object.


****** Inside the Constructor *******

The Length of the Rectangle set by the Constructor is = 2

The Breadth of the Rectangle set by the Constructor is = 4


****** Inside the Destructor *******
```

7. **Write a C++ program to demonstrate Unary Operator Overloading**

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

class complex {
   int a, b, c;
public:

   complex() {
   }

   void getvalue() {
      cout << "Enter the Two Numbers:";
      cin >> a>>b;
   }

   void operator++() {
      a = ++a;
      b = ++b;
   }

   void operator--() {
      a = --a;
      b = --b;
   }

   void display() {
      cout << a << "+\t" << b << "i" << endl;
   }
```

```
};

void main() {
    clrscr();
    complex obj;
    obj.getvalue();
    obj++;
    cout << "Increment Complex Number\n";
    obj.display();
    obj--;
    cout << "Decrement Complex Number\n";
    obj.display();
    getch();
}
```

Output

```
Enter the two numbers: 3 6
Increment Complex Number
4 +             7i
Decrement Complex Number
3 +             6i
```

8. Write a C++ program to demonstrate Binary Operator Overloading

```
#include <iostream>
using namespace std;
class complex
{
    int a, b;
```

```
    public:

    void get_data(){
        cout << "Enter the value of Complex Numbers a, b: "; cin >> a >> b;
    }

    complex operator+(complex ob)// overaloded operator function +
    {
        complex t;
        t.a = a + ob.a;
        t.b = b + ob.b;
        return (t);
    }

    complex operator-(complex ob)// overaloded operator function -
    {
        complex t;
        t.a = a - ob.a;
        t.b = b - ob.b;
        return (t);
    }

    void display(){
        cout << a << "+" << b << "i" << "\n";
    }
};

int main()
{
    complex obj1, obj2, result, result1;
    obj1.get_data();
    obj2.get_data();
```

```
    result = obj1 + obj2;
    result1 = obj1 - obj2;

    cout << "\n\nInput Values:\n";

    obj1.display();
    obj2.display();

    cout << "\nResult:";

    result.display();
    result1.display();

    return 0;
}
```


Output

```
Enter the value of Complex Numbers a, b: 7 5
Enter the value of Complex Numbers a, b: 3 4

Input Values:
7+5i
3+4i

Result:10+9i
4+1i
```

9. **Write a C++ program to demonstrate:**

- **Single Inheritance**
- **Multilevel Inheritance**
- **Multiple Inheritance**
- **Hierarchical Inheritance**
- **Hybrid Inheritance**

1. Single Inheritance in C++

```
#include <iostream>
using namespace std;
class A
{
   public:
     void display(){
        cout<<"Base class A content.";
     }
};
// sub class derived from base class
class B : public A
{

};

int main(){
   B obj;
   obj.display();
   return 0;
}
```

Output

Base class A content.

2. Multiple Inheritance in C++

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

// First base class
class Animal {
  public:
    Animal() {
      cout << "Animals can Run." << endl;
    }
};
// Second base class
class Bird {
  public:
    Bird() {
      cout << "Birds can Fly." << endl;
    }
};
// sub class derived from two base classes
class Parrot: public Animal, public Bird {

};
int main(){
```

```
    // creating object of sub class will invoke the constructor of base classes
    Parrot b1;
    return 0;
}
```

Output

```
Animals can Run.
Birds can Fly.
```

3. Hierarchical Inheritance in C++

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

// base class
class Animal
{
    public:
        Animal(){
            cout<<"Animals Can Run.";
        }
};
// First Sub class
class Dog : public Animal
{

};
// Second Sub class
class Cat : public Animal
{
```

```
};

int main(){
   Cat obj1;
   Dog obj;2

   return 0;
}
```

Output

Animals Can Run.

Animals Can Run.

4. Multilevel Inheritance in C++

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

class Animal
{
   public:
   Animal()
    {
        cout<<"This is a Animal.";
    }
};

class fourLegs : public Animal
{
```

```
    public:
    fourLegs()
     {
        cout<<"Objects with 4 legs are Animals.";
     }
};

class Cow : public fourLegs
{
    public:
    Cow()
     {
        cout<<"Cow has 4 Legs.";
     }
};

int main()
{
    Cow obj;
    obj.display();
    return 0;
}
```

Output

This is a Animal.

Objects with 4 legs are Animals.

Cow has 4 Legs.

5. Hybrid (Virtual) Inheritance in C++

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

class Animal{
   public:
   Animal()
    {
        cout<<"This is a Animal.";
    }
};

class Fare{
   public:
   Fare()
    {
        cout<<"Fare of Animal.";
    }
};

class Dog : public Animal
{

};

class Cat : public fourLegs, public Fare
{

};
int main(){

   Car obj2;
   return 0;
```

```
}
```

Output

```
This is a Animal.
Fare of Animals.
```

10. Write a C++ program to demonstrate Virtual Functions.

```
#include <iostream>
using namespace std;
class base {
public:
virtual void show(){
cout << "show base class" << endl;
}
void print(){
cout << "print base class" << endl;
}
};
class derived : public base {
public:
void show(){
cout << "show derived class" << endl;
}
void print(){
cout << "print derived class" << endl;
}
};
int main(){
base* bpointr;
derived dev;
```

```
bpointr = &dev;
// runtime binding
bpointr->show();
// compile time binding
bpointr->print();
}
```

Output:

```
show derived class
print base class
```

11. **Write a C++ program to manipulate a Text File.**

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
  fstream file; //object of fstream class

  //opening file "sample.txt" in out(write) mode
  file.open("sample.txt",ios::out);

  if(!file)
  {
     cout<<"Error in creating file!!!"<<endl;
     return 0;
```

```
    }

    cout<<"File created successfully."<<endl;
    //write text into file
    file<<"ABCD.";
    //closing the file
    file.close();

    //again open file in read mode
    file.open("sample.txt",ios::in);

    if(!file)
    {
        cout<<"Error in opening file!!!"<<endl;
        return 0;
    }

    //read untill end of file is not found.
    char ch; //to read single character
    cout<<"File content: ";

    while(!file.eof())
    {
        file>>ch; //read single character from file
        cout<<ch;
    }

    file.close(); //close file

    return 0;
}
```

Output

File created successfully.
File content: ABCD.

12. **Write a C++ program to perform Sequential I/O Operations on a file.**

```
#include <iostream>
#include <fstream>
#include <string.h >
int main()
{
char string[80];
std::cout << "Enter a string \n";
std::cin >> string;
int len=strlen(string);
std::fstream file; // input and output stream
file.open("TEXT",std::iostream::in | std::iostream::out);
for (int i=0;i < len;i++)

file.put(string[i]); // put a character to file
file.seekg(0); // go to the start
char ch;
while (file)
{
file.get(ch); // get a character from file
std::cout << ch; // display it on screen
```

```
}
}
```

Output

Enter a string
input : forton programming
output: forton programming

13. **Write a C++ program to find the Biggest Number using Command Line Arguments**

```
#include <stdio.h>

int main(int argc, char *argv[])

{

        int c[10];

        int i,temp,j,greatest;

        j = 0;

        for(i=1; i<argc; i++)

{

        temp = atoi(argv[i]);

        c[j] = temp;
```

```
        j++;

}

        greatest = c[0];

        for (i = 0; i < 10; i++) {

        if (c[i] > greatest) {

        greatest = c[i];

          }

   }

        printf("Greatest of ten numbers is %d", greatest);

        return 0;

}
```

14. Write a C++program to demonstrate Class Template

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

// Declaring a template class named Test.
template <class T>
class Test
```

```
{
private:
  // A variable (answer) of type T so that it can store results of various types.
  T answer;

public:
  // Constructor of Test class.
  Test(T n) : answer(n)
  {
    cout << "Inside constructor" << endl;
  }

  T getNumber()
  {
    return answer;
  }
};

// Main function
int main()
{
  // Creating an object with an integer type.
  Test<int> numberInt(60);

  // Creating an object with double type.
  Test<double> numberDouble(17.27);

  // Calling the class method getNumber with different data types:
  cout << "Integer Number is: " << numberInt.getNumber() << endl;
  cout << "Double Number = " << numberDouble.getNumber() << endl;

  return 0;
```

```
}
```

Output :

```
Inside constructor
Inside constructor
Integer Number is: 60
Double Number = 17.27
```

15. Write a C++ program to demonstrate Function Template.

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

template <typename T>
T add(T num1, T num2) {
    return (num1 + num2);
}

int main() {
    int result1;
    double result2;
    // calling with int parameters
    result1 = add<int>(2, 3);
    cout << "2 + 3 = " << result1 << endl;

    // calling with double parameters
    result2 = add<double>(2.2, 3.3);
    cout << "2.2 + 3.3 = " << result2 << endl;

    return 0;
```

```
}
```

Output

```
2 + 3 = 5
2.2 + 3.3 = 5.5
```

16. Write a C++ program to demonstrate Exception Handling.

```
// program to divide two numbers
// throws an exception when the divisor is 0

#include <iostream>
using namespace std;

int main() {

    double numerator, denominator, divide;

    cout << "Enter numerator: ";
    cin >> numerator;

    cout << "Enter denominator: ";
    cin >> denominator;

    try {

        // throw an exception if denominator is 0
        if (denominator == 0)
            throw 0;
```

```
        // not executed if denominator is 0
        divide = numerator / denominator;
        cout << numerator << " / " << denominator << " = " << divide << endl;
    }

    catch (int num_exception) {
        cout << "Error: Cannot divide by " << num_exception << endl;
    }


    return 0;
}
```

Output

```
Enter numerator: 72
Enter denominator: 0
Error: Cannot divide by 0
```