LAB MANUAL

Subject : Object Oriented Programming Lab
Code : LC-CSE-214

Department of Computer Science Engineering
RPS College of Engineering and Technology
Mahendergarh, Haryana - 123029
### INSTRUCTIONS

#### LIST OF PROGRAMS

<table>
<thead>
<tr>
<th>PROGRAMNO.</th>
<th>NAME OF PROGRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Study of C++ Standard library functions.</td>
</tr>
</tbody>
</table>
| 2          | a) Write a C++ program to find the sum of individual digits of a positive integer.  
b) Write a C++ program to generate the first n terms of the sequence. |
| 3          | a) Write a C++ program to generate all the prime numbers between 1 and n, where n is a value supplied by the user.  
b) Write a C++ program to find both the largest and smallest number in a list of integers. |
| 4          | a) Write a C++ program to sort a list of numbers in ascending order.  
b) Write a Program to illustrate New and Delete Keywords for dynamic memory allocation. |
| 5          | a) Write a program Illustrating Class Declarations, Definition, and Accessing Class Members.  
b) Program to illustrate default constructor, parameterized constructor and copy constructors. |
| 6          | a) Write a Program to Demonstrate the:-  
Operator overloading  
ii) Function Overloading.  
b) Write a Program to Demonstrate Friend Function and Friend Class. |
| 7          | a) Write a Program to Access Members of a STUDENT Class Using Pointer to Object Members.  
b) Write a Program to Generate Fibonacci Series use Constructor to Initialize the Data Members. |
| 8          | Write C++ programs that illustrate how the following forms of inheritance are supported:  
a) Single inheritance  
b) Multiple inheritance  
c) Multi level inheritance  
d) Hierarchical inheritance |
| 9          | Write a C++ program to illustrate the order of execution of constructors and destructors. |
| 10         | a) Write a Program Containing a Possible Exception. Use a Try Block to Throw it and a Catch Block to Handle it Properly. |
LAB REQUIREMENTS

HARDWARE REQUIREMENT:

- Processor: Intel(R) corei3 – 8100CPU 3.60GHz
- RAM: 8GB
- HDD: 250GB SSD
- Operating System: Windows10

SOFTWARE REQUIREMENT:

- DEV C++
- VS Code
- TURBOC++ IDE

PROGRAM OUTCOMES:

- Understand the features of C++ supporting object oriented programming
- Understand the relative merits of C++ as an object oriented programming language
- Understand how to produce object-oriented software using C++
- Understand how to apply the major object-oriented concepts to implement object oriented programs in C++, encapsulation, inheritance and polymorphism
- Understand advanced features of C++ specifically stream I/O, templates and operator overloading
PROGRAM 1: Write a Program Study of C++ Standard library functions.

C++ Standard Library: The C++ standard library also makes available the facilities of the C standard library, suitably adjusted to ensure static type safety. The descriptions of many library functions rely on the C standard library for the semantics of those functions.

The C++ Standard Library can be categorized into two parts:

- **The Standard Function Library**: This library consists of general-purpose, stand-alone functions that are not part of any class. The function library is inherited from C.

- **The Object Oriented Class Library**: This is a collection of classes and associated functions.

Standard C++ Library incorporates all the Standard C libraries also, with small additions and changes to support type safety.

The Standard Function Library:
The standard function library is divided into the following categories:

- I/O
- String and character handling
- Mathematical
- Time, date, and localization
- Dynamic allocation
- Miscellaneous
- Wide-character functions

The Object Oriented Class Library:
Standard C++ Object Oriented Library defines an extensive set of classes that provide support for a number of common activities, including I/O, strings, and numeric processing. This library includes the following:

- The Standard C++ I/O Classes
- The String Class
- The Numeric Classes
- The STL Container Classes
- The STL Algorithms
- The STL Function Objects
- The STL Iterators
- The STL Allocators
- The Localization library
- Exception Handling Classes
- Miscellaneous Support Library
PROGRAM 2(a): Write a C++ program to find the sum of individual digits of a positive integer.

Program:
#include<iostream.h>
int sum_of_digits(int n)
{
    int digit, sum=0;
    while(n!=0)
    {
        digit=n%10;
        sum=sum+digit;n=n/10;
    }
    int main()
    return sum;
    int number, digits_sum;
    cout<<"Enter Positive integer within the range: ";
    cin>>number;
    digits_sum=sum_of_digits(number);
    cout<<"sum of digits of "<number<" is "<digits_sum;return 0;
}

Input:
Enter Positive integer within the range: 4321

Output:
sum of digits of 4321 is 10.
PROGRAM 2(b): Write a C++ program to find the sum of individual digits of a positive integer.

**Program:**
#include<iostream.h>
void fib(int n)
{
    int f0,f1,f,count=0;f0=0;
f1=1;
    while(count<n)
    {
        cout<<f0<<endl;
        count++;
f=f0+f1;
f0=f1;
f1=f;
    }
}

int main()
{
    int terms;
    cout<<"Enter How many terms to be printed:";cin>>terms;
fib(terms);
    return 0;
}

**Input:**
Enter How many terms to be printed:10

**Output:**
0 1 1 2 3 5 8 13 21 34
PROGRAM 3(a): Write a C++ program to generate all the prime numbers between 1 and n, where n is a value supplied by the user.

Program:
#include<iostream.h>
void prime(int n)
{
    int factors;
    cout<<"prime numbers are... ";
    for(int i=1;i<=n;i++)
    {
        factors=0;
        for(int j=1;j<=i;j++)
        {
            if(i%j==0)
                factors=factors+1;
        }
        if(factors<=2)
            cout<<i<<"t";
    }
}
int main()
{
    int n;
    cout<<"Enter a integer value:"; 
    cin>>n;
    prime(n);
    return 0;
}

Input:
Enter a integer value:10

Output:
prime numbers are... 1 2 3 5 7
**PROGRAM 3(b):** Write a C++ Program to find both the largest and smallest number in a list of integers.

**Program:**

```cpp
#include<iostream.h> int
main()
{
 int a[50],i,n,small,large; cout<<"Enter The Array Size:";
 cin>>n;
 cout<<"ENTER ELEMENTS OF ARRAY";
 for(i=0;i<n;i++)
 cin>>a[i];
 small=a[0];
 large=a[0];
 for(i=0;i<n;i++)
 {
  if(a[i]<small)
   small=a[i];
  if(a[i]>large)
   large=a[i];
 }
 cout<<"largest value is"<<large<<endl;
 cout<<"smallest value is:"<<small<<endl;
 return 0;
}
```

**Input:**
Enter The Array Size:5
ENTER ELEMENTS OF ARRAY 5 4 3 2 1

**Output:**
largest value is 5
smallest value is: 1
PROGRAM 4(a): Write a C++ program to sort a list of numbers in ascending order.

Program:
#include<iostream.h>
void sort(int data[], int n)
{
    for(int i=0; i<n; i++) // read the elements of an array
    for(int j=0; j<n-1; j++)
    {
        int t; if(data[j] > data[j+1])
        {
            t = data[j];
            data[j] = data[j+1];
            data[j+1] = t;
        }
    }
}
int main()
{
    int a[50], i, n;
    cout << "Enter How many elements to sort: "; cin >> n;
    cout << "Enter Elements: ";
    for(i=0; i<n; i++) // read the elements of an array
    cin >> a[i];
    cout << "Sorted array is \n";
    for(i=0; i<n; i++)
    cout << a[i] << " \t";
    return 0;
}

Input:
Enter How many elements to sort: 5
Enter Elements: 5 4 3 2 1

Output:
Sorted array is
5   4   3   2   1
PROGRAM 4(b): Write a Program to illustrate New and Delete Keywords for dynamic memory allocation.

Program:
#include<iostream.h>
int sum(int *a,int n)
{
    int s=0;
    for(int i=0;i<n;i++)
      s=s+*(a+i);
    return s;
}
int main()
{
  int *p,i,n;
  cout<"enter how many values to be read:"
  cin>>n;
p=new int [n];
cout<"Enter values :":
  for(int i=0;i<n;i++)
cin>>p[i];
int Array sum=sum(p,n);
cout<"sum of all values are ",Array_sum;
return 0;
}

Input:
Enter how many values to be read:4
Enter values :1
2
3
4

Output:
sum of all values are 10
**Program 5(a):** Write a program Illustrating Class Declarations, Definition, and Accessing Class Members.

**Program:**
```cpp
#include<iostream.h>
class sample
{
private:
    char b;
    float c;

    void get_data ()
    public:
    { int a;
        cout<<"Enter an integer value:"; cin>>a;
        cout<<"Enter a character:"; cin>>b;
        cout<<"Enter a float value:"; cin>>c;
    }
    void print_data ()
    {
        cout<<"Values read from keyboard are:\n"
        cout<<"Integer value:"<<a<<endl;
        cout<<"character is :"<<b<<endl;
        cout<<"float value is :"<<c<<endl;
        cin>>c;
    }
    int main()
    {
        sample s;//creation of object
        s.get_data();
        s.print_data();
    }
}
```

**Output:**
Enter an integer value:12
Enter a character:S
Enter a float value:12.12
Values read from keyboard are
Integer value:12
character is :S
float value is :12.12
PROGRAM 5(b): Program to illustrate default constructor, parameterized constructor and copy constructors.

Program:
#include<iostream.h>
class code
{
    int id;
    int      count;
public:
    code()
    {
        cout<<"Default constructor called\n";
        id=0;
        cout<<"id="<<id<<endl;
    }
    code(int a)
    {
        cout<<"Parameterized constructor called\n";
        id=a;
        cout<<"id="<<id<<endl;
    }
    code(code&x)
    {
        cout<<"copy constructor called\n";
        id=x.id;
        cout<<"id="<<id<<endl;
    }
    void display()
    {
        ~code()
        {
            cout<<"id="<<id<<endl;
            cout<<"Object Destroyed"<<endl;
        }
    }
};
int main()
{
code a(100); // calls parameterized constructor
code b(a); // calls copy constructor
code c(a); // calls copy constructor
code d; // calls default constructor
cout << "n For object d id="; d.display();
cout << "n For object a id="; a.display();
cout << "n For object b id="; d.display();
cout << "n For object c id="; d.display();
return 0;
}

Output:
Parameterized constructor called
id=100
copy constructor called
id=100
copy constructor called
id=100
Object Destroyed
Object Destroyed
Object Destroyed
Object Destroyed
**PROGRAM 6(a):** Write a Program to Demonstrate the:-i) Operator overloading   ii) Function Overloading.

Syntax:

```
return type classname:: operator op(list of arguments)
{
……………………………….
}
```

**i) Program:**

```cpp
#include<iostream.h>
class complex
{
    float real, img;
    public:
    complex(); complex(float x,float y);
    void read_complex();
    complex operator+(complex);
    complex operator-(complex);
    void display();
};
complex::complex()
{
    real=img=0;
}
complex::complex(float x,float y)
{
    real=x; img=y;
}
void complex::display()
{
    char sign;
    if(img<0)
    {
    }
    else
    {
    sign='>'; img=-img;
    sign='+';
    cout<real<<sign<"i"<<img<<endl;
    }
    complex complex::operator+(complex c)
    {
    complex r;
    r.real=real+c.real;
    r.img=img+c.img;
    return r;
    }
    complex complex::operator-(complex c)
    {
    complex r;
    r.real=real-c.real;
```
ii) **Function Overloading**

```cpp
#include<iostream>

using namespace std;

class printData
{
public:
    void print(int i)
    {
        cout << "Printing int: " << i << endl;
    }

    void print(double f)
    {
        cout << "Printing float: " << f << endl;
    }

    void print(char*c)
    {
        cout << "Printing string: " << c << endl;
    }
};

int main()
{
    printData pd;
    // Call print to print integer
```
```cpp
pd.print(5);
// Call print to print float
pd.print(500.263);
// Call print to print character
pd.print("Hello C++");
return 0;
}

**Output:**
Printing int: 5
Printing float: 500.263
Printing string: Hello C++
PROGRAM 6(b): Write a Program to Demonstrate Friend Function and Friend Class.

Program:
#include<iostream>
using namespace std;

class sample2;

class sample1
{
    int x; public:
    sample1(int a);
    friend void max(sample1 s1,sample2 s2);
};
sample1::sample1(int a)
{
    x=a;
}

class sample2
{
    int y; public:
    sample2(int b);
    friend void max(sample1 s1,sample2 s2);
};
sample2::sample2(int b)
{
    y=b;
}

void max(sample1 s1,sample2 s2)
{
    if(s1.x>s2.y)
        cout<"Data member in Object of class sample1 is larger "<endl;
    else
        cout<"Data member in Object of class sample2 is larger "<endl;
}
int main()
{
    int main()
    {
        sample1 obj1(3);
        sample2 obj2(5);
        max(obj1, obj2);
    }

    Output

    Data member in Object of class sample2 is larger
PROGRAM 7(a): Write a Program to Access Members of a STUDENT Class Using Pointer to Object Members.

Program: #include<iostream.h>
class student
{
    int roll no;
    char name[50];
public: void get data();
    void print();
};
void student::getdata()
{
    cout<<"Enter roll number"<<endl;
    cin>>rollno;
    cout<<"Enter Name ",
    cin>>name;
}

void student::print()
{
    cout<<"Name :"<<name<<endl;
    cout<<"Roll no:"<<rollno<<endl;
}
int main()
{
    student a;
    a.get data();
    a.print();
    cout<<"Pointer to class\n";
    student *ptr;
    ptr=&a;
    ptr->print();
}

Output:
Enter roll number123
Enter Name  jayapalName :jayapal
Roll no:123
Pointer to class Name:  jayapal Roll no:123
PROGRAM 7(b): Write a Program to Generate Fibonacci Series use Constructor to Initialize the Data Members.

Program: #include<iostream>
using name space std;
class fibonacci
{
    int f0,f1,f;
    public:
    Fibonacci ()
    {
        f0=0;f1=1;
    }
    void series(int n)
    {
        int count=0;
        f0=0;
        f1=1;
        while(count<n)
        {
            cout<<f0<<"\n";
            count++;
            f=f0+f1;
            f0=f1;
            f1=f;
        }
    }
};
Int main()
{
    Fibonacci obj;
    int terms;
    cout<<"Enter How many terms to be printed: ";
    cin>>terms;
    obj.series(terms);
    return 0;
}

Output: Enter How many terms to be printed: 5
0 1 1 2 3
**PROGRAM 8**: Write C++ programs that illustrate how the following forms of inheritance are supported:

a) Single inheritance  
b) Multiple inheritance  
c) Multi level inheritance  
d) Hierarchical inheritance

The mechanism of deriving a new class from an old one is called inheritance or derivation

**Syntax:**

```
class derived-class-name : visibility-mode base-class-name
{
    ..........  
    ..........  
}
```

**Program:**

a) Write a C++ Program that illustrate single inheritance.

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

class A
{
    protected:
        int a,b;
    public:
        void get()
        {
            cout<<"Enter any two integer values";
            cin>>a>>b;
        }
};
class B:public A
{
    int c; public:
    void add()
    {
        c=a+b;
        cout<<a<<"+"<<b<<"="<<c;
    }
};
int main()
{  
    B b;
    b.get();
    b.add();
}
```

**Output:**

```
Enter any two integer values 1, 2
1+2=3
```
**Program:** b) Write a C++ Program that illustrate multiple inheritance.

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

class student
{
protected:
    int rno,m1,m2;
public:
    void get()
    {
        cout<<"Enter the Roll no :";
        cin>>rno;
        cout<<"Enter the two marks :";
        cin>>m1>>m2;
    }
};
class sports
{
protected:
    int sm; // sm = Sports mark
public:
    void getsm()
    {
        cout<<"Enter the sports mark :";
        cin>>sm;
    }
};

Class statement: public student,public sports
{
    Int tot, avg;
public:
    void display()
    {
        tot=(m1+m2+sm);
        avg=tot/3;
        cout<<"Roll No :"<<rno<<"Total :"<<tot;
        cout<<"Average :"<<avg;
    }
};
```
void main()
{
clrscr(); statement
obj.get();
obj.getsm();
obj.display();
getch();
}

Output:
Enter the Roll no: 100
Enter two marks 90
80
Enter the Sports Mark: 90
Roll No: 100
Total : 260
Average: 8
Program: c) Write a program for Multi level inheritance

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

class top  //base class
{
public:
int a;
void getdata()
{
    cout<<"Enter first Number ::::t";
    cin>>a;
}
void putdata()
{
    cout<<"First Number Is ::::t"<<a;
}
};

//First level inheritance
class middle:public top  // class middle is derived_1
{
public:
int b;
void square()
{
    getdata();
b=a*a;
    cout<<"Square Is ::::t"<<b;
}
};

//Second level inheritance
class bottom:public middle  // class bottom is derived_2
{
public:
int c;
void cube()
{
    square();
c=b*a;
    cout<<"Cube ::::t"<<c;
}
};

int main()
{
    clrscr(); bottom b1;
b1.cube();
    getch();
}
```

Input:
Enter first number :::: 4

Output: Square Is :::: 16 Cube :::: 64
**Program: c)** Write a program for Hierarchical inheritance.

```cpp
#include<iostream.h>
#include<conio.h>
class A //Base Class
{
    public:
        int a, b;
    void get_number()
    {
        cout << "Enter Number ::: ";
        cin >> a;
    }
};
class B : public A //Derived Class 1
{
    public:
        void square()
        {
            get_number(); //Call Base class property
            cout << "Square of the number ::: " << (a * a);
        }
};
class C : public A //Derived Class 2
{
    public:
        void cube()
        {
            get_number(); //Call Base class property
            cout << "Cube of the number ::: " << (a * a * a);
        }
    int main()
    {
        clrscr();
        B b1;  //b1 is object of Derived class 1 b1.square();
        C c1;  //call member function of class BC c1;
        c1.cube();  //call member function of class C
        getch();
    }
}

**Input:**
Enter number ::: 2

**Output:**
Square of the number ::: 4
PROGRAM 9: Write a C++ program to illustrate the order of execution of constructors and destructors.

Program: #include<iostream.h>
class Base
{
public:

    Base ()
    {
        cout<< "Inside Base constructor" <<endl;
    }

    ~Base ()
    {
        cout<< "Inside Base destructor" <<endl;
    }
};
class Derived : public Base
{
public:
    Derived ()
    {
        cout<< "Inside Derived constructor" <<endl;
    }

    ~Derived ()
    {
        cout<< "Inside Derived destructor" <<endl;
    }
};

void main()
{
    Derived x;
}

Output:
Inside Base constructor
Inside Derived constructor
Inside Derived destructor
Inside Base destructor
Program: 10: Write a C++ program containing a possible exception. Use a try block to throw it and a catch block to handle it properly.

Program:
```cpp
#include<iostream>
using namespace std;

int main()
{
    int x = -1;
    cout<< "Before try \n";
    try
    {
        cout<< "Inside try \n"; if (x < 0)
        {
            throw x;
            cout<< "After throw (Never executed) \n";
        }
    }
    catch (int x)
    {
        cout<< "Exception Caught \n";
    }
    cout<< "After catch (Will be executed) \n";
    return 0;
}
```

Output:
Before try
Inside try
Exception Caught
After catch (Will be executed)