

Delhi Technological University

Department Of Computer Science And Engineering



Object Oriented Programming Lab

Lab File for the course CO203 (P)

Submitted by : Vishal Das (2K21/CO/523)

Submitted to : Dr. R.K. Yadav

Contents

S. No.	Program	Date	Signature
1	Write a program to print first 50 terms of Fibonacci Series.	August 23, 2022	
2	Write a program to calculate factorial of a number.		
3	Write a program to print all prime number less than a given number.		
4	Write a program to implement use of Class, it's data members and method.	August 30, 2022	
5	Write a program to implement different kind of Constructors and Destructor.	September 06, 2022	
6	Write a program to implement friend function.	September 13, 2022	
7	Write a program to implement single-level inheritance.	October 13, 2022	
8	Write a program to implement multilevel inheritance.		
9	Write a program to implement hierarchical inheritance.		
10	Write a program to implement functional overloading.	October 20, 2022	
11	Write a program to implement operator overloading.		
12	Write a program to implement functional overriding.		
13	Write a program to implement exception handling.	November 20, 2022	
14	Write a program to implement class templates.		
15	Write a program to implement virtual functions.		

Program 01 : Write a program to print first 50 terms of Fibonacci Series.

Theory :

The Fibonacci sequence is a series where the next term is the sum of previous two terms. The first two terms of the Fibonacci sequence is 0 followed by 1.

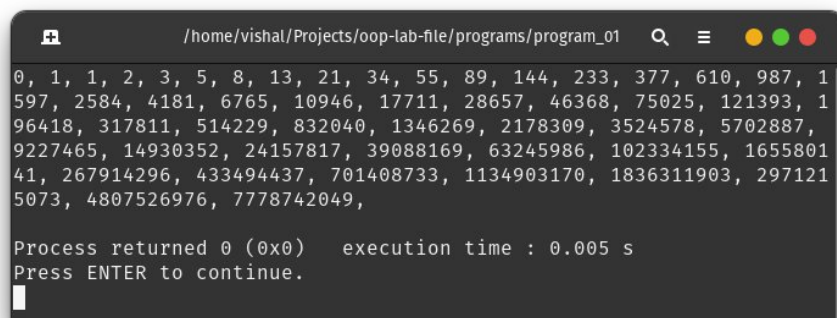
Code :

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

void fibonacci(int k)
{
    unsigned long int n2 = 1, n1 = 0, temp = 1;
    for (int n = 1; n <= k; n++)
    {
        cout << n1 << ", ";
        temp = n2 + n1;
        n1 = n2;
        n2 = temp;
    }
    cout << endl;
}

int main(void)
{
    fibonacci(50);
    return 0;
}
```

Output :



```
/home/vishal/Projects/oop-lab-file/programs/program_01
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1
597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 1
96418, 317811, 514229, 832040, 1346269, 2178309, 3524578, 5702887,
9227465, 14930352, 24157817, 39088169, 63245986, 102334155, 1655801
41, 267914296, 433494437, 701408733, 1134903170, 1836311903, 297121
5073, 4807526976, 7778742049,
Process returned 0 (0x0)  execution time : 0.005 s
Press ENTER to continue.
```

Summary : We learn how to print Fibonacci series using the concept of recursion.

Program 02 : Write a program to calculate factorial of a number.

Theory :

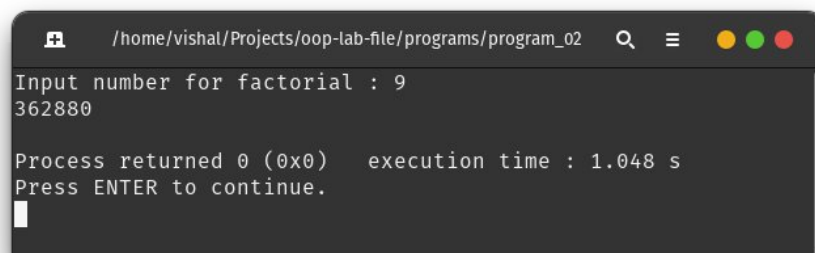
The factorial of a number is the product of all the integers from 1 up to that number. The factorial can only be defined for non-negative integers.

Code :

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

int main(void)
{
    long unsigned int factorial = 1;
    int n;
    cout << "Input number for factorial : ";
    cin >> n;
    for (int k = 1; k <= n; k++)
        factorial *= k;
    cout << factorial << endl;
    return 0;
}
```

Output :

A screenshot of a terminal window with a dark background. The title bar at the top shows the file path "/home/vishal/Projects/oop-lab-file/programs/program_02" and standard window control buttons. The terminal output shows the program's execution: it prompts for an input number, receives '9', and outputs '362880'. Below this, it shows 'Process returned 0 (0x0)' and 'execution time : 1.048 s'. The prompt 'Press ENTER to continue.' is visible at the bottom with a cursor.

Summary : We learn how to calculate and print factorial of a non-negative positive number using the concept of loop.

Program 03 : Write a program to print all prime number less than a given number.

Theory :

A number which has exactly two factors i.e. : 1 and the number itself is called Prime Number. To identify a prime number we have to check if it is divisible by a positive number less than the square root of the number other than 1.

Code :

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

bool isPrime(int n){
    if (n == 1) return false;
    else if (n == 2) return true;
    else if (!(n % 2)) return false;
    else
        for (int k = 3; k < sqrt(n); k = k + 2)
            if (n % k == 0) return false;
    return true;
}

int main(void){
    int n;
    cout << "Input a number to check upto : ";
    cin >> n;
    cout << "Prime numbers less than " << n << " : ";
    while (n)
    {
        if (isPrime(n))
            cout << n << ", ";
        n--;
    }
    cout << endl;
    return 0;
}
```

Output :

```
/home/vishal/Projects/oop-lab-file/programs/program_03
Input a number to check upto : 29
Prime numbers less than 29 : 29, 25, 23, 19, 17, 13, 11, 9, 7, 5, 3, 2,
Process returned 0 (0x0)   execution time : 1.000 s
Press ENTER to continue.
```

Summary : We learn the concept of function and functional call using while loop.

Program 04 : Write a program to implement use of Class, it's data members and method.

Theory :

A class in C++ is the building block that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object.

Code :

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

class Student
{
public:
    string name;
    int roll_number;
    int marks[3];

    float getAverageMarks()
    {
        float average = 0.0;
        for (int i = 0; i < 3; i++)
            average += *(marks + i);
        average /= 3;
        return average;
    }
};

int main(void)
{
    int n;
    cout << "Input number of students: ";
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        cout << "Student " << i + 1 << ": " << endl;
        Student s;

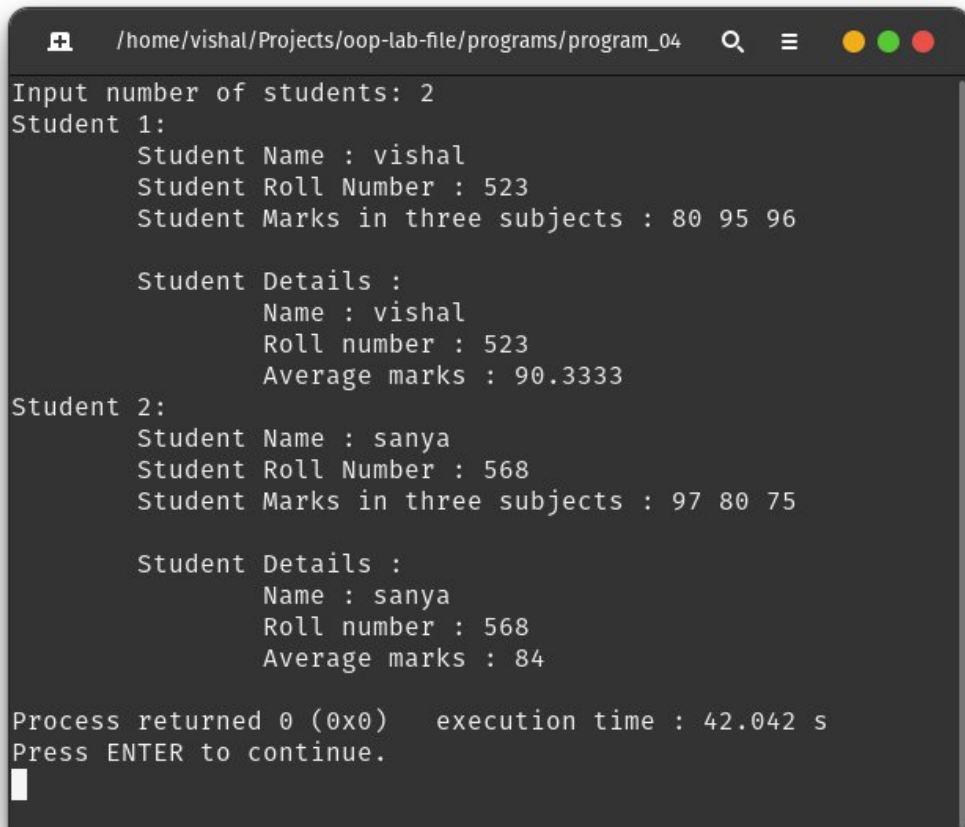
        cout << "\tStudent Name : ";
        cin >> s.name;
        cout << "\tStudent Roll Number : ";
        cin >> s.roll_number;
        cout << "\tStudent Marks in three subjects : ";
        for (int i = 0; i < 3; i++)
            cin >> *(s.marks + i);
    }
}
```

```

    cout << endl
        << "\tStudent Details : " << endl;
    cout << "\t\tName : " << s.name << endl;
    cout << "\t\tRoll number : " << s.roll_number << endl;
    cout << "\t\tAverage marks : " << s.getAverageMarks() << endl;
}
return 0;
}

```

Output :



```

/home/vishal/Projects/oop-lab-file/programs/program_04
Input number of students: 2
Student 1:
    Student Name : vishal
    Student Roll Number : 523
    Student Marks in three subjects : 80 95 96

    Student Details :
        Name : vishal
        Roll number : 523
        Average marks : 90.3333
Student 2:
    Student Name : sanya
    Student Roll Number : 568
    Student Marks in three subjects : 97 80 75

    Student Details :
        Name : sanya
        Roll number : 568
        Average marks : 84

Process returned 0 (0x0)   execution time : 42.042 s
Press ENTER to continue.

```

Summary : We learn the concept of classes and public variables and methods in OOP.

Program 05 : Write a program to implement different Constructors and Destructor.

Theory :

A constructor is a member function of a class that has the same name as the class name. It helps to initialize the object of a class. Destructor is also a member function of a class that has the same name as the class name preceded by a tilde(~) operator. It helps to deallocate the memory of an object. It is called while the object of the class is freed or deleted.

Code :

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

class Student {
private:
    static int count;

public:
    string name;
    int age;
    int roll;
    // default constructor
    Student() {
        name = "";
        age = 0;
        roll = 0;
        count++;
    }
    // parameterized constructor
    Student(string name, int age, int roll) {
        this->name = name;
        this->age = age;
        this->roll = roll;
        count++;
    }
    // copy constructor
    Student(const Student &s) {
        name = s.name;
        age = s.age;
        roll = s.roll;
        count++;
    }
    // destructor
    ~Student() {
        cout << "Destructor called for " << name;
        cout << "(Object Count : " << --count << ")" << endl;
    }
}
```



```

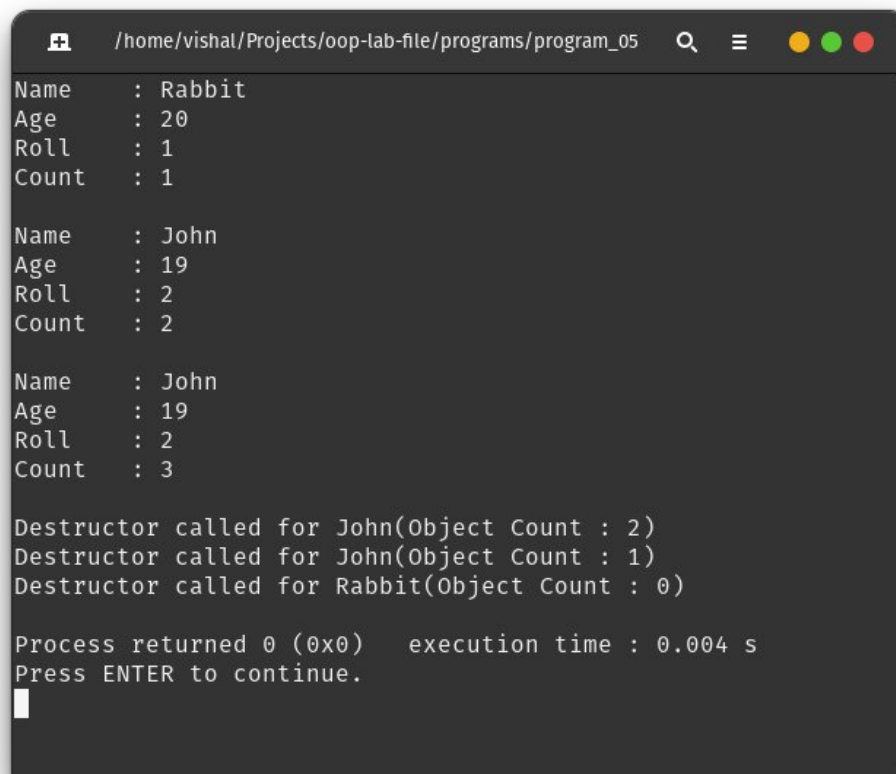
void display() {
    cout << "Name\t: " << name << endl;
    cout << "Age\t: " << age << endl;
    cout << "Roll\t: " << roll << endl;
    cout << "Count\t: " << count << endl << endl;
}
};

int Student::count = 0;

int main(void) {
    Student s; // default constructor
    s.name = "Rabbit";
    s.age = 20;
    s.roll = 1;
    s.display();
    Student s1("John", 19, 2); // parametric constructor called
    s1.display();
    Student s2(s1); // copy constructor called
    s2.display();
    return 0;
}

```

Output :



```

/home/vishal/Projects/oop-lab-file/programs/program_05
Name : Rabbit
Age : 20
Roll : 1
Count : 1

Name : John
Age : 19
Roll : 2
Count : 2

Name : John
Age : 19
Roll : 2
Count : 3

Destructor called for John(Object Count : 2)
Destructor called for John(Object Count : 1)
Destructor called for Rabbit(Object Count : 0)

Process returned 0 (0x0) execution time : 0.004 s
Press ENTER to continue.

```

Summary : We learn the concept of constructor and destructor and their usage cases.

Program 06 : Write a program to implement friend function.

Theory :

A friend function can be given a special grant to access private and protected members. A friend function can be :

- a) A member of another class
- b) A global function

Code :

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

class Student {
private:
    static int objCreated;
    int uniqueId;

public:
    string name;
    int age;
    int roll;
    Student() {
        name = "";
        age = 0;
        roll = 0;
        uniqueId = ++objCreated;
    }
    Student(string name, int age, int roll) {
        this->name = name;
        this->age = age;
        this->roll = roll;
        uniqueId = ++objCreated;
    }
    Student(Student &s) {
        name = s.name;
        age = s.age;
        roll = s.roll;
        uniqueId = ++objCreated;
    }
    void display() {
        cout << "Name\t: " << name << endl;
        cout << "Age\t: " << age << endl;
        cout << "Roll\t: " << roll << endl;
        cout << "ID\t: " << uniqueId << endl << endl;
    }
    friend void updateUniqueID(Student &, int);
};
```

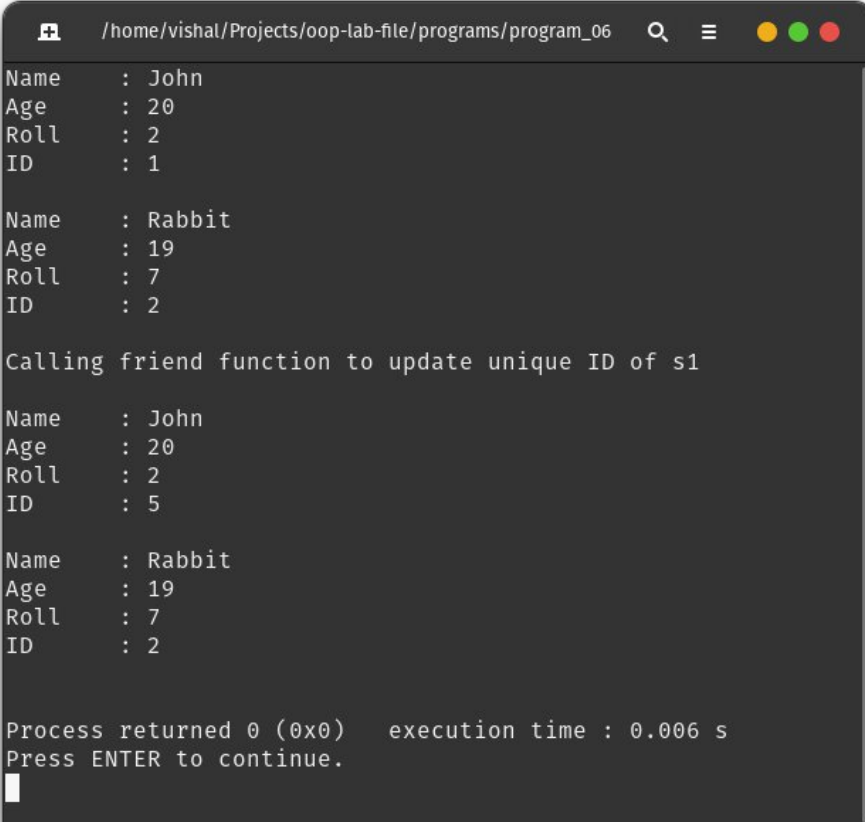
```
};

int Student::objCreated = 0;

void updateUniqueID(Student &s, int id) { s.uniqueId = id; }

int main(void) {
    Student s1("John", 20, 2);
    Student s2("Rabbit", 19, 7);
    s1.display();
    s2.display();
    cout << "Calling friend function to update unique ID of s1\n" << endl;
    updateUniqueID(s1, 5);
    s1.display();
    s2.display();
    return 0;
}
```

Output :



```
/home/vishal/Projects/oop-lab-file/programs/program_06
Name      : John
Age       : 20
Roll      : 2
ID        : 1

Name      : Rabbit
Age       : 19
Roll      : 7
ID        : 2

Calling friend function to update unique ID of s1

Name      : John
Age       : 20
Roll      : 2
ID        : 5

Name      : Rabbit
Age       : 19
Roll      : 7
ID        : 2

Process returned 0 (0x0)   execution time : 0.006 s
Press ENTER to continue.
```

Summary : We learn the concept of friend function which can access the private and protected members of a class despite of not being its member function.

Program 07 : Write a program to implement single-level inheritance.

Theory :

The capability of a class to derive properties and characteristics from another class is called Inheritance. The inheritance in which a single derived class is inherited from a single base class is known as the Single Inheritance.

Code :

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

class Person {
protected:
    string name;
    int age;

public:
    Person(string name, int age) : name(name), age(age) {}
    void display() {
        cout << "Person Name\t: " << name << endl;
        cout << "Person Age\t: " << age << endl;
    }
};

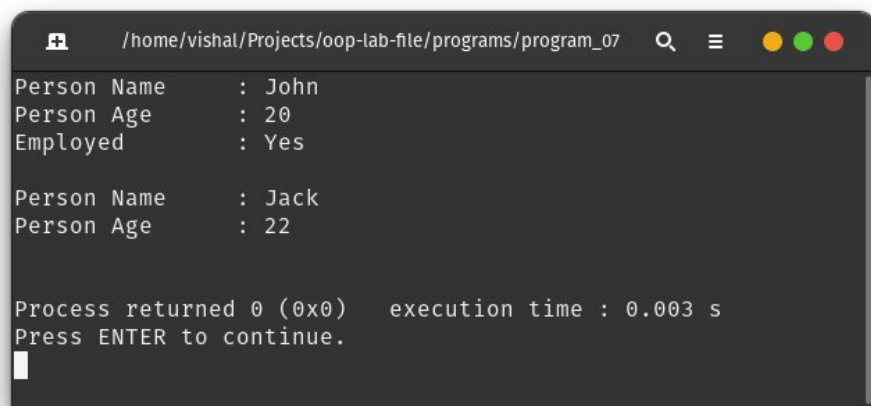
class Literate : public Person { // single inheritance
protected:
    bool employmentStatus;

public:
    Literate(string name, int age, bool is_employed)
        : Person(name, age), employmentStatus(is_employed) {}
    void display() {
        Person::display();
        cout << "Employed\t: " << (employmentStatus ? "Yes" : "No") << endl;
    }
};

int main(void) {
    Literate s1("John", 20, true);
    Person s2("Jack", 22);

    s1.display();
    cout << endl;
    s2.display();
    cout << endl;

    return 0;
}
```

Output :A terminal window with a dark background and light gray text. The title bar at the top shows a file icon, the path "/home/vishal/Projects/ooop-lab-file/programs/program_07", a search icon, a menu icon, and three window control buttons (yellow, green, red). The output text is as follows:

```
Person Name      : John
Person Age       : 20
Employed         : Yes

Person Name      : Jack
Person Age       : 22

Process returned 0 (0x0)   execution time : 0.003 s
Press ENTER to continue.
█
```

Summary : We learn the concept of inheritance and single-level inheritance with a real life example.

Program 08 : Write a program to implement multilevel inheritance.

Theory :

The capability of a class to derive properties and characteristics from another class is called Inheritance. The multi-level inheritance includes the involvement of at least two or more than two classes.

Code :

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

class Person {
protected:
    string name;
    int age;

public:
    Person(string name, int age) : name(name), age(age) {}
    void display() {
        cout << "Person Name\t: " << name << endl;
        cout << "Person Age\t: " << age << endl;
    }
};

class Literate : public Person { // single inheritance
protected:
    bool employmentStatus;

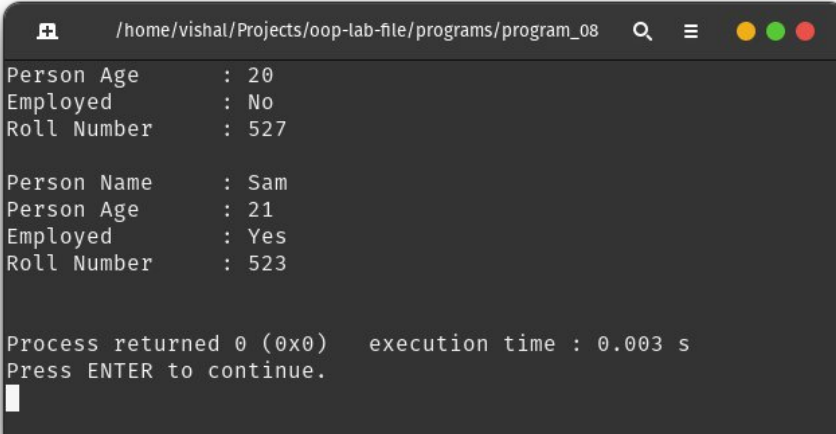
public:
    Literate(string name, int age, bool is_employed)
        : Person(name, age), employmentStatus(is_employed) {}
    void display() {
        Person::display();
        cout << "Employed\t: " << (employmentStatus ? "Yes" : "No") << endl;
    }
};

class Student : public Literate { // multilevel inheritance
protected:
    int roll;

public:
    Student(string name, int age, bool is_employed, int roll)
        : Literate(name, age, is_employed), roll(roll) {}
    void display() {
        Literate::display();
        cout << "Roll Number\t: " << roll << endl;
    }
}
```

```
};  
  
int main(void) {  
    Student s1("John", 20, false, 527);  
    Student s2("Sam", 21, true, 523);  
  
    s1.display();  
    cout << endl;  
    s2.display();  
    cout << endl;  
  
    return 0;  
}
```

Output :



```
/home/vishal/Projects/oop-lab-file/programs/program_08  
Person Age      : 20  
Employed        : No  
Roll Number     : 527  
  
Person Name     : Sam  
Person Age      : 21  
Employed        : Yes  
Roll Number     : 523  
  
Process returned 0 (0x0)   execution time : 0.003 s  
Press ENTER to continue.  
█
```

Summary : We learn the concept of inheritance and single-level inheritance with a real life example.

Program 09 : Write a program to implement hierarchical inheritance.

Theory :

The capability of a class to derive properties and characteristics from another class is called Inheritance. Hierarchical inheritance describes a situation in which a parent class is inherited by multiple subclasses.

Code :

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

class Person {
protected:
    string name;
    int age;
    bool employmentStatus;

public:
    Person(string name, int age, bool is_employed)
        : name(name), age(age), employmentStatus(is_employed) {}
    void display() {
        cout << "Person Name\t: " << name << endl;
        cout << "Person Age\t: " << age << endl;
        cout << "Employed\t: " << (employmentStatus ? "Yes" : "No") << endl;
    }
};

class Student : public Person {
    // multiple inheritance from base class Literate
protected:
    int roll;

public:
    Student(string name, int age, bool is_employed, int roll)
        : Person(name, age, is_employed), roll(roll) {}
    void display() {
        Person::display();
        cout << "Roll Number\t: " << roll << endl;
    }
};

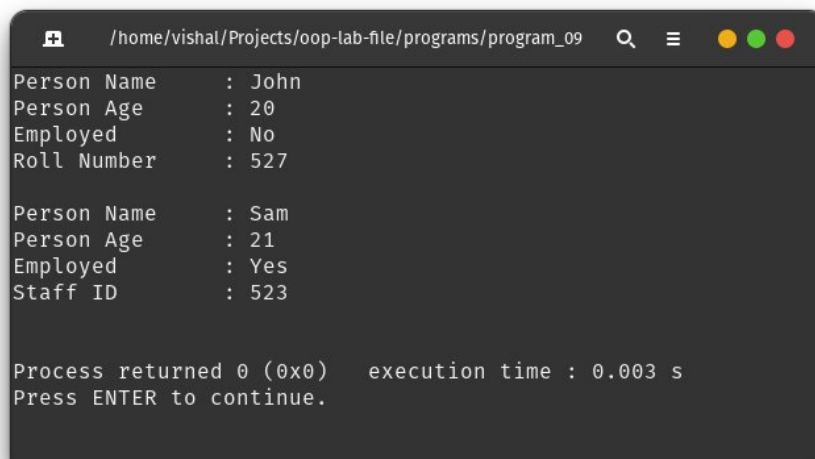
class Staff : public Person {
    // multiple inheritance from base class Literate
protected:
    int staffId;

public:
    Staff(string name, int age, bool is_employed, int staff_id)
        : Person(name, age, is_employed), staffId(staff_id) {}
};
```



```
void display() {  
    Person::display();  
    cout << "Staff ID\t: " << staffId << endl;  
}  
};  
  
int main(void) {  
    Student s1("John", 20, false, 527);  
    Staff s2("Sam", 21, true, 523);  
  
    s1.display();  
    cout << endl;  
    s2.display();  
    cout << endl;  
  
    return 0;  
}
```

Output :



```
/home/vishal/Projects/oop-lab-file/programs/program_09  
Person Name      : John  
Person Age       : 20  
Employed         : No  
Roll Number      : 527  
  
Person Name      : Sam  
Person Age       : 21  
Employed         : Yes  
Staff ID         : 523  
  
Process returned 0 (0x0)   execution time : 0.003 s  
Press ENTER to continue.
```

Summary : We learn the concept of inheritance and single-level inheritance with a real life example.

Program 10 : Write a program to implement functional overloading.**Theory :**

Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters. When a function name is overloaded with different jobs it is called Function Overloading.

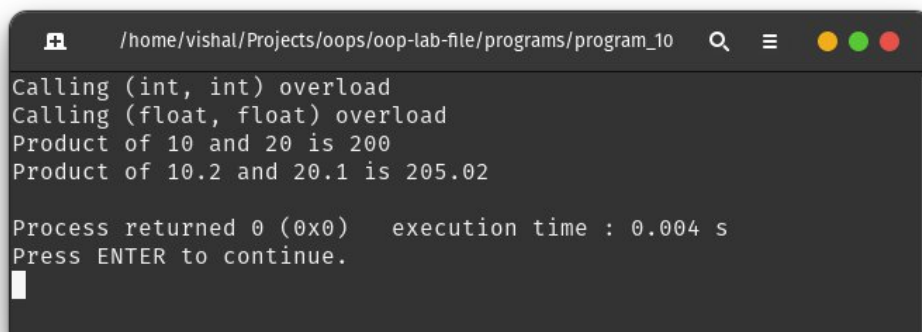
Code :

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

int multiply(int a, int b) {
    cout << "Calling (int, int) overload\n";
    return a * b;
}

float multiply(float a, float b) {
    cout << "Calling (float, float) overload\n";
    return a * b;
}

int main(void) {
    int a = 10, b = 20;
    float c = 10.2, d = 20.1;
    auto x = multiply(a, b);
    auto y = multiply(c, d);
    cout << "Product of " << a << " and " << b << " is " << x << endl;
    cout << "Product of " << c << " and " << d << " is " << y << endl;
    return 0;
}
```

Output :

```
/home/vishal/Projects/oops/oop-lab-file/programs/program_10
Calling (int, int) overload
Calling (float, float) overload
Product of 10 and 20 is 200
Product of 10.2 and 20.1 is 205.02

Process returned 0 (0x0)   execution time : 0.004 s
Press ENTER to continue.
```

Summary : We learn the concept of functional overloading by using different functional signatures.

Program 11 : Write a program to implement operator overloading.

Theory :

In C++, we can change the way operators work for user-defined types like objects and structures. This is known as operator overloading.

Code :

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

class str {
    string s;

public:
    str(string s) : s(s) {}
    str operator+(str &s2) { return str(s + s2.s); }
    str operator*(int n) {
        if (n <= 0)
            return str("");

        string temp = s;
        for (int i = 1; i < n; i++)
            s += temp;

        return str(s);
    }

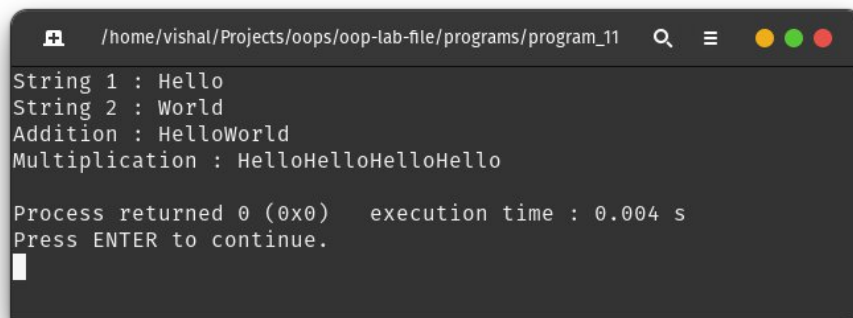
    void show() { cout << s << endl; }
};

int main(void) {
    str s1("Hello"), s2("World");
    str s3 = s1 + s2;

    cout << "String 1 : ";
    s1.show();
    cout << "String 2 : ";
    s2.show();
    cout << "Addition : ";
    s3.show();

    s3 = s1 * 4;
    cout << "Multiplication : ";
    s3.show();

    return 0;
}
```

Output :A terminal window with a dark background and light gray text. The window title bar shows the file path "/home/vishal/Projects/oops/oop-lab-file/programs/program_11" and standard window control buttons. The output text is as follows:

```
String 1 : Hello
String 2 : World
Addition : HelloWorld
Multiplication : HelloHelloHelloHello

Process returned 0 (0x0)   execution time : 0.004 s
Press ENTER to continue.

```

Summary : We learn the concept to overload an operator to perform desired action relevant to the class.

Program 12 : Write a program to implement functional overriding.

Theory :

Function overriding in C++ is a feature that allows us to use a function in the child class that is already present in its parent class.

Code :

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

class shape {
    int l, b, h;

public:
    shape(int l, int b, int h) : l(l), b(b), h(h) {}
    float volume() { return l * b * h; }
};

class cuboid : public shape {
public:
    cuboid(int l, int b, int h) : shape(l, b, h) {}
    float volume() { return shape::volume(); }
};

class cylinder : public shape {
    int r;

public:
    cylinder(int r, int h) : shape(r, r, h), r(r) {}
    float volume() { return 3.14 * r * r * shape::volume(); }
};

class cube : public shape {
public:
    cube(int l) : shape(l, l, l) {}
    float volume() { return shape::volume(); }
};

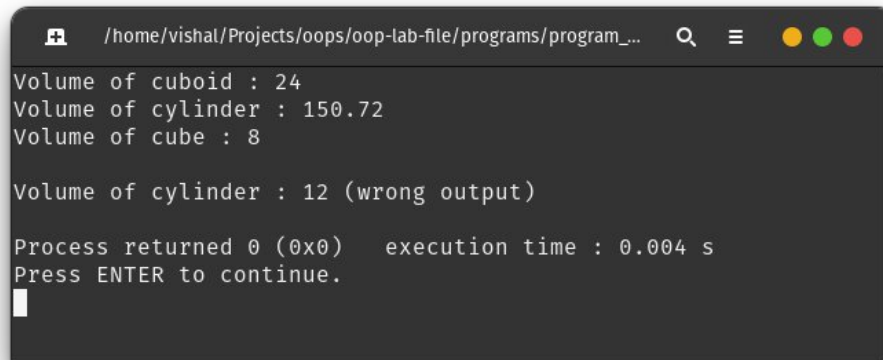
int main(void) {
    cuboid c1(2, 3, 4);
    cylinder c2(2, 3);
    cube c3(2);

    shape *s1 = &c2;

    cout << "Volume of cuboid : " << c1.volume() << endl;
    cout << "Volume of cylinder : " << c2.volume() << endl;
    cout << "Volume of cube : " << c3.volume() << endl;
}
```

```
    cout << "\nVolume of cylinder : " << s1->volume() << " (wrong  
output)\n";  
    return 0;  
}
```

Output :

A terminal window with a dark background and light text. The window title bar shows the path /home/vishal/Projects/oops/oop-lab-file/programs/program_... and standard window controls. The output text is as follows:

```
Volume of cuboid : 24  
Volume of cylinder : 150.72  
Volume of cube : 8  
  
Volume of cylinder : 12 (wrong output)  
  
Process returned 0 (0x0)   execution time : 0.004 s  
Press ENTER to continue.  
█
```

Summary : We learn the concept of functional overriding and their behavior of objects to call function which are assigned during the compile time.

Program 13 : Write a program to implement exception handling.

Theory :

Exceptions are runtime anomalies or abnormal conditions that a program encounters during its execution.

C++ provides the following specialized keywords for this purpose :

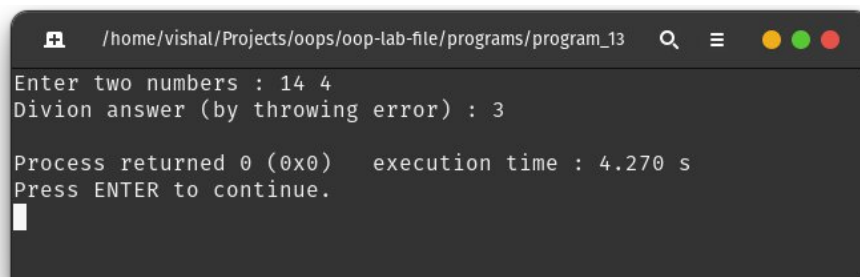
- *try* : a block of code that can throw an exception.
- *catch* : a block of code that is executed when a particular exception is thrown.
- *throw* : used to throw an exception

Code :

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

int main(void) {
    int a, b;
    cout << "Enter two numbers : ";
    try {
        cin >> a >> b;
        if (cin.fail())
            throw "Invalid input";
        if (b == 0)
            throw "Division by zero";
        throw a / b;
    } catch (const char *e) {
        cout << e << endl;
    } catch (int x) {
        cout << "Divion answer (by throwing error) : " << x << endl;
    } catch (...) {
        cout << "Unknown exception" << endl;
    }
    return 0;
}
```

Output :



```
/home/vishal/Projects/oops/oop-lab-file/programs/program_13
Enter two numbers : 14 4
Divion answer (by throwing error) : 3

Process returned 0 (0x0)   execution time : 4.270 s
Press ENTER to continue.
█
```

Summary : We learn the concept of error handling for various cases using try-catch block.

Program 14 : Write a program to implement class templates.

Theory :

A template is a simple yet very powerful tool in C++. The simple idea is to pass data type as a parameter so that we don't need to write the same code for different data types. Templates are expanded at compile time.

Code :

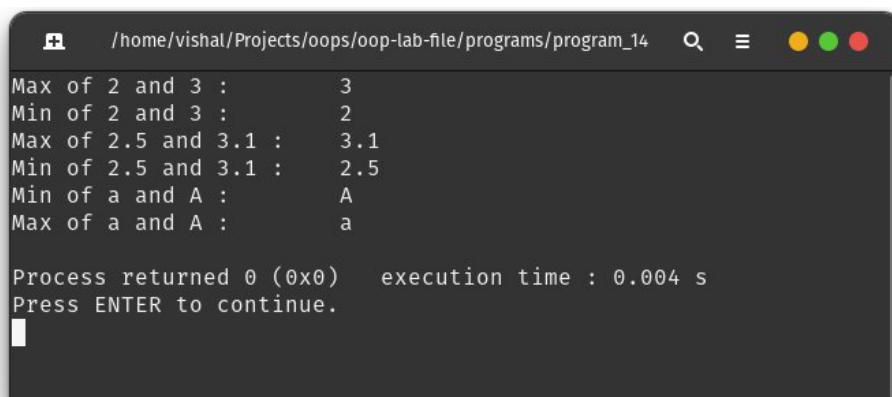
```
#include <iostream>
using namespace std;

template <class T> class compare {
    T a, b;

public:
    compare(T a, T b) : a(a), b(b) {}
    T max() { return a > b ? a : b; }
    T min() { return a < b ? a : b; }
};

int main(void) {
    compare<int> c1(2, 3);
    compare<float> c2(2.5, 3.1);
    compare<char> c3('a', 'A');
    cout << "Max of 2 and 3 :\t" << c1.max() << endl;
    cout << "Min of 2 and 3 :\t" << c1.min() << endl;
    cout << "Max of 2.5 and 3.1 :\t" << c2.max() << endl;
    cout << "Min of 2.5 and 3.1 :\t" << c2.min() << endl;
    cout << "Min of a and A :\t" << c3.min() << endl;
    cout << "Max of a and A :\t" << c3.max() << endl;
    return 0;
}
```

Output :



```
/home/vishal/Projects/oops/oop-lab-file/programs/program_14
Max of 2 and 3 :      3
Min of 2 and 3 :      2
Max of 2.5 and 3.1 :  3.1
Min of 2.5 and 3.1 :  2.5
Min of a and A :      A
Max of a and A :      a

Process returned 0 (0x0)   execution time : 0.004 s
Press ENTER to continue.
```

Summary : We learn the concept of class template which generates different classes required in the program during the compile time.

Program 15 : Write a program to implement virtual functions.

Theory :

A virtual function is a member function which is declared within a base class and is re-defined (overridden) by a derived class.

When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.

Code :

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

class shape {
    int l, b, h;

public:
    shape(int l, int b, int h) : l(l), b(b), h(h) {}
    virtual float volume() { return l * b * h; }
};

class cuboid : public shape {
public:
    cuboid(int l, int b, int h) : shape(l, b, h) {}
    float volume() { return shape::volume(); }
};

class cylinder : public shape {
    int r;

public:
    cylinder(int r, int h) : shape(r, r, h), r(r) {}
    float volume() { return 3.14 * r * r * shape::volume(); }
};

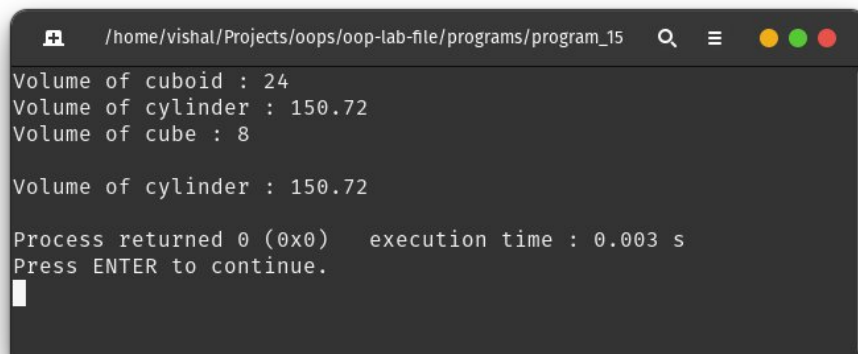
class cube : public shape {
public:
    cube(int l) : shape(l, l, l) {}
    float volume() { return shape::volume(); }
};

int main(void) {
    cuboid c1(2, 3, 4);
    cylinder c2(2, 3);
    cube c3(2);
```

```
shape *s1 = &c2;

cout << "Volume of cuboid : " << c1.volume() << endl;
cout << "Volume of cylinder : " << c2.volume() << endl;
cout << "Volume of cube : " << c3.volume() << endl;
cout << "\nVolume of cylinder : " << s1->volume() << endl;
return 0;
}
```

Output :

A terminal window with a dark background and light text. The title bar shows the file path "/home/vishal/Projects/oops/oop-lab-file/programs/program_15". The output of the program is displayed as follows:

```
Volume of cuboid : 24
Volume of cylinder : 150.72
Volume of cube : 8

Volume of cylinder : 150.72

Process returned 0 (0x0)   execution time : 0.003 s
Press ENTER to continue.
```

Summary : We learn the concept of virtual function and its run-time behavior making it different from regular functional overriding.