

Department of Computer Science and Engineering (CSE)

Laboratory Report Cover Page Semester: Fall-2023

**Students must complete all details except the faculty use part*.***

**Lab Number:**  **01-20**

**Course Code:**  **0613-CSE-1210**

**Course Name: Object Oriented Programming I Lab**

**Submission Date: 29.12.2023**

**Course Teacher/Instructor**

**Ankan Roy, Lecturer, CSE, RPSU**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Batch No.** | **Student Name** | **Student ID** | **Student**  **Signature** | **Date** |
| **Submitted by:** | | | | |
| 26 | Md. Siam Hossain | 23100084 | | | |

|  |  |
| --- | --- |
| **For faculty use only:** | **Total Marks:** **Marks Obtained:** |
| **Faculty comments** | |

|  |  |  |
| --- | --- | --- |
| Lab No | Topics Name | Page No |
| 01 | Data type, Variables, Input, Output | 03 - 04 |
| 02 | Loops | 05 - 06 |
| 03 | Function | 07 – 09 |
| 04 | Class and object | 10 – 11 |
| 05 | Members and Methods | 12 – 13 |
| 06 | Encapsulation | 14 – 15 |
| 07 | Static properties | 16 – 17 |
| 08 | Constructor | 18 – 19 |
| 09 | Friend function | 20 |
| 10 | Friend Class | 21 – 22 |
| 11 | Inheritance (single, multilevel) | 23 – 26 |
| 12 | Access specifier | 27 – 28 |
| 13 | Inheritance (Hierarchical) | 29 – 30 |
| 14 | Inheritance (Multiple) | 31 – 32 |
| 15 | Inheritance (Hybrid) | 33 – 34 |
| 16 | Compile Time Polymorphism (Overloading) | 35 – 36 |
| 17 | Run Time Polymorphism (Overriding) | 37 – 38 |
| 18 | Virtual function | 39 – 40 |
| 19 | Abstraction | 41 – 42 |
| 20 | Exception handling | 43 - 44 |

**INDEX**

**Lab No: 01**

**Topics Name:** Data type, Variables , Input, Output.

**Source Code:**

#include <iostream>

#include<iomanip>

using namespace std;

int main()

{

    // Here string,char,int,float,bool are the Data Types and

    // name,symbol,int\_num,float\_num,flag are the Variables

    string name;

    char symbol;

    int int\_num;

    float float\_num;

    bool flag;

    // Input...

    cout<< "Enter your Name: ";

    getline(cin,name);

    cout<< "Enter symbol: ";

    cin>> symbol ;

    cout<< "Enter any Integer type Number: ";

    cin>> int\_num;

    cout<< "Enter any Floating type Number: ";

    cin>> float\_num;

    flag = true;

    // Output...

    cout<< "Your Name is: " << name << endl;

    cout<< "Symbol is: " << symbol << endl;

    cout<< "Integer type Number that you entered is: " << int\_num << endl;

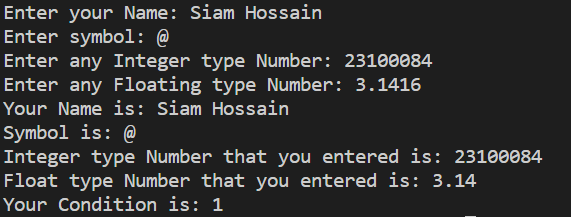
    cout<< "Float type Number that you entered is: " << fixed << setprecision(2) << float\_num << endl;

    cout<< "Your Condition is: " << flag << endl;

    return 0;

}

**Input and Output:**



**Lab No: 02**

**Topics Name:** Loops

**Source Code:**

#include <iostream>

using namespace std;

int main()

{

    int n, i;

    cout << "Enter the Range (n) to print from (0-n): ";

    cin >> n; // Read the user input and store it in the variable n

    cout << "Print Number using for loops: ";

    for (i = 0; i <= n; i++)

    {

        cout << i << " "; // Output the current value of i followed by a space

    }

    cout << endl

         << "Print Number using while loops: "; // Move to the next line

    i = 0;                                      // Reset the value of i to 0

    while (i <= n)

    {

        cout << i << " "; // Output the current value of i followed by a space

        i++;              // Increment the value of i

    }

    cout << endl

         << "Print Number using do-while loops: "; // Move to the next line

    i = 0;                                         // Reset the value of i to 0

    do

    {

        cout << i << " "; // Output the current value of i followed by a space

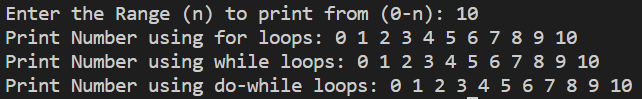
        i++;              // Increment the value of i

    } while (i <= n);     // Continue looping while the condition is true

    return 0;

}

**Input and Output:**

****

**Lab No: 03**

**Topics Name:** Function

**Source Code:**

#include <stdio.h>

// Function to calculate the sum of three numbers

int sumation(int num1, int num2, int num3)

{

    int sum = num1 + num2 + num3;

    return sum;

}

// Function to calculate the multiplication of two numbers

int multiplication(int num1, int num2)

{

    int m;

    m = num1 \* num2;

    return m;

}

// Function to calculate the subtraction of two numbers

int substraction(int m, int sum)

{

    int sub = m - sum;

    return sub;

}

// Function to find the maximum of five numbers

int maximum(int num1, int sum, int num3, int m, int sub)

{

    int max = num1;

    if (sum > max)

        max = sum;

    if (num3 > max)

        max = num3;

    if (m > max)

        max = m;

    if (sub > max)

        max = sub;

    return max;

}

// Function to find the minimum of five numbers

int minimum(int num1, int sum, int num3, int m, int sub)

{

    int min = num1;

    if (sum < min)

        min = sum;

    if (num3 < min)

        min = num3;

    if (m < min)

        min = m;

    if (sub < min)

        min = sub;

    return min;

}

int main()

{

    int num1, num2, num3, sum, sub, m, max, min;

    printf("\nEnter numbers to use later: ");

    scanf("%d %d %d", &num1, &num2, &num3);

    // Calculate sum of three numbers

    sum = sumation(num1, num2, num3);

    // Calculate multiplication of sum and num3

    m = multiplication(sum, num3);

    // Calculate subtraction of m and sum

    sub = substraction(m, sum);

    // Find the maximum and minimum among the given numbers

    max = maximum(num1, sum, num3, m, sub);

    min = minimum(num1, sum, num3, m, sub);

    // Display results

    printf("\nSum Results of %d, %d and %d: %d", num1, num2, num3, sum);

    printf("\nMult Results of %d and %d: %d", sum, num3, m);

    printf("\nSub Results of %d and %d: %d", m, sum, sub);

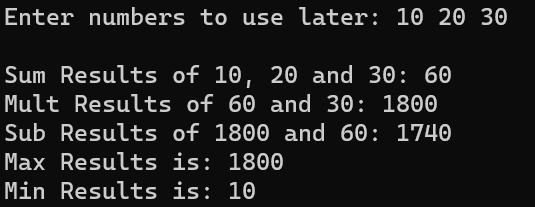
    printf("\nMax Results is: %d", max);

    printf("\nMin Results is: %d", min);

    return 0;

}

**Input and output:**

****

**Lab No: 04**

**Topics Name:** Class and object

**Source Code:**

#include <iostream>

using namespace std;

// Define a class named A

class A

{

public:

    int a, b, c, d;

    // Method to display the values of member variables

    void display()

    {

        cout << a << endl;

        cout << b << endl;

        cout << c << endl;

        cout << d << endl;

    }

};

int main()

{

    // Create an object of class A named obj1

    A obj1;

    // Initialize values for the member variables of obj1

    obj1.a = 5;

    obj1.b = 6;

    obj1.c = 7;

    obj1.d = obj1.a + obj1.b + obj1.c;

    // Call the display method of obj1 to print the values

    obj1.display();

    // Create another object of class A named obj2

    A obj2;

    // Initialize values for the member variables of obj2

    obj2.a = 50;

    obj2.b = 70;

    obj2.c = 100;

    obj2.d = obj2.a - obj2.b - obj2.c;

    // Call the display method of obj2 to print the values

    obj2.display();

    return 0;

}

**Input and Output:**

****

**Lab No: 05**

**Topics Name:** Members and Methods

**Source Code:**

#include <iostream>

using namespace std;

// Class definition for 'person'

class person

{

private:

    // Private member variables

    int id, age;

    string name, gender;

public:

    // Public member method inside class to set information

    void setInfo()

    {

        cout << "Enter your Full Name: ";

        getline(cin, name);

        cout << "Enter your ID: ";

        cin >> id;

        cout << "Enter your Age: ";

        cin >> age;

        cout << "Enter your Gender: ";

        cin >> gender;

        cout << endl;

    }

    // Public member method to get and display information

    void getInfo();

};

// Definition of the 'getInfo' member method outside the class

void person::getInfo()

{

    cout << "Full Name: " << name << endl;

    cout << "ID: " << id << endl;

    cout << "Age: " << age << endl;

    cout << "Gender: " << gender << endl;

}

int main()

{

    // Create an instance of the 'person' class

    person p1;

    // Call the 'setInfo' and 'getInfo' functions for the created object

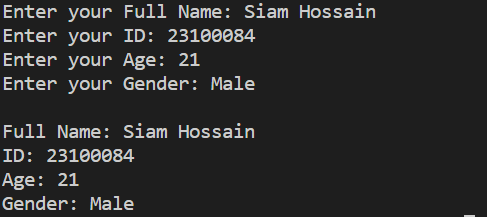
    p1.setInfo();

    p1.getInfo();

    return 0;

}

**Input and Output:**

****

**Lab No: 06**

**Topics Name:** Encapsulation

**Source Code:**

#include <iostream>

using namespace std;

// Define a class named 'worker'

class worker

{

private:

    // Private member variables

    int id, age;

    string name, gender;

public:

    // Public member function to set information for the worker

    void setInfo()

    {

        cout << "Enter your Full Name: ";

        getline(cin, name);

        cout << "Enter your ID: ";

        cin >> id;

        cout << "Enter your Age: ";

        cin >> age;

        cout << "Enter your Gender: ";

        cin >> gender;

        cout << endl;

    }

    // Public member function to display information about the worker

    void getInfo()

    {

        cout << "Full Name: " << name << endl;

        cout << "ID: " << id << endl;

        cout << "Age: " << age << endl;

        cout << "Gender: " << gender << endl;

    }

};

int main()

{

    // Create an object of the 'worker' class

    worker p1;

    // Call the setInfo method to input information

    p1.setInfo();

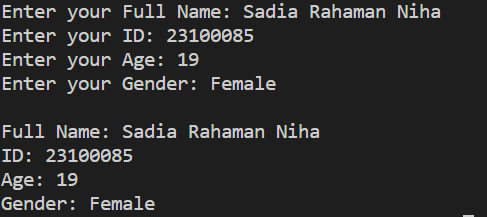
    // Call the getInfo method to display information

    p1.getInfo();

    return 0;

}

**Input and Output:**

****

**Lab No: 07**

**Topics Name:** Static properties

**Source Code:**

#include <iostream>

using namespace std;

class MyClass

{

public:

    // Static property

    static int staticProperty;

    // Regular member property

    int regularProperty;

    // Constructor

    MyClass(int Value)

    {

       regularProperty = Value;

    }

    // Static method that can access static property

    static void printStaticProperty()

    {

        cout << "Static Property in Static method: " << staticProperty << "\n";

    }

    // Regular method that can access both static and regular properties

    void printProperties()

    {

        cout << "Static Property in Regular method: " << staticProperty << "\n";

        cout << "Regular Property in Regular method: " << regularProperty << "\n";

    }

};

// Initializing the static property

int MyClass::staticProperty = 42;

int main()

{

    // Creating instances of MyClass

    MyClass obj1(10);

    MyClass obj2(20);

    // Calling static method

    obj1.printStaticProperty();

    // Calling regular method

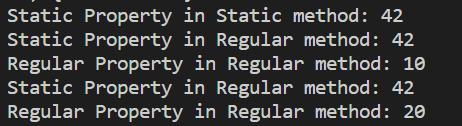
    obj1.printProperties();

    obj2.printProperties();

    return 0;

}

**Input and Output:**



**Lab No: 08**

**Topics Name:** Constructor

**Source Code:**

#include <iostream>

using namespace std;

class person

{

public:

    string name, gender;

    int id, age;

    person()  // Default Constructor

    {

        cout << "Default Constructor is called!" << endl;

    }

    person(string a, int b) // 2 parameterized constructor

    {

        name = a;

        id = b;

    }

    person(string n, string g, int a) // 3 parameterized constructor

    {

        name = n;

        gender = g;

        age = a;

    }

    void display()  // Display name and id of 2 parameterized constructor

    {

        cout << name << " " << id << endl;

    }

    void show()  // Display name, gender and age of 3 parameterized constructor

    {

        cout << name << " " << gender << " " << age << endl;

    }

};

int main()

{

    // Creating object 'p' for calling Default Constructor

    person p;

    // Object 'p1' for Calling 2 parameterized Constructor

    person p1("Topu", 18);

    p1.display();

    // Object 'p2' for Calling 3 parameterized Constructor

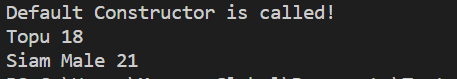
    person p2("Siam", "Male", 21);

    p2.show();

    return 0;

}

**Input and Output:**



**Lab No: 09**

**Topics Name:** Friend function

**Source Code:**

#include <iostream>

using namespace std;

class add

{

private:  int a;

protected: int b;

public:

    void getData();

    friend void display(add &obj); // Declare display() as a friend function

};

// Member function definition for getData()

void add::getData()

{

    cout << "Enter Two Numbers to Addition: ";

    cin >> a >> b;

}

// Friend function definition for display()

void display(add &obj)

{

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

}

int main()

{

    add p1;       // Create an object of class add

    p1.getData(); // Call the getData() member function to input values

    display(p1);  // Call the friend function display() to display the addition result

    return 0;

}

**Input and Output:**



**Lab No: 10**

**Topics Name:** Friend Class

**Source Code:**

#include <iostream>

using namespace std;

class c1

{

private:

    int a;

public:

    void setA(int x)

    {

        a = x;

    }

    // Member function to display the value of 'a' inside c1

    void display()

    {

        cout << "Value of a inside c1: " << a << endl;

    }

    // Friend class declaration for c3

    friend class c3;

};

class c2

{

private:

    int a;

public:

    void setA(int x)

    {

        a = x;

    }

    // Member function to display the value of 'a' inside c2

    void display()

    {

        cout << "Value of a inside c2: " << a << endl;

    }

    // Friend class declaration for c3

    friend class c3;

};

class c3

{

public:

    // Member function to swap the 'a' values between c1 and c2 objects

    void swapc1c2(c1 &o1, c2 &o2)

    {

        int temp = o1.a;

        o1.a = o2.a;

        o2.a = temp;

    }

};

int main()

{

    // Create an object of c1

    c1 obj1;

    obj1.setA(10);

    obj1.display();

    // Create an object of c2

    c2 obj2;

    obj2.setA(30);

    obj2.display();

    // Create an object of c3

    c3 obj3;

    // Call the swapc1c2 function to swap 'a' values between c1 and c2 objects

    obj3.swapc1c2(obj1, obj2);

    cout << "After swap: " << endl;

    // Display the values of 'a' inside c1 and c2 after the swap

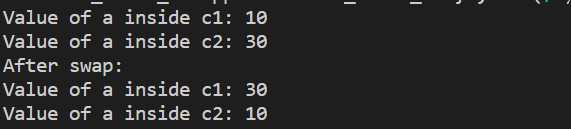
    obj1.display();

    obj2.display();

    return 0;

}

**Input and Output:**



**Lab No: 11**

**Topics Name:** Inheritance (single, multilevel)

**11.1|Source Code of Single Inheritance:**

#include <bits/stdc++.h>

using namespace std;

// Defining a base class 'person'

class person

{

public:

    string name;

    int id;

    void display1()

    {

        cout << "Student Name is: " << name << endl;

        cout << "Student ID is: " << id << endl;

    }

};

// Child class 'employee' inherits from the 'person' class

// It can access only public and protected members of the base class

class employee : public person

{

public:

    int age;

    int salary;

    void display2()

    {

        // Invoking the display1() function from the base class

        display1();

        cout << "Student Age is: " << age << endl;

        cout << "Student Salary is: " << salary << " TK" << endl;

    }

};

int main()

{

    employee e1;

    // Setting values for the data members of the 'employee' class

    e1.name = "Siam Hossain";

    e1.age = 21;

    e1.id = 23100084;

    e1.salary = 20000;

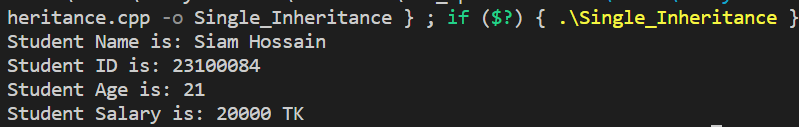
    // Displaying the information using the 'display2()' function

    e1.display2();

    return 0;

}

**11.1|Input and Output Single Inheritance:**

****

**11.2|Source Code of Multilevel Inheritance:**

#include <iostream>

using namespace std;

// Class A definition

class A

{

public:

    int a;

    int b;

    void display()

    {

        cout << a + b << endl;

    }

};

// Class B inherits publicly from class A

class B : public A

{

public:

    int c;

    void display1()

    {

        cout << a + b + c << endl;

    }

};

// Class C inherits publicly from class B

class C : public B

{

public:

    int d;

    // Inherit Member Variable a, b from class A and c from class B

    void display2()

    {

        cout << "Addition of " << a << ", " << b << ", " << c << ", " << d << " is: " << a + b + c + d << endl;

    }

};

int main()

{

    C c1;

    // Assigning values to the data members of class C

    c1.a = 10;

    c1.b = 20;

    c1.c = 30;

    c1.d = 40;

    // Calling the display2 function to show the sum of a, b, c, and d

    c1.display2();

    return 0;

}

**11.2|Input and Output of Multilevel Inheritance:**

****

**Lab No: 12**

**Topics Name:** Access specifier

**Source Code:**

#include <iostream>

using namespace std;

class MyClass

{

public:

    // Public member variable

    int publicVar;

    // Public member function

    void publicFunction()

    {

        cout << "Public function called." << endl;

    }

private:

    // Private member variable

    int privateVar;

    // Private member function

    void privateFunction()

    {

        cout << "Private function called." << endl;

    }

protected:

    // Protected member variable

    int protectedVar;

    // Protected member function

    void protectedFunction()

    {

        cout << "Protected function called." << endl;

    }

public:

    // Public member function accessing private member

    void setPrivateVar(int value)

    {

        privateVar = value;

    }

    // Public member function displaying private member

    void displayPrivateVar()

    {

        cout << "Private variable: " << privateVar << endl;

    }

};

class protect : public MyClass

{

public:

    void display\_protect()

    {

        protectedFunction();

    }

};

int main()

{

    // Create an object of MyClass

    MyClass myObject;

    // Access public members

    myObject.publicVar = 42;

    myObject.publicFunction();

    // Access private members through public member functions

    myObject.setPrivateVar(100);

    myObject.displayPrivateVar();

    // Access protected members (only within derived classes)

    protect pro;

    pro.display\_protect();

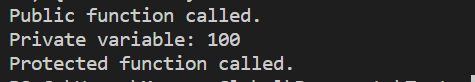
    // myObject.protectedVar; and myObject.protectedFunction();

    // This would be an error outside of a derived class.

    return 0;

}

**Input and Output:**

****

**Lab No: 13**

**Topics Name:** Inheritance (Hierarchical)

**Source Code:**

#include <iostream>

using namespace std;

// Base class representing a person

class person

{

public:

    int id;

    string name;

    int age;

    void display()

    {

        cout << "Full Name: " << name << endl;

        cout << "ID: " << id << endl;

        cout << "Age: "<< age << endl;

    }

};

// Derived class representing a programmer, inheriting from person

class programmer : public person

{

public:

    string pro\_lang;

    string phone;

    void display1()

    {

        // Call the display function of the base class (person)

        cout<< "Information of Programmers: ";

        display();

        cout << "Proficient Programming Language: " << pro\_lang << endl;

        cout << "Phone No: " << phone << endl;

    }

};

// Derived class representing a manager, inheriting from person

class manager : public person

{

public:

    string skill;

    void display2()

    {

        // Call the display function of the base class (person)

        cout<< "Information of Manager: ";

        display();

        cout << "Skill: " << skill << endl;

    }

};

int main()

{

    programmer p1;

    manager m1;

    // Setting values for the programmer object

    p1.id = 23100084;

    p1.name = "Siam Hossain";

    p1.age = 21;

    p1.pro\_lang = "C++, Java.";

    p1.phone = "+8801909967161";

    p1.display1();

    cout << endl;

    // Setting values for the manager object

    m1.id = 23100069;

    m1.name = "Talha";

    m1.age = 20;

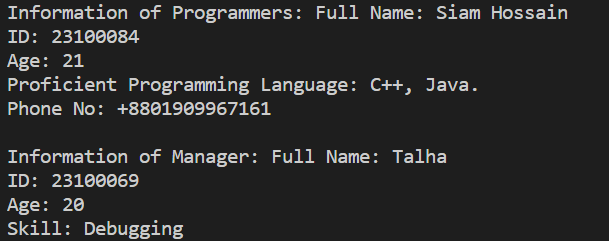
    m1.skill = "Debugging";

    m1.display2();

    return 0;

}

**Input and Output:**



**Lab No: 14**

**Topics Name:** Inheritance (Multiple)

**Source Code:**

#include <iostream>

using namespace std;

// Class A with a public member variable 'a' and two display functions

class A

{

public:

    int a = 10;

    void display()

    {

        cout << "The Number: " << a << endl;

    }

    void display1()

    {

        cout << "The Addition of " << a << " and " << a << " is: " << a + a << endl;

    }

};

// Class B with a public member variable 'b' and two display functions

class B

{

public:

    int b = 20;

    void display()

    {

        cout << "The Number: " << b << endl;

    }

    void display2()

    {

        cout << "The Addition of " << b << " and " << b << " is: " << b + b << endl;

    }

};

// Class C inherits publicly from both classes A and B

class C : public A, public B

{

};

int main()

{

    A a1;

    B b1;

    C obj1;

    a1.display();

    obj1.display1();

    b1.display();

    obj1.display2();

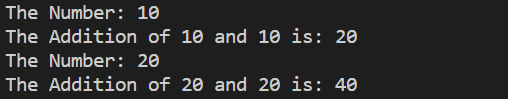
    // Uncommenting the following line will result in an error due to ambiguity

    // obj1.display();

    return 0;

}

**Input and Output:**

****

**Lab No: 15**

**Topics Name:** Inheritance (Hybrid)

**Source Code:**

#include <iostream>

using namespace std;

// Base Class - 01

class Animal

{

public:

    void eat()

    {

        cout << "Animal is eating." << endl;

    }

};

// Base Class - 02

class Mammal

{

public:

    void giveBirth()

    {

        cout << "Mammal is giving birth." << endl;

    }

};

// Sub Class - 01

class Dog : public Animal

{

public:

    void bark()

    {

        cout << "Dog is barking." << endl;

    }

};

// Sub Class - 02

class Bat : public Mammal, public Dog

{

public:

    void fly()

    {

        cout << "Bat is flying." << endl;

    }

};

int main()

{

    // Create object of the classes

    Dog myDog;

    Bat myBat;

    // Access methods from the Dog class

    myDog.eat();

    myDog.bark();

    // Access methods from both Animal and Mammal classes through the Bat class

    myBat.eat();

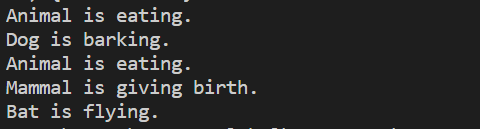
    myBat.giveBirth();

    myBat.fly();

    return 0;

}

**Input and Output:**

****

**Lab No: 16**

**Topics Name:** Compile Time Polymorphism (Overloading)

**Source Code:**

#include <iostream>

using namespace std;

class overloading

{

public:

    int sum(int a, int b)

    {

        cout << "Function with two INT parameters addition is: "<< endl;

        return a + b;

    }

    float sum(float a, float b)

    {

        cout << "Function with two FLOAT parameters addition is: " << endl;

        return a + b;

    }

    int sum(int a, int b, int c)

    {

        cout << "Function with three INT parameters addition is: " << endl;

        return a + b + c;

    }

};

int main()

{    overloading myObj;

    // Call the function to add two floating-point numbers

    cout << myObj.sum(5.5f, 6.6f) << endl;

    // Call the function to add three integers

    cout << myObj.sum(5, 6, 7) << endl;

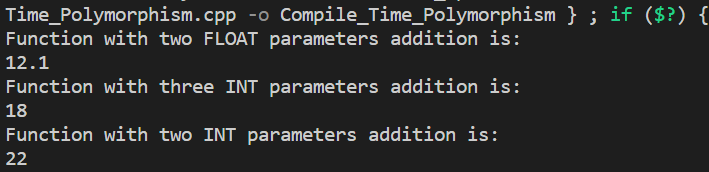
    // Call the function to add two integers

    cout<< myObj.sum(10, 12) << endl;

    return 0;

}

**Input and Output:**

****

**Lab No: 17**

**Topics Name:** Run Time Polymorphism (Overriding)

**Source Code:**

#include <iostream>

using namespace std;

class BaseClass // Base class

{

public:

    int a = 10;

    void display()

    {

        cout << "Base class a is: " << a << endl;

    }

};

// Derived class definition, inheriting publicly from BaseClass

class DerivedClass : public BaseClass

{

public:

    int b = 100; // Public member variable 'b' with an initial value of 100

    // Member function to display the values of 'a' and 'b'

    void display()

    {

        cout << "Derived class a is: " << a << endl;

        cout << "Derived class b is: " << b << endl;

    }

};

int main()

{

    DerivedClass obj1;

    // Calling the display function of the DerivedClass

    obj1.display();

    BaseClass base;

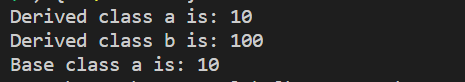
    // Calling the display function of the BaseClass

    base.display();

    return 0;

}

**Input and Output:**

****

**Lab No: 18**

**Topics Name:** Virtual function

**Source Code:**

// Only showing the functionalities to the user...

// Only Base Class can only be an Abstract Class...

// Abstract class can not creat an Object....

// The Base class virtual method must be override in the derived class...

#include <iostream>

#include <conio.h>

using namespace std;

// Using Abstract base class defining the Virtual Function for car drivers

class carDriver

{

public:

    virtual void information() = 0; // Pure virtual function for displaying information

    virtual void get()

    {

        // Virtual function for getting input

    }

};

// Derived class Siam implementing the carDriver interface

class Siam : public carDriver

{

public:

    int id;

    int salary;

    void get()

    {

        cout << "Enter the ID and Salary of Siam: ";

        cin >> id >> salary;

    }

    void information()

    {

        cout << "The ID No of Siam is: " << id << endl;

        cout << "The Monthly salary of Siam is: " << salary << endl;

    }

};

// Derived class Talha implementing the carDriver interface

class Talha : public carDriver

{

public:

    int id;

    string home\_add;

    void get()

    {

        cout << endl

             << "Enter the ID and Home Address of Talha: ";

        cin >> id;

        getline(cin, home\_add);

    }

    void information()

    {

        cout << "The ID No of Talha is: " << id << endl;

        cout << "The Home Address of Talha is: " << home\_add << endl;

    }

};

int main()

{

    carDriver \*v;

    Siam s;

    Talha t;

    // Using a pointer to the base class

    v = &s;

    v->get();

    v->information();

    v = &t;

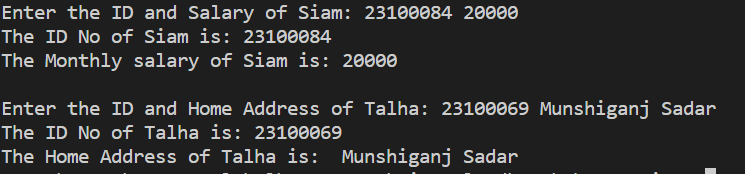
    v->get();

    v->information();

    return 0;

}

**Input and Output:**

****

**Lab No: 19**

**Topics Name:** Abstraction

**Source Code:**

#include <iostream>

using namespace std;

class Base // Base class definition

{

public:

    int a = 5;

    // Example of runtime polymorphism with the help of virtual method.

    virtual void display()

    {

        cout << "Base class" << endl;

        cout << "Base class a is " << a << endl;

    }

};

class derived : public Base // Derived class definition

{

public:

    int b = 100;

    // Override the display method in the derived class

    void display()

    {

        cout << "Derived class." << endl;

        cout << "Base class a is " << a << endl;

        cout << "Derived class b is " << b << endl;

    }

};

int main()

{

    Base \*base\_class\_pointer;

    Base obj1;

    derived obj2;

    // Point the base class pointer to the derived class object

    base\_class\_pointer = &obj2;

    base\_class\_pointer->display();

    // Declare a pointer to the derived class

    derived \*derived\_class\_pointer;

    // Point the derived class pointer to the derived class object

    derived\_class\_pointer = &obj2;

    derived\_class\_pointer->display();

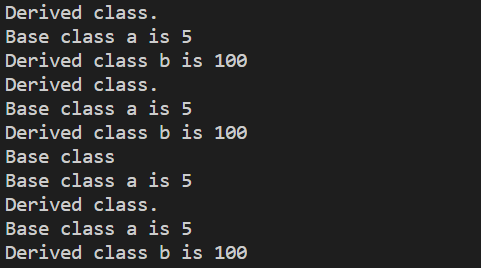
    obj1.display();

    obj2.display();

    return 0;

}

**Input and Output:**

****

**Lab No: 20**

**Topics Name:** Exception handling

**Source Code:**

// Error are mainly two types:

// 1. Compile time error : An error while the code is compiling

// 2. Run time Error: Errors which are occurring during the run time.

//  Exception is a run time error.

//  Exception Handling is a mechanism that can handle the exception.

#include <iostream>

using namespace std;

int main()

{

    int num1, num2;

    cout << "Enter number 01: ";

    cin >> num1;

    cout << "Enter number 02: ";

    cin >> num2;

    try

    {

        // If user input the num2 = 0 then program will return -1 directly to the catch()

        if (num2 == 0)

        {

            throw -1;

        }

        double result = (double)num1 / num2;

        cout << "Result: " << result << endl;

    }

    catch (int x)

    {

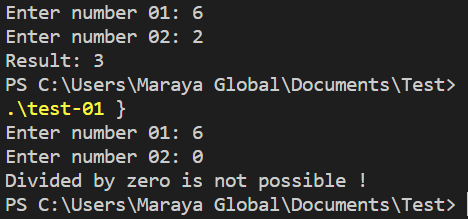
        cout << "Divided by zero is not possible !" << endl;

    }

    return 0;

}

**Input and Output:**

****

Thank You!