|  |  |  |
| --- | --- | --- |
| **Rubrics for Object Oriented Programming Lab** | | |
| **Lab #:** | **03** | |
| **Lab Title:** | *Class Scope and Accessing Class Members* | |
| **Submitted by:** | | |
| **Name** | | **Registration #** |
| **AMMAR**  **MUHAMMAD KALEEM ULLAH** | | **FA19-BCE-001**  **FA19-BCE-007** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rubrics name & number** | | | **Marks** | | |
| **In-Lab** | | **Post-Lab** |
| **Engineering Knowledge** | ***R2: Use of Engineering Knowledge and follow Experiment Procedures:***  *Ability to follow experimental procedures, control variables, and record procedural steps on lab report.* | |  | | |
| **Problem Analysis** | | ***R5: Data/Evidence Measurements:***  *Ability to record raw data / evidence.* | |  | |
| **Design** | | ***R8: Best Coding Standards:***  *Ability to follow the coding standards and programming practices.* | |  | |
| **Modern Tools Usage** | | ***R9: Understand Tools:*** *Ability to describe and explain the principles behind and applicability of engineering tools.* | |  | |
| **Individual and Teamwork** | | ***R12: Individual Work Contributions:*** *Ability to carry out individual responsibilities.* | |  | |
| ***R13: Management of Team Work:***  *Ability to appreciate, understand and work with multidisciplinary team members.* | |  | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rubrics #** | R2 | R5 | R8 | R9 | R12 | R13 |
| **In –Lab** |  |  |  |  |  |  |
| **Post- Lab** |  |  |  |  |  |  |

**Lab 03 – *Class Scope and Accessing Class Members***

1. **Objectives**

The objective of this lab is to teach the students, the scope of class data members and its member functions.

1. **Outcome**

At the end of this lab student will be familiar with the accessing rules of class data members and member functions

1. **Introduction**

In object oriented programming, methods and variables have various scope. Scope means that the method or variable may or may not be directly accessible to other objects or classes. Classes that do not have instances may be accessible to the system.

One of the techniques in object-oriented programming is encapsulation. It concerns the hiding of data in a class and making them available only through its methods. In this way the chance of making accidental mistakes in changing values is minimized. C++ allows you to control access to classes, methods, and fields via so-called access modifiers. The access to classes, constructors, methods and fields are regulated using access modifiers i.e. a class can control what information or data can be accessible by other classes. To take advantage of encapsulation, you should minimize access whenever possible.

* 1. **Class Scope**

Class variables and class methods are associated with a class. An instance of the class (object) is not required to use these variables or methods. Class methods cannot access instance variables or methods, only class variables and methods.

* 1. **Instance Scope**

Instance variables and instance methods are associated with a specific object. They can access class variables and methods.

* 1. **Private Scope**

A **private** member variable or function cannot be accessed, or even viewed from outside the class. Only the class and friend functions can access private members. Private variables and private methods are only accessible to the object they are contained in.

* 1. **Protected Scope**

Protected variables and protected methods are accessible by the class they are in and inheriting classes (sub classes) only.

* 1. **Public Scope**

A **public** member is accessible from anywhere outside the class but within a program. You can set and get the value of public variables without any member function

* 1. **Encapsulation**

The process of providing a public interface to interact with the object while hiding other information inside the object is called encapsulation.

1. **In Lab Tasks** 
   1. Code the example given above and check the errors if you try to access the private data members in main() function. Also modify the above task by making the scope of public member functions as private. Create access functions in public scope to access private member functions from main().

**Code:**

#include <iostream>

using namespace std;

class Subtract

{

private:

int Data\_1;

int Data\_2;

public:

void Assigner(int,int);

void Display\_Result();

int Subtractor() ;

};

void Subtract::Assigner(int A,int B)

{

Data\_1=A;

Data\_2=B;

}

int Subtract::Subtractor()

{

return(Data\_1-Data\_2);

}

void Subtract::Display\_Result()

{

cout<<"\n\t The Result From Subtraction Of Data-1 And Data-2 Is : "<<Subtractor();

}

int main()

{

Subtract Alpha;

int Data1,Data2;

cout<<"\n\t Input The Number 1 Data : ";cin>>Data1;

cout<<"\n\t Input The Number 2 Data : ";cin>>Data2;

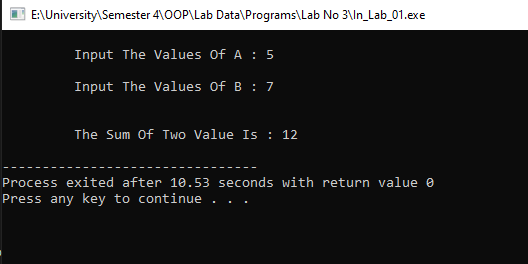
Alpha.Assigner(Data1,Data2);

Alpha.Subtractor();

Alpha.Display\_Result();

}

**Output:**



* 1. Create an employee class, The member data should comprise an int for storing the employee number and a float for storing the employee’s compensation. Member functions should allow the user to enter this data and display it. Write a main() that allows the user to enter data for three employees and display it.

**Code:**

#include <iostream>

using namespace std;

class Employee

{

private:

int Number;

float Compen;

public:

void Input()

{

cout<<"\n\t Input The Employee Number : ";cin>>Number;

cout<<"\n\t Input The Employee Compensation : ";cin>>Compen;

}

void Display()

{

cout<<"\n\n\t Employee Number : "<<Number<<"\n\t Compensation Is : "<<Compen<<endl;

}

};

int main()

{

Employee Alpha[3];

for(int i=0;i<3;i++)

{

Alpha[i].Input();

}

system("cls");

for(int j=0;j<3;j++)

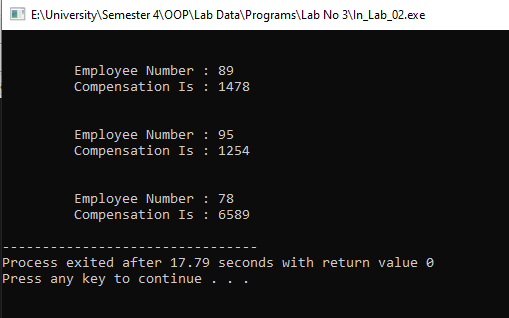
{

Alpha[j].Display();

}

}

**Output:**



**------------------------------------------------------------------------------------------------------------**

* 1. Create a class called time that has separate int member data for hours, minutes, and  
     seconds. One constructor should initialize this data to 0, and another should initialize it  
     to fixed values. Another member function should display it, in 11:59:59 format. The final member function should add two objects of type time passed as arguments.  
     A main() program should create two initialized time objects and  
     one that isn’t initialized. Then it should add the two initialized values together, leaving the result in the third time variable. Finally it should display the value of this third variable.

**Code:**

#include <iostream>

using namespace std;

class Time

{

public:

int Hours;

int Mins;

int Secs;

char Meri;

Time() // Constructor No 1

{

Hours=0;

Mins=0;

Secs=0;

}

Time(int H,int M,int S) // Constructor No 2

{

Hours=H;

Mins=M;

Secs=S;

}

void Display\_12()

{

if(Mins>60)

{

Hours++;

Mins=Mins-60;

}

if(Secs>60)

{

Mins++;

Secs=Secs-60;

}

if(Hours > 12)

{

Hours=Hours-12;

Meri='P';

}

if(Meri=='P')

{

cout<<"\n\n\t Time In Standered 12 Hour Format is :\t "<<Hours<<" Hour : "<<Mins<<" Miuntes : "<<Secs<<" Seconds\tPm";

}else

{

cout<<"\n\n\t Time In Standered 12 Hour Format is :\t "<<Hours<<" Hour : "<<Mins<<" Miuntes : "<<Secs<<" Seconds\tAm";

}

}

void Adder(Time A,Time B)

{

Hours=A.Hours+B.Hours;

Mins=A.Mins+B.Mins;

Secs=A.Secs+B.Secs;

}

void Display\_Final()

{

if(Mins>60)

{

Hours++;

Mins=Mins-60;

}

if(Secs>60)

{

Mins++;

Secs=Secs-60;

}

cout<<"\n\n\t The Sum Of Both Entered Time is :\t "<<Hours<<" Hour : "<<Mins<<" Miuntes : "<<Secs<<" Seconds";

}

};

int main()

{

Time Alpha\_1;

// Time No 1

int Ho,Mi,Se;

cout<<"\n\n\t Please Enter The Hours: "; cin>>Ho;

cout<<"\n\t Please Enter The Minutes: "; cin>>Mi;

cout<<"\n\t Please Enter The Seconds: "; cin>>Se;

Time Alpha\_2(Ho,Mi,Se);

// Time No 2

int Ho2,Mi2,Se2;

cout<<"\n\n\t Please Enter The Hours: "; cin>>Ho2;

cout<<"\n\t Please Enter The Minutes: "; cin>>Mi2;

cout<<"\n\t Please Enter The Seconds: "; cin>>Se2;

Time Alpha\_3(Ho2,Mi2,Se2);

// Displayer

Alpha\_2.Display\_12();

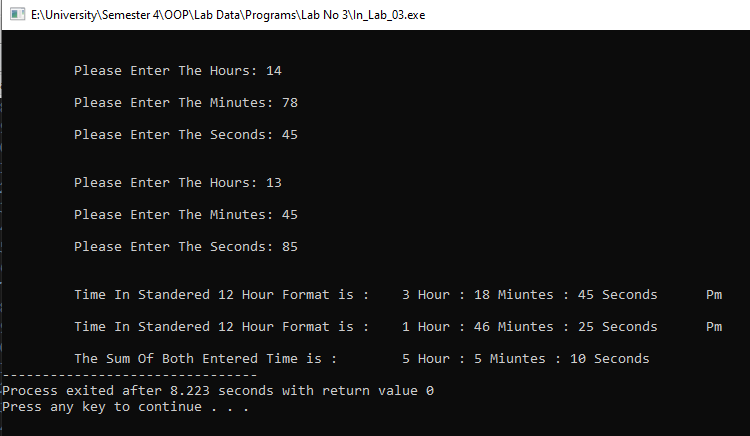
Alpha\_3.Display\_12();

Alpha\_1.Adder(Alpha\_2,Alpha\_3);

Alpha\_1.Display\_Final();

}

**Output:**



**------------------------------------------------------------------------------------------------------------**

* 1. Create a fraction class. Member data is the fraction’s numerator and denominator. Member functions should accept input from the user in the form 3/5, and output the fraction’s value in the same format. Another member function should add two fraction values. Write a main() program that allows the user to repeatedly input two fractions and then displays their sum. After each operation, ask whether the user wants to continue.

**Code:**

#include <iostream>

using namespace std;

class Fraction

{

private:

int Nomi;

int Denom;

public:

void Inputer(int A,int B)

{

Nomi=A;

Denom=B;

}

void Display()

{

cout<<"\n\n\t Value = "<<Nomi<<"/"<<Denom;

}

void Adder(Fraction A,Fraction B)

{

if(A.Denom==B.Denom)

{

cout<<"\n\n\t The Sum Of 2 Fractions Is : "<<A.Nomi+B.Nomi<<" / "<<A.Denom;

}

else

{

int X=A.Denom,Y=B.Denom;

A.Nomi=A.Nomi\*Y;

A.Denom=A.Denom\*Y;

B.Nomi=B.Nomi\*X;

B.Denom=B.Denom\*X;

cout<<"\n\n\t The Sum Of 2 Fractions Is "<<A.Nomi+B.Nomi<<" / "<<A.Denom;

}

}

};

int main()

{

Fraction Alpha[3];

int Nomii,Denomm;

for(int i=0;i<2;i++)

{

cout<<"\n\t"<<"("<<i+1<<")-"<<" Input The Values Of Nominator : ";cin>>Nomii;

cout<<"\n\t Input The Values Of Denominator : ";cin>>Denomm;

Alpha[i].Inputer(Nomii,Denomm);

}

system("cls");

// Displayer

for(int i=0;i<2;i++)

{

Alpha[i].Display();

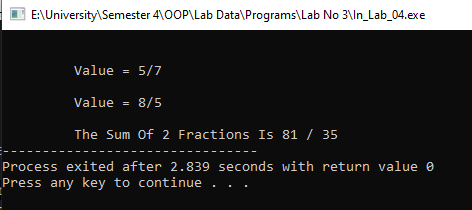
}

// Adder Caller

Alpha[2].Adder(Alpha[0],Alpha[1]);

}

**Output:**



1. **Post Lab Tasks** 
   1. Create a date class. Its member data should consist of three ints: month, day, and year. It should also have two member functions: getdate() , which allows the user to enter a date in 12/31/02 format, and showdate() , which displays the date.

**Code:**

#include <iostream>

using namespace std;

class Date

{

private:

int Year;

int Month;

int Date;

public:

void Get\_Data(int D,int M,int Y);

void Display() const;

};

void Date::Get\_Data(int D,int M,int Y)

{

Date=D;

Month=M;

Year=Y;

}

void Date::Display() const

{

cout<<"\n\n\t The Entered Date is :\t "<<Date<<" / "<<Month<<" / "<<Year;

}

int main()

{

Date Alpha;

int Da,Mo,Ye;

cout<<"\n\n\t Please Enter The Date: "; cin>>Da;

cout<<"\n\t Please Enter The Month: "; cin>>Mo;

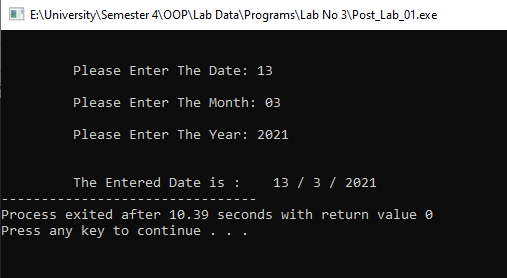
cout<<"\n\t Please Enter The Year: "; cin>>Ye;

Alpha.Get\_Data(Da,Mo,Ye);

Alpha.Display();

}

**Output:**



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

* 1. Create a class of subtraction having two private data members. Create member functions to get data from users and for subtraction of data members. Use appropriate access modifiers for class methods.

**Code:**

#include <iostream>

using namespace std;

class Subtract

{

private:

int Data\_1;

int Data\_2;

public:

void Assigner(int,int);

void Display\_Result();

int Subtractor() ;

};

void Subtract::Assigner(int A,int B)

{

Data\_1=A;

Data\_2=B;

}

int Subtract::Subtractor()

{

return(Data\_1-Data\_2);

}

void Subtract::Display\_Result()

{

cout<<"\n\t The Result From Subtraction Of Data-1 And Data-2 Is : "<<Subtractor();

}

int main()

{

Subtract Alpha;

int Data1,Data2;

cout<<"\n\t Input The Number 1 Data : ";cin>>Data1;

cout<<"\n\t Input The Number 2 Data : ";cin>>Data2;

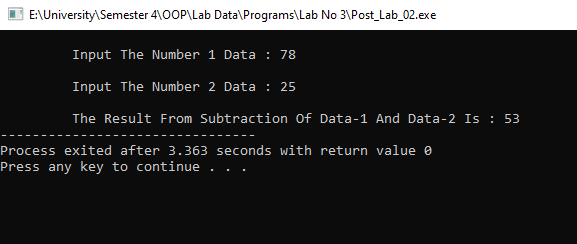
Alpha.Assigner(Data1,Data2);

Alpha.Subtractor();

Alpha.Display\_Result();

}

**Output:**



**Conclusions:**

* One of the biggest **advantages** of **C++** is the feature of object-oriented programming which includes concepts like **classes**, inheritance, polymorphism, data abstraction, and encapsulation that allow code reusability and makes a program even more reliable.
* **Access modifier** provides you the authority to control your data depending upon the scenarios. ...
* All base class public member becomes public members of the derived class. ...
* In private inheritance scenario, all base class public member becomes private members of the derived class.