|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  | **Programming Fundamentals** | |  | **(CL214)** | |  | **LABORATORY MANUAL** | |  | **Spring 2021** | |  | **C:\Users\Aamer\Desktop\nu-new.png**  **LAB 09** | |  | **Classes** | |  |  |  |  |  |  | | --- | --- | --- | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | \_\_\_\_\_\_\_\_\_\_ | \_\_\_ | | STUDENT NAME | ROLL NO | SEC | |  | | | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | LAB ENGINEER SIGNATURE & DATE | | | | **MARKS AWARDED: /10** | | | |  | | | | **NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES (NUCES), ISLAMABAD** | | | |  | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Prepared by: | Engr. Sana Saleh | Date: | 29 March, 2019 |
| Verified by: | Dr. Shahid Qureshi | Date: | 30 March, 2019 |

|  |  |
| --- | --- |
| **LAB 09** | **Classes** |

**Lab Objectives:**

1. To give introduction to classes.

**Software Required:**

* Code Blocks

**Introduction:**

1. **Object Oriented Programming:**

Instead of having function-oriented programs i.e. getting data and performing functions with it, we have now data that knows how to manipulate itself. This way, now the programming is object-oriented. It means that our programs revolve around data and the objects. Therefore, it would be nice to have some building blocks for programs so that these can be combined to write a program. We have so far talked about simple variables like integer, double and char, followed by strings of characters and arrays of different data types. But now, in the form of an object, we have a block which knows itself about contents and the behavior.

* 1. **Objects:**

While engaged in the process of programming, we try to solve a real-world problem. The real world is not only of integers, floats, doubles and chars, but there are other things like cycles, cars, buildings, schools and people. We perceive all these things as objects. Each object has a behavior associated with it. Consider the example of a man who can talk, walk, sit and stand etc. Similarly, we can think of a vehicle that has many functions. These objects also have attributes. For example, a man has height, weight, color of eyes and hair and so on. These all are his attributes. His actions will be referred as functions or methods. This principle may be applicable to vehicles, aero planes and all other real world things. An aero plane has attributes like its height, width, number of seats and number of engines etc. These are attributes called data members. Its actions include take off, flying and landing. These actions are its functions or methods.

1. **Classes:**

A class is a way of defining a user-defined data type. In a class, one may find data members and member functions that can manipulate that data.

Class is very similar to the structures, except that we can now include also functions.

* 1. **Access Specifiers:**

Access specifiers modify the access rights that the members following them acquire. It includes one of the following three keywords:

1. Private - Private members of a class are accessible only from within other members of the same class.
2. Public - Public members are accessible from anywhere where the object is visible.
3. Protected – discussed in a future lab.

By default, all members of a class declared with the class keyword have private access for all its members. Therefore, any member that is declared before any explicit class specifier automatically has private access.

//Example: A Simple Class

class rectangle

{ // A simple class

int height ; //height

int width ; //and width are private data members of class

public :

void Display (); //member function of class

};

void rectangle:: Display () //definition of function Display occurs outside of class

{

cout << height <<endl ;

cout << width <<endl ;

}

void main ()

{

rectangle Obj;

Obj. Display ();

}

Output of the above code:

-858993430

-858993430

Output is garbage value as the variables are not initialized before calling the Display () function.

The main purpose of classes is concept of data hiding i.e. encapsulation that means that the data of a class cannot be accessed from outside. However, it can be done through some defined functions (methods). These are the member functions of the class. To hide the data, we declare it private. If a data is private, it will be available only to member functions of the class. No other function outside the class (except friend functions – seen later) can access the private data.

Normally in a class we divide the private part which is typically what we call implementation of the class, from the functions that manipulate that private data which is called the interface (which is the front end which is publicly visible).

The example of a room can helps us understand private and public parts of a class. A class is a room having a curtain in its middle. The things behind the curtain (private) are visible to the residents (insiders) of the room. They know about everything present in the room. When the door opens, the outsiders see only the things in front of the curtain. This is the public interface of the class while behind the curtain is the private interface. A function inside the class (i.e. a member function) can access and manipulate all things in the class. A function outside the class can only access and manipulate its public interface part.

1. **Accessor functions:**

With encapsulation, a programmer defines labels for the data members and functions and specifies whether they are accessible by other classes. When the programmer labels data members "private," they cannot be accessed and manipulated by member functions of other classes. Accessors allow access to these private data members.

An accessor [function](https://www.thoughtco.com/definition-of-void-958182) in C++ and the mutator function are like the set and get functions in [C#](https://www.thoughtco.com/all-about-the-c-programming-language-958330). They are used instead of making a class [member](https://www.thoughtco.com/object-2034254) [variable](https://www.thoughtco.com/definition-of-variable-958320) public and changing it directly within an object. To access a private object member, an accessor function must be called.

Typically for a member such as Level, a function GetLevel() returns the value of Level and SetLevel() to assign it a value. For example:

class rectangle

{ // A simple class

int height ; //height

int width ; //and width are private data members of class

public :

void Get\_Height(); //member function of class

};

int rectangle:: Get\_Height () //definition of function Display occurs outside of class

{

return height;

}

### Characteristics of an Accessor Function

* An accessor doesn't need arguments
* An accessor has the same type as the retrieved variable
* The name of the accessor begins with the Get prefix
* A naming convention is necessary

1. **Mutator Function**

While an accessor function makes a data member accessible, it does not make it editable. Modification of a protected data member requires a mutator function.

Because they provide direct access to protected data, mutator and accessor functions must be written and used carefully.

Here's an example:

class rectangle

{

private:

int height; //height

int width; //and width are private data members of class

public:

int Get\_Height()

{ return Height; } *// accessor*

void Set\_Height (int New\_Height)

{ Height=New\_Height; } *// mutator*

}

It's best practice to make data members private (as in the example above) and only access them via accessors and mutators. This is for the following reasons:

* You know when they are accessed (and can debug this via a breakpoint).
* The mutator can validate the input to ensure it fits within certain constraints.
* If you need to change the internal implementation, you can do so without breaking a lot of external code -- instead you just modify the way the accessors/mutators reference the internal data.

1. **Constructor**
   1. **Simple Constructor:**

A constructor is a special class function which has the exact same name as the class yet it cannot return a value. If no constructor is expressly declared, the compiler automatically initializes one, however it doesn’t do anything. To actually initialize or reset class member data, we can write a constructor that takes no argument. The user defined constructor that takes no argument is called a simple constructor.

When we write such a constructor, it automatically assumes the role of the default-constructor. The compiler will then not call the default constructor. Rather, the constructor written by the programmer will be called whenever an object will be instantiated. It is a good programming practice to provide a default constructor (i.e. a constructor with no argument).

class rectangle

{

// A simple class

int height ;

int width ;

public :

rectangle (); // with a constructor ,

void Display ();

};

void rectangle:: Display ()

{

cout<< height <<endl;

cout<<width <<endl;

}

rectangle :: rectangle () // constructor

{

height = 2;

width = 2;

}

void main ()

{

rectangle Obj;

Obj.Display ();

}

Output:

2

2

* 1. **Parameterized Constructor**

A default constructor does not have any parameter, but if you need, a constructor can have parameters. This helps you to assign initial value to an object at the time of its creation as shown in the following example:

|  |
| --- |
| class rectangle  {  // A simple class  int height ;  int width ;  public :  rectangle (); // with a constructor ,  rectangle (int a, int b);  void Display ();  };  void rectangle:: Display ()  {  cout<< height <<endl;  cout<<width <<endl;  }  rectangle :: rectangle () // simple constructor  {  height = 2;  width = 2;  }  rectangle :: rectangle (int a, int b) // parameterized constructor  {  height = a;  width = b;  }  void main ()  {  rectangle Obj (4,5);  Obj.Display();  } |

Output:

4

5

**Practice Problems:**

1. Write a program that can store student’s data using class “Student”. The data members of class “Student” are mentioned below:

# include <iostream >

# include <string>

using namespace std;

class Student

{

private :

char name[50];

float gpa\_semster1;

float gpa\_semster2;

float cgpa;

public :

Student ();

Student(char \*name1, float gpa1, float gpa2);

float get\_gpa1(); //accessor

float get\_gpa2(); //accessor

void cgpa(float gpa1,float gpa2);

};

You are required to the implement following tasks:

* 1. Store 1st student’s data i.e. name, semester 1 GPA and semester 2 GPA, using the default constructor and 2nd student data using parameterized constructor.
  2. Calculate cgpa of each student using accessor() and cgpa\_calculator () function.
  3. Display each student data on console using accessor () function.

#include<iostream>

using namespace std;

class Student{

private :

char name[50];

float gpa\_semster1;

float gpa\_semster2;

float Cgpa;

public :

Student ();

Student(char \*name1, float gpa1, float gpa2);

float get\_gpa1(); //accessor

float get\_gpa2(); //accessor

void cgpa(float gpa1,float gpa2);

};

Student::Student (){

cout<<"enter first student name"<<endl;

cin.getline(name,50);

cout<<"enter first student first semester gpa"<<endl;

cin>>gpa\_semster1;

cout<<"enter first student second semester gpa"<<endl;

cin>>gpa\_semster2;

}

float Student::get\_gpa1(){

cout<<"first student name = "<<name<<endl;

cout<<"first student first semester gpa = "<<gpa\_semster1<<endl;

cout<<"first student second semester gpa = "<<gpa\_semster2<<endl;

Cgpa=(gpa\_semster1+gpa\_semster2)/2;

return Cgpa;

}

Student::Student(char \*name1, float gpa1, float gpa2){

for(int i=0;i<50;i++){

name[i]=name1[i];

}

gpa\_semster1=gpa1;

gpa\_semster2=gpa2;

}

float Student::get\_gpa2(){

cout<<"second student name "<<name<<endl;

cout<<"second student first semester gpa = "<<gpa\_semster1<<endl;

cout<<"second student second semester gpa = "<<gpa\_semster2<<endl;

Cgpa=(gpa\_semster1+gpa\_semster1)/2;

cout<<"cgpa is of second student is"<<endl<<Cgpa<<endl;

}

void Student::cgpa(float gpa1,float gpa2){

cout<<"cgpa is of first student is"<<endl<<get\_gpa1()<<endl;

}

int main(){

Student s;

float gpa1,gpa2;

char name1[50];

cout<<"enter second student name"<<endl;

cin>>name1;

cout<<"enter second student first semester gpa"<<endl;

cin>>gpa1;

cout<<"enter second student second semester gpa"<<endl;

cin>>gpa2;

Student s1(name1,gpa1,gpa2);

//s.get\_gpa1();

s1.get\_gpa2();

s.cgpa( gpa1, gpa2);

}

1. Write a C++ program that can perform the following operations on a class “Circle”:

|  |
| --- |
| class Circle  {  private:  double radius;  public:  Circle ( );  void set\_radius();  double Area();  void Circumference(double&); //keep the return type void  }; |

a.    Create three objects of a class "Circle". By using the default constructor assigns zero value to the radius of each circle.

b.        Ask the user, to enter the radius value for each circle. (use “set\_value” function)

**c.**       Calculate the area of each circle. The formula for finding the area of the circle is:        **Area= π\*r2           (π=22/7=3.14)**

d.     Calculate the circumference of the circle using the following formula:

**Circumference=2\*π\*r**

e.    Mention the circle that has the largest area.

Declare accessor and mutator functions inside the class and define the functions out of the class using the scope operator **‘ ::** **’**

#include<iostream>

using namespace std;

class circle{

private:

float radius;

float area;

float circumference;

char max[50];

public:

circle();

float set\_value(float radius,char \*max);

void get\_value();

};

circle::circle(){

radius=0;

area=0;

circumference=0;

}

float circle::set\_value( float r,char \*m){

radius=r;

area=(3.14\*(r\*r));

circumference=2\*3.14\*r;

for(int i=0;i<50;i++){

max[i]=m[i];

}

}

void circle::get\_value(){

cout<<"radius of first circle is ="<<radius<<endl;

cout<<"Area of the that circle is ="<<area<<endl;

cout<<"Circumference of that circle is"<<circumference<<endl;

cout<<max;

}

int main(){

float r1,r2,r3;

cout<<"enter the value of radius of first circle"<<endl;

cin>>r1;

cout<<"enter the value of second of first circle"<<endl;

cin>>r2;

cout<<"enter the value of third of first circle"<<endl;

cin>>r3;

circle c1,c2,c3;

if(r1>r2&&r1>r3){

char m[50]="first circle has maximum area";

c1.set\_value(r1,m);

c2.set\_value(r2,m);

c3.set\_value(r3,m);

}

if(r2>r3&&r2>r1){

char m[50]="second circle has largest area";

c1.set\_value(r1,m);

c2.set\_value(r2,m);

c3.set\_value(r3,m);

}

if(r3>r2&&r3>r1){

char m[50]="third circle has largest area";

c1.set\_value(r1,m);

c2.set\_value(r2,m);

c3.set\_value(r3,m);

}

/\*circle c1,c2,c3;

c1.set\_value(r1,m);

c2.set\_value(r2,m);

c3.set\_value(r3,m);

\*/

c1.get\_value();

c2.get\_value();

c3.get\_value();

}