National University of Computer & Emerging Sciences, Karachi
Computer Science Department
Spring 2022, Lab Manual - 03

Course Code: CL-1004	**Course : Object Oriented Programming Lab**
Instructor(s) :	**Anam Qureshi**

Lab # 03

Outline

- Classes
- Objects
- Structures VS Classes
- Access Specifiers
- Setter/Getter Functions
- Example Programs
- Exercise

Object-Oriented Programming:

OOP languages permit *higher level of abstraction* for solving real-life problems. The traditional procedural language (such as C and Pascal) forces you to think in terms of the structure of the computer (e.g. memory bits and bytes, array, decision, loop) rather than thinking in terms of the problem you are trying to solve. The OOP languages (such as Java, C++, C#) let you think in the problem space, and use software objects to represent and abstract entities of the problem space to solve the problem.

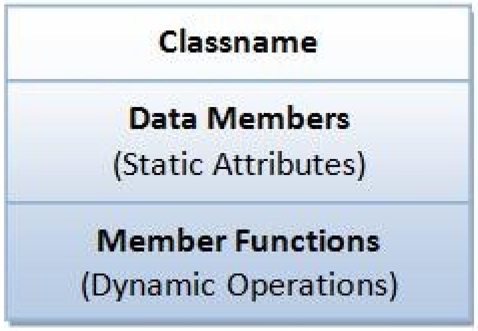
As an example, suppose you wish to write a computer soccer games (which I consider as a complex application). It is quite difficult to model the game in procedural-oriented languages. But using OOP languages, you can easily model the program accordingly to the "real things" appear in the soccer games.

- Player: attributes include name, number, location in the field, and etc; operations include run, jump, kick-the-ball, and etc.
- Ball:
- Reference:
- Field:
- Audience:
- Weather:

Most importantly, some of these classes (such as Ball and Audience) can be reused in another application, e.g., computer basketball game, with little or no modification.

Object-Oriented Programming Basics:

Classes and Instances:



A class is a 3-compartment box encapsulating data and functions

Class: A *class is a definition of objects of the same kind*. In other words, a *class* is a blueprint, template, or prototype that defines and describes the *static attributes* and *dynamic behaviors* common to all objects of the same kind.

Instance: An *instance* is *a realization of a particular item of a class*. In other words, an instance is an *instantiation* of a class. All the instances of a class have similar properties, as described in the class definition. For example, you can define a class called "Student" and create three instances of the class "Student" for "Peter", "Paul" and "Pauline".

The term "*object*" usually refers to *instance*. But it is often used quite loosely, which may refer to a class or an instance.

A Class is a 3-Compartment Box encapsulating Data and Functions:

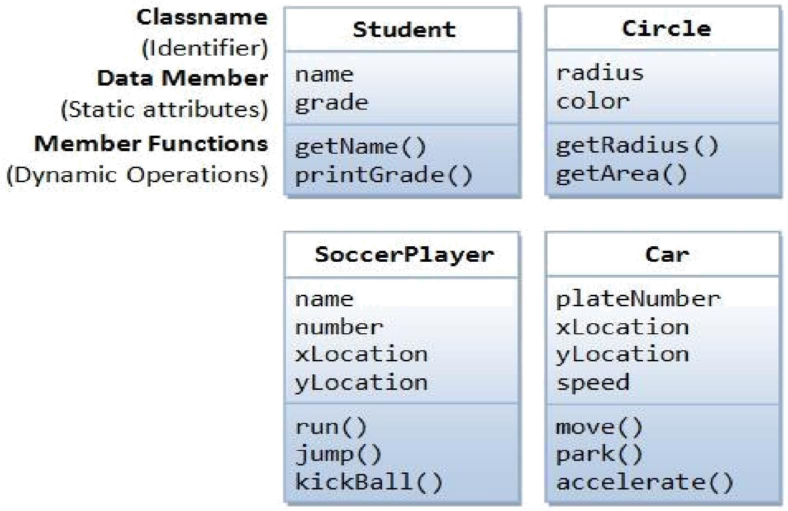
A class can be visualized as a three-compartment box, as illustrated:

1. ***Classname*** (or identifier): identifies the class.
2. ***Data Members*** or ***Variables*** (or *attributes*, *states*, *fields*): contains the *static attributes* of the class.
3. ***Member Functions*** (or *methods*, *behaviors*, *operations*): contains the *dynamic operations* of the class.

In other words, a class encapsulates the static attributes (data) and dynamic behaviors (operations that operate on the data) in a box.

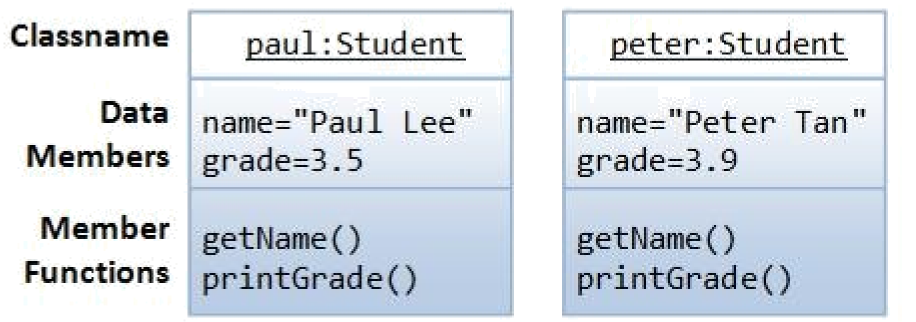
Class Members: The *data members* and *member functions* are collectively called *class members*.

The followings figure shows a few examples of classes:



Examples of classes

The following figure shows two instances of the class Student, identified as "paul" and "peter".



Two instances of the Student class

Class Definition:

In C++, we use the keyword class to define a class. There are two sections in the class declaration: private and public, which will be explained later. For examples,

```
class Circle {        // classname
private:
   double radius;     // Data members (variables)
   string color;
public:
   double getRadius(); // Member functions
   double getArea();
}
```

```
class SoccerPlayer {  // classname
private:
   int number;        // Data members (variables)
   string name;
   int x, y;
public:
   void run();        // Member functions
   void kickBall();
}
```

Creating Instances of a Class:

To create *an instance of a class*, you have to:

1. Declare an instance identifier (name) of a particular class.
2. Invoke a constructor to construct the instance (i.e., allocate storage for the instance and initialize the variables).

For examples, suppose that we have a class called Circle, we can create instances of Circle as follows:

```
/  Construct 3 instances of the class Circle: c1, c2, and
c3 Circle c1(1.2, "red"); // radius, color
Circle c2(3.4);// radius, default color
Circle c3;            // default radius and color
```

Alternatively, you can invoke the constructor explicitly using the following syntax:

```
Circle c1 = Circle(1.2, "red");  // radius, color
Circle c2 = Circle(3.4);        // radius, default color
Circle c3 = Circle();           // default radius and color
```

Dot (.) Operator

To reference a *member of a object* (data member or member function), you must:

1. First identify the instance you are interested in, and then
2. Use the *dot operator* (.) to reference the member, in the form of *instanceName.memberName*.

For example, suppose that we have a class called Circle, with two data members (radius and color) and two functions (getRadius() and getArea()). We have created three instances of the class Circle, namely, c1, c2 and c3. To invoke the function getArea(), you must first identity the instance of interest, says c2, then use the *dot operator*, in the form of c2.getArea(), to invoke the getArea() function of instance c2.

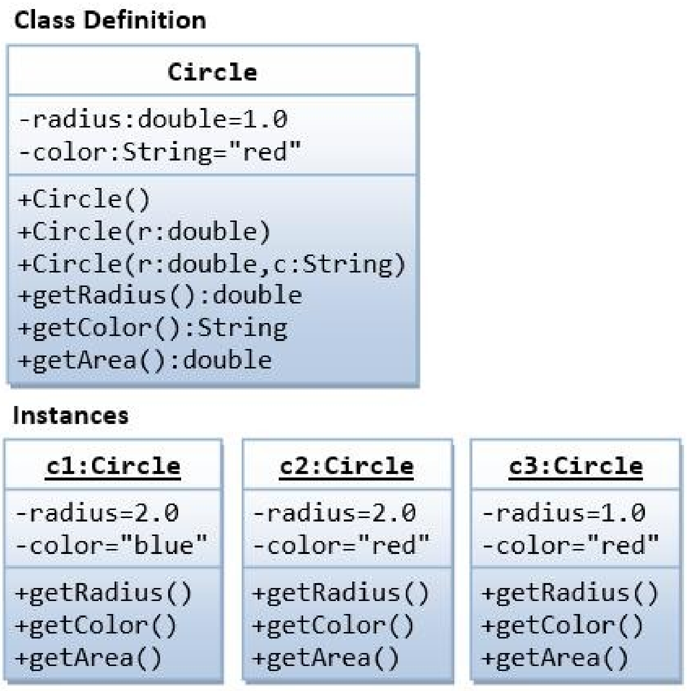
For example,

```
//Declare and construct instances c1 and c2 of the class Circle
Circle c1(1.2, "blue");
Circle c2(3.4, "green");
//Invoke member function via dot operator
cout << c1.getArea() << endl;
cout << c2.getArea() << endl;
//Reference data members via dotoperator
c1.radius = 5.5;
c2.radius = 6.6;
```

Calling getArea() without identifying the instance is meaningless, as the radius is unknown (there could be many instances of Circle - each maintaining its own radius).

Putting them together an OOP Example:

A class called Circle is to be defined as illustrated in the class diagram. It contains two data members: radius (of type double) and color (of type String); and three member functions: getRadius(), getColor(), and getArea(). Three instances of Circles called c1, c2, and c3 shall then be constructed with their respective data members, as shown in the instance diagrams.



"Public" vs. "Private" Access Control Modifiers

An *access control modifier* can be used to control the visibility of a data member or a member function within a class. We begin with the following two access control modifiers:

1. **public**: The member (data or function) is accessible and available to *all* in the system.
2. **private**: The member (data or function) is accessible and available *within this class only*.

For example, in the above Circle definition, the data member radius is declared private. As the result, radius is accessible inside the Circle class, but NOT outside the class. In other words, you cannot use "c1.radius" to refer to c1's radius in main(). Try inserting the statement "cout << c1.radius;" in main() and observe the error message:

Information Hiding and Encapsulation

A class encapsulates the static attributes and the dynamic behaviors into a "3-compartment box". Once a class is defined, you can seal up the "box" and put the "box" on the shelve for others to use and reuse. Anyone can pick up the "box" and use it in their application. This cannot be done in the traditional procedural-oriented language like C, as the static attributes (or variables) are scattered over the entire program and header files. You cannot "cut" out a portion of C program, plug into another program and expect the program to run without extensive changes.

Data member of a class are typically hidden from the outside world, with private access control modifier. Access to the private data members e.g., are provided via public assessor functions, getRadius() and getColor().

This follows the principle of *information hiding*. That is, objects communicate with each other using well-defined interfaces (public functions). Objects are not allowed to know the implementation details of

others. The implementation details are hidden or encapsulated within the class. Information hiding facilitates reuse of the class.
Rule of Thumb: Do not make any data member public, unless you have a good reason.

Getters and Setters

To allow other to *read* the value of a private data member says xxx, you shall provide a *get function* (or *getter* or *accessor function*) called getXxx(). A getter need not expose the data in raw format. It can process the data and limit the view of the data others will see. Getters shall not modify the data member.

To allow other classes to *modify* the value of a private data member says xxx, you shall provide a *set function* (or *setter* or *mutator function*) called setXxx(). A setter could provide data validation (such as range checking), and transform the raw data into the internal representation.

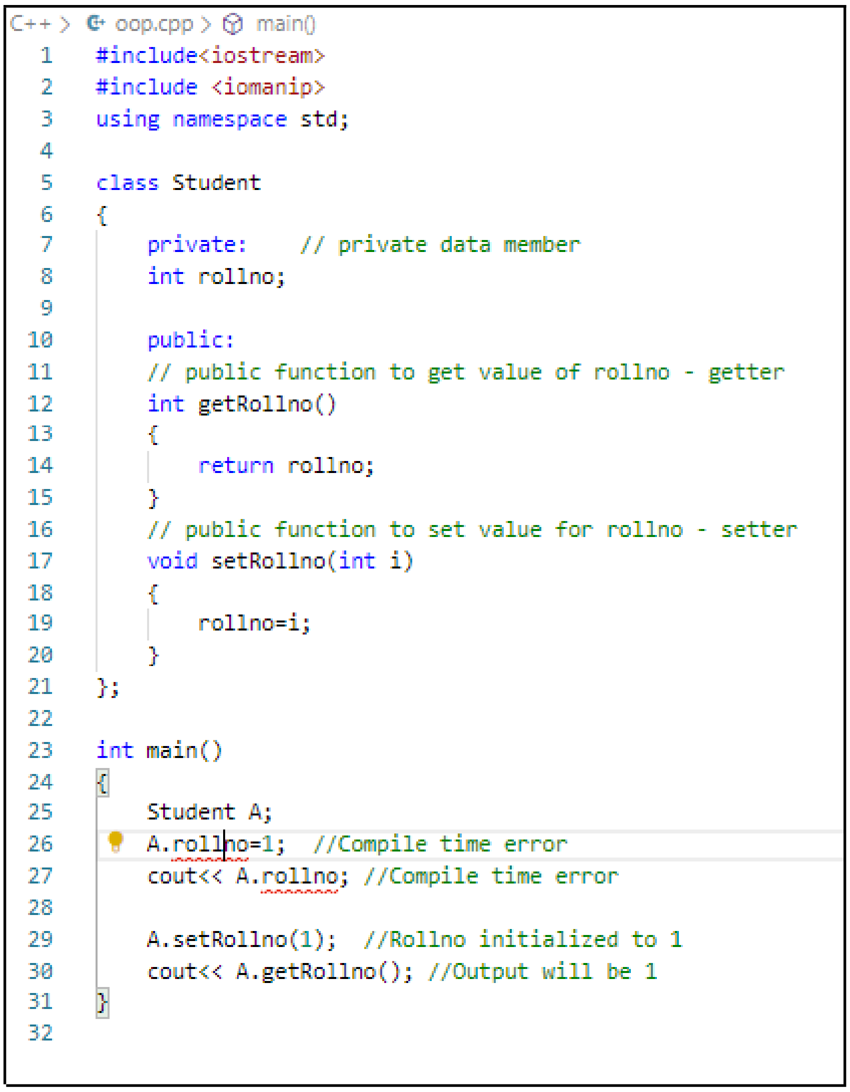
For example, in our Circle class, the data members radius and color are declared private. That is to say, they are only available within the Circle class and not visible outside the Circle class - including main(). You cannot access the private data members radius and color from the main() directly - via says c1.radius or c1.color. The Circle class provides two public accessor functions, namely, getRadius() and getColor(). These functions are declared public. The main() can invoke these public accessor functions to retrieve the radius and color of a Circle object, via says c1.getRadius() and c1.getColor().

There is no way you can change the radius or color of a Circle object, after it is constructed in main(). You cannot issue statements such as c1.radius = 5.0 to change the radius of instance c1, as radius is declared as private in the Circle class and is not visible to other including main().
If the designer of the Circle class permits the change the radius and color after a Circle object is constructed, he has to provide the appropriate setter, e.g.,

```
// Setter for color
void setColor(string c) {
  color = c;
}

// Setter for radius
void setRadius(double r) {
  radius = r;
}
```

With proper implementation of *information hiding*, the designer of a class has full control of what the user of the class can and cannot do.

Example of accessing Private Data Members Using Setter and Getter Functions



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

class Student
{
    private:    // private data member
    int rollno;

    public:
    // public function to get value of rollno - getter
    int getRollno()
    {
        return rollno;
    }
    // public function to set value for rollno - setter
    void setRollno(int i)
    {
        rollno=i;
    }
};

int main()
{
    Student A;
    A.rollno=1;  //Compile time error
    cout<< A.rollno; //Compile time error

    A.setRollno(1);  //Rollno initialized to 1
    cout<< A.getRollno(); //Output will be 1
}
```

Member Functions in Classes

There are 2 ways to define a member function:

- Inside class definition
- Outside class definition

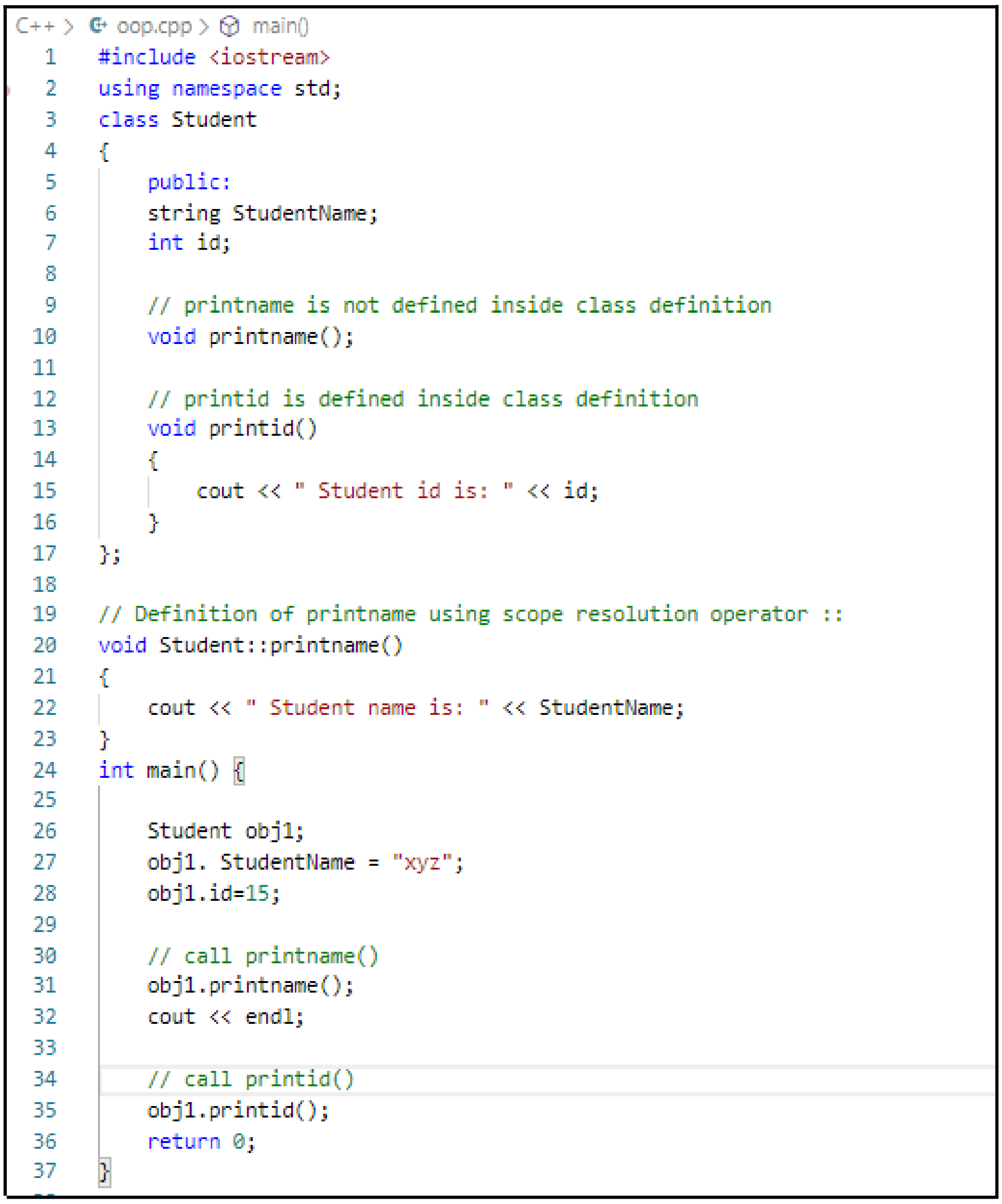
1. Inside class definition

With an inline function, the compiler tries to expand the code in the body of the function in place of a call to the function.

Note that all the member functions defined inside the class definition are by default **inline**, but you can also make any non-class function inline by using keyword inline with them. Inline functions are actual functions, which are copied everywhere during compilation, like pre-processor macro, so the overhead of function calling is reduced.

2. Outside class definition

To define a member function outside the class definition we have to use the scope resolution :: operator along with class name and function name.



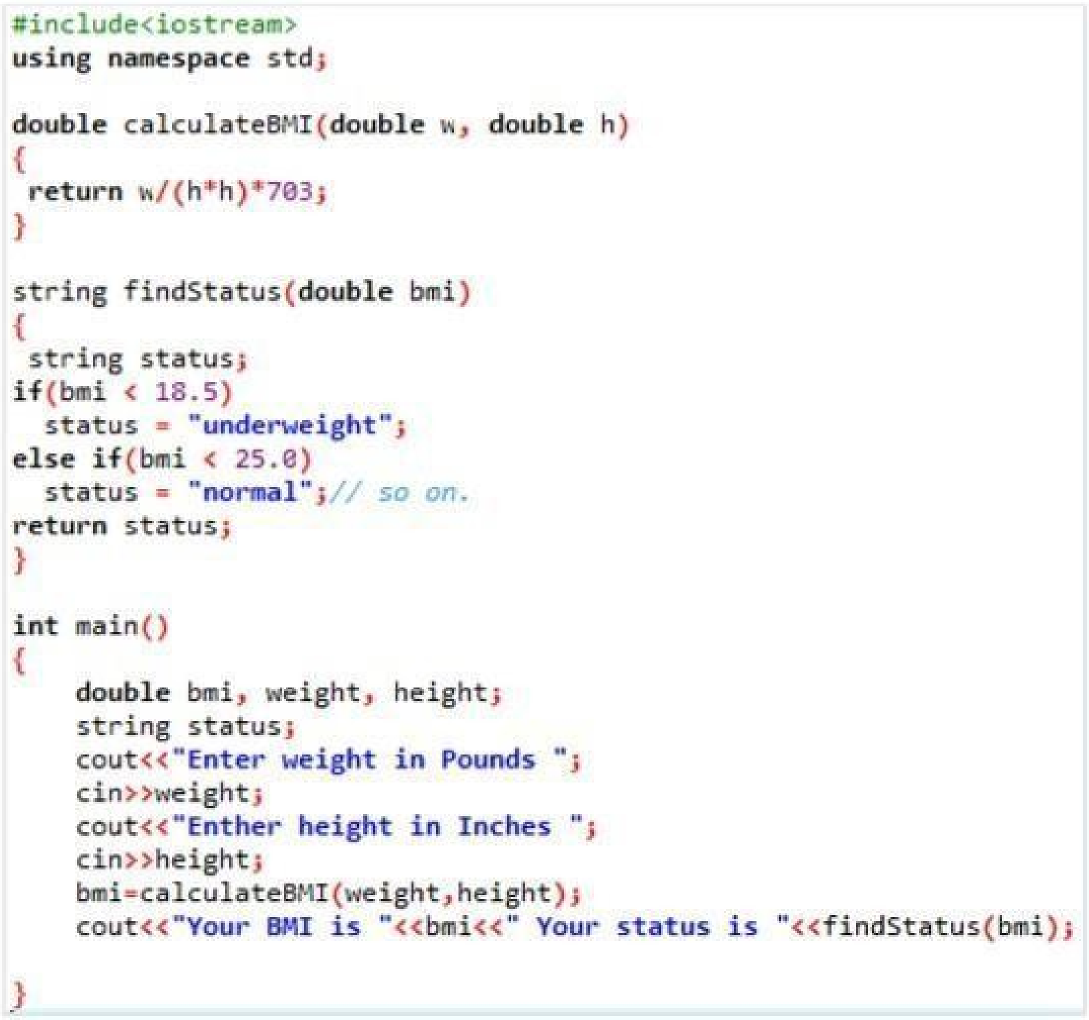
```cpp
#include <iostream>
using namespace std;
class Student
{
    public:
    string StudentName;
    int id;

    // printname is not defined inside class definition
    void printname();

    // printid is defined inside class definition
    void printid()
    {
        cout << " Student id is: " << id;
    }
};

// Definition of printname using scope resolution operator ::
void Student::printname()
{
    cout << " Student name is: " << StudentName;
}
int main() {

    Student obj1;
    obj1. StudentName = "xyz";
    obj1.id=15;

    // call printname()
    obj1.printname();
    cout << endl;

    // call printid()
    obj1.printid();
    return 0;
}
```

Structures VS Classes

By default, all structure fields are public, or available to functions (like the main() function) that are outside the structure. Conversely, all class fields are private. That means they are not available for use outside the class. When you create a class, you can declare some fields to be private and some to be public. For example, in the real world, you might want your name to be public knowledge but your Social Security number, salary, or age to be private.

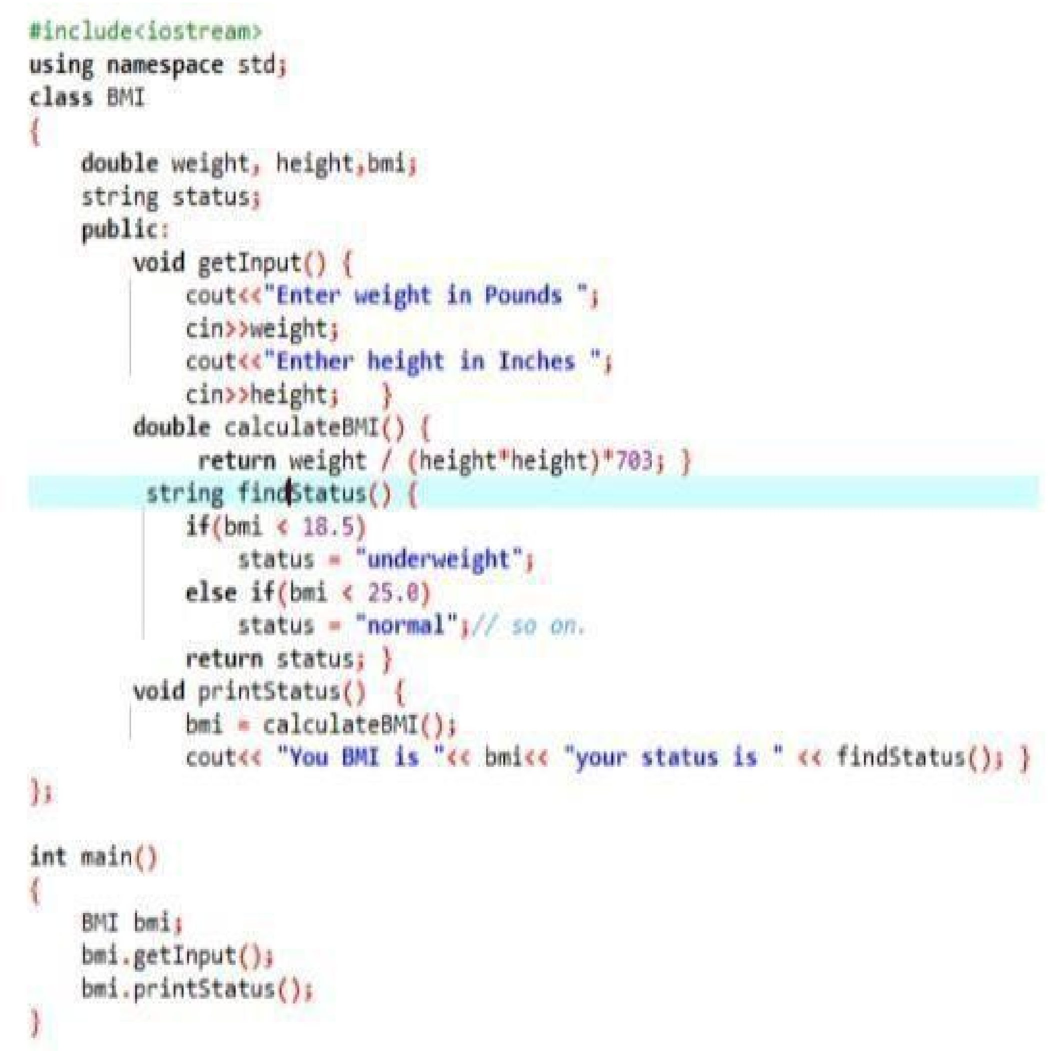
Example of Procedural and Object Oriented Approach

Procedural Approach



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

double calculateBMI(double w, double h)
{
 return w/(h*h)*703;
}

string findStatus(double bmi)
{
 string status;
if(bmi < 18.5)
  status = "underweight";
else if(bmi < 25.0)
  status = "normal";// so on.
return status;
}

int main()
{
    double bmi, weight, height;
    string status;
    cout<<"Enter weight in Pounds ";
    cin>>weight;
    cout<<"Enther height in Inches ";
    cin>>height;
    bmi=calculateBMI(weight,height);
    cout<<"Your BMI is "<<bmi<<" Your status is "<<findStatus(bmi);

}
```

Object Oriented Approach



```
#include<iostream>
using namespace std;
class BMI
{
    double weight, height,bmi;
    string status;
    public:
        void getInput() {
            cout<<"Enter weight in Pounds ";
            cin>>weight;
            cout<<"Enther height in Inches ";
            cin>>height;    }
        double calculateBMI() {
             return weight / (height*height)*703; }
         string findStatus() {
            if(bmi < 18.5)
                status = "underweight";
            else if(bmi < 25.0)
                status = "normal";// so on.
            return status; }
        void printStatus()  {
            bmi = calculateBMI();
            cout<< "You BMI is "<< bmi<< "your status is " << findStatus(); }
};

int main()
{
    BMI bmi;
    bmi.getInput();
    bmi.printStatus();
}
```

Exercise Lab 03

Question # 01: Define a class to represent a student. Include the following members
Data Members
Name of the student
ID number
Department name
Six Subjects
Marks obtained
Total marks
Member Function
Write setter and getter property to set all attributes
To calculate percentage and grade
To display name and ID number, percentage and grade

Question # 02: A bank wants a simple application module to manage the accounts of its customers. For every new customer, the app must let us fill in the details including his Name, Age, NIC#, Address, Opening Balance, Current Balance, Contact# & PIN. These details may be modified later except for the PIN. At any given time, the customer can check his balance.
Also, tax must be calculated (Tax is 0.15% of the current balance for customers aged 60 or above and 0.25% for all other customers).

Question # 03: Create a class called Bank Employee that includes three pieces of information as data members—a first name (type char* (DMA)), a last name (type string) and a monthly salary (type int). Your class should have a setter function that initializes the three data members. Provide a getter function for each data member. If the monthly salary is not positive, set it to 0. Write a test program that demonstrates class Employee's capabilities. Create five Employee objects and display each object's yearly salary. Then give each Employee a 20 percent raise and display each Employee's yearly salary again. Identify and add any other related functions to achieve the said goal.

Question # 04: Define a class for a type called **CounterType**. An object of this type is used to count things, so it records a count that is a nonnegative whole number. Include a mutator function that sets the counter to a count given as an argument. Include member functions to increase the count by one and to decrease the count by one. Also, include a member function that returns the current count value and one that outputs the count. Embed your class definition in a test program.

Question # 05: You are a programmer for the **Standard Charted Bank** assigned to develop a class that models the basic workings of a bank **account**. The class should perform the following tasks:
- Save the account balance.
- Save the number of transactions performed on the account.
- Allow deposits to be made to the account.
- Allow with drawls to be taken from the account.
- Calculate interest for the period.
- Report the current account balance at any time.
- Report the current number of transactions at any time.

Menu

1. Display the account balance
2. Display the number of transactions
3. Display interest earned for this period
4. Make a deposit
5. Make a withdrawal
6. Add interest for this period
7. Exit the program

Practice Problems

Task 01:
Create a 'DISTANCE' class with :
- feet and inches as data members
- member function to input distance
- member function to output distance
- member function to add two distance objects

Write a main function to create objects of DISTANCE class. Input two distances and output the sum.

Task 02:
Write a program in which a class named student has member variables name, roll_no, semester and section. Create 4 objects of this class to store data of 4 different students, now display data of only those students who belong to section A.

Task 03:
Create a class called distance that has a separate integer member data for feet and inches. One constructor should initialize this data to zero and another should initialize it to fixed values. A member function should display it in feet inches format.

Task 04:
Create a class called Employee that includes three pieces of information as data members—a first name (type char*), a last name (type string) and a monthly salary (type int). Your class should have a setter function that initializes the three data members. Provide a getter function for each data member. If the monthly salary is not positive, set it to 0. Write a test program that demonstrates class Employee's capabilities. Create two Employee objects and display each object's yearly salary. Then give each Employee a 10 percent raise and display each Employee's yearly salary again. Identify and add any other related functions to achieve the said goal.

Task 05:
Create a class called Invoice that a hardware store might use to represent an invoice for an item sold at the store. An Invoice should include four data members—a part number (type string), a part description (type string), a quantity of the item being purchased (type int) and a price per item (type float). Your class should have a functions that initializes the four data members. Provide a get function for each data member. In addition, provide a member function named getInvoiceAmount that calculates the invoice amount (i.e., multiplies the quantity by the price per item), then returns the amount as a float value. If the quantity is not positive, it should be set to 0. If the price per item is not positive, it should be set to 0. Write a test program that demonstrates class Invoice's capabilities.

Task 06:
Create a class Sales with 3 private variables SaleID of type integer, ItemName of type string , and Quantity of type integer.
Part (a) Use a default constructor to initialize all variables with any values.
Part (b) Use a constructor to take user input in all variables to display data.
Part (c) Use a parameterized constructor to initialize the variables with values of your choice.

Task 07:
Write a program in which a class named student has member variables name, roll_no, semester and section. Use a parameterized constructor to initialize the variables with your name, roll no, semester and section. Print all data calling some public method.

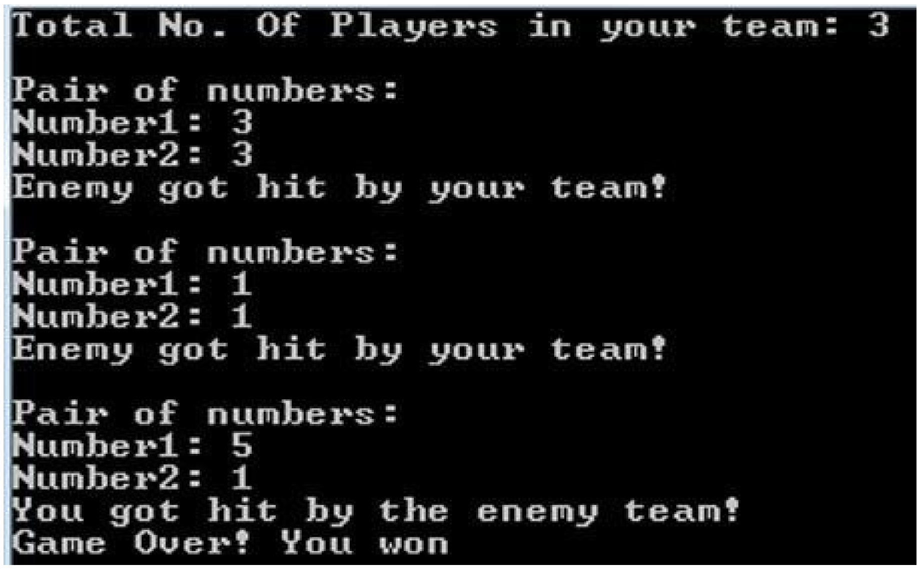
Task 08:
Write C++ code to represent a hitting game. The details are as follows:
This game is being played between two teams (i.e. your team and the enemy team).
The total number of players in your team is randomly generated and stored accordingly.
The function generates a pair of numbers and matches each pair. If the numbers get matched, the following message is displayed:

"Enemy got hit by your team!"

Otherwise, the following message is displayed:

"You got hit by the enemy team!"

The number of hits should be equal to the number of players in your team.
The program should tell the final result of your team by counting the hits of both the teams.
Consider the following sample output:



```
Total No. Of Players in your team: 3

Pair of numbers:
Number1: 3
Number2: 3
Enemy got hit by your team!

Pair of numbers:
Number1: 1
Number2: 1
Enemy got hit by your team!

Pair of numbers:
Number1: 5
Number2: 1
You got hit by the enemy team!
Game Over! You won
```

Task 09:
Write a class named Vehicle that can represent both the Rickshaw and Bike on the basis of number of wheels it has. Each vehicle has the following details

- year. An int that holds the vehicle's model year.
- manufacturer. A string that holds the manufacturer name of that vehicle.
- speed. An int that holds the vehicle's current speed.

In addition, the class should have the following member functions.

- accelerate. The accelerate function should add 5 to the speed member variable each time it is called.
- brake. The brake function should subtract 5 from the speed member variable each time it is called.

Demonstrate the class in a program that creates a Vehicle object for a Rickshaw and for a Bike both, and then calls the accelerate function five times. After each call to the accelerate function, get the current speed of the car and display it. Then, call the brake function two times. After each call to the brake function, get the current speed of the car and display it.

Task 10:
Create a class Rectangle with attributes length and width, each of which defaults to 1. Provide member functions that calculate the perimeter and the area of the rectangle. Also, provide set and get functions for the length and width attributes. The set functions should verify that length and width are each floating-point numbers larger than 0.0 and less than 20.0.