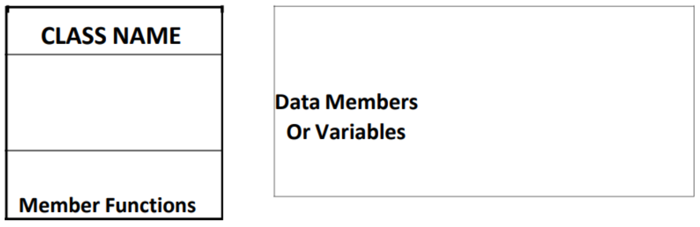
Object Oriented Programming Lab 03

Course: Object Oriented Programming (CL1004) **Semester**: Spring 2025
Instructor: Shafique Rehman

Note:
- Maintain discipline during the lab.
- Listen and follow the instructions as they are given.
- Just raise hand if you have any problem.
- Completing all tasks of each lab is compulsory.
- Get your lab checked at the end of the session.

Classes in C++

A class is a programmer-defined data type that describes what an object of the class will look like when it is created. It consists of a set of variables and a set of functions. We can think of class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. based on these descriptions we build the house. House is the object.
As, many houses can be made from the same description, we can create many objects from a class.
Classes are created using the keyword class. A class declaration defines a new type that links code and data. This new type is then used to declare objects of that class. A Class is a user defined data-type which has data members and member functions. Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.
A class member can be defined as public, private or protected. By default, members would be assumed as private.
In the UML, a class icon can be subdivided into compartments. The top compartment is for the name of the class, the second is for the variables of the class, and the third is for the methods of the class.



Class Name

By convention, the name of a user-defined class begins with a capital letter and, for readability, each subsequent word in the class name begins with a capital letter.

Data Members

Consider the attributes of some real-world objects:

RADIO – station frequency, volume level.
CAR – speedometer readings, amount of gas in its tank and what gear it is in.

These attributes form the data in our program. The values that these attributes take (the blue color of the petals, for example) form the state of the object.

Member Functions

Consider the operations of some real-world objects:

RADIO – setting its station and volume (invoked by the person adjusting the radio's controls)
CAR – accelerating (invoked by the driver), decelerating, turning and shifting gears. These operations form the functions in program. Member functions define the class's behaviors.

<u>Objects in C++</u>

In C++, when we define a variable of a class, we call it instantiating the class. The variable itself is called an instance of the class. A variable of a class type is also called an object. Instantiating a variable allocates memory for the object. Below is the syntax to define an object in C++:

className objectVariableName

Following is an example of an object created from a class named as radio:

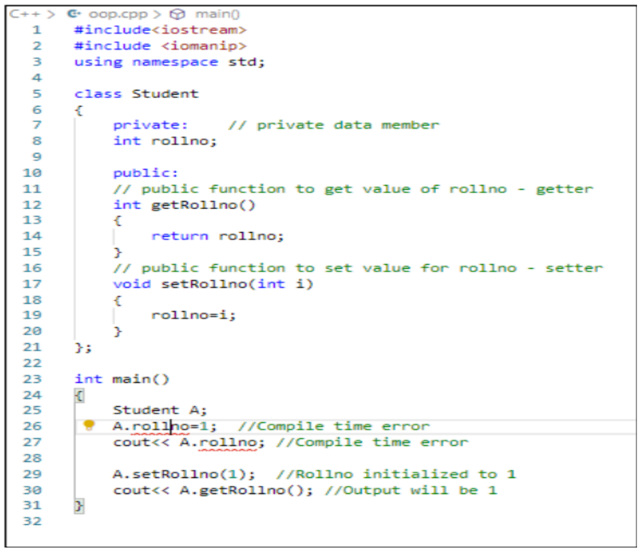
Radio r;

Accessing Public Data Members

The public data members of objects of a class can be accessed using the direct member access operator (.). However, the private data members are not allowed to be accessed directly by the object. Accessing a data member depends solely on the access control of that data member.

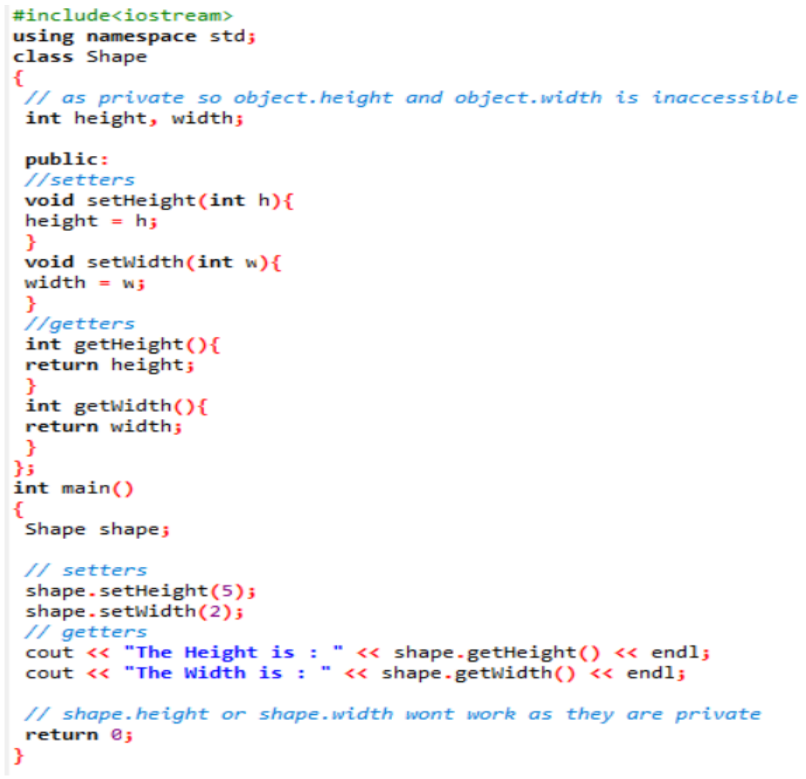
Accessing Private Data Members

To access, use and initialize the private data member you need to create getter and setter functions, to get and set the value of the data member. The setter function will set the value passed as argument to the private data member, and the getter function will return the value of the private data member to be used. Both getter and setter function must be defined public.



```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
}
```

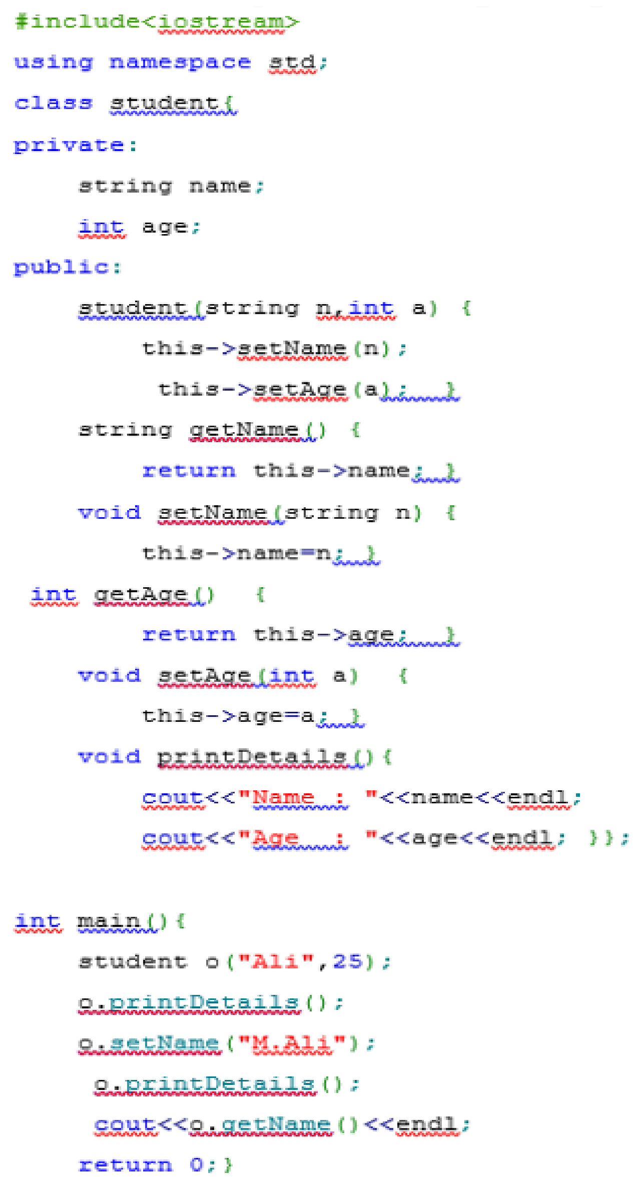
Consider another example below:



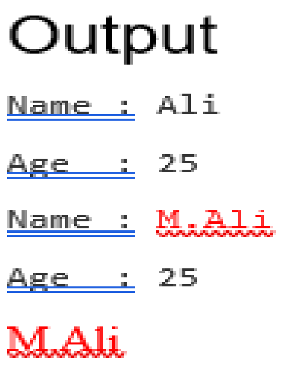
```cpp
#include<iostream>
using namespace std;
class Shape
{
 // as private so object.height and object.width is inaccessible
 int height, width;

 public:
 //setters
 void setHeight(int h){
 height = h;
 }
 void setWidth(int w){
 width = w;
 }
 //getters
 int getHeight(){
 return height;
 }
 int getWidth(){
 return width;
 }
};
int main()
{
 Shape shape;

 // setters
 shape.setHeight(5);
 shape.setWidth(2);
 // getters
 cout << "The Height is : " << shape.getHeight() << endl;
 cout << "The Width is : " << shape.getWidth() << endl;

 // shape.height or shape.width wont work as they are private
 return 0;
}
```

Getter & Setter , this Keyword in C++ Programming

The getter function is used to retrieve the variable value and the setter function is used to set the variable value. They this is a keyword that refers to the current instance of the class. They are getters and setters the standard way to provide access to data in Java classes. Setters and Getters allow for an object to contain private variables which can be accessed and changed with restrictions.



```
#include<iostream>
using namespace std;
class student{
private:
    string name;
    int age;
public:
    student(string n,int a) {
        this->setName(n);
         this->setAge(a);   }
    string getName() {
        return this->name; }
    void setName(string n) {
        this->name=n; }
  int getAge()   {
        return this->age;  }
    void setAge(int a)   {
        this->age=a; }
    void printDetails(){
        cout<<"Name  :  "<<name<<endl;
        cout<<"Age   :  "<<age<<endl;  }};

int main(){
    student o("Ali",25);
    o.printDetails();
    o.setName("M.Ali");
     o.printDetails();
     cout<<o.getName()<<endl;
    return 0;}
```



Output

```
Name : Ali
Age  : 25
Name : M.Ali
Age  : 25
M.Ali
```

Member Functions in Classes

There are 2 ways to define a member function:

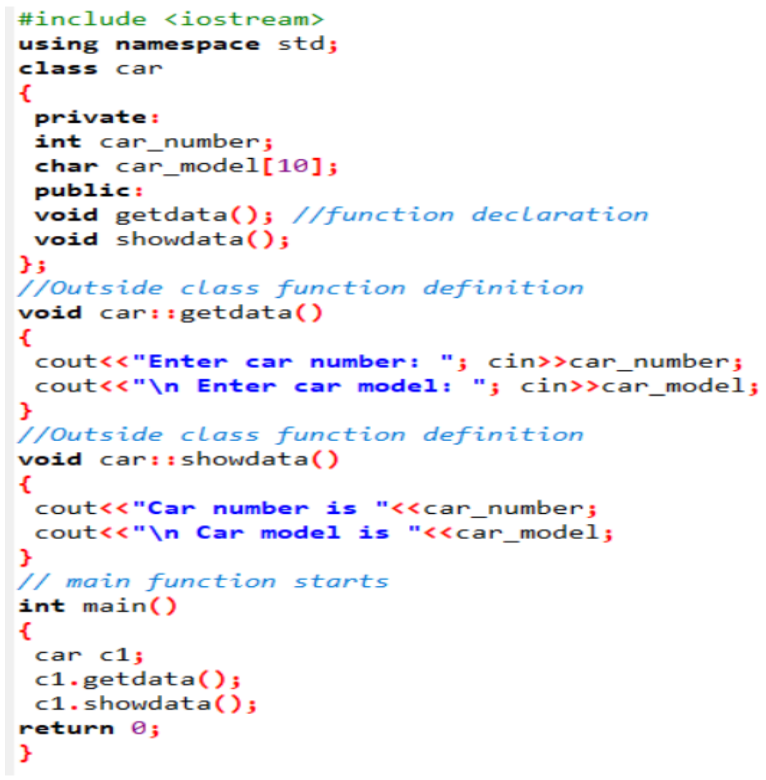
1. Inside class definition.
2. Outside class definition.

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. This type of definition is already shown in the above examples.

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.

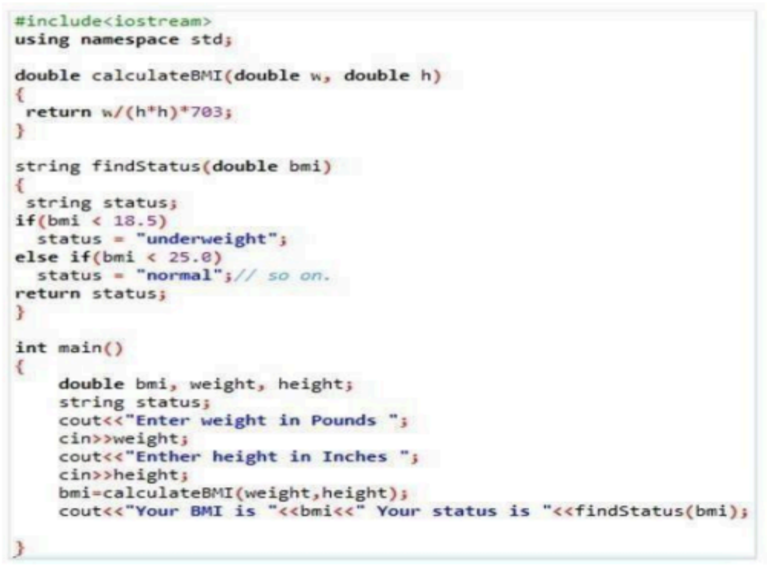


```cpp
#include <iostream>
using namespace std;
class car
{
 private:
 int car_number;
 char car_model[10];
 public:
 void getdata(); //function declaration
 void showdata();
};
//Outside class function definition
void car::getdata()
{
 cout<<"Enter car number: "; cin>>car_number;
 cout<<"\n Enter car model: "; cin>>car_model;
}
//Outside class function definition
void car::showdata()
{
 cout<<"Car number is "<<car_number;
 cout<<"\n Car model is "<<car_model;
}
// main function starts
int main()
{
 car c1;
 c1.getdata();
 c1.showdata();
return 0;
}
```

Struct VS Classes

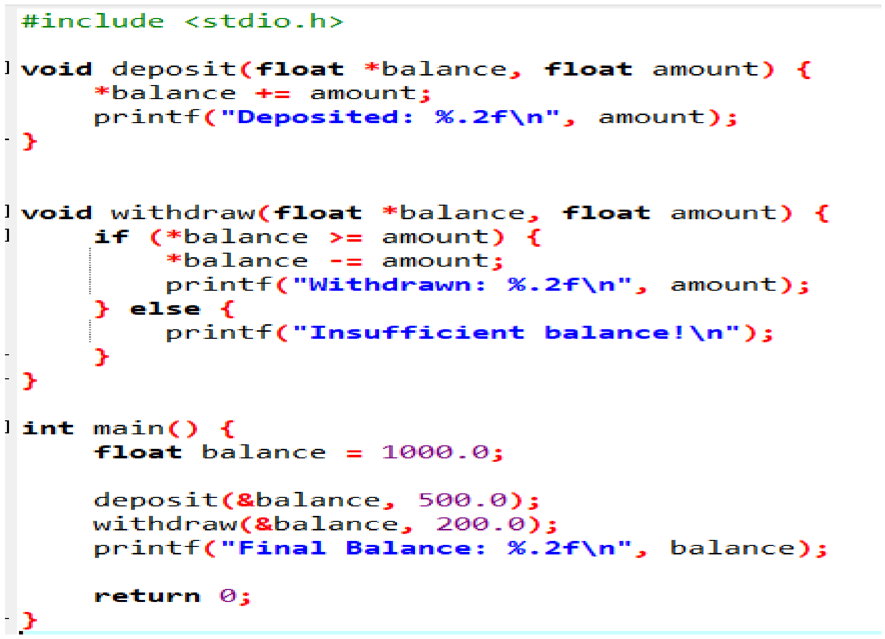
By default, all structure fields are public, or available to functions (like the main() function) that are outside the structure. Conversely, all class fields are private. That means they are not available for use outside the class. When you create a class, you can declare some fields to be private and 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.

Transformation from Procedural to Object Oriented Programming



```cpp
#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);

}
```

Procedural Approach



```c
#include <stdio.h>

void deposit(float *balance, float amount) {
    *balance += amount;
    printf("Deposited: %.2f\n", amount);
}


void withdraw(float *balance, float amount) {
    if (*balance >= amount) {
        *balance -= amount;
        printf("Withdrawn: %.2f\n", amount);
    } else {
        printf("Insufficient balance!\n");
    }
}

int main() {
    float balance = 1000.0;

    deposit(&balance, 500.0);
    withdraw(&balance, 200.0);
    printf("Final Balance: %.2f\n", balance);

    return 0;
}
```

Object Oriented Programming Approach

```
#include<iostream>
using namespace std;
class Account
{
private:
        double balance; // Account balance
public: //Public interface:
        string name; // Account holder long accountNumber;
        // Account number void setDetails(double bal)
        {
                balance = bal;
        }
        double getDetails()
        {
                return balance;
        }
        void displayDetails()
        {
                cout<<"Details are: "<<endl;
           cout<<"Account Holder:
           "<<name<<endl;
                cout<<"Account Number:
                "<<
           accountNumber <<endl; cout<<"Account
            Balance: "<<getDetails()<<endl;
        }
 };
```

Set and get functions to manipulate

Private data member

```
int main()
{ double accBal;
Account
currentAccount;
currentAccount.getDetails();

cout<<"Please enter the details"<<endl;
cout<<"Enter Name:"<<endl; getline(cin,
currentAccount.name); cout<<"Enter
Account Number:"<<endl;
cin>>currentAccount.accountNumber;

cout<<"Enter Account
Balance:"<<endl; cin>>accBal;
currentAccount.setDetails(accBal);
```

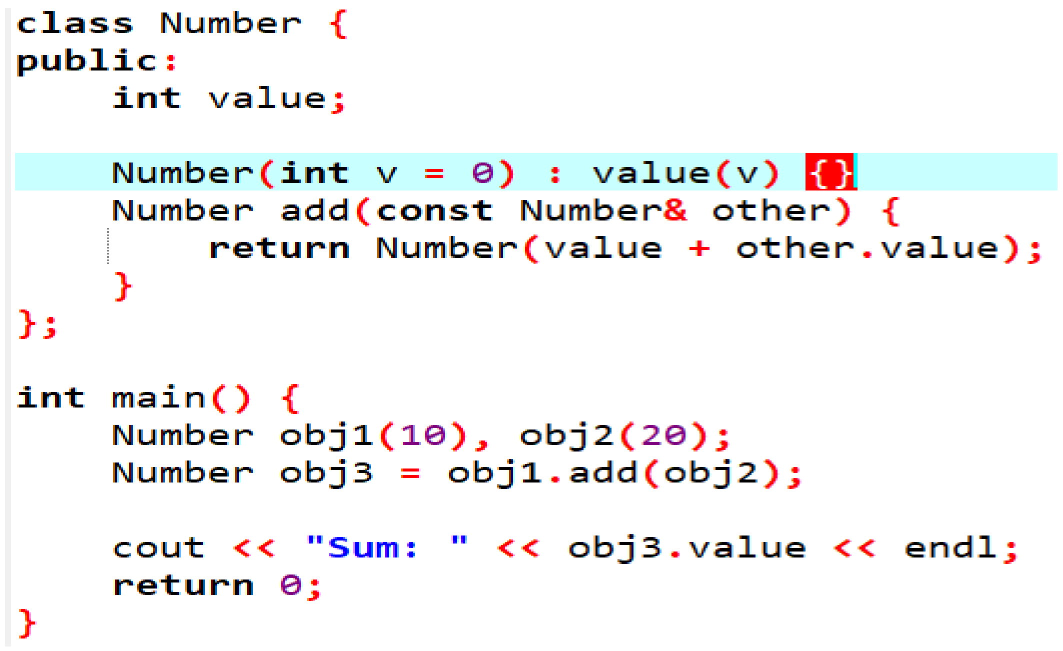
Publically available data:
Assigning values from

Private data:

```
cout<<endl;
currentAccount.displayDetails();
return 0;
        }
```

Accessing private data using member function

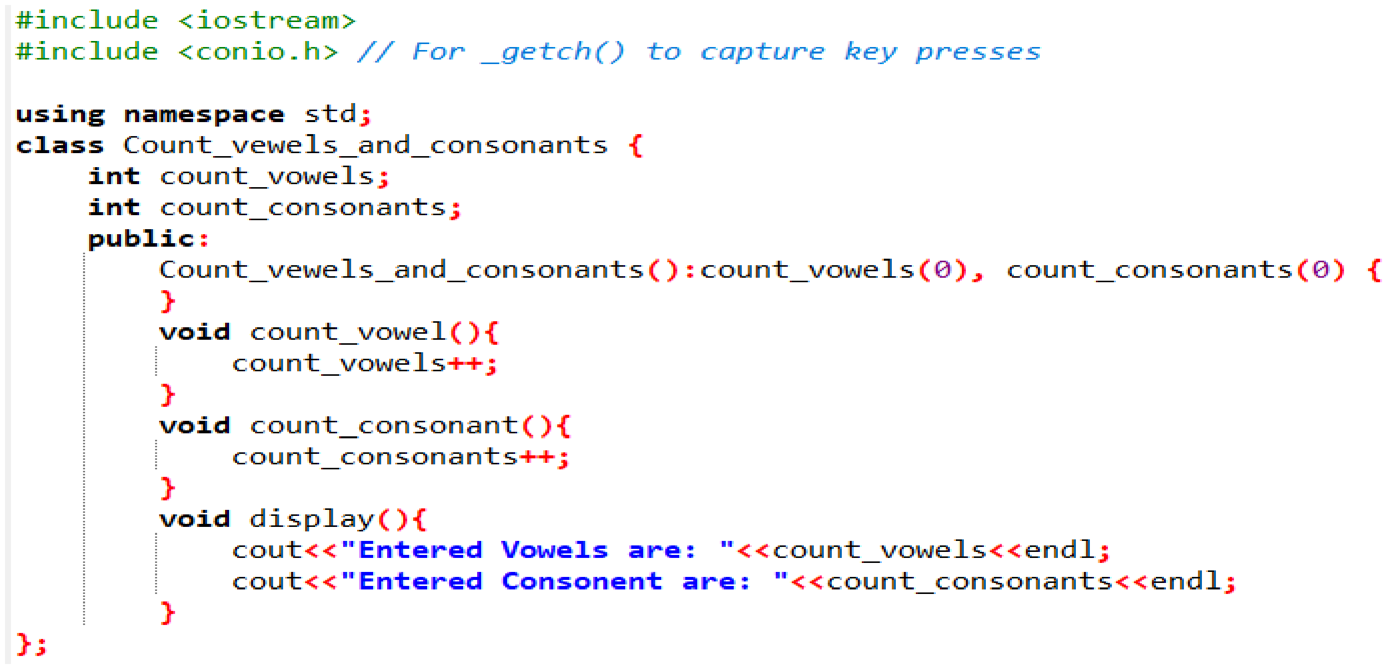
<u>Example 1: Adding two Objects of class and returning result as Object</u>



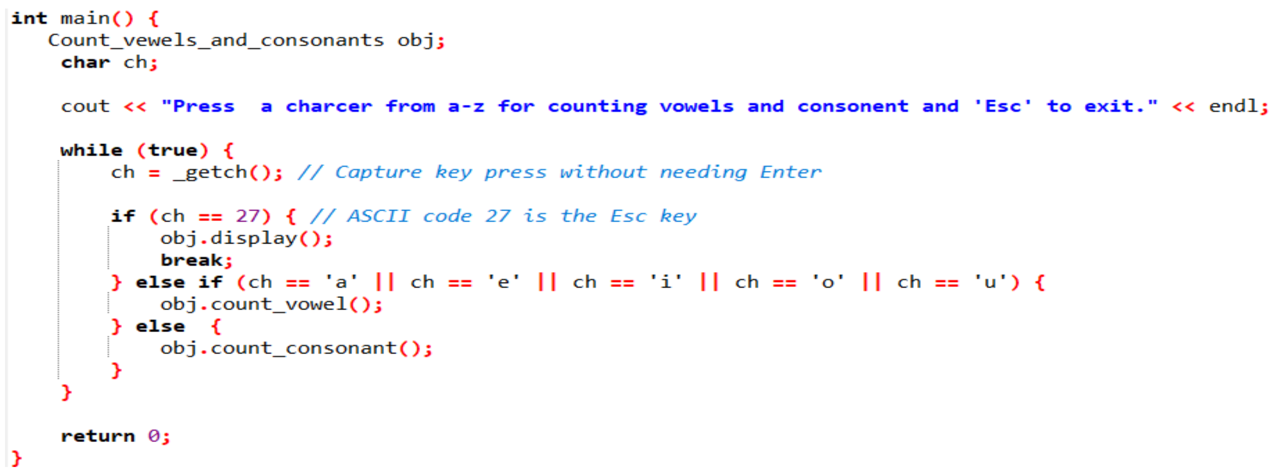
```
class Number {
public:
    int value;

    Number(int v = 0) : value(v) {}
    Number add(const Number& other) {
        return Number(value + other.value);
    }
};

int main() {
    Number obj1(10), obj2(20);
    Number obj3 = obj1.add(obj2);

    cout << "Sum: " << obj3.value << endl;
    return 0;
}
```

Output: Sum: 30

Usage of <conio.h> library and _getch() function
Example: counting Vowels and consonants



```
#include <iostream>
#include <conio.h> // For _getch() to capture key presses

using namespace std;
class Count_vewels_and_consonants {
    int count_vowels;
    int count_consonants;
    public:
        Count_vewels_and_consonants():count_vowels(0), count_consonants(0) {
        }
        void count_vowel(){
            count_vowels++;
        }
        void count_consonant(){
            count_consonants++;
        }
        void display(){
            cout<<"Entered Vowels are: "<<count_vowels<<endl;
            cout<<"Entered Consonent are: "<<count_consonants<<endl;
        }
};
```



```cpp
int main() {
   Count_vewels_and_consonants obj;
    char ch;

    cout << "Press  a charcer from a-z for counting vowels and consonent and 'Esc' to exit." << endl;

    while (true) {
        ch = _getch(); // Capture key press without needing Enter

        if (ch == 27) { // ASCII code 27 is the Esc key
            obj.display();
            break;
        } else if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
            obj.count_vowel();
        } else  {
            obj.count_consonant();
        }
    }

    return 0;
}
```

Header Files

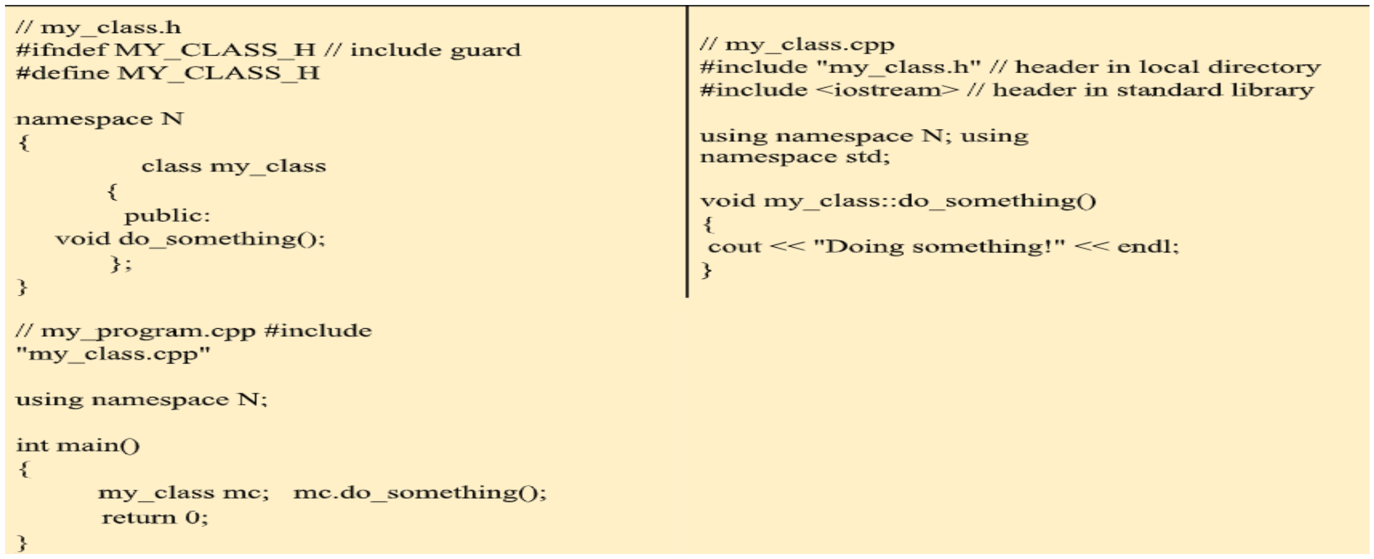
Header files are used for declaration. In OOP you should use header files to declare classes and functions. It will make your program look cleaner and more professional.
Header file for a class will include:

- Include guards.
- Class definition:
  - Member variables.
  - Function declaration (only prototype).

Implementation File will include:

- Include directive for "header.h".
- Necessary include directives.
- Function definitions for all the functions of the class.

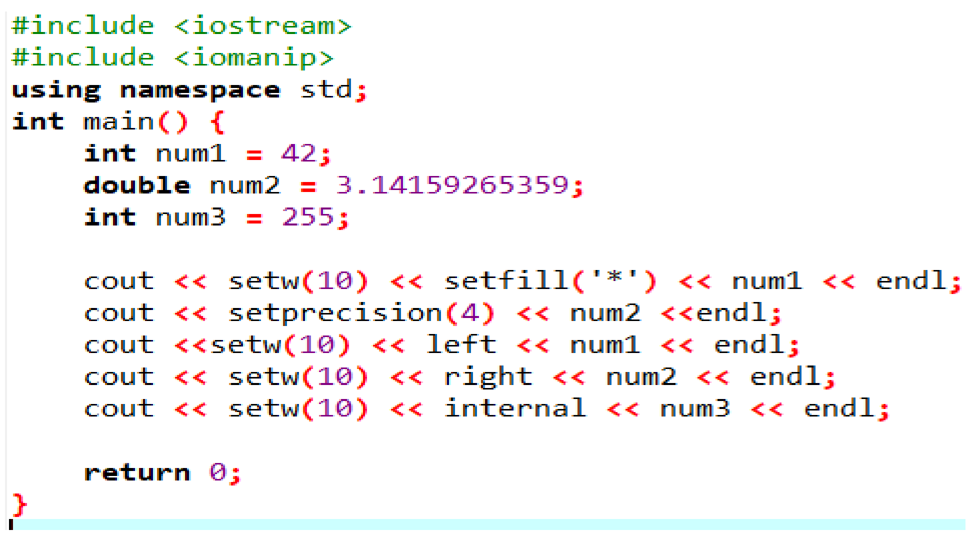
Consider the below example:



```cpp
// my_class.h
#ifndef MY_CLASS_H // include guard
#define MY_CLASS_H

namespace N
{
            class my_class
        {
          public:
    void do_something();
         };
}

// my_program.cpp #include
"my_class.cpp"

using namespace N;

int main()
{
        my_class mc;   mc.do_something();
        return 0;
}
```

```cpp
// my_class.cpp
#include "my_class.h" // header in local directory
#include <iostream> // header in standard library

using namespace N; using
namespace std;

void my_class::do_something()
{
 cout << "Doing something!" << endl;
}
```

iomanip library functions and explanations



```
#include <iostream>
#include <iomanip>
using namespace std;
int main() {
    int num1 = 42;
    double num2 = 3.14159265359;
    int num3 = 255;

    cout << setw(10) << setfill('*') << num1 << endl;
    cout << setprecision(4) << num2 <<endl;
    cout <<setw(10) << left << num1 << endl;
    cout << setw(10) << right << num2 << endl;
    cout << setw(10) << internal << num3 << endl;

    return 0;
}
```

Explanation:

- For num1, I use setw(10) and setfill('*') to ensure the number takes up a width of 10 characters, padding with * if necessary.
- For num2, I apply setprecision(4) to limit the number of significant digits to 4.
- For num1 again, I use sleft to left-align the output within the specified width.
- For num2 again, I use right to right-align the output.
- For num3, I use internal to ensure any padding occurs on the left side of the number (typically used for numbers with signs, though not applicable here).

Exercises Lab 3:

Note: Inputs must be handled via the main () function

Task 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 (should they be const?) 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. Make appropriate member functions const.

Task 2:

Imagine a tollbooth at a bridge. Cars passing by the booth are expected to pay a 50 cent toll. Mostly they do, but sometimes a car goes by without paying. The tollbooth keeps track of the number of cars that have gone by, and of the total amount of money collected.
Model this tollbooth with a class called tollBooth. The two data items are a type unsigned int to hold the total number of cars, and a type double to hold the total amount of money collected. A constructor initializes both of these to 0. A member function called payingCar() increments the car total and adds 0.50 to the cash total. Another function,
called nopayCar(), increments the car total but adds nothing to the cash total. Finally, a member function called display() displays the two totals. Make appropriate member functions const.
Include a program to test this class. This program should allow the user to push one key to count a paying car, and another to count a nonpaying car. Pushing the Esc key should cause the program to print out the total cars and total cash and then exit.
Note: _getch() (from <conio.h>) reads a key press instantly without requiring Enter.
The loop keeps running until the user presses Esc.the ASCII code 27 is the Esc key
Task 3:
Create a class that includes a data member that holds a "serial number" for each object created from the class. That is, the first object created will be numbered 1, the second 2, and so on. To do this, you'll need another data member that records a count of how many objects have been created so far. (This member should apply to the class as a whole; not to individual objects. What keyword specifies this?) Then, as each object is created, its constructor can examine this count member variable to determine the appropriate serial number for the new object.
Add a member function that permits an object to report its own serial number. Then write a main() program that creates three objects and queries each one about its serial number. They should respond I am object number 2, and so on.

Task 4:
In ocean navigation, locations are measured in degrees and minutes of latitude and longitude. Thus if you're lying off the mouth of Papeete Harbor in Tahiti, your location is 149 degrees 34.8 minutes west longitude, and 17 degrees 31.5 minutes south latitude. This is written as 149°34.8' W, 17°31.5' S. There are 60 minutes in a degree. (An older system also divided a minute into 60 seconds, but the modern approach is to use decimal minutes instead.) Longitude is measured from 0 to 180 degrees, east or west from Greenwich, England, to the international dateline in the Pacific. Latitude is measured from 0 to 90 degrees, north or south from the equator to the poles.
Create a class angle that includes three member variables: an int for degrees, a float for minutes, and a char for the direction letter (N, S, E, or W). This class can hold either a latitude variable or a longitude variable. Write one member function to obtain an angle value (in degrees and minutes) and a direction from the user, and a second to display the angle value in 179°59.9' E format. Also write a three-argument constructor.
Write a main() program that displays an angle initialized with the constructor, and then, within a loop, allows the user to input any angle value, and then displays the value. You can use the hex character constant '\xF8', which usually prints a degree (°) symbol.
Note:

- **fixed**: This manipulator is used to display floating-point numbers in fixed-point notation (i.e., a set number of digits after the decimal point).
- **setprecision(n)**: This manipulator sets the number of digits to be displayed after the decimal point. For example, setprecision(1) ensures that only one digit will be displayed after the decimal point, as seen in your angle minutes.

Task 5:

Create a class called calendar. The calendar should have 12 arrays for each month of the year, and a variable that stores information about the current year. The user is allowed to store their tasks to do against each day. Assume only one entry is needed per day.
Create the following methods for your class:

- Add a task. Thisfunction accepts three parameters: task details, date and month. It should add
- a task on the day specified.
- Remove a task. Accepts the date and month as a parameter to remove the task from.
- Show tasks. This method should go through all of your months and print all the tasks allocated.

Your task is to create one calendar object, then hardcode 5-6 tasks for your calendar. Then demonstrate how you'll remove a task, and display the updated task list.
Task 6:
Create a class called Smartphone with the following attributes:

- Company
- Model
- Display Resolution
- RAM
- ROM
- Storage

Create getter and setter methods for your attributes. A smartphone has some specific actions that it can perform.
For example:

1. Make a phone call
2. Send a message
3. Connect to the wifi
4. Browse the internet

Create different smartphone objects. Set their attributes using the setter functions and display their attributes after using the getter functions to fetch the attributes.
Task 7:
Create a class for a stationary shop. The stationary shop maintains a list for all the items that it sells (hint: array of strings), and another list with the prices of the items (hint: array of prices).
Create a menu-driven program to:

1. Allow the shop owner to add the items and their prices.
2. Fetch the list of items
3. Edit the prices of the items
4. View all the items and their prices

Create a receipt that the shopkeeper can share with their customers. The receipt can only be created after the shopkeeper inputs the items and their amounts bought by the customer.