DHA SUFFA UNIVERSITY
Department of Computer Science
Object Oriented Programming
CS-1002L
Spring 2020

Lab 03
Introduction to OOP

Objectives

- To learn about classes , objects , member variables , member functions , constructors with its types, destructors, accessors and mutators

Class

A class is the building block of OOP. It is the prototype that defines methods and variables that are common to all specified types of objects. It is the way to bind the data and its logically related functions together. Its an abstract data type that can be treated like any other built in data type.

Syntax

```
//Syntax of a class
class ClassName
{
  member variables;
  member functions;
};
```

Example 3.1

```
//FileName:cellPhone.h
//Defining class
class CellPhone
{
    //Declaring instance variables
    const char* name;
    const char* processor;
    const char* memory;
    const char* os;

    //Declaring member function
    void display();
};
```

```
//FileName:cellPhone.cpp
#include "cellPhone.h"
//defining member function
void CellPhone::display()
{
        std::cout<<"\====Displaying Phone Information====\n";
        std::cout<<"Device Name = "<<name<<std::endl;
        std::cout<<"Processor = "<<processor<<std::endl;
        std::cout<<"Memory = "<<memory<<std::endl;
        std::cout<<"OS = "<<os<<std::endl;
}
```

Access Modifiers

Access modifiers are keywords that are used to set accessibility of classes, methods and variables. In OOP there are three types of access modifiers

- `public`
- `private`
- `protected`

Scope of Access Modifiers

Scope of access modifiers are as follows

Keywords	Scope
`private`	The members of class that are declared `private` can be accessible within class only
`public`	The members of class that are declared `public` can be accessible by other classes too. They can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.
`protected`	The members of class that are declared `protected` can be accessible within class and within derived class only

Example 3.2

```
//FileName:cellPhone.h
#include <iostream>

class CellPhone
{
  private:
    //Declaring instance variables
    char* name;
    char* processor;
    char* memory;
    char* os;
    //Specifying access modifier
  public:
    //Declaring member function
    void display();
};
//FileName:cellPhone.cpp
#include "cellPhone.h"
    void cellPhone::display()
    {
        std::cout<<"\n====Displaying Phone Information====\n";
        std::cout<<"Device Name = "<<name<<std::endl;
        std::cout<<"Processor = "<<processor<<std::endl;
        std::cout<<"Memory = "<<memory<<std::endl;
        std::cout<<"OS = "<<os<<std::endl;
    }
```

const keyword

`const` keyword is used when you want to restrict any modification.It can be used with variables,pointers,function arguments,function return types,member variables,member functions and objects

const function

const function is used to restrict the function to modify values of data members of a class

Syntax

```
returnType functionName()const
{
     //function body
}
```

const parameters

const parameters function implies that the member function will not modify values of the object.

Syntax

```
returnType functionName(const datatype varName)
{
    //function body
}
```

Constructor

Constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new objects. The constructor has the same name as class. It does not return any value. If we do not specify a constructor , C++ compiler generates a default constructor for us (expects no parameters and has an empty body). There can be two types of constructors in C++.

- Non-Parameterized constructor
- Parameterized constructor

Non-Parameterized constructor

Constructor that does not takes any arguments are non-Parameterized constructors

```
class ClassName
{
  accessModifier:
  //Syntax of creating a non-parameterized constructor
  className()
  {
  }
};
```

Example 3.3

```
//FileName:cellPhone.h
#include <iostream>

class CellPhone{
  private:
    //Declaring instance variables
    const char* name;
    const char* processor;
    const char* memory;
    const char* os;
    //Specifying access modifier
  public:
    //Declaring non-parameterized constructor
    cellPhone();
    //Declaring member function
    void display()const;
};
```

```
//FileName:cellPhone.cpp
#include "cellPhone.h"
//defining constructor
cellPhone::cellPhone()
    {
       name="Samsung Galaxy Note 8";
      processor="Octa-core";
       memory="64GB built-in, 6GB RAM";
       os="Android v7.1.1 Nougat";
    }
    //defining member function
    void cellPhone::display() const
    {
        std::cout<<"\n===Displaying Phone Information===\n";
        std::cout<<"Device Name = "<<name<<std::endl;
        std::cout<<"Processor = "<<processor<<std::endl;
        std::cout<<"Memory = "<<memory<<std::endl;
        std::cout<<"OS = "<<os<<std::endl;
    }
```

Parameterized constructor

Constructor that takes arguments are Parameterized constructors.

```
class className
{
  //Syntax of creating a non-parameterized constructor
  className(const datatype )
  {
  }
};
```

Example3.4

```
//FileName:cellPhone.h
#include <iostream>
//Defining CellPhone class

class CellPhone
{
  private:
    //Declaring instance variables
    const char* name;
    const char* processor;
    const char* memory;
    const char* os;
    //Specifying access modifier
  public:
    //Declaring parameterized constructor
    cellPhone(char* ,char*, char* , char* );
    //Declaring member functions
    void display()const;
};
```

Object

An object is an instance of a class. Its an abstraction of real world entity.

Syntax

```
//Syntax of creating an object
classname  objectName;
```

Creating an object of cellPhone class

Syntax

```
CellPhone c1;
```

Calling member function through an object

Syntax

```
//syntax of calling member function of a class through an object
objectName.member_function;
```

Calling display function of cellPhone class

Syntax

```
 c1.display();
```

Header Guards

Header guards is a piece of code that is used for eliminating duplicate inclusion of header files.

Syntax

```
#ifndef FILENAME_H
#define FILENAME_H
class ClassName
{
   //Attributes
   //Functions
};
#endif
```

Separating Interface from Implementation

Example 3.5

```
//FileName:cellPhone.h
#ifndef CELLPHONE_H
#define CELLPHONE_H
//FileName:cellPhone.h
#include <iostream>
class CellPhone
{
  private:
    //Declaring instance variables
    const char* name;
    const char* processor;
    const char* memory;
    const char* os;
    //Specifying access modifier
  public:
    //Declaring non-parameterized constructor
    CellPhone();
    //Declaring member function
    void display()const;
};
#endif
```

```
//FileName:cellPhone.cpp
#include "cellPhone.h"
    //defining constructor
    CellPhone::CellPhone()
    {
        name="Samsung Galaxy Note 8";
         processor="Octa-core";
        memory="64GB built-in, 6GB RAM";
        os="Android v7.1.1 Nougat";
    }

    //defining member function
  void CellPhone::display()const
    {
        std::cout<<"\n===Displaying Phone Information===\n";
        std::cout<<"Device Name = "<<name<<std::endl;
        std::cout<<"Processor = "<<processor<<std::endl;
        std::cout<<"Memory = "<<memory<<std::endl;
        std::cout<<"OS = "<<os<<std::endl;
       }
```

```
//FileName:driver.cpp
#include "cellPhone.h"
int main()
{
       //creating object of cellPhone class
       //way of creating object through new keyword
       CellPhone *c1=new CellPhone();

       //calling display method through cellPhone class object
     c1->display();

       //creating another object of cellPhone class
       CellPhone *c2=new CellPhone();

     //calling display method through cellPhone class object
      c2->display();
      return 0;
}
```

Output

```
========Displaying Phone Information==========
Device Name = Samsung Galaxy Note 8
Processor = Octa-core
Memory = 64GB built-in, 6GB RAM
OS = Android v7.1.1 Nougat

========Displaying Phone Information==========
Device Name = Samsung Galaxy Note 8
Processor = Octa-core
Memory = 64GB built-in, 6GB RAM
OS = Android v7.1.1 Nougat
```

Destructor

Destructor is a member function which destructs or deletes an object when it goes out of scope. Destructors deallocates memory when

- When a local object goes out of scope
- For a global object, operator is applied to a pointer to the object of the class

```
class ClassName
{
  //Syntax of creating a non-parameterized constructor
  className()
  {
  }
  //Syntax of creating destructor
  ~className()
  {
  }
};
```

Example 3.6

```
//FileName:cellPhone.h
```

```
class cellPhone
{    //Specifying access modifier
  public:
    .
    .
    .
    //Declaring Destructors
    ~cellPhone();
};
#endif
```

```
//FileName:cellPhone.cpp
#include "cellPhone.h"
    //defining Destructors
    cellPhone :: ~cellPhone()
    {
    }
```

Accessors and Mutators

Accessors

An accessor is a member function of class which allows access to private or protected data member of a class outside the class. It is also called Getter.

There are a few things that accessor/getter must meet:

1. Must have a return type same as the data member.
2. It does not take any argument/s.
3. Its name should start with `get` prefix.

syntax

```
data-type GetAccessor-Name ()
{
     return data-member;
}
```

Mutators

Mutator is a member function of class which allows manipulating the contents of private or protected data member of a class outside the class. It is also called Setter.

There are a few things that Mutator/setter must meet:

1. Does not return any value.
2. It receives one argument and data member is set to that value.
3. Its name should start with **`set`** prefix.

syntax

```
void SetMutator-Name (data-type variable-Name)
{
     data-member = variable-Name;
}
```

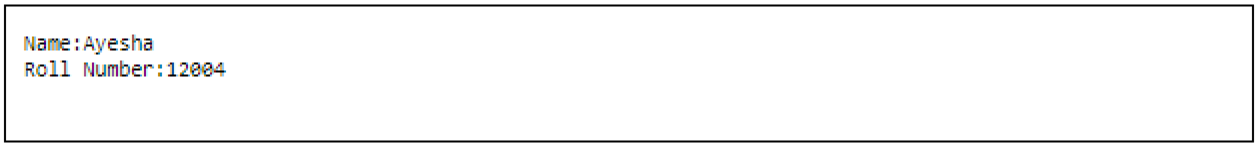
Example 3.7

```
//Defining a class's interface with Function prototypes
//filename:Student.h
#include <iostream>
//Defining class
class Student
{
 //data members
 private:
    std::string name;
    int rollNo;
 //member functions
 public:
     Student();
     ~Student();
     void setName(const std::string& n);
     std::string getName();
     void setRollNo(int r);
     int getRollNo();
};
```

```
//Defining Member Functions in a Separate Source-Code File
//File Name:Student.cpp
#include <iostream>
#include "Student.h"
//constructor definition
Student::Student()
{
    name="";
    rollNo=0;
}
Student::~Student()
{
    std::cout<<"Destructor";
}
//Mutator
void Student::setName(const std::string& n)
{
    name=n;

}
//Acessor
std::string Student::getName()
{
    return name;

}
//Mutator
void Student::setRollNo(int r)
{
    rollNo=r;

}
//Accessor
int Student::getRollNo()
{
    return rollNo;

}
```

```
//File Name:main.cpp
#include <iostream>
#include "Student.h"
//Defining main method
int main()
{
  Student s1;
  s1.setName("Ayesha");
  std::cout <<"Name:" << s1.getName();
  s1.setRollNo(12004);
  std::cout <<"\nRoll Number:" << s1.getRollNo();
  return 0;
}
```

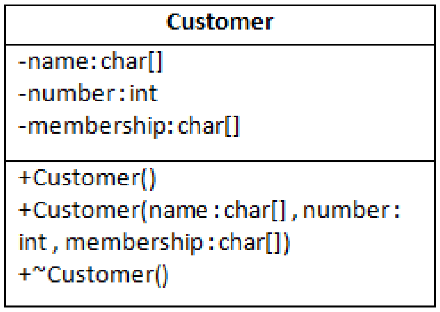
Output



```
Name:Ayesha
Roll Number:12004
```

Lab Tasks

Task 3.1
Implement the following UML into c++



```
Customer
-name: char[]
-number: int
-membership: char[]
+Customer()
+Customer(name : char[] , number :
int , membership : char[])
+~Customer()
```

Task 3.2

Create a calculator class that will calculate sum, subtraction, product and division of two numbers will be taken from user

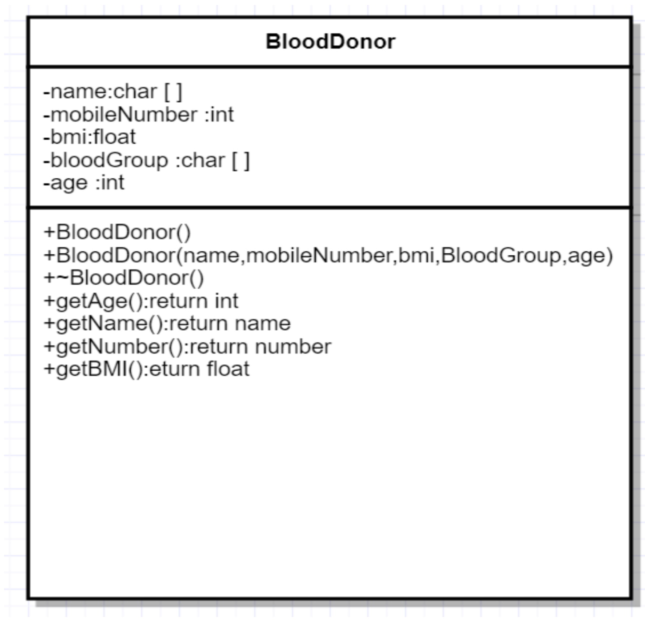
Task 3.3

Create an employee class. The member data should comprise an int for storing the employee number,int for year of joining and a float for storing the employee's salary. Write a main() that allows the user to enter data for three employees and display it.There will following member functions

1. displayInfo() will be used to display employee info
2. calculateNetSalary() will be used to display net salary after deductions(consider loans,late comings and leaves for the deduction criteria of salary)

Task 3.4 (Home Assignment)

Indus hospital is visiting universities on a blood drive to overcome the shortage of blood. Being helpful Human beings, you have decided to help Indus Hospital in setting up Computerized records of the blood donors. To help them achieve this you will be creating a class which will store the details of Blood donor. Class requirements are given in UML diagram below:



BloodDonor

-name:char [ ]
-mobileNumber :int
-bmi:float
-bloodGroup :char [ ]
-age :int

+BloodDonor()
+BloodDonor(name,mobileNumber,bmi,BloodGroup,age)
+~BloodDonor()
+getAge():return int
+getName():return name
+getNumber():return number
+getBMI():eturn float

Task 3.5

Create a class BankAccount whose attributes are: account number, account holder’s name and balance. BankAccount constructor initializes the values of account number, account holder’s name and balance. Make one credit function which credits the amount either passed by arguments or the value entered by user. Create three accounts for three persons and display the same on your screen.

Submission Instructions

1. Number your solution folders as question number e.g. Q1, Q2, etc. (Q is in upper case)
2. Every folder should contain three files (one header, one implementation and one driver)
3. Create a new folder named cs192abc where abc is your 3 digit roll #. e.g. cs192111.
4. Copy all the project folders into this folder.
5. Now make sure a zip file named cs192abc.zip is created e.g. cs192111.zip
6. Upload the assignment solution on LMS under the assignment named Lab 03