Birla Institute of Technology and Science, Pilani Hyderabad Campus
BITS F232: Foundations of Data Structures and Algorithms
1st Semester 2023-24 Lab Sheet No:2

General Tips

- If you are working with the gcc compiler, a common line to add at the top of your code file is:

  ```
  #include <bits/stdc++.h>
  ```

  This will include all the required files that you will need for the purpose of the labs. It is also recommended to find out which files are included in this for your own understanding.

- Indent your code appropriately and use proper variable names, if not already provided in the question. Also, use comments wherever necessary.

- Make sure to debug your code after every task or after every code block to avoid having to debug your entire file at once.

- Use a proper IDE or text editor like Sublime Text or VSCode (Or Vim) as they help to run and test your code on multiple test-cases easily. You can install Windows Subsystem Linux (WSL) or MinGW, if you are Windows user to compile and run your programs . However, all the lab programs must run on the Linux distributions loaded in I-015 (regular labs in CS department's Data Science lab).

Overview:

Having learned C in your first year, you must be at least somewhat familiar with its syntax and features. C++ was introduced by Bjarne Stroustrup in the late 1970s as a way to bring Object Oriented Programming, along with some other incredibly useful features, to C. In this lab, we will take a look at some of the most prevalent features with a focus on the object oriented programming. Since some of you might not have done OOPS yet, so we will give a brief summary of the same.

Class Structure:

Let's start by understanding how to declare a class in C++. A class is defined using the **class** keyword, followed by the class name. The class contents are enclosed within curly braces, and looks something like this:

```
class ClassName
{
    // body of the class
};
```

Object:

Object is a run time entity, which is an instance of a Class. Class is an ADT. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

Access Specifiers:

Access specifiers are used to define the scope of access for members and methods. There are three access specifiers:

- ***Public:*** Public members are accessible from outside the class through an object of the class (typically with the use of the dot operator). This means that any class can use an object of this class type to get access to a public member or method.

- ***Protected****:* Protected members can be accessed only by classes derived **(inherited)** from the current class. This means that only child classes will be allowed to directly access these members.

- ***Private**:* Private members cannot be accessed by anything other than the class itself. Any attempt to access a private member will result in the compilation error. Any member declared without an access specifier is **private by default**.

The access specifiers are declared with the following syntax:

```
class ClassName
{
// public members can be accessed by any other class or function
public:
// public members

// protected members can only be accessed by classes that inherit this class
protected:
// protected members

 // private members cannot be accessed by anything except this class
 private:
 // private members
 };
```

C++ offers an alternative approach for defining class methods. These methods can be defined similarly to regular functions. However, to associate a function with a class, the scope resolution operator :: is employed. It is used to explicitly specify that a function belongs to a particular class. An example is shown below:

```
class ClassName
{
    private:
       int value;
    public:
       void increment_value();
       // This is a method of the ClassName class and can be accessed with the dot operator
};
// The increment_value() method of ClassName class is defined below
void  ClassName::increment_value() {
   value++;
}
```

Constructors:

A constructor is a special type of member function of a class which initializes objects of a class. In C++, Constructor is automatically called when an object (instance of class) is created. It is used to initialize the data members of new objects. It is special member function of the class because it does not have any return type. A constructor differs from regular member functions in the following ways:

- Constructor has the same name as the class name.
- Constructors don't have a return type.
- A constructor is automatically called when an object is created.

- It must be placed in public section of class.
- If we **do not** specify a constructor, C++ compiler generates a default constructor for object (expects no parameters). In C++, the default constructor created by the compiler has an empty body i.e. it doesn't assign default values to data members unlike in Java where default values are assigned.

```
class construct
{
 public:
        int a, b;
        // The constructor is called when an object is instantiated and sets the values of a and b
           construct()
           {
              a = 10;
              b = 20;
           }
};
       int main()
       {
           // Default constructor called automatically when the object is created
           construct c;
           cout << "a: " << c.a << std::endl << "b: " << c.b;
       }
```

There are 3 types of constructors in C++.

1. **Default Constructor:**
   A constructor defined without any arguments (or parameters) is referred to as a default constructor. This type of constructor is commonly utilized to initialize data members with actual values. It serves as a means to provide initial values to the class members when an object is instantiated.
2. **Parameterized Constructor:**
   Constructors that accept one or more arguments (or parameters) are known as parameterized constructors. The constructor's body will then utilize these parameters for object initialization.
3. **Copy Constructor:**
   The member function initializes an object using another object of the same class. It helps to copy data from one object to another. When a copy constructor is not defined, the C++ compiler automatically generates a default copy constructor that copies all the data of the object to the new object.

```
class Sample
{
 public:
        int a;

           Sample(int x)
           {
              a = x;
           }
```

```
10          Sample(Sample &obj)    //This is a copy constructor
11          {
12              a = obj.a;
13          }
14          };
13
14      int main()
15      {
16          Sample s1(10);
17          Sample s2(s1);      //or Sample s2 = s1;
18          cout << "a of s2: " << s2.a;
19      }
```

The 4 Pillars of Object Oriented Programming:

1. Inheritance:

The capability of a class to derive properties and characteristics from another class is called Inheritance.

Sub Class: The class that inherits properties from another class is called Sub class or Derived Class.

Super Class: The class whose properties are inherited by sub class is called Base Class or Super class.

For creating a sub-class which is inherited from the base class we have to follow the below syntax.

```
1   class subclass_name : access_mode base_class_name
2   {
3   //body of subclass
4   };
```

Here, subclass name is the name of the sub class, access mode is the mode in which you want to inherit this sub class for example: public, private etc. and base class name is the name of the base class from which you want to inherit the sub class.

Note: A derived class doesn't inherit access to private data members. However, it does inherit a full parent object, which contains any private members which that class declares.

An example is given below:

```
1       //Base class
2       class Parent
3       {
4       public:
```

```
5           int id_p;
6           };
7
8           // Sub class inheriting from Base Class (Parent)
9           class Child : public Parent
10          {
11          public:
12          int  id_c;
13          };
```

The modes of inheritance are:

- **Public mode:** If we derive a sub class from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in derived class.
- **Protected mode:** If we derive a sub class from a Protected base class. Then both public member and protected members of the base class will become protected in derived class.
- **Private mode:** If we derive a sub class from a Private base class. Then both public member and protected members of the base class will become Private in derived class.

Types of Inheritance:

i **Single Level Inheritance**: When one class inherits another class, it is known as single level inheritance.

ii **Multiple Inheritance:** Multiple inheritance is the process of deriving a new class that inherits the attributes from two or more classes.

iii **Hierarchical inheritance**: Hierarchical inheritance is defined as the process of deriving more than one class from a base class.

iv **Multilevel inheritance:** Multilevel inheritance is a process of deriving a class from another derived class.

v **Hybrid inheritance:** Hybrid inheritance is a combination of simple, multiple inheritance and hierarchical inheritance.

2.Encapsulation:

It is the process in which the data is not accessed directly, but with the help of functions. In normal words, public getters and setter's functions are used to access private members. It helps in data hiding. Data hiding ensures exclusive data access to class members and protects object integrity by preventing unintended or intended changes. e.g. "protected", "private".

3.Abstraction:

Abstraction means displaying only essential information and **hiding** the details/ **implementation** behind it. Using classes, it can be implemented using **access specifiers**. A Class can decide which data member will be visible to outside world and which is not.

Another way of implementing is using **header files**, for example- *pow ()* present in math.h header file. Whenever we need to calculate power of a number, we simply call the function *pow ()* present in the math.h header file and pass the numbers as arguments without knowing the underlying algorithm.

It helps to **increase security** of an application or program as only important details are provided to the user.

3.**Polymorphism:**

As the name suggests, poly means 'many' and morphism means 'form'. Anything which is present in more than one form is said to follow polymorphism.

It is of 2 types:

1. **Compile Time Polymorphism**- This type of polymorphism is achieved by function overloading or operator overloading. Function overloading takes place when there are multiple function with same name but with different parameters.

2. **Run Time Polymorphism**- This type of polymorphism is achieved by Function Overriding.

   Function overriding occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.

Friend Classes and Functions:

A friend class can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class. For example, a LinkedList class may be allowed to access private members of Node.

```
class Node
{ private:
    int key;
    Node* next;
    /* Other members of Node Class */

    // Now class  LinkedList  can
    // access private members of Node
    friend class LinkedList;
    };
```

Friend Function Like friend class, a friend function can be given a special grant to access private and protected members. A friend function can be:

- A member of another class
- A global function

```
class Node
{  private:
    int key;
    Node* next;

    /* Other members of Node Class */
    friend int LinkedList::search();
    // Only search() of linkedList // can access internal members
};
```

The following points should be kept in mind while using friend classes and functions:

- Friends should be used only for limited purpose. too many functions or external classes are declared as friends of a class with protected or private data, it lessens the value of encapsulation of separate classes in object-oriented programming.
- Friendship is not mutual. If class A is a friend of B, then B doesn't become a friend of A automatically.
- Friendship is not inherited.

Templates:

Templates in C++ are a powerful feature that allows you to write generic code that works seamlessly with different data types. They provide a way to create functions and classes that can be used with various data types without the need to duplicate code. This not only promotes code reusability but also enhances the efficiency of your programs by eliminating redundant implementations.

Templates enable us to define functions or classes with placeholders for data types. When we use a template function or class, we provide the specific data type as an argument, and the compiler generates the appropriate code for that data type.

For example, if we need to write functions for tasks like sorting and searching that handles data of different datatypes, we can write a single template function and pass the datatype as an argument.

```
1.  template <typename T>
2.  T add(T a, T b)
3.  {
4.      T result = a + b;
5.      return result;
6.  }
7.
8.  int main()
9.  {
10.     cout << add<int>(1,2)<<endl;          // Call add for int
11.     cout <<add<double>(1.2,2.3)<<endl;    // Call add for double
12. }
```

Here, the **template** keyword signifies that the following function or class will be a template. The **typename** keyword denotes that the T specifier that follows it is a data type parameter. The word **typename** can also be replaced with the word **Class** like so**:**

```
template < class T>
```

In the above example, the compiler creates two overloaded functions, for int and double because those are the data types which are being used in the function calls.

Course Class:

Hopefully you weren't overloaded with information in the last section, because now you have to complete the following task. You are given a Course class representing an under-graduate student level course at BITS, say BITS F232. You will then be asked to modify the class as well as create new classes with certain functionality.

The class is as follows:

```
1. class Course {
2. private:
3.              vector<int>courseMarks;
4.              public:
5.              string name;
6.              string IC;
7.              int numberOfStudents;
8.              int maxMarks;
9.              Course(){
10.
11.             }
12.             vector<int> getMarks(){
 13.
14.             }
```

```
15.     void setCourseMarks(vector<int>&v){
16.
17.     }
18.     bool evaluateAnswer(int givenAnswer, int expectedAnswer){
19.         return (givenAnswer ==expectedAnswer);
20.     }
21.     };
22.     class Lab: public Course{
23.     private:
24.      public:
25.     void setLabMarks(vector<int>&v){
27.
28.     }
29.     int countNC(int minMarks){
30.
31.     }
32.     };
```

Code link - https://p.ip.fi/6cZ3

Task 1:

- First, create a new constructor for Course class that takes 4 arguments and initializes the class members name, IC, numberOfStudents, and maxMarks in any order.
- Now, complete the child class of Lab class. This class should have an extra member called labMarks, which is a vector of int, and the size of the vector is equal to the number of students in the private section.
- In the Lab class, complete function called countNC, which can access the private variables of the Course class. This function should take in a single parameter, minMarks which is an integer. Then, it should sum each value in courseMarks vector of the Course object with its corresponding value in the labMarks vector and return the total number of students whose total marks are less than the minMarks and got NC (Not clear).

You can use public getter and setter functions to access the private members of the class. Just complete the implementation of these functions.

Try to see what happens if you modify the default constructor instead of making a new one.

Task 2:

Implement polymorphism by overloading the function called evaluateAnswer that is part of the Course class. It takes in two parameters: givenAnswer and expectedAnswer. These two parameters are of type int, you are required to overload this function for the float and string data types.

The function returns true if both of the answers match and false if they don't. Note: Global functions can also be overloaded in a similar fashion.
Can you tell which type of polymorphism is implemented here?

Task 3:

Write a C++ program using templates to find minimum of two numbers. Use function templates to implement your task where both integers and float values could be given as input.

End of lab sheet…

More read@: https://cplusplus.com/doc/tutorial/