CC-211L

Object Oriented Programming

Laboratory 05

OOP Primitives and Composition / STL Containers

Version: 1.0.0

Release Date: 27-02-2024

Department of Information Technology

University of the Punjab

Lahore, Pakistan

Contents:

Learning Objectives:

- Returning Reference to private Data Members
- Default Memberwise Assignment
- const Objects
- const Member Functions
- Object Composition and Aggregation

Resources Required:

- Desktop Computer or Laptop
- Microsoft ® Visual Studio 2022

General Instructions:

- In this Lab, you are **NOT** allowed to discuss your solution with your colleagues, even not allowed to ask how is s/he doing, this may result in negative marking. You can **ONLY** discuss with your Teaching Assistants (TAs) or Lab Instructor.
- Your TAs will be available in the Lab for your help. Alternatively, you can send your queries via email to one of the followings.

Teachers:		
Course Instructor	Prof. Dr. Syed Waqar ul Qounain	swjaffry@pucit.edu.pk
Lab Instructor	Mam Sanam	
Teacher Assistants	Syed M.Jafar Ali Naqvi	bitf21m032@pucit.edu.pk
	M. Usman Munir	bitf19m020@pucit.edu.pk
	Aqdas Shabir	bitf21m022@pucit.edu.pk
	Ibad Nazir	bitf21m024@pucit.edu.pk
	Umer Farooq	bitf21m010@pucit.edu.pk

Background and Overview:

Reference to an Object:

A reference to an object in C++ is an alias that allows you to access an object by another name. It is similar to a pointer, except that you do not need to use the dereference operator (*) to access the object. A reference is created using the & operator. References are typically used to create a more concise and efficient way of referring to an object without needing to copy the object itself. Additionally, references can be used to pass an object into a function without needing to make a copy of the object.

Const Keyword

The const keyword in C++ is used to declare a constant. It is a modifier that indicates that the value of the variable or object it is applied to cannot be changed. Constants are useful for declaring values that will not change throughout the program, such as mathematical constants or other unchanging values.

Composition

Composition in C++ is a way to combine objects or classes into a bigger class or object. This type of relationship is known as a "has-a" relationship, as the composed class "has a" reference to the other class. Composition is used to model a "part-of" relationship, where one object can contain multiple smaller objects. This allows for complex behavior with simpler parts. For example, a car class may contain a steering wheel, an engine, and a transmission as separate objects. All of these objects work together to make up the car, but can be managed and updated independently. The composition also allows for greater flexibility and reuse of code, as the objects can be changed and reused in different contexts.

Member Initializer List:

Member Initializer Lists in C++ are a way to initialize data members of a class. They are used to initialize member variables to a known state before the body of the constructor is run.

Activities:

Pre-Lab Activities:

Returning Reference to a private data member:

Returning a reference to a private data member is a way for a class to allow other classes and functions to access a private variable without making the variable public. This allows the private variable to remain protected while still providing a way for it to be accessed.

The use of the const keyword is of utmost importance in this regard, as it prevents external code from modifying private data members by not allowing the reference to be used to change their values. If the reference return type is declared const, the reference is a nonmodifiable value, meaning it cannot be used to modify the data. On the other hand, if the reference return type is not declared const, it can lead to subtle errors.

Example:

```cpp
#include<iostream>
using namespace std;
class Time {
private:
    unsigned int hour{ 0 }, minute{ 0 }, second{ 0 };
public:
    void setTime(int h, int m, int s) {
        // validate hour, minute and second
        if ((h >= 0 && h < 24) && (m >= 0 && m < 60) && (s >= 0 && s < 60)) {
            hour = h; minute = m; second = s;
        }
        else
            throw invalid_argument("hour, minute and/or second was out of range");
    }
    unsigned int getHour() const {
            return hour;
    }
    unsigned int& badSetHour(int hh) {
        if (hh >= 0 && hh < 24)
            hour = hh;
        else
            throw invalid_argument("hour must be 0-23");
        return hour; // dangerous reference return
    }
};
int main() {
    Time t;
    unsigned int& hourRef = t.badSetHour(20);
    cout << "Valid hour before modification: " << hourRef;
    hourRef = 30;
    cout << "\nInvalid hour after modification: " << t.getHour();
    t.badSetHour(12) = 74;
    cout << "\n\nPOOR PROGRAMMING PRACTICE\n" << "t.badSetHour(12) as an lvalue, invalid hour:" << t.getHour()<<"\n";
}
```

Fig. 01 (Returning reference to private data member)

Output:

```
Microsoft Visual Studio Debug Console
Valid hour before modification: 20
Invalid hour after modification: 30

POOR PROGRAMMING PRACTICE
t.badSetHour(12) as an lvalue, invalid hour:74
```

Fig. 02 (Returning reference to private data member)

If we use the 'const' keyword as a return type, the compiler will prevent any modification of private data members of a class, thus preserving the concept of encapsulation while still allowing access to those members. As seen in the code snippet below, the compiler will throw an error on any lines where we attempt to change the values of private data members.

```
#include<iostream>
using namespace std;
class Time {
private:
    unsigned int hour{ 0 }, minute{ 0 }, second{ 0 };
public:
    void setTime(int h, int m, int s) {
        // validate hour, minute and second
        if ((h >= 0 && h < 24) && (m >= 0 && m < 60) && (s >= 0 && s < 60)) {
            hour = h; minute = m; second = s;
        }
        else
            throw invalid_argument("hour, minute and/or second was out of range");
    }
    unsigned int getHour() const {
            return hour;
    }
    const unsigned int& badSetHour(int hh) {
        if (hh >= 0 && hh < 24)
            hour = hh;
        else
            throw invalid_argument("hour must be 0-23");
        return hour; // dangerous reference return
    }
};
int main() {
    Time t;
    const unsigned int& hourRef = t.badSetHour(20);
    cout << "Valid hour before modification: " << hourRef;
    hourRef = 30;
    cout << "\nInvalid hour after modification: " << t.getHour();
    t.badSetHour(12) = 74;
    cout << "\n\nPOOR PROGRAMMING PRACTICE\n" << "t.badSetHour(12) as an lvalue, invalid hour:" << t.getHour()<<"\n";
}
```

Fig. 03 (Returning reference to private data member)

Error List:

abc	E0137	expression must be a modifiable lvalue	Project10	Source.cpp	30
abc	E0137	expression must be a modifiable lvalue	Project10	Source.cpp	32

Fig. 04 (Returning reference to private data member)

Task 01: Count the Bill **[Estimated time 20 minutes / 20 marks]**

Create a function countBill which will get number of units consumed and will count the total bill based on it. Rate for consumed first 100 units is 25 rupees per unit. Then for next 50 its 30, for next 50 its 35 and so on.

Task 02: Queue Reverse **[20 minutes / 20 marks]**

Ask the user for range, input elements in queue. Then reverse the elements in Queue.

Input: 5

4 3 1 5 2

Output: 2 5 1 3 4

Task 03: Recursion meets Vector **[Estimated time 20 minutes / 20 marks]**

Ask the user for number of sentences to be inserted. Then insert that many senetences in vector from user without using Loop. Then create another function to Display all the sentences inserted in vector without a loop.

In-Lab Activities:

Default Memberwise Assignment:

The assignment operator (=) can be used to assign an object to another object of the same type. By default, such assignment is performed by **Memberwise assignment** each data member of the object on the right of the assignment operator is assigned individually to the same data member in the object on the left of the assignment operator.

Example:

```
#include <iostream>
using namespace std;
class Test {
    int x;
public:
    Test(int i)
    {
        x = i;
    }
    void print() { cout << x << endl; }
};
// Driver Code
int main()
{
    Test t1(10);
    Test t2(20);
    t2 = t1;
    t1.print();
    t2.print();
    return 0;
}
```

Fig. 07 (Memberwise Assignment)

Output:

```
Microsoft Visual Studio Debug Console
10
10
```

Fig. 08 (Memberwise Assignment)

const Objects and const Member functions:

Some objects need to be modifiable, and some do not. You may use const to specify that an object is not modifiable and that any attempt to modify the object should result in a compilation error.

const Time noon {12,0,0};

The statement above declares a const object noon of class Time and initializes it to 12 noon. It's possible to instantiate const and non-const objects of the same class.

Constant member functions are those functions that are denied permission to change the values of the data members of their class. To make a member function constant, the keyword "const" is appended to the function prototype and to the function definition header.

Example:

```cpp
#include<iostream>
using namespace std;
class Demo
{
    int x;
public:
    void set_data(int);
    int get_data() const;

};
void Demo::set_data(int a)
{
    x = a;
}
int Demo::get_data() const
{
    x++;
    return x;
}
int main()
{
    Demo d;
    d.set_data(10);
    cout << endl << d.get_data();
    return 0;
}
```

Fig. 09 (const Objects & Member functions)

Error List:

E0137	expression must be a modifiable lvalue	Project10	Source.cpp	17

Fig. 10 (const Objects & Member functions)

Composition:

In real-life complex objects are often built from smaller and simpler objects. For example, a car is built using a metal frame, an engine some tires, a transmission system, a steering wheel, and many other parts. This process of building complex objects from simpler ones is called C++ composition. It is also known as object composition.

A class can have one or more objects of other classes as members. A class is written in such a way that the object of another existing class becomes a member of the new class. this relationship between classes is known as C++ Composition. It is also known as containment, part-whole, or has-a relationship. In C++ Composition, an object is a part of another object. The object that is a part of another object is known as a sub-object. When a C++ Composition is destroyed, then all its sub-objects are destroyed as well. Such as when a car is destroyed, then its motor, frame, and other parts are also destroyed with it. It is a do or die relationship.

Example:

```cpp
#include <iostream>
using namespace std;
class X
{
private:
    int num;
public:
    void set_value(int k){
        num = k;
    }
    void show_sum(int n){
        cout << "Sum of " << num << " and " << n << " = " << num + n << endl;
    }
};
class Y
{
public:
    X a; //Object of Class X
    void print_result(){
        a.show_sum(10);
    }
};
int main()
{
    Y b;
    b.a.set_value(50);
    b.a.show_sum(50);
    b.print_result();
}
```

Fig. 11 (Composition)

Output:

```
Microsoft Visual Studio Debug Console
Sum of 50 and 50 = 100
Sum of 50 and 10 = 60
```

Fig. 12 (Composition)

Explanation:

In this program, class X has one data member ‘num’ and two member functions ‘set_value()’ and ‘show_sum()’. The set_value() function is used to assign value to ‘num’.

The show_sum() function uses an integer type parameter. It adds the value of parameter with the value of ‘num’ and displays the result on the Console.

Another class Y is defined after the class X. the class Y has an object of class X that is the C++ Composition relationship between classes X and Y. This class has its own member function print_result().

In the main() function, an object ‘b’ of class Y is created. The member function set_value() of object ‘a’ that is the sub-object of object ‘b’ is called by using two dot operators. One dot operator is used to access the member of the object ‘b’ that is object ‘a’, and second is used to access the member function set_value() of sub-object ‘a’ and ‘num’ is assigned a value 20.

In the same way, the show_sum() member function is called by using two dot operators. The value 50 is also passed as a parameter. The member function print_result of object ‘b’ of class Y is also called for execution. In the body of this function, the show_sum() function of object ‘a’ of class X is called for execution by passing value 10.

Task 01: Queue Showdown [Estimated: 15 marks/ mins]

Take range from user of numbers, put all those elements in queue. Remove even numbers from queue and display whole queue. Now queue must have only odd numbers in it.

Task 02: Fair Election [Estimated 50 minutes / 50 marks]

class Voter:
Attributes:

- cnic //make sure that cnic is unique
- Name
- hasVoted (boolean)

Methods:
Create all necessary functions.

class Candidate:
Attributes:

- candidateID
- name
- Party
- Symbol
- votes

Methods:
Create all necessary functions.

Class VotingMachine

Attributes:
vector of candidates;
vector of voters;

Methods:

- Make all necessary functions.
- Add Candidate
- Remove Candidate
- Display all candidates
- Add Voter
- Remove Voter
- Display all Voters
- Cast Vote () *// You will take voter id from user, display all candidates to him and then user will select the candidate he/she wants, you will check if the voter and candidates ids are valid and also if the voter has already voted or not. If all good then cast the Vote, increment votes of candidate and also change state of voter.*
- Declare Winner.

Main function:

Your main function should be menu driven. Provide all the functions in VotingMachine in Menu along side Exit option.

(This Menu should keep on displaying until user selects Exit options.)

Task 3: Stack Galore **[Estimated: 15 marks/ 30 mins]**

A number is palindrome if it is equal to its reverse. Create a function is isPalindrome, take a number input from user and check if the number is palindrome or not.

Input: 12321
Output: true;

Input: 9999
Output: true;

Input: 1234
Output: false;

Task 4: Prime Mania: **[Estimated: 20 marks/ 30 mins]**

Ask the user, number of Prime numbers and create a function "PrintPrimes" , to print that many prime numbers from start. Use Recursion.

Input: 5

Ouput: 2 3 5 7 11

Input: 3

Output: 2 3 5

Post-Lab Activities:

Task 01: Encryption Master **[Estimated 50 minutes / 50 marks]**

Create a class Encryption, Attributes:

- Paragraph dynamic array.
- Number of paragraphs
- Keys (number) vector

You will get number of lines to be inserted in paragraph from user in parameterized constructor. Then make a function to insert that many lines in paragraph. A paragraph has many lines in it so it is a array of strings. Also ask the user to set a key for each line and store the key in keys vector. Then there will be another function, Encrypt. Encryption works such that key number is added to ascii value of each character of that string. So if we have string abc and key 12, then encrypted string is mno. If adding key to a character acii is getting out of bound then don't add. You will do this to every string with the respective key of that string. Now you have the whole paragraph in Encrypted form. Create a function to display the Encrypted Paragrapgh.

Create another func to Decrypt the whole paragraph using the above mechanism mentioned above in reverse order.

Handle all corner cases, use Exception Handling, your program must not crash.

Submissions:

- For In-Lab Activity:
  - Save the files on your PC.
  - TA's will evaluate the tasks offline.
- For Pre-Lab & Post-Lab Activity:
  - Submit the .cpp file on Google Classroom and name it to your roll no.

Evaluations Metric:

- All the lab tasks will be evaluated offline by TA's
- **Division of Pre-Lab marks:** **[30 marks]**
  - Task 01: Array Updating [20 marks]
  - Task 02: Forbid Change [10 marks]
- **Division of In-Lab marks:** **[60 marks]**
  - Task 01: Composition with two classes [20 marks]
  - Task 02: Rational Numbers [30 marks]
  - Task 03: Course Composition [10 marks]
- **Division of Post-Lab marks:** **[20 marks]**
  - Task 01: Composition with three classes [20 marks]

References and Additional Material:

- **Return By Reference**
  https://www.scaler.com/topics/cpp-return-reference/

- **Const object and data members**
  https://www.linuxtopia.org/online_books/programming_books/thinking_in_c++/Chapter08_024.html

- **Composition**
  https://www.geeksforgeeks.org/object-composition-delegation-in-c-with-examples/

Lab Time Activity Simulation Log:

- Slot – 01 – 00:00 – 00:15: Class Settlement
- Slot – 02 – 00:15 – 00:40: In-Lab Task
- Slot – 03 – 00:40 – 01:20: In-Lab Task
- Slot – 04 – 01:20 – 02:20: In-Lab Task
- Slot – 05 – 02:20 – 02:45: Evaluation of Lab Tasks
- Slot – 06 – 02:45 – 03:00: Discussion on Post-Lab Task