**FAST School of Computing**

**Object Oriented Programming – Spring 2025**

**Software Engineering Department**

**LAB 15**

**Abstraction & Templates in C++**

**Learning Outcomes**

In this lab you are expected to learn the following:

* Basic and Adanced Concept and Implementation of Polymorphism & Abstraction

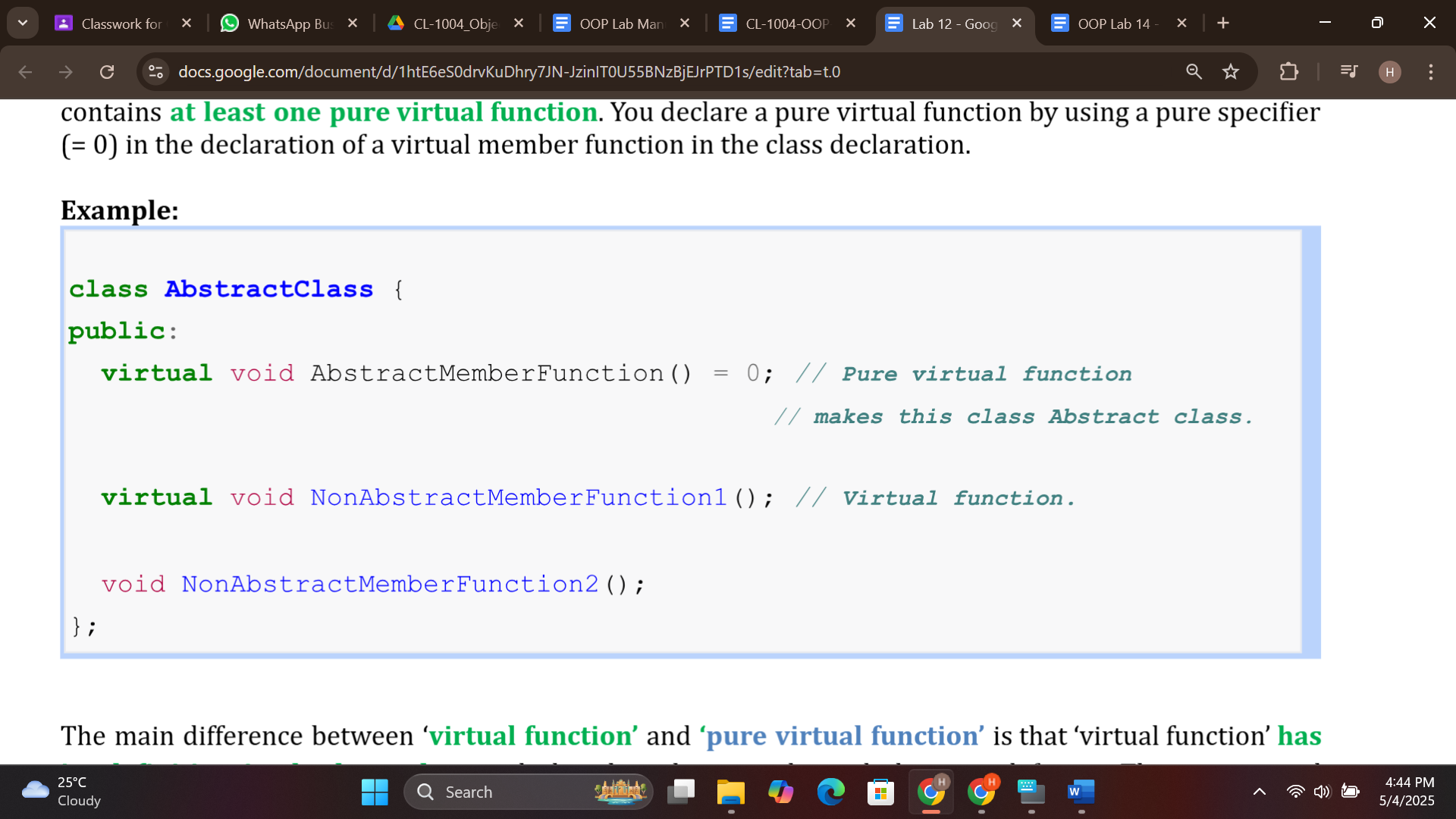
**Note:** Plagiarism(from some else or internet) in any 1 question will lead to zero marks in the whole lab task.

***\*Before attempting the task read the concepts discussed below***

**Abstract Classes:**

An **abstract class** is a class that is designed to be specifically used as a base class. An abstract class contains **at least one pure virtual function**. You declare a pure virtual function by using a pure specifier (= 0) in the declaration of a virtual member function in the class declaration.

**Example:**



The main difference between ‘**virtual function’** and **‘pure virtual function’** is that ‘virtual function’ **has its definition in the base class** and also the inheriting derived classes redefine it. The pure virtual function **has no definition in the base class**, and all the inheriting derived classes have to redefine it.

**Be Careful !**

Abstract class **cannot** be used as a parameter type, a function return type, and **not** to declare an object of an abstract class. It **can be** used to declare pointers and references to an abstract class.

**For initialization of base class members:** Constructor of base class can only be called using Initializer List.

**Templates**

Templates in C++ provide a powerful mechanism for generic programming, allowing you to write code that can be used with different data types. Templates enable the creation of functions and classes that can be parameterized with one or more types, allowing the code to be reused with different data types without writing separate implementations for each type. Templates are defined using the template keyword, followed by the template parameters enclosed in angle brackets (<>). There are two main types of templates in C++: function templates and class templates. a) Function Templates: Function templates allow you to define generic functions that can operate on different data types. You can parameterize a function template with one or more types, which are represented by template parameters. Here's an example of a simple function template that swaps two values of any type:

template <typename T>

void swapValues(T& a, T& b) {

T temp = a;

a = b;

b = temp;

}

In this example, T is a template parameter representing the type of the values being swapped. The swapValues() function can be used with various data types, such as integers, floating-point numbers, or custom-defined types. The compiler

generates a specific version of the function for each data type it is used with.

[**Function Templates**](https://programiz.com/cpp-programming/function-template)

#include <iostream>

using namespace std;

// One function works for all data types.  This would work

// even for user defined types if operator '>' is overloaded

template <typename T> T myMax(T x, T y)

{

    return (x > y) ? x : y;

}

int main()

{

    cout << myMax<int>(3, 7) << endl; // Call myMax for int

    cout << myMax<double>(3.0, 7.0)

         << endl; // call myMax for double

    cout << myMax<char>('g', 'e')

         << endl; // call myMax for char

    return 0;

}

Similar to function templates, we can use class templates to create a single class to work with different data types.

template <class T>

class className {

  private:

    T var;

    ... .. ...

  public:

    T functionName(T arg);

    ... .. ...

};

**Class Template Example**

#include <iostream>

using namespace std;

// Class template

template <class T>

class Number {

   private:

    // Variable of type T

    T num;

   public:

    Number(T n) : num(n) {}   // constructor

    T getNum() {

        return num;

    }

};

int main() {

    // create object with int type

    Number<int> numberInt(7);

    // create object with double type

    Number<double> numberDouble(7.7);

    cout << "int Number = " << numberInt.getNum() << endl;

    cout << "double Number = " << numberDouble.getNum() << endl;

    return 0;

}

**Output**

**int Number = 7**

**double Number = 7.7**

**Problem 01:**

Create a base class named Person which has a variable *name(string)* and methods as follows: constructor, destructor, virtual void print() const (print function will display the data stored in name variable of Person class).

Create a derived class named Student which only state is gpa(double) and methods: Constructor, Destructor, virtual void print() const (it invokes the Person::print() and also display gpa of student)

Finally, In main function create a pointer array of type Person with size 4, create objects of Person and Student class with these pointers and call print() to see the polymorphic behavior

**Problem 02:**

**Multilevel Inheritance**

Write C++ class Drink. Publicly inherit "Drink" class to "Water" class and "Water" class to "Carbonated" class. i.e.

**Water: Drink** and **Carbonated: Water**

Class Drink should have the following attributes:

Flavor (string)

Temperature for best serve (float) Price (float)

Expiry date (string)

* For **Drink** class, write default constructor to set all **string** values to “ “ and all float values to  0, and overloaded constructor for "Drink" to set Flavor, Temperature, price and Expiry date.
* Write getter/setter functions for Drink class.
* Inside **Water** class, declare a **string** variable **supplier**
* Write an overloaded constructor for Water class and a **Display** method to display all the attributes of Water.
* Inside **Carbonated** class, declare a **string** variable **type.**

**Carbonated** class should have default, parameterized constructor and **void Display** function to display all the attributes of the class.

**Problem 03:**

**(a)**

Write a program which is capable of calculating Power, Square Root, Sin, Cos and Tan of any data type. For mathematical calculations you are allowed to use C++ ‘cmath’ library. Your program should consist of generic function (Function templates) for each operation. Pass the operands to desired function which returns the result after performing appropriate operation.

**(b)**

Write a class Calculator using templates. The template class shall allow you to perform addition, multiplication and division on integer, double, float and long data types.

**Submission Details:**

1. Save single .cpp file with your roll no and lab number e.g. i22-XXXX\_Lab#.cpp
2. Take screen shot of running test cases of tasks.
3. Zip the .cpp file and screen shots (Do not create .rar file) with roll no and lab no. e.g. i22-XXXX\_Lab#.zip.
4. Submit the zip file on google class room.