UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS110 - Program design: Introduction

Practical 4 Specification

More operator Overloading and Basic Inheritance

Release Date: 05-09-2022 at 06:00

Due Date: 09-09-2022 at 23:59

Total Marks: 42

Contents

1 General instructions

- This practical should be completed individually, no group effort is allowed.
- Be ready to upload your assignment well before the deadline, as **no extension will be granted.**
- If your code does not compile, you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence of certain functions or classes).
- Failure of your program to successfully exit will result in a mark of 0.
- Read the entire assignment thoroughly before you start coding.
- **To ensure that you did not plagiarize, your code will be inspected with the help of dedicated software.**
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at http://www.ais.up.ac.za/plagiarism/index.htm.
- Unless otherwise stated, the usage of C++11 or additional libraries outside of those indicated in the assignment, will not be allowed. Some of the appropriate files that you have submit will be overwritten during marking to ensure compliance to these requirements **Please ensure you use C++98**

2 Overview

This practical will test your ability to:

- Create classes derived from a base class
- Implement pure virtual functions of a base class, in derived classes
- Perform operator overloading and conversion to a specified datatype

Ensure that you are familiar with theses concepts either from the lecture slides or the textbook (preferably both).

Inheritance relationships can be used to create a wide variety of interesting structures that can make programming more intuitive. On the surface level, a derived class is just a "subset" of a base class (the is-a relationship), only you can add more data to it. In reality it acts more like an extension to a base class.
Virtual functions make the structure more flexible; an operation defined for a base class could behave differently depending on which derived class performs that operation. Instead of just "adding" to a class, you can change almost everything about how it behaves to suite certain situations. For example the core data of the base class would be accessible to a child class, but how that data is used depends entirely on the child class.
You will be implementing a basic example of that in this practical, along with some operator overloading.

3 Your Task:

You will create a wrapper class for a 2D integer array called **TwoDArray**. Overloading some operators will provide output for usage outside the class.
For this use case once the 2D array is contained by an object of this class, we don't want it to be modified again. Thus, all the operations performed by the child classes will be read-only.
In Task 2, you will create the actual derived classes that each perform specific operations on the data contained in the 2-dimensional array. These classes will have access to the 2D array of the parent class, but will use the data in different ways despite the fact that the same function call is used ([ ]).

It's important to be exposed to some of the problems you may face when implementing inheritance and virtual functions. So for this prac, you need to implement all aspects of the classes yourself.
Be sure to pay attention to member names and case, as the header files will be overwritten.

A *main.cpp* file has been provided for testing, along with two other cpp files, one for each task. Be sure to *#include* and use the correct file for each task (since you will make an important change between tasks 1 and 2). Also pay attention to how the files are actually included; compile accordingly.
Hint: Remember what the *#include* directive actually does, in terms of the pre-processor.

what the fuck is a pre-processor?

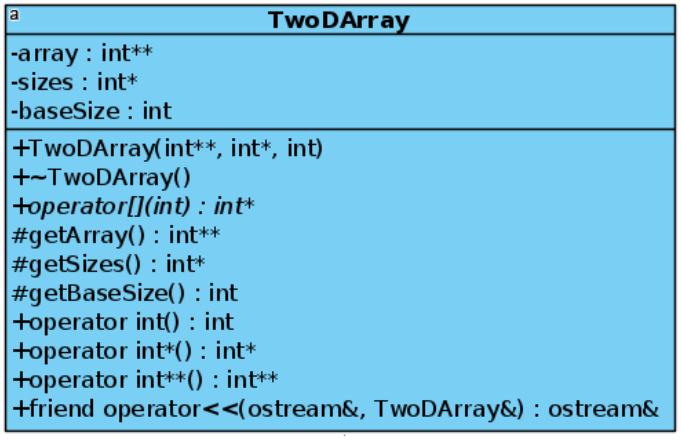

Figure 1: Class Diagram for TwoDArray. **Hint:** In UML, access modifiers are represented with −(private), +(public), #(protected)

3.1 Task1

This task lays the groundwork for the different functionality that will be provided by subclasses. For now the inheritance aspect can be ignored, and only the relevant methods should be implemented. You are given an empty header file, called *TwoDArray.h*, in which to define the class. Implement the methods in a CPP file, called *TwoDArray.cpp*.

3.1.1 TwoDArray

- Members:
  - `int** array`
    - This is the wrapped 2D array of integers
  - `int* sizes`
    - This is a one-dimensional array that contains the sizes of the arrays in *array* (remember that a 2D array is an array of arrays)
  - `int baseSize`
    - This is the base size of the array, or the size of array *sizes*
- Functions:
  - `TwoDArray(int** array, int* sizes, int baseSize)`
    - The constructor for the class
    - Initialize all members appropriately
    - Remember to make a deep copy
  - `~TwoDArray()`
    - The destructor, which de-allocates memory used by the class
  - `friend std::ostream& operator<<(std::ostream& os, TwoDArray& tda)`
    - The overload of the *stream extraction operator* which conveniently puts formatted output into the ostream object it operates on.
    - Each value in a row is inserted into the output stream separated by spaces (no space at the end), and after all values have been inserted they are terminated by a newline. This is repeated for all rows.
  - `int** getArray()`
    - This is used to get access to the wrapped array. Since anyone can create class that derives from TwoDArray, we don't necessarily want them to intentionally or unintentionally modify the array data. For this reason we have this method declared as protected, to be used by the class itself and child classes only. It should return a **copy that does not modify the original data**.
  - `int* getSizes()`
    - The same reasoning as before is applied here. This method will be used by child classes to get a deep copy of the *sizes* array.
  - `int getBaseSize()`
    - Returns the base size member.
  - `operator int**()`

    - When assignment occurs of an object of this class to a variable of type *int***, this conversion method is called. This is to allow "the outside" access to the contained array. Normally the getters from above would simply be made public instead of private, but this is to demonstrate an alternative. **NB:** This sort of conversion realistically means that only one operation can be performed per type. They can however still be convenient.
    - This *int*** conversion functions the same as *getArray()*.
- `operator int*()`
    - This conversion functions the same as *getSizes()*.
- `operator int()`
    - This conversion returns the *baseSize*.
- `virtual int* operator[]() =0`
    - A pure virtual overload of the subscript operator
    - This is implemented in Task 2, and can be commented/omitted for now in the header file. There will **not** be an implementation of this method in the *TwoDArray* class.

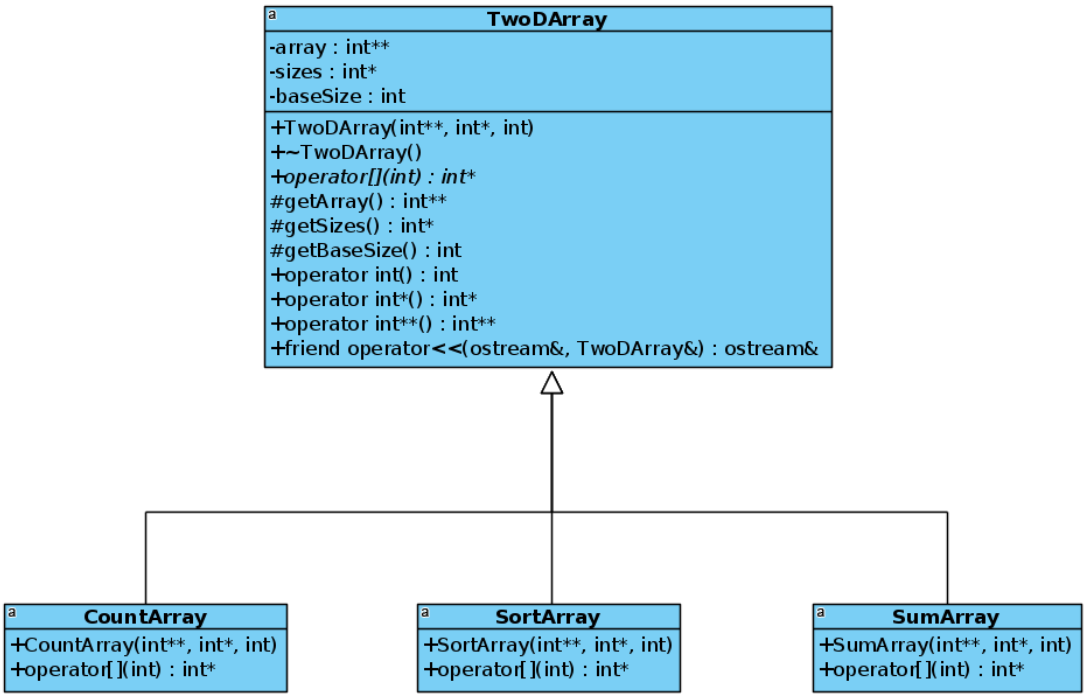

Figure 2: The arrow represents classes inheriting from a base class

3.2 Task2

Now you will implement the inherited classes and the means to access their functionality, *operator*[] (subscript). Each will have different operations performed on the contained array. More specifically, a subarray at a specific index will be operated upon. All of these classes derive **publicly** from *TwoDArray*. While implementing the constructors for these classes, keep in mind that the parameters should also be passed into the constructor for the base class (remember the member variables are **private**, not **protected**). Remember to include/uncomment the pure virtual function in *TwoDArray.h*.
Also note the return type of *operator[]*, which is *int**. In this way, one can return both a numeric value (in the form of a dynamic *int*) and a dynamic array.

In C++ fashion for all the derived classes, you may assume that **the index passed into the function will always be valid**. It is, of course, the responsibility of the programmer using your class to know what they are doing.

3.2.1 CountArray

This class counts the number of integers in a subarray at an index. For example, if the *int* array at index *k* of the *int*** array has 5 elements, the value 5 is returned.

- Functions:
  - `CountArray(int** array, int* sizes, int baseSize)`
    - The parameters passed in are the same as those passed into *TwoDArray*
  - `int* operator[](int);`

    - The implementation of the virtual function from *TwoDArray*.
    - Accessing an instance of the object with the subscript operator and an index, should perform the calculation on the array at the specified index of the 2D array.

3.2.2 SumArray

This class sums all the elements of a subarray. For example, if a small array had only the values *2, 0, 4*, the value 6 would be returned (in the form of a dynamic int).

- `SumArray(int** array, int* sizes, int baseSize)`
    - The parameters passed in are the same as those passed into *TwoDArray*
- `int* operator[](int);`
    - The implementation of the virtual function from *TwoDArray*.
    - Accessing an instance of the object with the subscript operator and an index, should perform the calculation on the array at the specified index of the 2D array.

3.2.3 SortArray

This class sorts a given subarray, smallest to largest, then returns the sorted array. The internal arrays should remain unchanged.

umm... the class returns something?
or is it the array that is a member of the class that is returned

- `SortArray(int** array, int* sizes, int baseSize)`
    - The parameters passed in are the same as those passed into *TwoDArray*
- `int* operator[](int);`
    - The implementation of the virtual function from *TwoDArray*.
    - Accessing an instance of the object with the subscript operator and an index, should return a sorted array.

Remember to test your code well. Feel free to use and extend the given main.cpp, Task1.cpp and Task2.cpp files.

These are basic classes, more for proof of concept. This idea could be expanded to create subclasses for any operation, on any type of data that the class might hold.

4 Uploading

Upload the following files in an archive:

- TwoDArray.h, TwoDArray.cpp
- SumArray.h, SumArray.cpp
- CountArray.h, CountArray.cpp
- SortArray.h, SortArray.cpp