Part D – Polymorphism

Templates

Workshop 9

(V0.9 – on the day of your lab, before submission, make sure there are no new updates or corrections)

In this workshop, you will work with a templated class that will allow for its member data and functions to operate on types supplied through a parameter list. The class will be a container similar to an array. It will hold an array of numbers of unknown time and perform some operations on them via operator overloads.

LEARNING OUTCOMES

Upon successful completion of this workshop, you will have demonstrated the abilities

- To code a templated class
- To specify the type to implement in a call to a templated class
- To specify a constant numeric parameter to a templated class
- To describe to your instructor what you have learned in completing this workshop

SUBMISSION POLICY

This workshop has only in lab section.

> *The DIY parts are now replaced with the project milestones under the **OOP-Project** repository.*

Workshop 9 is to be submitted during the workshop period from the lab. To get the mark for this workshop you must attend the lab.

Start the workshop few days earlier and come to the workshop sessions with questions and attendance.

Remember: you must be present at the lab in order to get credit for the workshop.

If you attend the lab period and cannot complete the workshop during that period, ask your instructor for permission to complete after the period.

All your work (all the files you create or modify) must contain your name, Seneca email and student number.
You are responsible to back up your work regularly.
Ask your professor if there are any additional requirements for your specific section.

Citation and Sources

When submitting Workshops, Project and assignment deliverables, a file called **sources.txt** must be present. This file will be submitted with your work automatically.
You are to write either of the following statements in the file "**sources.txt**":

I have done all the coding by myself and only copied the code that my professor provided to complete my workshops and assignments.
Then add your name and your student number as signature

OR:

Write exactly which part of the code of the workshops or the assignment are given to you as help and who gave it to you or which source you received it from.
You need to mention the workshop name or assignment name and also mention the file name and the parts in which you received the code for help.
Finally add your name and student number as signature.

By doing this you will only lose the mark for the parts you got help for, and the person helping you will be clear of any wrongdoing.

Late Submission Penalties:

Late submission: 1 to 6 days -50% after that submission rejected.
- If the content of sources.txt is missing, the mark for the whole workshop will be **0.**

WORKSHOP DUE DATES

You can see the exact due dates of all assignments by adding -due after the submission command:
Run the following script from your account (use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

~profname.proflastname/submit 244/NXX/WS09/lab -due<ENTER>

COMPILING AND TESTING YOUR PROGRAM

All your code should be compiled using this command on matrix:
g++ -Wall -std=c++11 -o ws (followed by your .cpp files)
After compiling and testing your code, run your program as follows to check for possible memory leaks: (assuming your executable name is "ws")
valgrind ws <ENTER>

Utils Module

For those of you who would like to reuse their previously developed functions and classes in later workshops, an empty module is added to the workshop called `Utils` (`Utils.cpp` and `Utils.h`). These files are empty and will be compiled with your workshop.
If you don't have any interest in reusing your previous functions, leave these files as they are, and they will not have any effect in the workshop.
If you have anything that you would like to add to the project, place their code in the `".cpp"` file and the definitions and prototypes in the `".h"` file. This header file will be included in the tester program.
In any case when submitting your work, make sure that these two files (empty or not) are submitted along with your workshop.

Lab

In this workshop we will be writing a container class called **NumbersBox**. It's a box of numbers which will have some functionality via operators overloads mainly. The structure is similar to an **array** with some defined behavior. It is a templated class so it may work with different numeric types.

NUMBERSBOX MODULE

The **NumbersBox** module will be the only module in this workshop and it is composed of just a header file due to being a *template*. As such all code will be in **NumbersBox.h**.

Be sure incorporate proper use of header guards, namespaces and avoid memory leaks.

TEMPLATE PARAMETERS:

Create a templated class **NumbersBox** with the following template parameters:

1. A **generic type** (T type for example) that will be the representative type of Numbers held in the NumbersBox.

PRIVATE MEMBERS:

A **NumbersBox** will have the following data members:

- `name` – This is a statically allocated **character array** with a maximum length of 15 characters excluding the nullbyte. It's the name of the Box.
- `size` – This is an **integer** that is the size of the Box and the amount of numbers it carries currently.
- `items` – This is a **dynamically allocated array** of the **generic type** (eg T type) declared from the template parameters. These are our numbers in the Box.

PUBLIC MEMBERS:

Constructors

This class will have two constructors:

- A Default constructor that should set the **NumbersBox** to a safe empty state.
- A two arg constructor that takes in an **integer** representing the size of the Box and a **constant character pointer** representing the name of the Box. There is no maximum size for the Box but it should hold at least one item. The number of items in the Box should be based on the passed in size and initialized with the *needed memory*. The items themselves do not need to be set values at this stage.

  If all the parameters are valid, set the state of the Box appropriately and if invalid set it to a safe empty state.

Member Operator Overloads

```
T& operator[](int i)
```
This operator overload allows access to the Box's items via an index much like what you'd expect from an array of a fundamental type. The purpose of this overload is to return the value of the item in the Box at the given index **i**.

For example if we had a NumbersBox object called mybox that has values (1, 2, 3) in its items array, **mybox[0]** will call this overload to return the value from the 0 index in the items array member data (the value of 1).

```
NumbersBox<T>& operator*=(const NumbersBox<T>& other)
```
This operator will take a secondary NumbersBox reference and multiply each item in our Box by the opposing Box's item thus modifying the current object's items. It returns the current object.

For example, if the current box has values (1, 2, 3) and the opposing box has (2, 2, 2), the resulting values of our current box will be (2, 4, 6).

This function will only attempt to perform the above operation if the **sizes of the both Boxes** are equal and do nothing other than return the current object.

```
NumbersBox<T>& operator+=(T num)
```
This operator will resize our items array **dynamically** and incorporate the num passed into the resized array. If given a box mybox with values (1, 2, 3) and we were to perform:

mybox += 4;

mybox following this operator should now have values (1, 2, 3, 4) in its items array.

Other members

```
ostream& display(ostream& os) const
```
If the Box is in a safe empty state, the display function prints
`Empty Box<newline>`
and returns the os.
Otherwise the display function prints out the details of the Box and its items in the following format. Given a box with the values (1, 2, 3) the output should look like this:

```
Box name: [name]<newline>
1, 2, 3<newline>
```

Refer to the sample main output for reference

Helper operators

`ostream& operator<<(ostream& os, NumbersBox<T>& box)`
This **templated** operator will allow the NumbersBox class to interact directly with ostream objects to display the details of the Box in the manner of:

cout << mybox;

MAIN MODULE

```
/*************************************************************************
// OOP244 Workshop 9: Templates
// File NumbersBoxTester.cpp
// Version 1.0
// Date       2020/3/20
// Author     Hong Zhan (Michael) Huang
// Description
// Tests the Calculator template and its functions
//
// Revision History
// -----------------------------------------------------------
// Name            Date            Reason
// Michael
/////////////////////////////////////////////////////////////////
*************************************************************************/
#define _CRT_SECURE_NO_WARNINGS

#include <iostream>
#include <iomanip>
#include "NumbersBox.h"
using namespace std;
using namespace sdds;

ostream& line(int len, char ch) {
  for (int i = 0; i < len; i++, cout << ch);
  return cout;
}
ostream& number(int num) {
  cout << num;
  for (int i = 0; i < 9; i++) {
    cout << " - " << num;
  }
  return cout;
}

int main() {

  cout << "Create an int and double NumbersBox" << endl;
  line(64, '-') << endl;
  number(1) << endl;
  NumbersBox<int> intbox(3, "Int Box");
  NumbersBox<double> doublebox(3, "Double Box");

  cout << "Populate intbox and doublebox and display" << endl;
  line(64, '-') << endl;
  number(2) << endl;
  intbox[0] = 22;
  intbox[1] = 33;
```

```
  intbox[2] = 44;
  doublebox[0] = 1.5;
  doublebox[1] = 2.5;
  doublebox[2] = 3.5;
  cout << intbox << endl;
  cout << doublebox << endl;

  cout << "Peform *= operation on intboxs, first of not matching sizes then matching siz-
es" << endl;
  line(64, '-') << endl;
  number(3) << endl;
  NumbersBox<int> bigintbox(5, "big int box");
  intbox *= bigintbox;
  cout << intbox << endl;

  intbox *= intbox;
  cout << intbox << endl;

  cout << "Perform *= operation on doubleboxs, first of not matching sizes then matching
sizes" << endl;
  line(64, '-') << endl;
  number(4) << endl;
  NumbersBox<double> emptydoublebox;
  doublebox *= emptydoublebox;
  cout << doublebox << endl;

  doublebox *= doublebox;
  cout << doublebox << endl;

  cout << "Perform += on intbox and doublebox and add a new item to it" << endl;
  line(64, '-') << endl;
  number(5) << endl;
  intbox += 999;
  doublebox += 999.999;
  cout << intbox << endl;
  cout << doublebox << endl;

  return 0;
}
```

Main Output

```
Create an int and double NumbersBox
----------------------------------------------------------------
1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1
Populate intbox and doublebox and display
----------------------------------------------------------------
2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2
Box name: Int Box
22, 33, 44

Box name: Double Box
1.5, 2.5, 3.5

Perform *= operation on intboxs, first of not matching sizes then matching sizes
----------------------------------------------------------------
3 - 3 - 3 - 3 - 3 - 3 - 3 - 3 - 3 - 3
Box name: Int Box
```

```
22, 33, 44

Box name: Int Box
484, 1089, 1936

Perform *= operation on doubleboxs, first of not matching sizes then matching sizes
---------------------------------------------------------------
4 - 4 - 4 - 4 - 4 - 4 - 4 - 4 - 4 - 4
Empty Box

Box name: Double Box
1.5, 2.5, 3.5

Box name: Double Box
2.25, 6.25, 12.25

Perform += on intbox and doublebox and add a new item to it
---------------------------------------------------------------
5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5 - 5
Box name: Int Box
484, 1089, 1936, 999

Box name: Double Box
2.25, 6.25, 12.25, 999.999
```

LAB SUBMISSION

To test and demonstrate execution of your program use the same data as the output example above.

If not on matrix already, upload sources.txt, NumbersBox module, NumbersBoxTester.cpp program to your matrix account. Compile and run your code and make sure that everything works properly.

Then, run the following script from your account during the lab (use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

~profname.proflastname/submit 244/NXX/WS09/lab<ENTER>

and follow the instructions generated by the command and your program.

IMPORTANT: Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

REFLECTION

Study your final solutions for each deliverable of the Workshop, reread the related parts of the course notes, and make sure that you have understood the concepts covered by this workshop. **This should take no less than 30 minutes of your time and the result is suggested to be at least 150 words in length.**

Create a file named `reflect.txt` that contains your **detailed description of the topics that you have learned** in completing this workshop and mention any issues that caused you difficulty and how you solved them. Add any other comments you wish to make.

The reflection should also contain any thoughts and learning from the final project milestones when possible.

Reflection Submission

You can submit your reflection 4 to 6 days after your lab session. Upload reflect.txt to your matrix account. Then, run the following command from your account (use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

~profname.proflastname/submit 244/NXX/WS09/reflect <ENTER>

and follow the instructions generated by the command.

To see the when you can submit your reflection and when the submission closes, (like any other submission) add **-due** to the end of your reflection submission command:

~profname.proflastname/submit 244/NXX/WS09/reflect -due<ENTER>