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 two components:

- **lab**: to be completed by the end of the day of your lab;
- **Reflection**: to be completed by the end of the week of your lab (Sunday @ 23:59:59);

You must be present at the lab for the duration of the entire session in order to get credit for the *lab* portion.

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.

Reflection

After the workshop is fully completed (meaning the **lab** was already submitted), create a text file named **reflect.txt** that contains your detailed description of the topics that you have learned in completing this particular workshop and mention any issues that caused you difficulty and how you solved them. Add any other comments you wish to make.

A section below provides some questions and expectations you should aim to answer at a minimum for the reflection. Aiming higher than that will be to your benefit as a poorly done reflection can incur a **mark reduction** (see Submission Penalties below).

This reflection is a mandatory part of the workshop submission.

Late Submission Penalties:

If you do not follow the submission the policies, the following penalties will be applied:

- If the **lab** is submitted past the due date, it will be receive **0%.**
- If the submitted **reflection** is deemed to be insufficient in depth or does not demonstrate a strong understanding of the core concepts used in the Workshop, a **maximum reduction of 40%** may be applied the overall mark of the Workshop.
- If any of *lab* or *reflection* is missing on rejection date, the total mark of the workshop will be **0%**.

To see the dues for a deliverable, use:

```
  ~cornel.barna/submit 200/NXX/WSXX/lab -due<ENTER>
~cornel.barna/submit 200/NXX/WSXX/reflect -due<ENTER>
```

Compiling and Testing Your Program

Compile all your code using this command on matrix:

```
g++ -Wall -std=c++11 -o ws file1.cpp file2.cpp ...<ENTER>
```

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>
```

Lab

In this workshop, we will be writing a container class called **NumbersBox**. It's a collection of numbers that will have some functionality via operator 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;

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 200/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 should 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.

You can submit your reflection until Sunday @ 23:59:59 of the week of your lab session. Upload reflect.txt to your matrix account. Then, run the following command from your account (use your section ID to replace NXX, i.e., NAA, NBB, etc.):

```
~cornel.barna/submit 200/NXX/WSXX/reflect<ENTER>
```

and follow the instructions generated by the command.