Reflection

----------------------------------------------------------------------------------------------------------------------------

Name: Hashmeet Singh Saini

Seneca ID : hsaini28

Student No. : 153070214

Date : Septemeber 29th, 2022

I have done all the coding by myself and only copied the code that my professor provided to complete my workshops and assignments.

----------------------------------------------------------------------------------------------------------------------------

I found this week’s workshop to be the toughest so far in this course. It helped me learn a variety of new concepts like instantiating a template class, specializing member function of a templated class, coding a template class and making as well as specializing a template class variable. I found this workshop to be tough because of the syntactic problems in specializing anything. I solved this problem by going through the notes, visiting the campus for an in-class lab and clarifying my doubts with the professor.

1)the reason for specializing the push() member function.

I specialized the push function for derived template class UniqueQueue. The normal push function for this class works by comparing every element in the queue with the item to be pushed. If any repeated element is found, the function ignore it. This process of comparison is handled by line 26 of my UniqueQueue.h file as:-

if (Queue<T, 100>::operator[](i) == item)

This works fine for the other testing types like int and Dictionary but there are additional specifications for comparison of float type which ignore the precision of 0.005. To achieve that it became necessary for me to create a specialized push function which is accessed whenever the UniqueQueue has type double. Whenever this condition is true, the comparison is governed by line 43 of UniqueQueue.h file as-

if ((std::abs(operator[](i)) >= std::abs(item) - 0.005) && (std::abs(operator[](i)) <= std::abs(item) + 0.005)) foundflag = true;

2) the reason for defining the class variable outside the class definition.

As you can see at line 22 of my Queue.h file:

static T m\_dummyObject;

that my class variable m\_dummyObject is static. This is a requirement of static member variables, since they are not unique to every individual class object, they are explicitly defined in the global scope, outside the class. In my Queue.h file this has been done at line number 69 and 70.

template <typename T, unsigned int CAPACITY>

T Queue<T, CAPACITY>::m\_dummyObject = {};

The m\_dummyObject was declared inside the class but was initialized outside the class. Since this is also a template class we must also resolve the class argument list during initialization. If one wants too initialize a class variable inside the class, they must use the inline keyword.

3) answers to the other questions in these specifications.

a) Explain in the reflection why we do not split this module into \*.h and \*.cpp like you did in the previous workshops.

Since normally classes are defined in the header file but implemented in the .cpp file, it is common for someone to try to implement a template class in the same way. But this process is not the same for template classes because it is defined on certain conditions in the header file mentioned in the template<> keyword whose scope is till the end of the definition of the class. For example in my code Queue.h has been templated by using the codeline18:

template <typename T, unsigned int CAPACITY>

the scope of this template is only till the closing bracket(}) of class. The only way I could implement its methods in a .cpp file, was to use the above-mentioned code line repeatedly before the implementation of every method in the .cpp file which is a lot more tedious job for the coder. So to prevent any sorts of errors and complicated coding, it is better to implement a template class inside the header file.

b)In the reflection **justify** why it was necessary to add each one of the members you have decided to add.

I added 4 members to the Dictionary module. First was a default class constructor to initialize the class attributes m\_term and m\_definition to their default rest state. The second member I added was the display function which defined how the dictionary class is outputted. Third was a helper function, a << operator overload that enabled its use with the dictionary class, making it more intuitive to code and output. Both the display function and the << operator helped in code line 54 of my Queue.h file:

os << m\_queue[i] << std::endl;

If I hadn’t added the said members, this code line couldn’t have worked with Dictionary objects.

The 4th was the helper function, == operator overload that defined how two instances of Dictionary class are to be compared. This enabled the comparison of two Dictionary objects later in the UniqueQueue.h file at line 26

if (Queue<T, 100>::operator[](i) == item) {

This operator overload defines that when == operator is called for two dictionary objects, it compares their m\_term attributes and if they are same, the objects are considered equal to each other.