|  |  |  |
| --- | --- | --- |
|  | **Rochester Institute of Technology**  **Golisano College of Computing and Information Sciences**  **School of Interactive Games and Media**  **2145 Golisano Hall – (585) 475-7680** |  |

**Data Structures & Algorithms for Games & Simulation II**

**IGME 309**

**A1: Templated Priority Queue**

**Summary:**

Following the Visual Studio project settings exercise and using C++ on Visual Studio 2019, code a templated priority queue data structure in such a way that you are able to use it as a container of variables of any data type; you will need to ***sort them by a metric***.

**Details:**

A queue is a data structure able to hold an undetermined amount of information; it can contain as many entries as necessary because it possesses the ability of resizing depending on the needs of the data to be allocated.

The C++ std library contains a class that behaves like a queue, it is named std::deque (not to be confused with a std::queue which behaves like a queue but offers no random access to the content). This data type starts with one space of allocated memory and then doubles the size of the contained information every time it runs out of space. std::queue is a FIFO class, which means that every time you take pop out something from the data structure, you would be taking out the first pushed entry (compared to the std::stack that removes the last pushed entry):

|\_| |\_| |\_| |2| |\_| |\_|

|\_| -> push(0) -> |\_| -> push(1) -> |1| -> push(2) -> |1| -> pop() -> |2| -> pop() -> |\_|

|\_| |0| |0| |0| |1| |2|

Notes:

* Push means add to the data structure, pop means take from it.
* std::deque offers both a FIFO and LIFO behavior but the provided example focusses on how to use the std::deque as a FIFO data structure.

For this homework assignment you will need to create a data structure that behaves as the std::deque only for the FIFO behavior, it needs to be a templated class so you can create a queue of any possible type. It also needs to sort the information by a metric, you are to decide that metric.

You need to call your class MyQueue exactly as it will be test with a unitest by the grader. I will also provide the unitest so you get an idea of what you are missing.

For example if you are allocating numbers you may sort those numbers from small to large, if you are allocating strings you may sort them by memory address or by length; as you do not know what data you will be getting as input due the Templated nature of this class the data type of the input needs to have an operator for sorting (your metric), Lets say you are allocating integers, then > or < make sense, now lets say that you are allocating “Alberto” class objects if the Alberto class does not provide an < operator then you cannot sort and your class is fine. Assume that I will try your class with an Alberto object that has the operator > and < defined.

Rephrasing what I said above, to create a Templated Priority Queue data structure, it needs to implement sorting and it needs to be able to accept any data type so long operations like <, >, <=, >= are defined. The only difference between a regular Queue and a Priority Queue is that the data gets automatically sorted as you push or pop. For example a priority queue that takes integers and receive as arguments: [10] [50] [2] [50] would sort it as [2-10-50-50] being 2 the front of the Queue, of course being a templated data structure it implies that you can store any data on your queue even another data structure, but if that object does not contain the methods mentioned above it would not be able to sort them, after all, how do you know if an object of the class “Alberto” is larger than another “Alberto” if the object does not define it on its own. At a minimum any class object thrown at your queue will have to define the < operation for sorting.

You will need to program the following:

A templated class able to contain any data type and have the behavior of a queue, therefore it should contain the following methods:

* **The rule of three** – As any other class aside of the rule of 3 a constructor is required for this class, the use of a queue is fundamentally an array of pointers, your constructor should allocate space for the queue; A copy constructor that takes another queue and clone its contents in a new object; A copy assignment operator that reinitialize an existing queue object and copy the content from the queue to be copied onto it; A destructor that deallocates all the memory of the queue.
* **Push** – Add information to the queue. If the queue ran out of space to contain data, it should allocate more space to contain the new information. You may sort here or make an additional helper method that does it for you. Any sorting algorithm is fine for this assignment.
* **Pop** – Removes information of the queue from the front, to make things easier you do not need to reallocate the queue contents into a smaller container when you remove information from it; you can just shift the entries one index down. You can keep the same size for your pointer array and have an index to know where the last element is stored. You do not need to return any information from the queue just eliminate the entry.
* **Print** – A method that writes in console the contents of the queue (provided the operation is defined for the data type, all basic data types as integers, doubles, chars, floats, shorts, etc are very well defined. When your queue use a class object, the class needs to have a definition on what happens when you call an std::cout via a friend function, you can assume any class object I use for testing the homework will contain this definition)
* **GetSize** – A method that returns an integer containing the amount of entries (not the size of the queue, but how many entries the queue has) as an integer.
* **IsEmpty** – A method that returns a Boolean that informs whether or not a queue contains no entries.

You should test your class with the provided main.cpp so we can grade your assignment. It is a templated class, so everything will be on the MyQueue.h file and that is the only thing you will need to provide. Take a look at the example code provided (using the std::deque).

Grade breakdown:

|  |  |
| --- | --- |
| List class | 100% of grade |
| The Rule of Three | 35% (10% Each + Constructor) |
| Push | 15% |
| Pop | 10% |
| Print | 10% |
| GetSize | 5% |
| IsEmpty | 5% |
| Sorting | 20% |

You are allowed to use any additional methods to aid you, but these helper methods should be called automatically from within the required methods if needed as you are not allowed to change the Main.cpp file at all. Should go without saying, but remember templates only use a .h or .hpp file DO NOT create a .cpp file for your class

For this assignment you are only required to upload one header file named MyQueue.h. The grader will download only the MyQueue.h file from MyCourses for grading.

You are allowed to look for a sorting algorithm online but if you do so, you need to cite your source, if you are using any AI engine you also need to provide your prompts. AI should only be used for finding errors on your code or extra bits. At the end of the day, you will be tested on the knowledge of your code.

**Penalties:**

If you copy an existing library (std::vector included) or someone else’s work that decision will result on an automatic failure (0%) of this assignment. You are expected to code everything from scratch; you are allowed to review others materials to verify formulas (if any) but all work must be original.

-100% Your class is not called MyQueue

-10% to 20% For memory leaks (depending on the gravity of the leak).

-10% Not commenting your code properly.

-10% Not following naming conventions, you can use any naming convention that you see fit for this assignment but be consistent with them: naming variables var, dog, house, thing, foo, etc., is NOT ok. (except in the case of a generic class in which you can use Foo, Bar, Baz, etc)

Deliverable:

*Just one file MyQueue.h NOT in a zip*

*Possibly a readme.txt if you need to clarify something. (additionally, not zipped)*

***Submit to the dropbox labeled: A01 – Templated Priority Queue***