CS 105 (C++)

Assignment 7: Centipede

I. Overview

In this assignment, you will implement a queue class template, to be used in a simple centipede-style interactive display. In addition to the use of a template, this project will require the implementation of copy control as well as dynamic memory management.

> **Note**: To allow you to focus on this week's C++ topics rather than on the complexities of data structures, this project is designed around a queue which has a ***fixed size***. From its first creation and throughout its entire existence, the queue will contain exactly `QUEUE_SIZE` (specified below) elements.

You will use the existing file `a7_main.cpp` (download here) to handle interaction and display, and you will complete your `Queue` template implementation entirely within the file `Queue.h`. The given `a7_main.cpp` should not be modified; doing so may cause your code to work incorrectly during grading.

Your implementation must meet the following requirements:

- `Queue.h` must properly use preprocessor directives to prevent multiple inclusion.
- Each method of class template `Queue` must be defined outside of the class declaration, but still within `Queue.h`. (Because of the way templates are compiled, both the declaration and the definition of the template must be in the same file.)
- `Queue.h` must use a preprocessor directive to define `QUEUE_SIZE` as `30`.
- Your Queue class template must include the following members (where `Type` refers to the parameterized type of the template):
  - A private pointer to `Type`, which will point to a dynamically allocated array of `Type` which is used for the queue's internal storage. (This will be the only private member and the only data member.)
  - A default constructor, which allocates the array with size `QUEUE_SIZE`.
  - A copy constructor.
  - A destructor.
  - An assignment operator.
  - An access function called `pushAndPop`, which takes a reference to a new item of type `Type` to be pushed onto one end of the queue, and returns an item of type `Type` that has been popped off the other end of the queue. (Note that the queue size is not changed by this operation.)
- As an exercise in programming to specifications, the `Queue` class template should contain exactly the members specified, and no others.

- As always, proper handling of dynamic memory will be extremely important because of the potential for damaging and hard-to-find bugs. (In this assignment, the memory management is simpler than in previous assignments, but must still be completed with great care.)

Note: This assignment's `a7_main.cpp` file makes use of the ncurses library for display, which requires the option `-lncurses` (that's the letter `l`, not the number `1`) to g++ during compilation, as shown here:

```
g++ a7_main.cpp -lncurses -o a7
```

II. Grading

The following is a list of specific assignment requirements, along with the grade value for each (out of a total of 10 points for the assignment).

- Minimum Requirements
  - `Queue` functions properly as used by `a7_main.cpp`.
  - Your work must be submitted in the file `Queue.h`.
  - This file (when combined with the given `a7_main.cpp`) must compile on a department UNIX machine with the following command:
    ```
    g++ a7_main.cpp -lncurses -o a7
    ```
  - Before evaluation, your code must be submitted via `turnin`, using the following command on a department UNIX machine:
    ```
    turnin --submit dlessin a7 Queue.h
    ```
- Graded Elements
  - Preprocessor directives used to prevent multiple inclusion in `Queue.h`.
  - All `Queue` member functions defined outside of the class declaration, but within Queue.h, as described in Section I.
  - `QUEUE_SIZE` defined as described in Section I.
  - Each member of Queue must be implemented as described in Section I.
  - No members beyond those specified in Section I can be included.
- Provisional Dealbreakers
  - Due to the importance of proper handling of dynamic memory, you will lose **50%** of your programming assignment grade if a valgrind evaluation of your work shows any memory leaks or errors *other than memory which is "still reachable"*. (Due to this assignment's implementation, there will be some memory listed as "still reachable" in valgrind's leak summary output. This is completely acceptable, and will not affect your grade.)
  - **However**, in recognition of the difficulty of this task, you may resubmit your work at any time before the end of the course for regrading on this portion of your score. (And of course, I'll be happy to help you find and fix any problems.)
  - Valgrind evaluation is performed as follows:
    - Prepare your executable for valgrind by compiling with debugging information on and optimization off. Use the following command for this (Note that "`O0`" is a capital letter "o" followed by the number zero.):
      ```
      g++ -g -O0 a7_main.cpp -lncurses -o a7
      ```
    - Run valgrind by prepending the following to your normal command (including all normal

arguments):

```
valgrind --leak-check=yes
```