21.6 Lab 3 - Stacks and Queues Part 2

In this assignment you are going to create a Stack and a Queue--two data structures which are similar in many ways in terms of how you write them, though their overall goals are different. The details for this assignment are in the file downloaded from Canvas.

In this second part, you will only be implementing the Queue data structure. Both stacks and queues can be implemented in a variety of ways, but in this case you will be implementing a queue using a dynamically allocated array.

You may NOT use std::vector for this assignment. Any submissions using std::vector will receive a grade of 0. The data must be stored in a dynamically allocated array.

Memory Leaks

When dealing with memory, you will inevitably encounter memory leaks in your program. Simply stated, a memory leak is memory that was allocated with new, and never deleted.

Detecting memory leaks can be quite a challenge, but for this assignment there are two files which can help with this:

- leaker.h
- leaker.cpp

To use these files, initially you don't have to do anything at all. Simply being present as part of your files will let you use their functionality.

These files track your memory allocations/deallocations--that is, they track every time your code calls `new`, and every time your code calls `delete`. At the end of the program, if the number of new/delete calls don't match up, errors will be reported. They will look something like this:

```
LEAKER: errors found!
Leaks found: 1 allocations (4 bytes).
unknown:unknown():0 memory leak: memory was not deallocated.
```

That tells you there is A memory leak… but doesn't tell you where. If you #include "leaker.h" in a file which calls new/delete in it (such as the .cpp file with a class definition in it, the error reporting will also print out the file and line number of where the memory was allocated, like this:

```
LEAKER: errors found!
Leaks found: 1 allocations (4 bytes).
main.cpp:main():13 memory leak: memory was not deallocated.
```

Very useful! This only tells you WHERE the memory was allocated... it's up to you to fix it. Other types of errors you might see could involve deleting something that hasn't been allocated (or wasn't initialized to **`nullptr`**), or possibly using the wrong **`delete/delete[]`**:

```
LEAKER: main.cpp:main():16 delete error: pointer was not allocated!
LEAKER: main.cpp:main():19 mismatch error: memory allocated at
main.cpp:main():17 with new[], deallocated with delete.
```

Proper usage of leaker.h

If you are going to **`#include "leaker.h"`** in any of your files, be sure to write the #include statement AFTER you include any built-in files, such as <iostream> or <string>. Why? If you #include "leaker.h" before any of the others, you will get a variety of compiler errors, as memory-leak detection involves changing the way new/delete work a little bit, and this will interfere with the built-in classes. (The exact details of all of this are a little more complex than this, but aren't necessary to use these files in your own projects.)

Due to the way zyBooks handles certain types of unit tests, in this assignment leaker.h will NOT be included in any files, so any leak errors will simply report file/lines as unkown:0. This is a relatively simple assignment, so it shouldn't be too problematic to find out where the new/delete issues are located.

LAB ACTIVITY | 21.6.1: Lab 3 - Stacks and Queues Part 2 | 18 / 18

Downloadable files

`main.cpp` , `leaker.h` , and `leaker.cpp` **Download**

File is marked as read only Current file: **main.cpp**

```
26  }
27
28
29  cout << "\nMaking float ABQ...\n";
30  ABQ<float> floatABQ(10);
31  cout << "Size: " << floatABQ.getSize() << endl;
32  cout << "Max Capacity: " << floatABQ.getMaxCapacity() << endl;
33  cout << "\nEnqueueing items...\n";
34  for (float i = 1; i < 5; i += 0.5f)
35  {
36     floatABQ.enqueue(i);
37     cout << "\nEnqueued " << i << endl;
```