CS 3370

Program 1 – "A dynamic circular buffer

Because output devices (screen, printer, disk, etc.) are generally slower than the processors, data to be output is often buffered – put into memory where the output device can get it whenever it is ready. Because these buffers are used over and over, they are sometimes implemented as a *circular buffer*.

In a circular buffer, you (the processor) start at the front, and add data. In the meantime, the output device would be pulling data off the buffer behind you. When you reach the end of the buffer, you go back to the beginning and continue to fill. Of course, this assumes that the output device has already cleared out the data at the beginning of the buffer! (More on that later.)

You implement a Circular Buffer class. Inside the class, you will have a dynamic (heap) array. Keep integer indexes (or pointers, if you prefer), for the head and the tail of the data in use. As you insert characters into the buffer, you will advance the head index, and as you remove characters, you advance the tail index. When either reaches the end of the buffer, it will wrap around to the beginning.

What happens when you want to add more characters than the buffer has room? You have to grow the internal array. Allocate a new array which is one CHUNK larger than the previous one. (CHUNK is currently defined as 8). Copy the data from the old array into the new one, preserving the order of the data, of course. While the data could start somewhere in the middle of the old array, and even wrap around the end, just start the data at the beginning of the new array. Be sure to clean up; don't leak memory!

Note that you have a size(), which is the number of elements currently in the buffer. You also have capacity(), which is the maximum number of elements you can currently have without growing the buffer. For example, if your buffer has only 'X' in it, the size is 1, but the capacity is 8.

The constructor takes an argument which is the number of elements you want to reserve space for. So if the argument is 5, the capacity will be 8; however, the initial size will still be 0. Note that if the argument is 0, the initial capacity will be 0; you will not allocate any space initially.

The elements of the Circular buffer are characters. There are insert functions for a single character, and for multiple characters, with a number of how many characters to read. For convenience, there is a string version as well. For removing characters, there is a get function that gets a single character, and one that gets multiple characters. For convenience, it returns them as a string. If you request more characters than are in the buffer, you get all of them.

For the purposes of verifying the correctness of your implementation, there is a function called examine(). It returns a string representing the contents of the buffer, enclosed in the square brace characters. Represent unused elements with '-'. Note: when you remove an element from the buffer, IT INCORRECT TO REPLACE THAT CHARACTER WITH A '-'. DO NOT change the element in the buffer; just advance the tail index/pointer.

It may be nice to occasionally shrink the buffer. Implement a function that reduces the size of the buffer. It's kind of the reverse of growing the buffer.

```
#include <string>
using std::string;

class CircBuf {
        const size_t CHUNK { 8 };
        // Insert your data and private functions here.

public:
        CircBuf(size_t reserve = 0);            // Number of elements you want it to be able to hold to start
with.
        ~CircBuf();
```

```
        size_t  size();
        size_t  capacity();

        void            insert(char);
        void            insert (const char*, size_t sz);
        void            insert(const string&);
        char            get();
        string  get(size_t);
        string  flush();        // Returns a string with all the characters, AND shrinks the buffer to zero.
        string  examine();
        void            shrink();       // Reduces the unused space in the buffer.
};
```

Implement your CircBuf class in a file called CircBuf.cpp. Turn in *only* your CircBuf.h and your CircBuf.cpp. Also turn in a screen shot of your program output. If you are using MS Visual Studio, do NOT turn in the entire VS project folder.

To build your project, compile your CircBuf.cpp with the main.cpp provided, as well as test.h. The test.h header file contains macros that execute test cases and keep a tally of the passes and failures. There are currently 51 test cases in main.cpp. You must pass all the test cases in main.cpp to get full credit. Note, however, that the grader reserves the right to run additional test cases against your program.

Professional Tip: When you craft a program, you not only must solve the problem, but you must also write code that presents the solution clearly to a reader of your code. Before you turn your program in, make an edit pass through it with the idea that you may need to explain it to someone else (or to yourself in the future :-). *Also*: summarize what you learned and problems that you overcame in finishing this project. Put this in a comment block in your source code. (5 points out of 100)

Assessment Rubric

Competency ↓	**Basic →**	**Proficient →**	**Exemplary**
Memory Management	Every **new** has a corresponding **delete** (no memory leaks); destructor does the right thing; Use **delete [ ]** when deleting heap arrays.	Data is properly re-centered when growing the array	
Memory efficiency	Every free space in the array is available to the user (follow the spec for **front_** and **back_**)		
Clean Code	No magic numbers (use named constant for CHUNK); use simple, meaningful variable names	No repeated code (refactor); No unnecessary code	Simplest possible logic to fulfill program requirements; Use **std::copy** instead of a loop when growing the array (*Don't* use memcpy!)
Other	Use **size_t** for non-negative quantities	Always #include <cstddef> for **size_t**; use #ifndef–#endif guards instead of #pragma once (portable)	Use std:: (either as a prefix or in a using *declaration*) in your header file for names imported from the standard library (*never* use **using namespace std** in header files)