P2 MYString v1

- Due Apr 17, 2018 by 12:20pm
- Points 100
- Available Apr 10, 2018 at 12am - Jun 15, 2018 at 11:59pm 2 months

This assignment was locked Jun 15, 2018 at 11:59pm.

Program Note: For this assignment the normal C++ string and cstring functions can not be used (except >> and << with cstrings). You may not #include them. You must write all the functions that will be used.

Classes, Objects, Pointers and Dynamic Memory

Program Description: This assignment you will need to create your own string class. For the name of the class, use your initials from your name. For example: my string class would be called LHString. Since each of you will be using a unique class name, I will use the name MYSting as the generic name of the class (but make sure for you program, that you use the correct name based on your name).

The MYString objects will hold a cstring and allow it to be used and changed. We will be changing this class over the next couple programs, to be adding more features to it (and correcting some problems that the program has in this simple version).

Below will be the details of the class. Since this is your first class that you will be writing, I would recommend that you write and test this class a few member functions at a time. For example you may want to start by writing the two constructors and the ***at*** function and then test them with some simple code in main like this:

```
MYString testStr("hello");

cout << "Testing testStr.at() function\n"; // testStr.at(0) should return 'h'
   for(int i = -1; i < 10; i++ ){          // purposely testing out of bounds
       cout << i << ":" << testStr.at(i) << ' '
            << static_cast<int>(testStr.at(i)) << endl;
   }
cout << testStr.c_str() << endl;
cout << endl << endl;
```

Your first job will be to create and test the MYString class. As you write each member function, you should then write some code in main to test that the member function works well. Notice in the above example I was purposely using indexes outside of the bounds of the string to check if this would cause problems.

Your MYString class needs to be written using the .h and .cpp format.

Inside the class we will have the following data members:

Member Data	Description
char * str	pointer to dynamic memory for storing the string
int cap	size of the memory that is available to be used (start with 20 char's and then double it whenever this is not enough)
int end	index of the end of the string (the '\0' char)

The class will store the string in dynamic memory that is pointed to with the pointer. When you first create an MYString object you should allocate 20 spaces of memory (using the new command). The string will be stored as a cstring in this memory. {**NOTE**: Here on Canvas in Files\Review Lectures from 131\ there is a pdf of the pointer and cstring lecture slides from CS 131 to use as a review}.

You need to be allow your string that is held in the class to be able to be larger than 20 chars. It should start with a capacity of 20, but when needed it would grow in increments of 20. The capacity should always be a multiple of 20.

For example if we were storing the string "cat" in a MYString object, our data member would have the following values:

str	starting addr of dynamic array
cap	20
end	3

Dynamic array:

c	a	t	\0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

The MYString class will need to have the following member functions:

Programming Note: Write and test one or two functions at a time	
Member Functions : return type	**Description**
MYString( )	**Default Constructor:** creates an empty string
MYString (const char*)	creates a string that contains the information from the argument **example:** MYString greeting("hello there wise one");
length( ) : int	the length of the string ("cat" would return 3)
capacity( ) : int	the total amount of memory available for use
at(int index) : char	returns the character at a certain location (at(0) for a "cat" would return 'c'). If the index is not inside the string (negative or too large) then return '\0'
read(istream& istr) : bool	read one string from the istream argument (could be from cin or an ifstream variable). This should work just like the >> operator. When reading in, you can assume that you will not read in a string longer than 99 characters. This function will return true if it was able to read (remember >> operator will return true if it is able to read from a file). **For simplicity sake,** you could create a local char array variable 100 that you first read into and then you could copy from this char array into your dynamic memory.
write(ostream& ostr) : void	write the string out to the ostream argument, but do not add any end of line (could be cout or an ofstream variable)
compareTo(const MYString& argStr) : int	compares the object string (Ostr) to the argument string (Astr) by subtracting each element of Astr from Ostr until a difference is found or until all elements have been compared Ostr < argStr returns a negative number Ostr > argStr returns a positive number Ostr ==argStr returns zero
c_str( ) : const char *	return a pointer to a constant cstring version of the MYString object
setEqualTo(const MYString& argStr): void	this does the assignment operation aStr.setEqualTo(bStr) would change aStr so that it would contain the same information as bStr

Main Program Requirements:

- create a vector of MYStrings that is size 100
- read each of the words from the file called "infile2.txt" (the file is out in Files section under Program Resources). You can call the read function directly on the

indexed vector<MYString>. (do not use vector's push_back function. See programming note below*). Example: while (words[count].read(fin)) {...}

- as you are reading the words in, keep a count of how many words were read in.
- After you have read in all the words from the file, resize your vector to the correct size based on your count of the number of words read in.

- sort the MYStrings from smallest to largest (this will be based on the ASCII encoding) using Bubble Sort
- output the sorted words to outfile.txt file
  - 6 words per line (use setw(13) to space them out....the setw command should **not** be in the write member function)

***Programming Note: Do not use** the **push_back()** member function of vector, because this won't work for this program (it calls the copy constructor of our MYString class, which we haven't written).

Turn in: For turn in, you will have three files: your main program, the .h file and the .cpp file. For all programs which include class definitions, I want you to place them in that order (main, interface/header (.h), and implementation (.cpp))

In addition to the program header documentation (above main), you should also have class documentation (and author info: name, Sect #, and explanation of the class) at the top of the .h file.

Ways to lose points:

- if your main file does not contain the program header with a program description and short function description to accompany the function prototypes.
- your interface (.h) file should have a class description about what the class does
- your code should also be consistently indented as talked about in class, and shown in the book
- you can not use global variables unless it is a const
- you should use good variable names (descriptive, and start with lower case letter)
- proper placement of { and } (a } should not be placed at the end of a line)
- you need staple to keep your papers together (folding a corner or using a paper clip are not good enough)
- you need to have the three source files (mystring.h, mystring.cpp, and the main) as well as a print out of the output file
- if you did not split the MYString class into separate files

Comments: Comments are a way of documenting a program (explaining who did what and how). All programs for the rest of the course are required to have the following program header documentation (above main, and in any interface files (.h)) and inline documentation to explain any tricky pieces of code.

```
////
// Name: Nancy Programmer
// Section: A, B, or S
// Program Name: Hello World
//
// Description: A brief description of the program.  What does the
//  program do (not how it does it: for example, it uses loops)?  Does
//  the program get input?  What kind?  What information is output
//  from the program and to where (screen or file)
////

#include <…>

.......the rest of the program
```