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

Gang of Four and other friends

Program Description: This assignment is continuing to make your MYString class so that it is more complete and usable. The data within your MYString class will be the same as before, but now we are adding the Gang of Four functions and some operators. I have highlighted the new/modified functions in red below.

Your Gang of four and overloaded operator functions should have "normal behavior". As you write each function, you should make a little test to verify that the function is working correctly.

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

Programming Note: *Write and test one or two functions at a time.* ***Remember*** *to mark member functions that do not change member data with* ***const***	
Member Functions : return type	**Description**
MYString( )	**Default Constructor:** creates an empty string
MYString(const MYString & mstr)	**Copy Constructor:** creates a string that is a duplicate of the argument
MYString (const char*)	creates a string that contains the information from the argument **example:** MYString greeting("hello there wise one");
~MYString()	**Destructor:** release the dynamic memory

= operator : lvalue&	**Replaces** setEqualTo(const MYString& str): void	**Assignment Operator:** this does the assignment operation aStr.setEqualTo(bStr) would change aStr so that it would contain the same information as bStr
Non-const [ ] operator : char	**Like:** at(int index) : char	returns the character at a certain location. No Error checking. Fast but dangerous
>> operator: istream&	**Replaces** read(istream& istr) : istr	read a string from the istream argument (could be from cin or an ifstream variable) {when reading in, you can assume that you will not read in a string longer than 99 characters} This function will now return the istream which is the normal behavior.
<< operator : ostream&	**Replaces** write(ostream& ostr) : ostr	write the string out to the ostream argument, but do not add any end of line (could be cout or an ofstream variable) This function will now return the ostream which is the normal behavior.
< operator > operator == operator : all return a bool	**Replaces/Uses** compareTo(const MYString& str) : int	You could either replace the compareTo function with the 3 comparison operators or you could make the compareTo function private and then have the operators call compareTo. If you have the operators use compareTo, then you should make compareTo private. 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 < Astr returns a negative number Ostr > Astr returns a positive number Ostr ==Astr returns zero
+ operator : MYString		this function will add the rvalue onto the back of the lvalue. For example if you had the following values inside two of your strings and added them “bat” +

	“man”, then you would return a MYString that contains “batman”
length( ) : int	the length of the string ("cat" would return 3)
capacity() : int	The amount of spaces that is currently reserved for possible use before growing
c_str( ) : const char *	return a pointer to a constant cstring version of the MYString object

Main Program Requirements:

Read the words from the input file into MYString variables. As the words are being read append them to each other until you have combined 5 words together into one jumbo MYString (**Note**: To keep the simpler, the input file will be the same as program 2). Then take this jumbo MYString and add it into the vector with pushback (so it will be using your copy constructor). The input file probably won’t have an even multiple of 5, so you will need to allow the loop to end due to reaching end of file (not due to knowing the word count).

Once you have read, combined, and pushed your MYStrings into the vector, then once again sort the strings into ascending order based on their ASCII value. Then output them to the screen with one combination string per line. When outputting the strings also include the length and capacity of the string on the same line. Here is an example of one line of output:

```
Theyseemedamazinglybusy.I  25:40
```

You much create your own sort routine (can’t call a library, bubble sort is fine) and you can’t use the swap function (I want you to be using your assignment operator).

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 this order (main, interface/header (.h), and implementation (.cpp), program output)

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

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
- make sure you mark member functions with const if they don't modify the member data
- 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