COEN 79L - Object-Oriented Programming and Advanced Data Structures

Lab 5

- **String Class**

You will complete the implementation for **a version of the Standard Library's string class**. Please download and read through this header file: `mystring.h`

You are to write the implementation for the above header file in a file named `mystring.cpp`. Please note that the name of the actual class that you are writing is `string`, not `mystring`. For this reason, the use of namespaces is particularly important to this lab, as your class will have the same name as that of the Standard Library's `string` class.

You are allowed to use functions from the library `<cstring>` to assist you in writing your functions for things like copying between strings or appending strings to one another. You are to make sure, however, that you use safe versions of these strings. For example, you should use `strncpy` and `strncat`, not their less secure alternatives of `strcpy` and `strcat`. Finally, you may NOT use the "`strdup`" function.

Some of the important functions to be implemented are:

- A new constructor that has one parameter (a character). The constructor initializes the string to have just this one character.
- An insertion function that allows you to insert a string at a given position in another string.
- A deletion function that allows you to delete a portion of a string.
- A replacement function that allows you to replace a single character in a string with a new character.
- A replacement function that allows you to replace a portion of a string with another string.
- A search function that searches a string for the first occurrence of a specified character.
- A search function that counts the number of occurrences of a specified character in a string.
- A more complex search function that searches through a string for an occurrence of some smaller string.

Please refer to the header file for the full list of functions that should be implemented.

Before submitting, make sure to write a test file to thoroughly test all of the functions of your class. I recommend that, for this test file, you do not use the namespace `std` (i.e. don't have the line "`using namespace std`" in your code). It will likely be easier to just use your namespace: `coen79_lab5`. If you don't like putting `std::` in front of `cout`, `cin`, and `endl` every time you write them, you can use lines like "`using std::cout`", "`using std::cin`", etc.

Hints:

With our design:
A programmer can use strings with no worries about how long a string becomes

- The plan is to have a private member variable that is a dynamic array to hold the null-terminated string
- Each member function ensures that the array has sufficient room, increasing the size of the array whenever necessary
- You can also explicitly set the size of the dynamic array that holds the null-terminated string, by calling the reserve function
- Similar to the dynamic bag class, explicit resizing is not required. It is just a convenience for efficiency

```
class string {
... private:
    char *characters;
    std::size_t allocated;
    std::size_t current_length;
};
```

The use of the member variables is controlled by the invariant of the class:

- The string is stored as a null-terminated string in the dynamic array that characters points to
- The total length of the dynamic array is stored in the member variable allocated
- The total number of characters prior to the null character is stored in current_length, which is always less than allocated

Implementation

Constructors

The constructor is responsible for initializing the three private member variables
The initialization occurs by copying a character sequence from an ordinary null-terminated string:

```
string::string(const char str[ ])
{
    current_length = strlen(str);
    allocated = current_length + 1;
    characters = new char[allocated];
    strcpy(characters, str);
}
```

The constructor makes use of the library function `strcpy` to copy the null- terminated string from the parameter str to the dynamic array characters

The destructor

Since the class uses dynamic memory, you must implement a destructor, your destructor will return the string's dynamic array to the heap

Comparison operators

The string class has six comparison operators.
The prototype for the equality comparison is:

```
bool operator ==(const string& s1, const string& s2)
// Postcondition: The return value is true if s1 is identical to
s2.
// Library facilities used: cstring
{ return (strcmp(s1.characters, s2.characters) == 0);}
```

Note that `strcmp` must be a friend, because it accesses private member variable characters

The reserve function

• Similar to the dynamic bag's reserve function

```
 void reserve(size_t n);
//  Postcondition:  All  functions  will  now  work  efficiently
(without
// allocating new memory) until n characters are in the string.
```

• Programmers who use our string class never need to activate reserve, but it can be used to improve efficiency
• Our implementations of other member functions can also activate reserve whenever a larger array is needed

The operator >> (the extraction operator)

• After skipping the initial white space, our string input operator reads a string—reading up to but not including the next white space character (or until the input stream fails, which might occur from several causes, such as reaching the end of the file)
• The function `isspace` (from the <cctype>l ibrary facility) can help
This function has one argument (a character)
It returns true if its argument is one of the white space characters
• We can skip any initial white space with this loop:

```
 while (ins && isspace(ins.peek( )))
     ins.ignore( );
```

The loop also uses three `istream` features:

- In a boolean expression, the name of the istream (which is ins) acts as a test of whether the input stream is bad:
- If ins returns a true value, then the stream is okay o A false value indicates a bad input stream
- The peek member function returns the next character to be read (without actually reading it)
- The ignore member function reads and discards the next character
- After skipping the initial white space, your implementation should set the string to the empty string, and then read the input characters one at a time, adding each character to the end of the string
- The reading stops when you reach more white space (or the end of the file)
- Once the target string reaches its current capacity, our approach continues to work

correctly