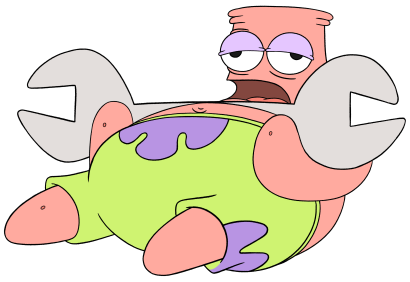
CS202

Assignment 6: Custom String Class



Description

As you know, the `string` class in C++ maintains a list of characters. Up to this point, you might have thought that a string is a primitive data type since it was introduced at a surface level in your previous CS course. In fact it is a custom type but it does seem to almost be a primitive type since strings are popular in programming. You will create a custom string class which we will call `myString`, the class can be seen below:

```
class myString
{
public:
  myString();
  myString(const char str[]);
  myString(const myString&);
  myString(char);
  char& operator[](int);
  int length() const;
  const myString& operator=(const myString&);
  const myString& operator=(const char[]);
  const myString& operator=(char);
  myString operator+(const myString&) const;
  myString operator+(const char[]) const;
  myString operator+(char) const;
  const myString& operator+=(const myString&);
  const myString& operator+=(const char[]);
  const myString& operator+=(char);
  bool operator==(const myString&) const;
  ~myString();
  friend myString operator+(const char[], const myString&);
  friend myString operator+(char, const myString&);
  friend ostream& operator<<(ostream& out, const myString&);
  friend istream& operator>>(istream& in, myString&);
private:
  int sLength;
  char * s;
};
```

Description of class

Each member of the class will perform/contain the following:

- `myString::myString()` - will be the default constructor that sets `s = NULL` and `sLength = 0`
- `myString::myString(const char str[])` - will be the constructor that takes a string literal `str` and assigns it to the `s` array, and the `sLength` field will be set to the length of the string literal, you will have to iterate through the `str` array to determine the size, details are explained on page 3, for example `myString s1("Hello");` invokes this constructor and it sets the `s` field to hold `Hello` and `sLength` will be set to 5
- `myString::myString(const myString& str)` - will be the default copy constructor that creates a deep copy of the `str` object, so `sLength = str.sLength` and we need to allocate `s` to size `str.sLength` and copy one character at a time from `str.s` to `s`, `str` could hold an empty string
- `myString::myString(char c)` - will be a constructor that sets the `sLength` to 1 and the only character in the array `s` will be `c`, the parameter passed
- `char& myString::operator[](int i)` - overloads the `[]` operator which will return `s[i]`
- `int myString::length() const` - returns the `sLength` field
- `myString& myString::operator=(const myString& str)` - will be the default assignment operator that assigns the `myString` object on the left side of the operator with the `str` object which is the object on the right side, you must perform a deep copy, so first you need to deallocate the `s` array if it is not pointing to `nullptr`, assign the `sLength = str.sLength` and set `s` to `str.sLength` by copying each character one at a time from `str.s` to `s`, `str` could be an empty string so in that case `sLength = 0` and `s = nullptr`
- `myString& myString::operator=(const char str[])` - will be an assignment operator that assigns the object that called the function (the left side object) with a string literal `str`, a deep copy must be performed, for example `myString s1 = "Hello World"` will trigger this function, so `s1`'s `s` field will contain `"Hello World"` and `sLength` will be set to 11, so once again first you will deallocate `s` if it is not pointing to `nullptrL`, and the count the number of characters in `str` and allocate `s` to that size and then perform the copying which will copy each character from `str` array into `s` array one character at a time, `str` could be an empty string so make sure your code sets everything correctly in that case
- `myString& myString::operator=(char rhs)` - will be an assignment operator that that assigns the object that called the function (the left side object) with a character `rhs`, a deep copy must be performed, for example `myString s1 = 'A'` will trigger this function, once again you will have to deallocate `s` if it is not pointing to `nullptr` and set `s[0] = rhs;` and `sLength = 1`, you will perform similar stuff as you did for the other assignment operators
- `bool myString::operator==(const myString& str) const` - will be the comparison operator that returns `true` if the `myString` object on the left side of the `==` sign is equal to the `myString` on the right side of the operator (the right side object will be denoted by `str`), if both strings are empty strings then return `false`
- `myString myString::operator+(const myString& str)` - will overload the + operator which will concatenate the `myString` object on the left side of the operator with the `str` object which will be on the right side, a new concatenated `myString` object will be returned, so you will need to create a temporary `myString tmp;` object and set `tmp.sLength = sLength + str.sLength;` and allocate `tmp.s` array to that length, and then `s` will be copied into `tmp.s` from 0 to `sLength - 1` and then you will copy `str.s` into `tmp.s` from `sLength` to `tmp.sLength - 1`, remember either `s` or `str.s` could be `nullptr`
- `myString myString::operator+(const char rhs[])` - will overload the + operator which will concatenate the `myString` object on the left side with a string literal `rhs` on the right side, a new concatenated `myString` object will be returned, for example `s1 + "Hello"` will trigger this function, **hint:**

you could use functions and constructors that you implemented already inside this function to avoid writing new code, make use of the `this` pointer which points to itself (the object that called this function), `*this` dereferences itself, once again `s` could be `nullptr` and `rhs` could be an empty string

- `myString myString::operator+(char rhs)` - will overload the + operator which will concatenate the `myString` object on the left side with a character `rhs` on the right side, a new concatenated `myString` object will be returned, for example `s1 + 'A'` will trigger this function, **hint:** you can once again reuse other functions, once again `s` could be `nullptr`
- `myString& myString::operator+=(const myString& rhs)` - will overload the += operator which will concatenate the `myString` object on the left side with the `rhs` object on the right side and will assign the new concatenated `myString` to the left side object, basically `s1 += s2` will be equivalent to `s1 = s1 + s2`, **hint:** make use of `this` pointer and the `operator+(const myString& str)` function, once again `s` or `rhs.s` could be `nullptr`
- `myString& myString::operator+=(const char rhs[])` - will overload the += operator which will concatenate the `myString` object on the left side with the string literal `rhs` on the right side and will assign the new concatenated `myString` to the left side object, basically `s1 += "Hello"` will be equivalent to `s1 = s1 + "Hello"`, **hint:** you can reuse again, but once again `s` could be `nullptr` and `rhs` could be an empty string
- `myString myString::operator+=(char rhs)` - will overload the += operator which will concatenate the `myString` object on the left side with a character `rhs` on the right side and will assign the new concatenated `myString` to the left side object, so `s1 += 'a'` will be equivalent to `s1 = s1 + 'a'`, **hint:** you can reuse again and once again `s` could be `nullptr`
- `myString::~myString()` - will be the destructor for the class that will deallocate the character array when the object goes out of scope
- `myString operator+(const char lhs[], const myString& rhs)` - will be a friend function that allows us to concatenate a string literal with a `myString` object, for example `"Hello"+ s1` will trigger this function, **hint:** create a temporary `myString` object from `lhs`, now you have two `myString` objects and the you could reuse some of the operators but once again be able to handle `nullptr` pointers
- `myString operator+(char lhs, const myString& rhs)` - will be a friend function that allows us to concatenate a character with a `myString` object, for example `'A' + s1` will trigger this function, **hint:** you can reuse other functions again but be able to handle `nullptr` pointers
- `ostream& operator<<(ostream& out, const myString& str)` - friend function that overload the `<<` operator that will output the string using the `ostream` variable `out`, you will need to output one character at a time until the end of `str.s` array is reached, if `str.s == nullptr`, then nothing will be printed
- `istream& operator>>(istream& in, myString& str)` - friend function that overloads the `>>` operator that will read in a string and assign it to `str.s` array, so basically you will need to use `in.get(ch);` to read a character from the input where `ch` will be a character declared inside the function, you will keep reading in a character and appending it to the `str.s` array until a `'\n'` character is read (**hint:** you could use `operator+=(char)` function to append the character read into a `myString` object assuming you implemented the `+=` function)
- `int sLength` - holds the length of the `myString` object
- `char * s` - holds the character array that will represent the string inside the `myString` object

Every string literal in C++ is terminated by a null character, `'\0'` character. Suppose you declare the following object in main

```
myString object("Sunday Funday");
```

Then the constructor `myString::myString(const char[] str)` would be invoked when `object` is being instantiated. You can traverse the string literal in the following way.

```
int i = 0;

while (str[i] != '\0')
  i++;

//the value of i contains the length of str
```

And then you can allocate the `char` array to `this->s` array, and then just deep copy `str` array into `this->s` array.

Sample Run

```
% make
g++ -Wall -Wextra -pedantic -std=c++11 -g -c main01.cpp myString.h
g++ -Wall -Wextra -pedantic -std=c++11 -g -c myString.cpp
g++ -Wall -Wextra -pedantic -std=c++11 -g -o m01 main01.o myString.o
g++ -Wall -Wextra -pedantic -std=c++11 -g -c main02.cpp myString.h
g++ -Wall -Wextra -pedantic -std=c++11 -g -o m02 main02.o myString.o
g++ -Wall -Wextra -pedantic -std=c++11 -g -c main03.cpp myString.h
g++ -Wall -Wextra -pedantic -std=c++11 -g -o m03 main03.o myString.o
g++ -Wall -Wextra -pedantic -std=c++11 -g -c main04.cpp myString.h
g++ -Wall -Wextra -pedantic -std=c++11 -g -o m04 main04.o myString.o
g++ -Wall -Wextra -pedantic -std=c++11 -g -c main05.cpp myString.h
g++ -Wall -Wextra -pedantic -std=c++11 -g -o m05 main05.o myString.o
g++ -Wall -Wextra -pedantic -std=c++11 -g -c main06.cpp myString.h
g++ -Wall -Wextra -pedantic -std=c++11 -g -o m06 main06.o myString.o
g++ -Wall -Wextra -pedantic -std=c++11 -g -c main07.cpp myString.h
g++ -Wall -Wextra -pedantic -std=c++11 -g -o m07 main07.o myString.o

% ./m01
HelloWorld
Hello#World
HelloRockview
World!

% ./m02
ByeByeBye
ByeEveryone
ByeEveryone!

% ./m03
GreenDay
ADay

% ./m04
Same string
Not the same string
Not the same string

% ./m05
Enter a string: Zip

Zip
Enter another string: ZipZap
```

```
ZipZap

% ./m06
B r a c k e t   o p e r a t o r   t e s t
BRACKET OPERATOR TEST
```

Specifications

- Document your functions
- Make sure your code is memory leak free
- Do not use `string` or `cstring` functions

Submission

Submit the source file to code grade by the deadline

References

- Supplemental Video https://youtu.be/LD-962yQOsk
- Link to the top image can be found at https://www.deviantart.com/lazoofficial/art/Patrick-SpongeBob-Where-s-the-Leak-Ma-am-Full-903350786