CS 202 - Assignment 4

Mr. Piotrowski

March 2022

1 Introduction

The past 3 assignments have hopefully been fun for you, working with some cool examples and concepts. This assignment will not be similar unfortunately. Your job is to implement a small string class. Best of luck!

The main focus of this assignment is dynamic memory. You will be working **only** with character arrays (*no std::strings allows*) which are navigated using a pointer inside your string class. Not only will you be graded on your assignment working as the output shows, but also the Valgrind output. We will make sure all memory you are using is allocated properly, deep copied, within bounds, and properly deallocated at the end. Any errors output by valgrind may be subject to point deductions.

You may not use the STL string class once in this assignment.

2 Coding

There are 3 files just like Assignment 3. Main.cpp hosts the process, CS202_String.h holds the class prototype, and CS202_String.cpp holds the class function definitions. All of your code will be in CS202_String.cpp.

2.1 Main.cpp

Please look inside this file. I have heavily commented main() to show you exactly which functions are being called from the CS202_String class. Consider commenting out all the code at the start. As you define functions for your class, uncomment a section of code in main to test it. It would be best to go in order.

2.2 NULL Terminated Strings

PLEASE READ THIS

NULL Terminated Strings are how C++ determines which character is the last character in a character string. For example, in your programs you may type the following code at some point:

```
1    string myString = "Patty";
```

When "Patty" is typed, is actually a character array, not a string. How does the string constructor know that "Patty" is 5 chars long without you telling it? Well, C++ sneaks a NULL character at end of the array. So the array actually looks like this:

```
1    "Patty\0"
```

Or when represented as an array of chars:

```
1    { 'P' , 'a' , 't' , 't' , 'y' , '\0' }
```

Two of the members within the class below require you to build a string from a **const char***. This is a character array. You must iterate through this array until you see the '\0', at which point you will know how long the string it.

2.3 CS202_String

MAS	Member	Const?	Static?	Type	Name	Params
private	variable	no	no	char*	charArr	N/A
private	variable	no	no	int	size	N/A
public	constructor	N/A	N/A	N/A	CS202_String	
public	constructor	N/A	N/A	N/A	CS202_String	const char*
public	constructor	N/A	N/A	N/A	CS202_String	const CS202_String&
public	destructor	N/A	N/A	N/A	~CS202_String	N/A
public	function	no	no	void	PushBack	const char*
public	function	no	no	void	DeleteAt	int
public	function	yes	no	void	PrintString	
public	function	yes	no	int	GetSize	
public	function	yes	no	char	At	int
public	function	yes	no	int	FindSubString	const CS202_String&
public	function	no	yes	CS202_String	Concatinate	const CS202_String&, const CS202_String&
public	operator	no	N/A	void	operator=	const CS202_String&

You have to include code for everything in this class except the prototype, the operator, and the PrintString function.

1. **Default Constructor -** This constructor initializes an empty string. The pointer should be initialized to nullptr and the size to 0.
2. **Parameter Constructor -** This constructor takes a character array as a parameter. I hope you read the above passage about NULL Terminated Strings. Figure out how long the array is, save that into the size variable, allocate enough memory to store the array into the class pointer, and then deep copy the contents of the parameter array into the class pointer array.
3. **Copy Constructor -** This constructor takes a CS202_String as a parameter. Initialize the new object to be exactly the same as the parameter. Remember, you must make a deep copy. Reference the Assignment Operator for help!
4. **Destructor -** In the destructor you need to deallocate the char array.
5. **void PushBack(const char*) -** This function acts like the same push_back function in strings and vectors. It takes in a character array and adds it to the end of the string. I hope you read the above passage about NULL Terminated Strings.
    - Figure out how long this parameter array is.
    - Allocate a new array of a new size which is the string objects size plus the size of the parameter array.
    - Deep copy the old object's array into the new array.

- Deep copy the elements of the parameter array into the remaining spots of the new array.
- Deallocate the old object's array.
- Copy the address of new array into the pointer for the old object's array.
- Adjust the size member to reflect the new size.

6. **void DeleteAt(int) -** This function will delete one character out of the string. The character to delete is located at the int parameter. For example, if the parameter is 1, delete the character at the 1 index.

   This function will require you to decrement the size, allocate a new array that is 1 spot smaller, and deep copy all elements except the element that was deleted into the new array. You will then need to delete the old object's array and copy the address of the new array into the pointer of the old object's array.

   If the index happens to be out of bounds, print this message and do nothing:

```
"WARNING: CS202_String::DeleteAt(int) - Nothing Was
Deleted."

```

7. **void PrintString() const -** I gave this function to you. Prints the string to the terminal.

8. **int GetSize() const -** Returns a copy of the size.

9. **char At(int) const -** Returns a copy of the character located at the index. The index is passed in as parameter. For example, if the string was "Patty", At(3) would return 't'. If the index happens to be out of bounds, print this message and return a NULL character:

```
 "ERROR: CS202_String::At(int) - Index out of bounds
    , returning NULL character."

```

10. **int FindSubString(const CS202_String&) const -** This will likely be the hardest function. The goal is to find the parameter string within the object string. Consider this example: "Patty" has lots of substrings. A substring is a string contained within a larger string. Listed are all possible substrings for "Patty" : "P", "a", "t", "y", "Pa", "at", "tt", "ty", "Pat", "att", "tty", "Patt", "atty", and "Patty".

    Let's pick one of the substrings to explain how the function works. We will use "att". This substring is located starting at index 1 within "P**att**y".

This function will return 1 because that is where it was found.

Try to find the parameter string within the object string. If it is found, return the index it starts at. If it is not found, the parameter string is empty, or the parameter string is larger than the object string, return -1.

11. **static CS202_String Concatinate(const CS202_String&, const CS202_String&) -** This static function takes in 2 strings and returns a brand new string that is a concatinated version of the Left parameter with the Right parameter. For example, "Loaf" concatinated with "Patty" is "LoafPatty". This function requires deep copies.

12. **void operator=(const CS202_String&) -** I gave this operator to you. I deallocate the old object array, copy the size, and then make a deep copy of the parameter string.

3 Compiling Your Code

Compile your code using

```
g++ -std=c++11 -g Main.cpp CS202_String.cpp
```

Run your code using

```
./a.out
```

Use valgrind!

```
valgrind --leak-check=full ./a.out
```

4 Comprehension Questions

Submit question answers via a WebCampus quiz.
Please answer the following questions. Your answers should be short (1 line!).

1. Can you allocate arrays of size 0 and size 1? If so, why do you think C++ lets you do it?

2. Explain the 3 things involved in the Rule of 3. Are they all present in this assignment?

3. What 2 things does the Rule of 3 aim to prevent?

5 Submission

Submit cpp file to CodeGrade.