Max Guidry

9-21-16

String Class Assessment

**1.0 Requirements Documentation**

**1.1 Description of Problem**

**Name:** Create a String Class

**Problem Statement:** Create a string utility class that makes working with character arrays easier to manage.

**Problem Specification:** This class needs to include functions that:

1. Get the length of a string.

2. Access a character at any given index of a string.

3. Compare two strings to see if they are exactly the same.

4. Append one string to another.

5. Prepend one string to another.

6. Return a string as a basic constant C-style string (const char \*).

7. Convert all letters to their lowercase counterparts.

8. Convert all letters to their uppercase counterparts.

9. Find a sub-string within a string.

10. Find a sub-string within a string starting at a certain index.

11. Replace a sub-string found in the string.

12. Set a string to an input C-style string.

**1.2 Input Information**

The user inputs two strings that are then used within the functions.

**1.3 Output Information**

The console displays results of each of the functions.

**1.4 User Interface**

Not Applicable.

**2.0 System Architecture**

**Member Functions in the class:**

PROTOTYPE: accessIndex(int)

DESCRIPTION: function of type char that finds whatever character is at the index that it is passed

PARAMETERS: function takes one parameter of type int that is used for the number of the index

PRECONDITION: must have an index

POST CONDITION:

VISIBILITY: public

PROTOTYPE: getLength();

DESCRIPTION: returns the length of the instance of MyString using the member variable m\_Length

PARAMETERS: takes no parameters

PRECONDITION: none

POST CONDTION:

VISIBILITY: public

PROTOTYPE: compareStrings(MyString)

DESCRIPTION: compares the string-(m\_Data)- from the instance being called to another one that is passed to it and returns true or false

PARAMETERS: takes one parameter of type MyString

PRECONDITION: must have a second string to pass into the function

POST CONDTION:

VISIBILITY: public

PROTOTYPE: append(MyString)

DESCRIPTION: appends the MyString that is passed to it to the end of the instance of the class and returns a pointer to the new string

PARAMETERS: takes one parameter of type MyString

PRECONDITION: must have a second string to pass into the function

POST CONDITION:

VISIBILITY: public

PROTOTYPE: prepend(MyString)

DESCRIPTION: prepends the MyString that is passed to it to the beggining of the current string of the instance being called and then returns a pointer to the new string

PARAMETERS: takes one parameter of type MyString

PRECONDITION: must have a second string to pass into the function

POST CONDITION:

VISIBILITY: public

PROTOTYPE: ToLower()

DESCRIPTION: changes all letters in the string to their lowercase counterparts using the ascii value and returns a pointer to the new string

PARAMETERS: takes no parameters

PRECONDITION: none

POST CONDITION:

VISIBILITY: public

PROTOTYPE: ToUpper()

DESCRIPTION: changes all letters in the string to their uppercase counterparts using the ascii value and returns a pointer to the new string

PARAMETERS: takes no parameters

PRECONDITION: none

POST CONDITION:

VISIBILITY: public

PROTOTYPE: subString(char \*)

DESCRIPTION: searches the string for a set of characters that are passed to it

PARAMETERS: takes one parameter of type char \*

PRECONDITION: must be passed a character pointer

POST CONDITION:

VISIBILITY: public

PROTOTYPE: subStringAtIndex(int,char\*)

DESCRIPTION: searches the string for a set of characters that are passed to it starting at a certain index that is also passed to it

PARAMETERS: takes two parameters one of type int which is the index and one of type char \* which is the sub string being searched for

PRECONDITION: must be passed an index and a character pointer

POST CONDITION:

VISIBILITY: public

PROTOTYPE: replace(char\*,char\*)

DESCRIPTION: searches for a sub string then if it is found replaces that sub string with a different sub string

PARAMETERS: takes two parameters both of type char\* one is the sub string being searched for and the other is the sub string that replaces it

PRECONDITION: must be passed two character pointers

POST CONDITION:

VISIBILITY: public

PROTOTYPE: constantCStyleString()

DESCRIPTION: sets a const char \* to the value of m\_Data and then returns it

PARAMETERS: takes no parameters

PRECONDITION: none

POST CONDITION:

VISIBILITY: public

**Member Variables:**

DESCRIPTION: Character array that stores the string char m\_Data[255]

DESCTIPTION: Integer that stores the length of the string int m\_Length

**CPP FILE:**

#include "Classes.h"

MyString::MyString(char \*first)//constructor definition to set the value of the string

{

int i;

for (i = 0; first[i] != '\0'; i++)//loops through the array to set values until it reaches a null character

{

m\_Data[i] = first[i];

}

m\_Data[i] = '\0';

m\_Length = i;

}

int MyString::getLength()

{

return m\_Length;

}

//function accessIndex

//takes one argument of type int

//takes in an integer which is used as the index to return a character at that index

char MyString::accessIndex(int index)

{

return m\_Data[index];

}

//funtion compareStrings

//takes one argument of type MyString

//compares each character in the strings to check for equality

bool MyString::compareStrings(MyString second)

{

bool equalStrings;

for (int i = 0;; i++)//loops through the characters in the string comparing it to the second string

{

equalStrings = (m\_Data[i] == second.m\_Data[i]) ? true : false;//if the characters are equal then the equalStrings is set to true

if (equalStrings == false)//if the equalStrings is false then break out of the loop

{

break;

}

if (i > m\_Length&&i > second.m\_Length)

{

break;

}

}

return equalStrings;

}

char \* MyString::append(MyString second) // appends the second string on the end of the first string

{

int oldLength = m\_Length;//get current length of the string

int i;

for (i = 0; second.m\_Data[i] != '\0'; i++)//loop setting the original string with the second string appended on the end

{

m\_Data[i + oldLength] = second.m\_Data[i];

}

m\_Data[i + oldLength] = '\0';

m\_Length = i + oldLength;

return m\_Data;

}

//function prepend

//takes one argument of type MyString

//prepends the second string on the beginning of the first string which now has the second string appended on the front of it

char \* MyString::prepend(MyString second)

{

char copy[255];//array of characters ment to copy the current string

int oldLength = second.m\_Length;//get current length of second string

int i = 0;

for (int i = 0; i < m\_Length; i++)//fills the copy array with the current first string

{

copy[i] = m\_Data[i];

}

for (int i = 0; i < m\_Length; i++)//uses the copy array to shift the current first string

{

m\_Data[i + oldLength] = copy[i];

}

for (i = 0; i < oldLength; i++)//places the second string where the first string is no longer

{

m\_Data[i] = second.m\_Data[i];

}

m\_Data[i + m\_Length] = '\0';

m\_Length += second.m\_Length;

return m\_Data;

}

//function lowercase

//takes no arguments

// makes all letters in the string lowercase

char \* MyString::ToLower()

{

for (int i = 0; i < m\_Length; i++)

{

if ((int)m\_Data[i] > 64 && (int)m\_Data[i] < 91)//check if the letters in the string are uppercase

{

(char)m\_Data[i] = (int)m\_Data[i] + 32;//if they are then change their integer value to the ascii value for lowercase

}

}

return m\_Data;

}

//function uppercase

// no arguments

//makes all letters in the string uppercase

char \* MyString::ToUpper()

{

for (int i = 0; i < m\_Length; i++)

{

if ((int)m\_Data[i] > 96 && (int)m\_Data[i] < 123)//check if the letters in the string are lowercase

{

(char)m\_Data[i] = (int)m\_Data[i] - 32;//if they are then change their integer value to the ascii value for uppercase

}

}

return m\_Data;

}

//function subString

//takes no arguments

//searches the string for a sequence of characters

bool MyString::subString(char \*sub)

{

bool isSubStringFound = false;

MyString temp = MyString(sub);//characters that are being searched for

int x = 0;

for (int i = 0; i < m\_Length; i++)

{

if (m\_Data[i] == sub[x])//check if the characters at the index i are the same

{

x++;

if (x == temp.m\_Length)//check if x is 2 if so the set the value of the of isSubStringFound to true

{

isSubStringFound = true;

break;

}

}

else//if the characters at the index are not equal then set isSubStringFound to false and reset x to 0

{

isSubStringFound = false;

x = 0;

}

}

return isSubStringFound;

}

//function subStringAtIndex

//takes one argument of type int

//searches the string for a sequence of characters starting at a certain index

bool MyString::subStringAtIndex(int index, char \* sub)

{

MyString temp = MyString(sub);

bool isSubStringFoundAtIndex = false;

int x = 0;

for (int i = index; i < m\_Length; i++)

{

if (m\_Data[i] == sub[x])//check if the characters at the index i are the same

{

x++;

if (x == temp.m\_Length)//check if x is 2 if so the set the value of the of isSubStringFoundAtIndex to true

{

isSubStringFoundAtIndex = true;

break;

}

}

else//if the characters at the index are not equal then set isSubStringFound to false and reset x to 0

{

isSubStringFoundAtIndex = false;

x = 0;

}

}

return isSubStringFoundAtIndex;

}

char \* MyString::replace(char\*sub, char\*rep)

{

MyString temp = MyString(sub);//characters that are being searched for

MyString newString = MyString(rep);//characters that are going to be replaced

MyString copy = MyString(m\_Data);

int x = 0;

int track = 0;

int CopyIndex = 0;

for (int i = 0; i < m\_Length; i++)

{

CopyIndex = track;

if (m\_Data[i] == sub[x])//check if the characters at the index i are the same

{

x++;

if (x == temp.m\_Length)//check to set the value of the of isSubStringFound to true

{

x = 0;

if (temp.m\_Length < newString.m\_Length)

{

m\_Length += (newString.m\_Length - temp.m\_Length);

}

else

{

m\_Length -= (temp.m\_Length - newString.m\_Length);

}

CopyIndex++;

for (int j = 0; j < newString.m\_Length; j++)

{

m\_Data[(j + i) - (temp.m\_Length - 1)] = newString.m\_Data[j];

}

{

int placeHolder = CopyIndex;

for (int DataIndex = i - temp.m\_Length + newString.m\_Length + 1; DataIndex < m\_Length; DataIndex++)//

{

m\_Data[DataIndex] = copy.m\_Data[CopyIndex];

CopyIndex++;

}

CopyIndex = placeHolder;

}

if (temp.m\_Length < newString.m\_Length)

{

i = i + (newString.m\_Length - temp.m\_Length);

}

else if (temp.m\_Length>newString.m\_Length)

{

i = i - (temp.m\_Length - newString.m\_Length);

}

}

}

else//if the characters at the index are not equal then set isSubStringFound to false and reset x to 0

{

x = 0;

}

m\_Data[m\_Length] = '\0';

track++;

}

return m\_Data;

}

//functinon getString

//takes no arguments

//gets the string from the users input

const char \* MyString::constantCStyleString()

{

const char \* constanceCString = m\_Data;

return constanceCString;

}

**READ ME:**

All code is accessible at <https://github.com/MaxGuidry/String-Class>

All functions work in their current state.

To start the project run the executable on github.

On startup two strings will need to be entered and all the information that is calculated will be displayed on the console