Zyron Melancon

10/6/16

String Class Assessment

1. **Requirements Documentation**
   1. **Description of Problem**

**Name:** Create a String Class

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

**Problem Specification:** The following features should be included in the String class:

* The ability to query the string’s length, returning an integer
* The ability to access a character at a certain index within the string class
* The ability to compare if the string is the same as another string class
* The ability to append one string to another
* The ability to prepend one string to another
* The ability to return the string as a basic constant C-style string (const char\*)
* The ability to convert the string to a duplicate containing all lowercase letters
* The ability to convert the string to a duplicate containing all uppercase letters
* The ability to set the string to input a C-style string
* The ability to find a sub-string within the string class
* The ability to find a sub-string within the string class, starting from a certain index within the string
* The ability to replace a sub-string found within the string with a different sub-string
  1. **Input Information**

The user inputs strings, and index numbers, depending on the function being used.

* 1. **Output Information**

Method of output is chosen by the user. In the example provided, output is shown on the console window.

* 1. **User Interface**

I/O Through console window.

1. **System Architecture**

**Member functions in the class:**

**Prototype:** strcnt()

**Type:** Integer

**Description:** Returns the length of the stored string, by counting the characters in the char variable **m\_str**.

**Parameters:** N/A

**Visibility:** Public

**Prototype:** strret(int)

**Type:** Character

**Description:** Returns the character of the string, stored at an input index number.

**Parameters:** One number variable of type int.

**Visibility:** Public

**Prototype:** strapp(string)

**Type:** Void

**Description:** Appends with an input string by copying the characters stored in the input string, starting at the index after the last character stored in the executing string.

**Parameters:** One input of type string.

**Visibility:** Public

**Prototype:** strpre(string)

**Type:** Void

**Description:** Prepends the executing string with the input string, by shifting the executing string’s characters by the length of the input string, then copying the input string into the executing string, starting at index 0.

**Parameters:** One input of type string.

**Visibility:** Public

**Prototype:** strlow()

**Type:** Void

**Description:** Converts all uppercase letter characters in the string to lowercase by adding 32 the ASCII value of every letter character.

**Parameters:** N/A

**Visibility:** Public

**Prototype:** strcap()

**Type:** Void

**Description:** Converts all lowercase letter characters in the string to uppercase by subtracting the ASCII value of every letter character by 32.

**Parameters:** N/A

**Visibility:** Public

**Prototype:** strsub(char[])

**Type:** Boolean

**Description:** Searches the string for a user-input sub-string, by comparing the input character array to the characters stored within the string. Returns true if found, false if not.

**Parameters:** An array of type char.

**Visibility:** Public

**Prototype:** strsub(char[],int)

**Type:** Boolean

**Description:** Searches the string for a user-input sub-string, starting at a specified index number. Returns true if found, false if not.

**Parameters:** An array of type char, and a number of type int.

**Visibility:** Public

**Prototype:** strcomp(string)

**Type:** Boolean

**Description:** Compares two strings by checking the equality of each character stored within the executing string and the input string. Returns true if all characters match, returns false if not.

**Parameters:** An input of type string.

**Visibility:** Public

**Prototype:** cstyle()

**Type:** Constant Character Pointer

**Description:** Returns the stored string as a C-style string.

**Parameters:** N/A

**Visibility:** Public

**Prototype:** strrep(char[],char[])

**Type:** Boolean

**Description:** Uses the sub-string finding algorithm from strsub(char[]), with additional functionality. It finds the index of where the substring was found, and shifts the rest of the string forwards or backwards to compensate for varying sizes in the user-input replacement string. It returns true if the substring was found, and false if not.

**Parameters:** An array of type char (the sub-string to find), and another array of type char (the string to replace the substring with)

**Visibility:** Public

**Member variables:**

**char m\_str[256]**

**Description:** Character array that stores the string data

**Visibility:** Private

.**CPP File:**

#include "string.h"

string::string(char in[])

{

int length = 0;

for (int i = 0; i < 255; i++) //Loop

{

if (in[i] == '\0') //Until end

break;

else

{

m\_str[i] = in[i]; //Copy all input characters into m\_str

length++;

}

}

m\_str[length] = '\0'; //Add terminating character at end

}

int string::strcnt()

{

int length = 0;

for (int i = 0; i < 256; i++) //check through every character

{

if (m\_str[i] == '\0') //break loop at terminating character

break;

else //But add one to length if not the end

length++;

}

return length; //return the length

}

char string::strret(int index)

{

return m\_str[index-1];

}

void string::strapp(string app)

{

int length = strcnt();

for (int i = 0; i < app.strcnt(); i++)

{

m\_str[app.strcnt() + i - 1] = app.m\_str[i]; //At the end of the string, add the appending characters

length++;

}

m\_str[length] = '\0'; //At the new ending, add terminating character

}

void string::strpre(string pre)

{

int length = strcnt();

int prelen = pre.strcnt();

int a;

/\* Move the existing string forward by the

amount of characters in the prepending string \*/

for (int i = strcnt(); i > -1; i--)

{

m\_str[i + prelen] = m\_str[i];

}

/\* Add the prepending string in the leftover slots \*/

for (int i = 0; i < pre.strcnt(); i++)

{

m\_str[i] = pre.m\_str[i];

length++;

}

m\_str[length] = '\0'; //Add terminating character

}

void string::strlow()

{

for (int i = 0; i < strcnt(); i++) //for every letter

{

if (m\_str[i] >= 65 && m\_str[i] <= 90) //if it's capital

m\_str[i] = m\_str[i] + 32; //bump the ASCII value to its lowercase counterpart

else if (m\_str[i] == '\0') //but if it's the end, break the loop

break;

else //if it's any other character, skip over and continue

continue;

}

}

void string::strcap()

{

for (int i = 0; i < strcnt(); i++) //for every letter

{

if (m\_str[i] >= 97 && m\_str[i] <= 122) //if it's lowercase

m\_str[i] = m\_str[i] - 32; //bump the ASCII value to its capital counterpart

else if (m\_str[i] == '\0') //but if it's the end, break the loop

break;

else //if it's any other character, skip over and continue

continue;

}

}

bool string::strsub(char sub[])

{

int sublength = 0;

int match = 0;

for (int i = 0; i < 255; i++)

{

if (sub[i] == '\0')

break;

else

sublength++;

}

/\* This loop goes through every character

in the string, and finds matches \*/

for (int i = 0; i < strcnt(); i++)

{

if (m\_str[i] == sub[0]) //If the first letter matches

{

for (int o = 0; o < sublength; o++) //Start a loop to match the rest of the substring

{

if (m\_str[i + o] == sub[o]) //If letter in string = letter of substring

match++; //Add 1 to matching letters

}

if (match != sublength) //If the entire string did not match, reset

match = 0;

}

if (match == sublength) //If the substring is found, break the loop

break;

}

if (match == sublength) //If matching charas equals charas in string, return true, if not, false

{

return true;

}

else

return false;

}

bool string::strsub(char sub[], int ind)

{

int sublength = 0;

int match = 0;

for (int i = 0; i < 255; i++)

{

if (sub[i] == '\0')

break;

else

sublength++;

}

/\* This loop goes through every character

in the string, and finds matches \*/

for (int i = ind; i < strcnt(); i++)

{

if (m\_str[i] == sub[0]) //If the first letter matches

{

for (int o = 0; o < sublength; o++) //Start a loop to match the rest of the substring

{

if (m\_str[i + o] == sub[o]) //If letter in string = letter of substring

match++; //Add 1 to matching letters

}

if (match != sublength) //If the entire string did not match, reset

match = 0;

}

if (match == sublength) //If the substring is found, break the loop

break;

}

if (match == sublength) //If matching charas equals charas in string, return true, if not, false

{

return true;

}

else

return false;

}

bool string::strcomp(string comp)

{

int match = 0;

for (int i = 0; i < strcnt(); i++) //For every character

{

if (m\_str[i] == comp.m\_str[i]) //Does it match?

match++;

else

continue;

}

if (match == comp.strcnt() && match == strcnt()) //If all characters match

return true;

else

return false;

}

const char \* string::cstyle()

{

return m\_str;

}

bool string::strrep(char in1[], char in2[])

{

int in1length = 0;

int in2length = 0;

int match = 0;

int index = 0;

for (int i = 0; i < 255; i++) //Finds the length of the substring to find

{

if (in1[i] == 0)

break;

else

in1length++;

}

for (int i = 0; i < 255; i++) //Finds the length of the replacement substring

{

if (in2[i] == 0)

break;

else

in2length++;

}

/\* This loop goes through every character

in the string, and finds matches \*/

for (int i = 0; i < strcnt(); i++)

{

index = i;

if (m\_str[i] == in1[0]) //If the first letter matches

{

for (int o = 0; o < in1length; o++) //Start a loop to match the rest of the substring

{

if (m\_str[i + o] == in1[o]) //If letter in string = letter of substring

match++; //Add 1 to matching letters

}

if (match != in1length) //If the entire string did not match, reset

match = 0;

}

if (match == in1length) //If the substring is found, break the loop

break;

}

if (match == in1length) //If matching charas equals charas in string

{

if (in1length < in2length) //If replacement string is larger than the substring

for (int i = strcnt(); i >= index; i--)

m\_str[i + (in2length - in1length)] = m\_str[i]; //Shift the rest of the string forwards

else if (in1length > in2length) //But if the replacement string is smaller

for (int i = (index + in2length - 1); i < strcnt(); i++)

m\_str[i] = m\_str[i + (in1length - in2length)]; //Shift the rest of the string backwards

for (int i = 0; i < in2length; i++) //Insert the replacement string

m\_str[index + i] = in2[i];

return true; //Yes, I found the substring and it has been replaced

}

else if (match != in1length)

return false; //No, I could not find the substring

}

**Read Me**

All source code and executables are found on Github.

<https://github.com/ZyronMelancon/StringClass>

All functions work in their current state.

Functions not currently included are:

The ability to input C-Style strings

To start example program, run the executable in the StringClass repository.

Console will ask for user input.