Zachary Briggs

October 11th 2017

Assessment One

1. **Requirements**

**1.1 Goal**

**Name:** String Class

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

**Problem Specifics:** The string class should contain the following functions:

· 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 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

· The ability to set the string to an input C-style string

**1.2. Input Information**

The program will ask the user to input anything, output what they user inputted, and then ask the user if what was outputted was the same as what the user inputted.

**1.3. Output Information**

Outputs whatever the user inputted. Also outputs String test results to a file named “test”.

**1.4. User Information**

N/A

1. **System Architecture**

**String.h**

Prototype: String String();

Description: A String with a default mString is created.

Arguments: None.

Precondition: None.

Postcondition: A default String is created.

Protection: Public.

Prototype: String ~String();

Description: A String is deleted.

Arguments: None.

Precondition: A String is created.

Postcondition: The String is deleted.

Protection: Public.

Prototype: String(char\* name);

Description: A String with a custom mString is created.

Arguments: None.

Precondition: None.

Postcondition: A custom String is created.

Protection: Public.

Prototype: int GetLength();

Description: Returns length of a string.

Arguments: None.

Precondition: None.

Postcondition: Length of string is returned.

Visibility: Public.

Prototype: void AccessIndex(int index);

Description: Accesses the string's index of the argument passed in.

Arguments: An int representing the index.

Precondition: A string.

Postcondition: The character at the index is returned.

Visibility: Public

Prototype: String AppendString(String rhs);

Description: Appends a string to the end of a string.

Arguments: String.

Precondition: Two seperated strings.

Postcondition: The two strings are now combined with the argument string

being on the right side.

Visibility: Public

Prototype: String PrependString(String lhs);

Description: Prepends a string to the beginning of a string.

Arguments: String.

Precondition: Two seperated strings.

Postcondition: The two strings are now combined with the argument string

being on the right side.

Visibility: Public

Prototype: const char\* ConstantString();

Description: Changes a string into a constant.

Arguments: None

Precondition: A string.

Postcondition: The string is now a constant character string.

Visibility: Public

Prototype: void UpperString();

Description: Uppercases all the characters in the string.

Arguments: None

Precondition: A string with non unified casing.

Postcondition: All the casing in the string is uppercased

Visibility: Public

Prototype: void LowerString();

Description: Lowercases all the characters in the string.

Arguments: None.

Precondition: A string with non unified casing.

Postcondition: All the casing in the string is lowercased.

Visibility: Public

Prototype: bool FindSubString(char\*);

Description: Searches for a sub-string in a string and returns true if it finds it.

Arguments: A character array representing the sub-string.

Precondition: A string.

Postcondition: True is returned if the substring is in the string.

Visibility: Public

Prototype: bool FindSubStringFromIndex(int, char\*);

Description: Searches for a sub-string in every index after the one passed in.

Arguments: An int representing the index and a character array for the sub string.

Precondition: A string.

Postcondition: True is returned if the substring is found after the index.

Visibility: Public

Prototype: String ReplaceSubString(char\*, char\*);

Description: Replaces a sub-string with another sub-string.

Arguments: Two sub-strings the first is the sub-string to be replaced and the second

is the sub-string that will replace it.

Precondition: A string.

Postcondition: The sub-string is replaced with the new sub-string.

Visibility: Public

Prototype: bool operator==(String & other);

Description: Returns true/false depending on if the two strings are

equal/exactly the same.

Arguments: Two strings.

Precondition: Two strings.

Postcondition: True/False is returned.

Visibility: Public.

Prototype: friend std::istream& operator>>(std::istream & input, String &returnString);

Description: Allows the user to insput a string.

Arguments: What the string is getting changed too.

Precondition: A string.

Postcondition: The string is changed to whatever the user inputted.

Visibility: Public.

Prototype: friend std::ostream& operator<<(std::ostream & stream, const String & a);

Description: Outputs the string.

Arguments: The string to be outputted.

Precondition: A string.

Postcondition: The contents of the string is outputted.

Visibility: Public.

**Testing.h**

Prototype: void successfulTest(int\* testNumber,int\* testsPassed, std::fstream &file);

Description: Increments testNumber and testsPassed if the test was successful.

Arguments: Two ints representing the number of tests that have happened and how many

were passed

Precondition: None.

Postcondition: testNumber and testPassed are incremented and the text saying the test was

successful is outputted into the text file.

Visibility: Public.

Prototype: void failedTest(int\* testNumber, int\* testsFailed, std::fstream &file);

Description: Increments testNumber and testFailed if the test was failed.

Arguments: None.

Precondition: None.

Postcondition: testNumber and testsFailed are incremented and the text saying the test was

failed is outputted into the text file.

Visibility: Public.

1. **Source Code**

**String.h**

#pragma once

#include <iostream>

class String

{

private:

const char\* mString;

public:

String();

~String();

String(char\* name);

int GetLength();

char AccessIndex(int index);

String AppendString(String rhs);

String PrependString(String lhs);

const char\* ConstantString();

String UpperString();

String LowerString();

bool FindSubString(String substring);

bool FindSubString(int index, String substring);

String ReplaceSubString(String substring, String newSubstring);

bool operator==(String & other);

friend std::istream& operator>>(std::istream & input, String &returnString);

friend std::ostream& operator<<(std::ostream & stream, const String & a);

**String.cpp**

#include "String.h"

String::String()

{

mString = new char[255];

}

String::~String()

{

}

String::String(char\* name)

{

mString = name;

}

int String::GetLength()

{

int i = 0;

while (mString[i] != '\0')

{

i++;

}

return i;

}

char String::AccessIndex(int index)

{

char letter = mString[index];

return letter;

}

String String::AppendString(String rhs)

{

char \*a = new char[255];

int i = 0;

int j = 0;

while (i < GetLength() + rhs.GetLength())

{

while (i < GetLength())

{

a[i] = mString[i];

i++;

}

while (j < rhs.GetLength())

{

a[i] = rhs.mString[j];

i++;

j++;

}

}

a[i] = '\0';

return String(a);

}

String String::PrependString(String lhs)

{

char \*a = new char[255];

int i = 0;

int j = 0;

while (i < GetLength() + lhs.GetLength())

{

while (i < lhs.GetLength())

{

a[i] = lhs.mString[i];

i++;

}

while (j < GetLength())

{

a[i] = mString[j];

i++;

j++;

}

}

a[i] = '\0';

return String(a);

}

const char \* String::ConstantString()

{

const char\* a = mString;

return a;

}

String String::UpperString()

{

char \*a = new char[255];

int i = 0;

while (i < GetLength())

{

if (mString[i] < 123 && mString[i] > 96)

a[i] = mString[i] - 32;

else

a[i] = mString[i];

i++;

}

a[i] = '\0';

return String(a);

}

String String::LowerString()

{

char\* a = new char[255];

int i = 0;

while (i < GetLength())

{

if (mString[i] < 91 && mString[i] > 64)

a[i] = mString[i] + 32;

else

a[i] = mString[i];

i++;

}

a[i] = '\0';

return String(a);

}

bool String::FindSubString(String substring)

{

int j = 0;

int i = 0;

while (i < GetLength())

{

if (mString[i] == substring.mString[j])

{

i = 0;

j++;

}

else

i++;

}

if (j == substring.GetLength())

return true;

else

return false;

}

bool String::FindSubString(int index, String substring)

{

int j = 0;

int i = index;

while (i < GetLength())

{

if (mString[i] == substring.mString[j])

{

i = index;

j++;

}

else

i++;

}

if (j == substring.GetLength())

return true;

else

return false;

}

String String::ReplaceSubString(String substring, String newSubstring)

{

char\* lhs = new char[255];

char\* rhs = new char[255];

char\* newString = new char[255];

int i = 0;

int j = 0;

bool running = true;

if (!FindSubString(substring))

return nullptr;

while (running == true)

{

if (mString[i] == substring.mString[0])

running = false;

else

{

newString[i] = mString[i];

i++;

}

}

// k keeps track of how long the front end is

int k = i;

while (newSubstring.mString[j] != '\0')

{

newString[i] = newSubstring.mString[j];

i++;

j++;

}

running = true;

k += substring.GetLength();

while (running == true)

{

newString[i] = mString[k];

i++;

k++;

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

running = false;

}

return String(newString);

}

bool String::operator==(String & other)

{

int i = 0;

int sameChar = 0;

while (i < GetLength())

{

if (mString[i] == other.mString[i])

{

sameChar++;

if (sameChar == GetLength())

return true;

}

else

return false;

i++;

}

}

std::istream & operator >> (std::istream & input, String &returnString)

{

char \*newString = new char[255];

input >> newString;

returnString = String(newString);

return input;

}

std::ostream & operator<<(std::ostream & stream, const String & a)

{

stream << a.mString;

return stream;

}

**Testing.h**

#pragma once

#include "String.h"

#include <fstream>

void successfulTest(int\* testNumber,int\* testsPassed, std::fstream &file);

void failedTest(int\* testNumber, int\* testsFailed, std::fstream &file);

**Testing.cpp**

#include "Testing.h"

void successfulTest(int\* testNumber,int\* testsPassed, std::fstream &file)

{

\*testNumber += 1;

\*testsPassed += 1;

file << "Test " << \*testNumber << " Successful.\n";

}

void failedTest(int\* testNumber,int\* testsFailed, std::fstream &file)

{

\*testNumber+=1;

\*testsFailed+=1;

file << "Test " << \*testNumber << " Failed.\n";

}

**Main.cpp**

#include "String.h"

#include "Testing.h"

int main()

{

std::fstream file;

String a = String("Zach");

String b = String("Blambo");

String c = String("am");

String d = String("Zach");

String test3 = String("ZACH");

String test4 = String("zach");

String test7 = String("ZachBlambo");

String test8 = String("BlamboZach");

String test9 = String("BlZachbo");

int tests = 0;

int testsPassed = 0;

int testsFailed = 0;

file.open("test.txt", std::ios\_base::out | std::ios\_base::app);

file << "Test 1: Length of String.\n";

if (a.GetLength() == 4)

successfulTest(&tests, &testsPassed, file);

else

failedTest(&tests, &testsFailed, file);

file << "Test 2: Specific Index.\n";

if (a.AccessIndex(0) == 'Z')

successfulTest(&tests, &testsPassed, file);

else

failedTest(&tests, &testsFailed, file);

file << "Test 3: Uppercasing an entire string.\n";

String test3Half = a.UpperString();

if(a.UpperString() == test3)

successfulTest(&tests, &testsPassed, file);

else

failedTest(&tests, &testsFailed, file);

file << "Test 4: Lowercasing an entire string.\n";

if (a.LowerString() == test4)

successfulTest(&tests, &testsPassed, file);

else

failedTest(&tests, &testsFailed, file);

file << "Test 5: Finding a substring.\n";

if (b.FindSubString(c) == true)

successfulTest(&tests, &testsPassed, file);

else

failedTest(&tests, &testsFailed, file);

file << "Test 6: Finding a substring from a specific index.\n";

if (a.FindSubString(2, c) == false)

successfulTest(&tests, &testsPassed, file);

else

failedTest(&tests, &testsFailed, file);

file << "Test 7: Appending a string.\n";

if (a.AppendString(b) == test7)

successfulTest(&tests, &testsPassed, file);

else

failedTest(&tests, &testsFailed, file);

file << "Test 8: Prepending a string.\n";

if (a.PrependString(b) == test8)

successfulTest(&tests, &testsPassed, file);

else

failedTest(&tests, &testsFailed, file);

file << "Test 9: Replacing a substring with another substring.\n";

if (b.ReplaceSubString(c, a) == test9)

successfulTest(&tests, &testsPassed, file);

else

failedTest(&tests, &testsFailed, file);

file << "Test 10: Returning string as const char\*.\n";

if (c.ConstantString() == "am")

successfulTest(&tests, &testsPassed, file);

else

failedTest(&tests, &testsFailed, file);

file << "Test 11: Overloading == operator.\n";

if(a == b == false && a == d)

successfulTest(&tests, &testsPassed, file);

else

failedTest(&tests, &testsFailed, file);

file << "Test 12: Inputting and Outputting C-String.\n";

std::cout << "Input something.\n";

std::cin >> a;

std::cout << "You inputted " << a << "." << " Correct? y/n ";

char correct;

std::cin >> correct;

std::cout << std::endl;

if(correct == 'y')

successfulTest(&tests, &testsPassed, file);

else if(correct == 'n')

failedTest(&tests, &testsFailed, file);

file << testsPassed << "/12 test(s) successful.\n";

file << testsFailed << " test(s) failed.\n";

}

1. **Read Me**
   1. Start up your computer.
   2. Sign into an account
   3. Open up a web browser
   4. Copy and paste: https://github.com/ZacharyBriggs/AssesmentOne/releases/tag/1.0 into the search bar of the browser.
   5. Click on String.Class.zip
   6. Find where the zip file was downloaded to.
   7. Extract the files.
   8. Read the Read\_Me.
   9. Run String\_Class.exe
   10. Input anything.
   11. Input ‘y’ if the string outputted is the same as what you inputted or ‘n’ if they aren’t the same.
   12. Have fun!(?)