# Group 3: Nguyễn Đức Duy Nguyễn Công Hoàng Trần Quốc Bảo

Project 3

## Programming Challenges

#### 1. String Length (1)

#### Write a function that returns an integer and accepts a pointer to a C-string as an argument. The function should count the number of characters in the string and return that number. Demonstrate the function in a simple program that asks the user to input a string, passes it to the function, and then displays the function’s return value.

**#include <iostream>**

**using namespace** std;

**int** getLength(**char** \*str) {

**int** r = 0;

**while** (str[r] != '\0') {

r++;

}

**return** r;

}

**int** main() {

**char** str[100];

cout << "Enter a string:\t";

cin.getline(str, 100);

cout << "Your string has " << getLength(str) << " characters." << endl;

**return** 0;

}

#### 2. Backward String (2)

#### Write a function that accepts a pointer to a C-string as an argument and displays its contents backward. For instance, if the string argument is “ Gravity ” the function should display “ ytivarG ”. Demonstrate the function in a program that asks the user to input a string and then passes it to the function.

#include <iostream>  
#include <cstring>

**using namespace** std;

**void** reverse(**char** \*str) {

size\_t l = strlen(str);

**for** (**int** i = 0; i < l / 2; ++i) {

**char** temp = str[i];

str[i] = str[l - i - 1];

str[l - i - 1] = temp;

}

cout << str << endl;

}

**int** main() {

**char** str[100];

cout << "Enter a string to reverse:\t";

cin.getline(str, 100);

cout << "\t-> ";

reverse(str);

**return** 0;

}

#### 3. Word Counter

#### Write a function that accepts a pointer to a C-string as an argument and returns the number of words contained in the string. For instance, if the string argument is “Four score and seven years ago” the function should return the number 6. Demonstrate the function in a program that asks the user to input a string and then passes it to the function. The number of words in the string should be displayed on the screen.

#### Optional Exercise: Write an overloaded version of this function that accepts a string class object as its argument.

#include <iostream>  
#include <cstring>

#include <cctype>

#include <string>

**using namespace** std;

**int** countWord(**char** \*str) {

size\_t len = strlen(str);

**int** count = 0;

**bool** isCurrentWord = **false**;

**for** (**int** i = 0; i < len; ++i) {

**if** (isalnum(str[i]) != 0) {

**if** (!isCurrentWord) {

isCurrentWord = **true**;

count++;

}

} **else** isCurrentWord = **false**;

}

**return** count;

}

**int** countWord(string str) {

**int** count = 0;

**bool** isCurrentWord = **false**;

**for** (**unsigned long** i = 0; i < str.length(); ++i) {

**char** c = str.at(i);

**if** ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) {

**if** (!isCurrentWord) {

isCurrentWord = **true**;

count++;

}

} **else** isCurrentWord = **false**;

}

**return** count;

}

**int** main() {

**char** s[256] = {0};

cout << "Enter a string:\t";

cin.getline(s, 256);

cout << "There are " << countWord(s) << " words using c-style string algorithm." << endl;

string s2(s);

cout << "There are " << countWord(s2) << " words using string object algorithm." << endl;

**return** 0;

}

#### 4. Average Number of Letters (4)

#### Modify the program you wrote for Problem 3 (Word Counter), so it also displays the average number of letters in each word.

#include <iostream>  
#include <cstring>

#include <cctype>

**using namespace** std;

**double** avgLetters(**char** \*str) {

size\_t len = strlen(str);

**int** count = 0;

**double** letters = 0.0;

**bool** isCurrentWord = **false**;

**for** (**int** i = 0; i < len; ++i) {

**if** (isalnum(str[i]) != 0) {

**if** (!isCurrentWord) {

isCurrentWord = **true**;

count++;

}

letters += 1;

} **else** isCurrentWord = **false**;

}

**return** letters / count;

}

**int** main() {

**char** str[256];

cout << "Enter a string:\t";

cin.getline(str, 256);

cout << "The string has an average of " << avgLetters(str)

<< " letters per word" << endl;

**return** 0;

}

#### 5. Most Frequent Character (9)

#### Write a function that accepts either a pointer to a C-string, or a string object, as its argument. The function should return the character that appears most frequently in the string. Demonstrate the function in a complete program.

#include <iostream>  
#include <string>

#include <cstring>

**using namespace** std;

**int** findMaxIndex(**int** l[], **int** size) {

**int** max = l[0], maxIndex = 0;

**for** (**int** i = 1; i < size; i++) {

**if** (max < l[i]) {

max = l[i];

maxIndex = i;

}

}

**return** maxIndex;

}

**char** mostFreqChar(**char** \*arr) {

**int** counter[95] = {0};

**for** (**int** i = 0; i < strlen(arr); i++) {

**if** (isprint(arr[i])) {

counter[(arr[i]) - 32]++;

}

}

**return static\_cast**<**char**>(findMaxIndex(counter, 95) + 32);

}

**char** mostFreqChar(string arr) {

**int** counter[95] = {0};

**for** (**int** i = 0; i < arr.length(); i++) {

**if** (isprint(arr[i])) {

counter[(arr[i]) - 32]++;

}

}

**return static\_cast**<**char**>(findMaxIndex(counter, 95) + 32);

}

**int** main() {

string arr;

cout << "Enter a string: ";

getline(cin, arr);

cout << "The character " << "'" << mostFreqChar(arr) << "'" << " appears most frequent" << endl;

**return** 0;

}

#### 6. replaceSubstring Function

#### Write a function named replaceSubstring . The function should accept three C-string or string object arguments. Let’s call them string1 , string2 , and string3 . It should search string1 for all occurrences of string2 . When it finds an occurrence of string2 , it should replace it with string3 . For example, suppose the three arguments have the following values:

#### String1 : “the dog jumped over the fence”

#### String2 : “the”

#### String3 : “that”

#### With these three arguments, the function would return a string object with the value “that dog jumped over that fence.” Demonstrate the function in a complete program.

#include <iostream>  
#include <cstring>

**using namespace** std;

/\*\*

\* Replace every pattern of original string that match with another string

\* using c-style string

\*

\* @note a string cannot be passed to string1 otherwise it will resulted in

\* stack memory error when C++ tries to clean up stack memory. This will occur

\* only when length of string3 larger than string2. In order to cope with this

\* we need to create new block of memory in heap so C++ will not alarm. This is

\* the difference between using string object and c-style string.

\*

\* @param string1 pointer to the original string

\* @param string2 the string to search for

\* @param string3 the string to replace with

\*/

**void** replaceSubstring(**char** \*\*string1, **const char** \*string2, **const char** \*string3) {

size\_t len1 = strlen(\*string1), len2 = strlen(string2), len3 = strlen(string3);

**bool** occurrence[len1];

**for** (**int** j = 0; j < len1; ++j) occurrence[j] = **false**;

**for** (**int** i = 0; i < len1 - (len2 > 0 ? len2 : 1) + 1; ++i) {

**if** ((\*string1)[i] == string2[0]) {

**bool** match = **false**;

**for** (**int** j = 1; j < len2; ++j) {

**if** (string2[j] != (\*string1)[i + j]) **break**;

**if** ((\*string1)[i + len2 - 1] == string2[len2 - 1])

match = **true**;

}

**if** (match) {

occurrence[i] = **true**;

i += len2 - 1;

}

}

}

size\_t len4 = len1;

**for** (**int** i = 0; i < len1; ++i) {

**if** (occurrence[i]) len4 += (len3 - len2);

}

**char** stringres[len4 + 1];

**int** offset = 0;

**for** (**int** i = 0; i < len1; ++i) {

**if** (occurrence[i]) {

**for** (**int** j = 0; j < len3; ++j) {

stringres[i + offset + j] = string3[j];

}

i += len2 - 1;

offset += len3 - len2;

} **else** stringres[i + offset] = (\*string1)[i];

}

stringres[len4] = '\0';

\*string1 = **new char**[len4 + 1];

strcpy(\*string1, stringres);

}

/\*\*

\* @note string2 and string 3 must be like const string &str because

\* if not the object of that string will be copied not the value. we can

\* make a promise not to change anything by using const keyword

\*

\* @param str1 original string

\* @param str2 the string to search for

\* @param str3 the string to replace with

\*/

**void** replaceSubstring(string &str1, **const** string &str2, **const** string &str3){

**for** (**unsigned long** i = 0; i <= str1.length() - str2.length() + 1; i++){

**if** (str1.compare(i, str2.length(), str2) == 0){

str1.replace(i, str2.length(), str3);

}

}

}

**int** main() {

string str1, str2, str3;

cout << "The ocurrences of string 2 in string 1 will be replaced by string 3.\n";

cout << "String 1: "; getline(cin, str1);

cout << "String 2: "; getline(cin, str2);

cout << "String 3: "; getline(cin, str3);

replaceSubstring(str1, str2, str3);

cout << "Edited string 1: " << str1;

**return** 0;

}

#### 11. Case Manipulator

#### Write a program with three functions: upper , lower , and reverse . The upper function should accept a pointer to a C-string as an argument. It should step through each character in the string, converting it to uppercase. The lower function, too, should accept a pointer to a C-string as an argument. It should step through each character in the string, converting it to lowercase. Like upper and lower , reverse should also accept a pointer to a string. As it steps through the string, it should test each character to determine whether it is upper- or lowercase. If a character is uppercase, it should be converted to lowercase. Likewise, if a character is lowercase, it should be converted to uppercase.

#### Test the functions by asking for a string in function main , then passing it to them in the following order: reverse , lower , and upper .