**Lab 2: C-strings and Functions**

Overview

In this lab you will:

1. Write a program that calls a function
2. Practice working with C-strings
3. Practice using I/O, including the getline() function

# Working with C-strings

Recall that a "C-string" is a character string defined as an array of type char; it is not the same thing as a C++ string object. The getline() function requires a C-string as its first argument.

The individual characters in a C-string can be accessed using array element notation. Suppose a program includes the line

char line[80] = "Hello, World!";

Then line[0] accesses the first character in line and, in this case, has the value 'H'. The second character is accessed with line[1] which has value 'e', etc. The last character is line[13] which has the value '\0' (null byte). Can you explain why?

Note that the index of the characters in a C-string starts at zero, not one; this is true of all arrays in C/C++. Also, although line was declared to be 80 characters long in the example above, only 13 characters were used. The end of a C-string is determined by the terminating null character ('\0'), notby the declared length.

## Determining the length of a C-string

The length of a C-string can be determined using the strlen() function. This is especially useful when the C-string is provided by the user so the program can't know the length a priori. The length returned by strlen()does not include the null terminator. To use strlen(), you must include the cstring library.

#include <iostream>

#include <cstring> // needed for strlen()

using namespace std;

int main() {

char line[80];

cout << "Enter a string (< 80 characters)" << endl;

cin.getline(line, sizeof(line));

cout << "The string you entered is " << strlen(line)

<< " characters long." << endl;

return 0;

}

If we run the program and provide the input

The Internet doesn't weigh anything!

the program will produce the output

The string you entered is 36 characters long.

## Looping Over Characters in a C-string

Combining strlen(), array index notation, and a for loop, we can loop over all the characters in a C-string; that is, we can examine or modify each successive character in the C-string. For example, the following code will read a line from the user and print each character on a separate line:

#include <iostream>

#include <cstring>

using namespace std;

int main() {

char line[80];

cout << "Enter a string (< 80 characters)" << endl;

cin.getline(line, sizeof(line));

for (int i = 0; i < strlen(line); i++ )

cout << line[i] << endl;

return 0;

}

Running the program with input Hello! produces the output

H

e

l

l

o

!

# Reading Input with getline()

We can read an entire line of user input with the getline() function. The function takes two arguments — the string variable in which to store the input and the maximum length that can be stored in the variable. For example:

#include <iostream>

using namespace std;

int main() {

char line[80];

cout << "Enter your name: ";

cin.getline(line, sizeof(line));

cout << "Your name is " << line;

return 0;

}

sizeof() is a built-in function that determines the size in bytes of a variable. Since line is an 80-long C-string, sizeof(line)is 80.

## Input Gotcha

When mixing getline() and other input methods, there is one problem to watch out for: getline() discards the ending "\n" character, but the other input methods do not. This can lead to unexpected results. For example:

#include <iostream>

using namespace std;

int main() {

int i1;

char s1[80], s2[80];

cin >> i1;

cin.getline(s1, sizeof(s1));

cin.getline(s2, sizeof(s2));

return 0;

}

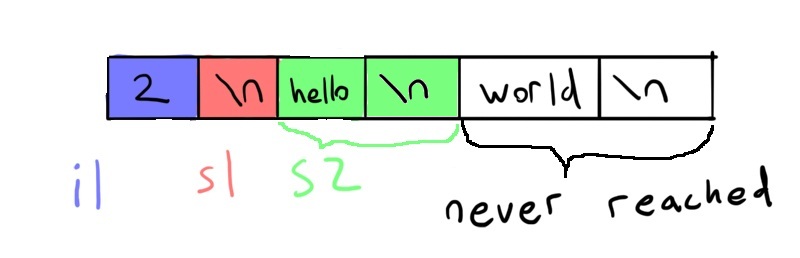
If we provide the input

2

Hello

World

the program will exit immediately after we input "Hello." i1 will have the value 2, but s1 will contain only a newline character, "\n", and s2 will contain "hello\n". To avoid this, an extra getline() can be inserted after cin >> num to discard the "\n". Here is a visual representation of what is happening on the input buffer:



The following code fixes the problem by adding a call to getline() to read the new line character before attempting to read s1:

#include <iostream>

using namespace std;

int main() {

int i1;

char s1[80], s2[80];

char dummy[80];

cin >> i1;

cin.getline(dummy, sizeof(dummy)); // reads the newline

cin.getline(s1, sizeof(s1));

cin.getline(s2, sizeof(s2));

return 0;

}

# Working with char values

Character constants are written using single quotes. For example, 'A' represents thecharacter A whereas "A" represents the C-string consisting of the letter A followed by a null terminator. Suppose we want to determine which letters in the C-string str are lowercase. This is can be done easily as follows:

for (int i = 0; str[i] != '\0'; i++)

if (str[i] >= 'a' && str[i] <= 'z')

cout << "Character " << i << " is lowercase." << endl;

Notice that determining whether a character is lowercase amounts to checking whether it is greater-than-or-equal-to 'a' and less-than-or-equal-to 'z'.

Internally, the computer represents characters as single-byte integers; that is, integers in the range 0 – 255. The reason the if statement in the previous example works is that the lowercase letters have numeric values 97 – 122, with 'a' represented by 97, 'b' by 98, etc., up to 'z' which has the value 122. So, if str[i] is a lowercase letter, then its numeric value is between 97 and 122, and the conditional evaluates to true.

The uppercase letters are represented by the integers 65 – 90; we can use this fact to convert uppercase to lowercase, or vice versa. For example, the following loop converts the lowercase letters in str to uppercase:

for (int i = 0; str[i] != '\0'; i++)

if (str[i] >= 'a' && str[i] <= 'z')

str[i] -= 32;

This can also be done without knowing the numeric representations as follows:

for (int i = 0; str[i] != '\0'; i++)

if (str[i] >= 'a' && str[i] <= 'z')

str[i] = str[i] - 'a' + 'A';

On GL (or any Linux system) you can find a list of characters and their numeric equivalents using the command

man ascii

In the output, the column labeled "Dec" gives the decimal numeric equivalent.

# Programming Assignment

You will be writing a program to convert lowercase to uppercase and space to underscore (\_) in a user-supplied string. The program must:

1. Prompt the user to enter a string
2. Read a string from the user with getline()
3. Call a function to convert lowercase to uppercase and space to underscore in the string
4. Display the modified string on the screen

Here is an example compilation and execution of the program:

linux3[1]% g++ -Wall lab2.cpp -o lab2

linux3[1]% ./lab2

Enter a string: The quick brown fox jumps over the lazy dog.

The modified string is: THE\_QUICK\_BROWN\_FOX\_JUMPS\_OVER\_THE\_LAZY\_DOG.

To get you started, a template of the program is provided below. You can copy and paste the template into a text editor such as Emacs. Alternatively, you may copy the file to your working directory from Prof. Marron's public folder on GL:

cp /afs/umbc.edu/users/c/m/cmarron/pub/cmsc202/lab2.cpp .

To complete the program, you must:

1. Add a global constants for the size of str
2. Add #include statements if you use additional libraries
3. Write code to prompt the user to enter a string
4. Write code to read the input using getline()
5. Implement the function ModifyString()
6. Write code to display the modified string in a user-friendly manner