1 Introduction
In this lab you will implement a "hangman" game where the user is shown blanks representing letter of a word and then tries to guess and fill in the letters with a limited number of guesses.

2 What you will learn
After completing this lab you should be able:
- Manipulate C-Strings (char *'s)
- Pass C-Strings to functions
- Utilize the C-String library (#include <cstring>) to perform various string operations
- Use cin to read a C-String

3 Background Information and Notes
Review your lectures regarding C-Strings and in particular the C-String library. This library provides several useful functions to copy, compare, and get the length of a string.

```c
char * strcpy ( char * destination, const char * source);
char * strncpy ( char * destination, const char * source, size_t num );
int strcmp ( const char * str1, const char * str2 );
size_t strlen ( const char * str );
```

strcpy():
This function takes in two pointers: the first to a destination character array, the second to a source character array. It copies the source string into the destination. The problem with strcpy() is that it only stops copying when it hits the NULL character in the source string. If the destination array does not have enough memory allocated, strcpy() will gladly overwrite out-of-bounds memory, which can lead to crashes, security vulnerabilities, or hacks in software that would let some rogue user take over the system. Search for 'buffer overrun vulnerability' to learn more.

strncpy():
This is a safer alternative to strcpy(). It provides a third argument which is a MAXIMUM number of characters that will be copied. It should usually be passed in as the LENGTH of the destination array to ensure we don't write off the end of the destination array. Google 'strncpy' and the top hit will be a great website: cplusplus.com. It has excellent documentation on all C++ library functions. Please read the description of strncpy(). What is the return value?
strcmp():

strcmp() takes two pointers to character strings and compares them lexicographically. It returns the integer value 0 if they are equal, -1 if str1 is "less-than" str2 and +1 if str1 is "greater than" str2. So comparing "abc" and "abc" would return 0. Comparing "ab" and "abc" would return -1 as would comparing "ab" with "ac". Comparing "ac" with "ab" would return +1 as would comparing "abc" with "ab".

strlen():

returns the length of a given string. The return type "size_t" is some kind of unsigned integer (on your VM it is an unsigned 64-bit integer) but you can assign the value returned into an 'int' variable without worrying too much (unless you have strings with over 4 billion characters 😊)

4 Procedure

4.1 [1 pt.] Finding C++ Documentation
Show your TA/CP the documentation for strcmp() on cplusplus.com

4.2 [2 pts.] Copying strings – Common Errors
Type in the following program to a file (stringdemo.cpp). Recall that if we have the following declarations of character arrays and pointers, we cannot simply copy one string to the other via an assignment statement. Similarly pointing one char* at another string means we will just reference the original and not make a copy.

```cpp
1: #include <iostream>
2: #include <cstring>
3: using namespace std;
4: int main()
5: {
6:     char mystring[80] = "ComputerScience";
7:     char yourstring[80];
8:     char* astring;
9:     yourstring = mystring;
10:    strncpy(astring, mystring, 80); // make a copy?
11:    astring = mystring; // make a copy?
12:    strncpy(yourstring, mystring, 80); // make a copy?
13:    cout << astring << endl;
14:    cout << yourstring << endl;
15: mystring[0] = '*'; // which one actually made a copy?
16:    cout << astring << endl;
17:    cout << yourstring << endl;
18: return 0;
19: }
```
Try to compile your code:

```
$ g++ -g -o stringdemo stringdemo.cpp
```

You should get the following error

```
stringdemo.cpp: In function 'int main()':
stringdemo.cpp:12:16: error: invalid array assignment
```

Recall that array names when used in C yield the starting address. So you are really saying take the starting address of mystring and make it the starting address of yourstring. But yourstring has to stay as the address of the memory corresponding to it. Thus you should think of array names as constant pointers that cannot be changed/assigned to.

The moral of the story is you must use the `strncpy()` function to copy over the characters one at a time. An example of this is line 16 of the program.

So now comment that line (i.e. change it to)

```
//yourstring = mystring;
```

Look at the next line of code and think what this will do.

```
strncpy(astring, mystring, 80);
```

It will attempt to copy the characters one at a time from mystring to the memory that astring points to. But what does astring point to? We haven't initialized it. So whatever garbage bits are in the astring pointer will be used as an address and it will attempt to copy the characters there...and likely fail/crash. Uh-oh! Compile and run the program and see what happens:

```
$ g++ -g -o stringdemo stringdemo.cpp
```

(If you have --Wall enabled, you’ll actually get a warning about the error.) Running

```
$ ./stringdemo
```

should (likely) yield:

```
Segmentation fault (core dumped)
```

This is because you tried to copy something to a bogus address. (If it doesn’t, change the declaration of astring to `char* astring = 0;` and try again.)

The moral of the story is you can only copy data to pointers that actually POINT at memory you have allocated!
Whenever you get a segmentation fault (which will be a lot in your CS career), you can use GDB to find the line of code causing it. Start 'gdb stringdemo' and at the prompt just type 'run' to run the program in the debugger. When it crashes in gdb, type

```
backtrace
```

This will show you the function calls (and line numbers). You should see that the last function that you recognize (i.e. you wrote) is `main()` at `stringdemo.cpp:13`. This means main() called some other function at line 13 and that other function triggered the error. So this can always be used to tell you where your segmentation fault is.

So now comment line 13 as well:

```
// strncpy(astring, mystring, 80);
```

Well we can initialize astring to point at mystring as on line 15. But are we making a copy? No, just referencing it (i.e. pointing at the original mystring array). On line 17 we print out astring which is really just printing out the data from mystring (since astring just points to mystring). But in line 20 we change mystring. So when we print out astring again on line 21 we will see the update.

**The moral of the story is pointing one pointer to someone else's data DOES NOT make a copy of the data. It only references that data. A change in the original is seen by the referencing pointer.**

On line 16 we actually make a copy of all the data from mystring to yourstring. Since we actually allocated an array of characters for yourstring, we have room to put the copied data. Now on line 20 when we change mystring, yourstring is unaffected and when we print it on line 22 we have the original.

**Show your TA you know how to find the source of a 'segmentation fault' by uncommenting the line of code that will cause the seg fault to occur and then running gdb in front of your TA to display the line number that caused the fault.**

### 4.3 [7 pts.] Hangman

Download the skeleton code.

```
$ wget http://ee.usc.edu/~redekopp/cs103/hangman.cpp
```

We have already created an array called wordBank which has 10 words defined in it. Your program should do the following:
1. Select a word at random from the wordBank
2. On each turn display the word, with letters not yet guessed showing as *'s, and letters that have been guessed showing in their correct location
3. The user should have 10 attempts (“lives”). Each unsuccessful guess costs one attempt. Successful guesses do NOT count as a turn.
4. You must use the C-String library (#include <cstring>) functions strlen() and strcmp() in your program [strcmp() can compare when the guessed word matches the target/selected word from the wordBank...meaning the user won/finished]
5. You must complete and use the function
   int processGuess(char* word, const char* targetWord, char guess)

   This function should take the pointer to the current value of the word guessed thus far, the pointer to the target/selected word from the wordBank, and the character that the user guessed. It should change any *'s to actual good characters for the letter the user guessed and return a count of how many times the guessed letter appears in that word. In this way, if you return 0, it means the user guessed a letter that did NOT appear and thus should lose a turn back in your main().

6. At every turn display the current version of the guessed word and how many turns remain
7. At the end of the game give a descriptive message whether the user won or lost

Demonstrate your program and show your TA/CP the two functions you wrote explaining how it works.

Sample runs of our solution to this program are shown below.

A winning example:

<table>
<thead>
<tr>
<th>Current word: ******</th>
<th>10 remain...Enter a letter to guess:</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td></td>
</tr>
<tr>
<td>Current word: ******</td>
<td>9 remain...Enter a letter to guess:</td>
</tr>
<tr>
<td>l</td>
<td></td>
</tr>
<tr>
<td>Current word: ******</td>
<td>8 remain...Enter a letter to guess:</td>
</tr>
<tr>
<td>c</td>
<td></td>
</tr>
<tr>
<td>Current word: <em>c</em>**c*</td>
<td>8 remain...Enter a letter to guess:</td>
</tr>
<tr>
<td>s</td>
<td></td>
</tr>
<tr>
<td>Current word: sc**<em>c</em></td>
<td>8 remain...Enter a letter to guess:</td>
</tr>
<tr>
<td>g</td>
<td></td>
</tr>
</tbody>
</table>
Current word:  sc***c*
7 remain...Enter a letter to guess:
i
Current word:  sci**c*
7 remain...Enter a letter to guess:
e
Current word:  scie*ce
7 remain...Enter a letter to guess:
n
The word was:  science.  You win!

A losing example:

Current word:  *****
10 remain...Enter a letter to guess:
z

Current word:  *****
9 remain...Enter a letter to guess:
z

Current word:  *****
8 remain...Enter a letter to guess:
z

Current word:  *****
7 remain...Enter a letter to guess:
z

Current word:  *****
6 remain...Enter a letter to guess:
z

Current word:  *****
5 remain...Enter a letter to guess:
z

Current word:  *****
4 remain...Enter a letter to guess:
z

Current word:  *****
3 remain...Enter a letter to guess:
z

Current word:  *****
2 remain...Enter a letter to guess:
z

Current word:  *****
1 remain...Enter a letter to guess:
z
Too many turns...You lose!