

LAB 7

Each lab will begin with a recap of last lab and a brief demonstration by the TAs for the core concepts examined in this lab. As such, this document will not serve to tell you everything the TAs will in the demo. It is highly encouraged that you ask questions and take notes.

In order to get credit for the lab, you need to be checked off by the end of lab. For non-zero labs, you can earn a maximum of 3 points for lab work completed outside of lab time, but you must finish the lab before the next lab. For extenuating circumstance, contact your lab TAs and Jennifer Parham-Mocello.

Pair Programming: In this lab, you can choose a partner for pair programming. **You must be checked off together. You only need one computer for pair programming.** One person will be the driver, who controls the computer and codes, while the other is the navigator, who observes and advises. After 20 minutes, you will switch driver and navigator, continuing this pattern until the task is complete. Please read more about pair programming and the benefits: [Pair Programming Student Handout](#)

Take 30 minutes and get caught up on anything you didn't complete in last week's lab and get checked off for Lab 6, and you can get up to 4 points added to the last lab!!!

(4 pts) Understand Pointers/Dynamic Memory

Write 3 different functions in C++ to create memory on the heap without causing a memory leak. Hint: You will need to assign the address to a pointer that was created outside the function. Remember, you can return an address or change what a pointer points to (the contents of a pointer) in a function by passing the pointer by reference or by passing the address of the pointer.

What if you want to change the contents of what the pointer points to? **Make a function that will set the contents of the space on the heap.** Do you still need to pass by reference or the address of the pointer? Why or why not?

How will you delete the memory off the heap? Try doing it outside a function, now inside a function. **Make sure your delete function is setting your pointer back to NULL**, since it is not supposed to be pointing anywhere anymore. ☺

You can check to see if you have any memory leaks using valgrind.

%valgrind program_exe

(3 pts) Static 1-d arrays: C-style Strings

Change your implementation from lab #6 to use C-style strings, instead of C++ strings. A C-style string is a string of characters ended by the null character, '\0'. Since you don't know how big the message will be, you need to declare a character array large enough to hold a sentence, usually 256 characters will do! This means that the string can hold a maximum of 255 characters, plus one null character. ☺

```
char mssg[256];
```

Create another C-style string for your replace string too. Your `getline()` call needs to change to `cin.getline()`, and this function will automatically insert a null character at the end of the characters read from the user. Since you are using a C-style string now, you will need to use the `cstring` (or `string.h`) library, instead of `string`.

<http://www.cplusplus.com/reference/cstring/?kw=cstring>

Notice, there are still functions for copying, searching, and finding the length of C strings, but **assigning one string to another will not work anymore!!!** Your function prototypes from lab #6 will change to use C strings, rather than C++ strings. Remember, C strings use character arrays, which are pointers. This means you do not need to pass by reference anymore. By default, the contents of an array can change inside a function, when you pass the name of the array as an argument to the function. You will need to change your function prototypes and definitions to character pointers or character arrays:

```
void get_string(char *);
void set_replace_string(char *, char *);
int get_search_replace(char *, char *);
OR
void get_string(char []);
void set_replace_string(char [], char []);
int get_search_replace(char [], char []);
```

Now, re-write these functions to work with C strings, instead of C++ strings.

(3 pts) Dynamic C-style Strings

Now, what if the string of characters is more than 256? What if it is a lot less than 256? Do we really need all that space or maybe we don't have enough! Let's try create a C-style string that is just right! **Don't forget to leave room for the null character, '\0'.**

Use a function from the first part of the lab to create space on the heap that holds the exact amount of space needed. You need to ask the user how many characters they would like to make, and then you need to make that many, plus one for the null character!

You can make a new array on the heap with a specific number of characters by using `new` with the type and how many.

```
new char[max_chars];
```

Now, make sure you delete the whole array off the heap by specifying the array with `[]`.

```
delete [] mssg;
```

Extended Learning:

Can you grow the array to be just right without asking the user for the number of characters? What if you read one character at a time, including the '\n'? How can you use this to grow your array to just the right length? Make sure you don't have any memory leaks!!! ☺