LAB FIVE

HEAP MEMORY, FUNCTION POINTERS, MODULES, VALGRIND

CS2263, Fall 2021

LEARNING OUTCOMES

At the conclusion of the lab, students should be able to

- Write C programs that use pointers-to-pointers
- Use C's `typedef` functionality
- Write and use a comparator function for use with `qsort()`
- Use the `valgrind` utility to discover potential memory leaks in their programs

EXERCISE ONE

Review the current state of your Strings module. At this stage it should include:

```
typedef char* String;
```
(L16)

```
String mallocString(int stringsize);
```
(L10)

```
void freeString(String s);
```
(L10)

```
String duplicateString(String s);
```
(L10)

```
String* duplicateStringList(String* sl);
```
(L11)

```
int compareStrings(void* s1, void* s2);
```
(L18)

Ensure that it does, that the function declarations are in the .h file and the definitions (code) are in the .c file. Also add the getString() from the L15 (Code Along) presentation.

SUBMIT:

- A screen shot of the make command output for a successful compile.

NOTE:

While completing the rest of this lab you may discover you have issues with your functions and need to correct them. That's part of the lab's intent.

EXERCISE TWO

Using your Strings module write a simple program called `stringTest1` that creates a duplicate String from `argv[0]` (the program name) and prints it out, but does not call `freeString()`.

Ensuring that your `make` for the target includes the `-g` option, compile and run your program. You know that there's memory leak, but it won't necessarily show up when you run the program. Use `valgrind` to examine your program and report for the problem. Note that it might turn up other memory issues that you hadn't expected. *That's its job!*

Once you've found the memory leak using `valgrind`, add the statement to your function to free the String to your program, compile and run with `valgrind` again to demonstrate that your program manages memory correctly. Of course, you may have other issues that need fixing too.

USING `valgrind`

- Use of `valgrind` within a `Makefile` and output is shown and discussed in chapter 9.1.4 of the text. You can safely use the form (remember to substitute your specifics):

  ```
  $ valgrind --tool=memcheck --leak-check=full --verbose --log-file=<log-file-name> <your-program>
  ```

- Note also that you can view the manual from the command line:

  ```
  $ man valgrind
  ```

  (use the spacebar to page and 'q' to quit)

SUBMIT:

- the output from running your `Makefile`
- the log-file outputs from both runs of `valgrind`, and your final `stringTest1.c` file.
- A screen-shot of the program running (with and without `freeString()`) and its output

EXERCISE THREE

Using your Strings module write a program called `stringListTest` that duplicates the command line arguments (a list of strings) using your `duplicateStringList()` function and prints the list out. You should use `valgrind` to test for leaks

SUBMIT:

- the output from running your `Makefile`
- the log-file outputs from the final run of `valgrind`
- your `stringListTest.c` file.
- A screen-shot of the program running and its output

EXERCISE FOUR

Using your Strings module write a program called `stringListSortTest` based on `stringListTest` (from Exercise Three) that takes the duplicated list of strings and uses qsort and your compareStrings function to sort the list and then print the sorted list of strings. You should use `valgrind` to test for leaks

SUBMIT:

- the output from running your `Makefile`
- the log-file outputs from the final run of `valgrind`
- your `stringListSortTest.c` file.
- A screen shot of the program running and its output

EXERCISE FIVE

Push your modified Strings module files to the FCS git

SUBMIT:

- the modified source code for the Strings module, including the Makefile
- the screenshot of you pushing the program source to the FCS git

SUBMISSION

Before the due date for this lab, students should submit a single zip or tar file (named *LastName_FirstName_Lab5.zip or LastName_FirstName_Lab5.tar*) online to the lms containing:

- the required material for each question (use the headings indicating the question number) in a single pdf file (named *LastName_FirstName_Lab5.pdf)*
- Your source code directory:
    - This should include all of your source files, including any test programs.
    - This should not include object (.o) files and executables. Nobody needs to see those.