

# PERFORMANCE, DATA STRUCTURES AND ALGORITHMS

Exercise 13

Exercise 13

Debugging a memory lEAk

# Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

PURPOSE

The purpose of this exercise is to give you practice with memory management.

Memory management is crucial in C programming. A program that consumes more and more memory ties up resources in the system that other processes need to function.

There are many strategies to track memory usage and pinpoint the location of memory leaks. We examine two of them in this exercise: memory tracking and memory tracing. Both of these can (and should) be implemented using conditional compilation.

Memory tracking simply involves counting the number of times malloc and free are called. For example, you can create a global variable in your program that is incremented when malloc is called and decremented when free is called. If the value of this variable at the end of the program is not zero, there is a problem with the way you have managed memory in your program.

Using printf, you can display the values of pointers by using %p in your format string. This displays the value of a pointer (the memory address). With judicious placement of debugging print statements in your code, you can trace the memory usage of your program.

You can refer to the material you were given for Project 02 for examples of how to use memory tracing.

For this exercise, you are given the solution to Exercise 12 as a starting point.

ACTIVITIES

Perform each of the following activities. If you have questions, issues, or doubts, please ask for help and do not just guess.

1. Modify the program to allow the user to remove a node from the linked list. This includes:
2. Adding an option to the menu for removing a node.
3. Modifying main() to handle the option to remove a node.
4. Creating a function to remove the node.
5. Use conditional compilation and appropriate code to assess whether or not your program has a memory leak and, if it does, to find the leak and correct it. Your code should use both memory tracing and memory tracking as described above.
6. Test your program thoroughly to verify that it works correctly. Be sure to include special cases in your testing (e.g. remove a node from an empty list, remove the node from a list of one item, etc.)
7. Document your code carefully.
8. In your Engineering Notebook, explain your approach to creating your code and your testing strategy. Include examples of the test input you used and the corresponding output from your program, as well as your memory tracking and tracing output. Also include the limitations to your solution, if any.
9. When you are ready to submit your work, first remove all intermediate files from your src directory.
10. Save and archive your finished C program and upload it to the LMS.