CSCI 1730 Breakout Lab 08

GDB and Valgrind

Learning Outcomes

- Use gdb to debug programs with various errors.
- Use valgrind to find memory leaks in programs. Implement programs that use dynamic memory allocation and deallocation and have no memory leaks.

Problem / Exercise

The purpose of this lab is to get experience using gdb to debug C programs and valgrind to check for memory leaks.

Requirements

1. Read, understand, and work through the examples in the `gnu_debugger_tutorial.pdf`, which is on eLC. You may skip the section on installing gdb since gdb is already installed on odin. Do not submit any examples from `gnu_debugger_tutorial.pdf`; these examples are for practice only. Also, if you would like to generate a core dump binary file when your program crashes, you can run the Unix command `ulimit -c unlimited` and then compile and rerun the program that crashed, which should generate a binary file that begins with `core`.
2. Read through the The Valgrind Quick Start Guide at `https://valgrind.org/docs/manual/quick-start.html` to learn how to use this software to check for memory leaks. Additional documentation about valgrind can be found at their website, `https://valgrind.org`.
3. Fix the syntax error(s) in the file lab08.c, which is a program that has multiple errors and is provided to you on eLC. After you fix the syntax error(s), use gdb to find bugs in the provided file lab08.c, and fix all of the bugs in lab08.c. You may assume the comments in lab08.c are correct. You must use a C array dynamically allocated on the program's heap to fix lab08.c. The fixed program should not contain any compiler errors, compiler warnings, nor any compiler notes when compiled with the provided Makefile and the -Wall and -pedantic-errors compiler options. Also, the fixed program should not contain any memory leaks (you'll need to use valgrind to check for memory leaks). Your corrected version of lab08.c must compile into an executable called lab08.out that matches the examples in the example section. Lastly, do not change the function prototype for the sum function.

Examples

After you finish correcting lab08.c, the I/O from running lab08.out should match exactly what is shown below. Each example is a single execution of a correctly working program for this assignment.

```
./lab08.out
*x is 0
y is 0
z is -1
```

```
./lab08.out 8 1 2
*x is 8
y is 11
z is -1
```

```
./lab08.out 3 6 2 5 4
*x is 3
y is 20
z is -1
```

```
./lab08.out 100 200 300 400 500 600 700 800
*x is 100
y is 3600
z is -1
```

```
./lab08.out 21
*x is 21
y is 21
z is -1
```

1 C Program

1.1 General Requirements

1. Place the files for this assignment in a directory called `lab08`.
2. Your program's source file(s) must compile on odin or a vcf node using the provided Makefile and using the required compiler for this course (`gcc version 11.2.0`).
3. Place your C source code for this assignment in a file named `lab08.c`. See the Examples section to see what your C program should output. Compile your C program using the provided Makefile for this assignment. Do NOT modify any commands, targets, or dependencies that are associated with compiling source code in the provided Makefile for this assignment. If there are any warnings or errors from the compiler, then you should fix them before you submit this assignment to us for grading.
4. Your program cannot contain a memory leak when executed. You are responsible for using valgrind to test your program to ensure it does not contain a memory leak, and valgrind should be run for every test case for your program. A program that contains one or more memory leaks (definitely, indirectly, or possibly lost) identified by valgrind will lose points.
5. The only include directives that you are authorized to use in your source code for this assignment are listed below.

   ```
   #include <stdio.h>
   #include <string.h>
   #include <stdlib.h>
   #include <stdbool.h>
   ```

   You CANNOT use any other include directives anywhere in your source code.
6. All written and verbal instructions stated by the teaching staff (lecture instructor, lab instructor(s), etc.) must be followed for this assignment. Failure to follow instructions may result in losing points.

1.2 Coding Style Requirements

1. All functions must be commented. Comments must include a brief description of what the function does, its input(s), its output, and any assumptions associated with calling it. If your function has a prototype and an implementation separated into different parts of your source code, then you only need to put the comments for that function above its prototype (there is no need to comment both the prototype and implementation; commenting the prototype is sufficient).
2. All structs, unions, and enums must be commented.
3. All global variables and static variables must be commented.
4. All identifiers must be named well to denote their functionality. Badly named identifiers are not allowed. For example, identifiers like a, aaa, b, bbb, bbbb are bad names for identifiers.
5. Every line of source code must be indented properly and consistently.

1.3 README.txt File Requirements

Make sure to include a `README.txt` file (use a .txt file extension) that includes the following information presented in a reasonably formatted way:

- Your First and Last Name (as they appear on eLC) and your 810/811#
- Instructions on how to compile and run your program. Since we are using a Makefile for this assignment, then these instructions should pretty simple based on the provided Makefile.

2 Submission

Submit your files before the due date and due time stated on eLC. Submit your files on eLC under the Assignment named Lab 08. Submit only the following files.

1. All source files required for this lab: lab08.c
2. A README.txt file filled out correctly

Do not submit any compiled code. Also, do not submit any zipped files or directories. Do not submit a Makefile. We will use our own copy of the provided Makefile to compile your program. We only need the files mentioned above with the file extensions aforementioned.

3 Breakout Lab Attendance: 3 points

Breakout lab attendance is required for this assignment, and it is worth three points. Students must physically attend the entire period of the breakout lab section that they registered for during the week of this assignment to earn these three points for attendance. If you complete the assignment before the end of your breakout lab period during the week of this assignment, then you are still required to attend the entire breakout lab period to earn attendance points, and you may use this time for independent study for the next exam in this course.

4 Grading: 7 points

If your program does not compile on odin or a vcf node using the provided Makefile and using the required compiler for this course `(gcc version 11.2.0)`, then you'll receive a grade of zero on this part of the assignment. Otherwise, your program will be graded using the criteria below.

Program runs correctly with various test cases on odin	7 points
README.txt file missing or incorrect in some way	-1 point
Not submitting to the correct Assignment on eLC	-1 point
Late penalty for submitting 0 hours to 24 hours late	-2 points
One or more compiler warnings	-2 points
Not adhering to one or more coding style requirements	-2 points
Valgrind identifies a memory leak (definitely, indirectly, or possibly lost)	-2 points
Not submitting this assignment before its late period expires	-7 points
Penalty for not following instructions (invalid I/O, etc.)	Penalty decided by grader

You must test, test, and retest your code to make sure it compiles and runs correctly on odin with any given set of valid inputs. This means that you need to create many examples on your own (that are different than the aforementioned examples) to ensure you have a correctly working program. Your program's I/O must match the examples given in the Examples section. We will only test your program with valid sets of inputs.