CSCI-1200 Data Structures — Spring 2017
Homework 4 Debugging (& List Iterators)

IMPORTANT: Read this whole assignment before you download or start working on the code. The format is different from our other assignments, so be sure you have read and understood all of the instructions.

Studying and documenting a software error is an essential skill that allows you to reproduce the error, fully describe the problem to other software developers on your team, and ultimately verify that your proposed code fix actually solves the problem.

In this assignment we provide legacy source code for an existing, complete program that decrypts an input text file. While the program is "complete", it is in need of some serious debugging. Your job is to debug this program, and then submit both your debugged version of the program *AND* a detailed writeup of how you found and fixed the bugs.

Filing a Detailed Bug Report

To document your debugging process you should get into the habit of taking detailed step-by-step notes (and possibly screenshots) of the errors you encounter. Each bug formally filed with a software project is assigned an issue or bug #. A well-written and complete *issue or bug report* should contain the following relevant information (as appropriate):

- Overall development environment: What operating system? Compiler? Computer hardware? etc.
- Can you reproduce the problem? What were the command line arguments, input file(s), or runtime actions (keyboard input, mouse clicks, etc.)?
- Describe in words the erroneous behavior: E.g., compilation error/warning, linker error, runtime crash, runtime buggy output, program hangs, etc.
- Record the exact text of any error messages.
- Record the exact output (to the screen or to a file). How is this different from what you expected?
- If you are using a debugger: What file, function, and line of code? What variable had the incorrect value? etc.
- Include screenshots as appropriate to supplement the information above.

For this assignment, you should make your own notes about what tools helped you pinpoint the error. If you learned how to use a feature of your debugger, make notes for yourself on how to use that feature and why it was helpful (or not so helpful) to track down this bug.

Submitting a Bug Fix or Patch

Once you have a identified the exact bug and figured out what needs to be changed, you should submit a *patch* with the proposed modification of the source code.

- What bug or issue # does this patch fix?
- The file name, line number, and original line(s) of code.
- The new replacement line(s) of code.
- Describe in words why the new code fully addresses and solves the original problem.

After filing an official bug fix or patch for a large software project, your proposed change will be reviewed by the development team and you may be requested to make edits before the change is accepted and ultimately merged into the project source code.

You should use this assignment as an opportunity to practice making a detailed record of each bug you fix. However, you will not submit the full transcript of your debugging process (more later in the handout).

Goal: Lots of Practice with a Step-by-Step Debugger

The "Rules" for this assignment:

- **Pretend this is a huge, complex project.** It's so big you can't read all of the code initially. When you encounter a bug, study the relevant surrounding code as needed to understand the situation and propose a solution.
- **Try to avoid falling back on "Print Debugging".** Don't add `std::cout` or `printf` statements to the code. Don't comment out blocks of code. Instead, use the debugger to navigate through the code and examine specific values in the code. Find a good reference or tutorial on your chosen debugger and learn how to use its amazing features.
- **Propose the smallest change bug fix.** When working with code written by another developer, we are often tempted to make major changes because we personally would have written things differently. For this assignment, you should adopt the "If it ain't broke, don't fix it" policy. Also, when making a bug fix, if there is more than one way to correct the problem, chose a solution that is the smallest edit (fewest number of lines or characters added/edited/deleted).

After completing this assignment, you should have plenty of practice with all (or most) of these things with a debugger of your choice:

- Launch/run the program within the debugger (and specify any necessary command line arguments).
- Manage (create, list, & delete) breakpoints and watchpoints, including *conditional* breakpoints and watchpoints.
- Control execution of the program one line at a time, either stepping into or over helper functions (using `step`, `next`, and `continue`).
- Examine values of arguments and local variables in the current frame.
- View the overall call stack, and examine different frames on the stack.

In addition to these step-by-step debugger skills, you're encouraged to enable all warnings on the compiler (with `-Wall`) and use a memory debugger (e.g., Dr. Memory or Valgrind).

The Secret Message Decryption Program

The provided code is organized into four separate *operations* with additional helper functions. There are many intentionally-placed bugs within this code. Each operations function is designed to return a specific value that contributes to the decryption process; however, the bugs in these functions will cause the function to return the wrong value or crash! By finding and fixing all the bugs, you will decrypt the file piece by piece. The comments left behind by the original developers will tell you how the program is expected to work.

For example, consider the first function, `arithmetic_operations`. It starts by initializing 19 different variables with various mathematical operations, and its comments indicate what values they should hold to pass all the tests and return the correct value. However, it is fairly easy to notice that these variables are not being calculated correctly, since the program will crash or fail an assert on one of its own tests. Thus, the "bug fix" here is simple: make sure the variables get the correct values.

When all bugs are successfully fixed and the program is run and passed in the encrypted file as an argument, it will decrypt and print the contents of the file. It will be obvious when it has been decrypted correctly — you will know when you have it.

We have prepared multiple differently-buggy versions of the secret message decryption program. Login to Submitty to obtain your assigned version of the buggy decryption source code.

The program should be compiled with the following command. The `-lm` flag tells `g++` (or `clang++`) to explicitly include the math library, which otherwise might not get linked correctly on some platforms.

```
g++ operations.cpp -lm -o decrypt.exe
```

To run the program, provide 3 arguments: a flag to indicate which operations to run, the encrypted input file, and the name of an output file where the decrypted secret message output should be written.

```
./decrypt.exe  --arithmetic-operations  encrypted_input.txt  secret_message_output.txt
./decrypt.exe  --file-operations        encrypted_input.txt  secret_message_output.txt
./decrypt.exe  --array-operations       encrypted_input.txt  secret_message_output.txt
./decrypt.exe  --vector-operations      encrypted_input.txt  secret_message_output.txt
./decrypt.exe  --list-operations        encrypted_input.txt  secret_message_output.txt
./decrypt.exe  --all-operations         encrypted_input.txt  secret_message_output.txt
```

Your Writeup

From your detailed notes, create an approximately 1000-2000 word written report of the debugging process. Rather than detail every bug found and fixed, chose 5-10 bugs that are interesting and demonstrate a variety of debugging methods. Try to show off each of the debugger skills listed above in at least one of these bug plus bug fix descriptions. *NOTE: We're more interested in your thought process, the description of the mechanical debugging process, and the new skills you learned than the formal bug report and code patch.*

You may include screenshots or paste code blocks. But your report should be concise and well-written. You should not have font smaller than 10 point font, and your writeup should be no more than 6 pages in length. You should format this report as a `.pdf` document. We will not accept or grade any other file format for the report.

Grading Criteria

Your grade for this assignment will be split into the following components:

15 pts Successful decryption of the secret message. (Partial credit will be awarded for partial decryption.)

10 pts Source code changes to fix bugs are small. (See the "Rules".)

25 pts Writing Quality:

- Report organization and structure. Don't just submit a full chronological transcript of your debugging process.
- Readability: Be concise and use complete sentences. Avoid spelling and grammar mistakes.
- Clarity: Describe each selected bug and why it is causing the program to fail. Describe how you found the problem. For non-trivial bugs, also explain why the fix you provided is a good one.
- The Debugging Process: Include things you tried that did not work and tell us about different approaches you took. If you could not find a bug that you knew was there, show us your efforts to find it.
- Screenshots of Debugger Use: Include a select few that supplement your text.

Note on Academic Integrity

Normally, talking about the solutions to problems with other students is fine as long as you do not share code. However, the whole point of this homework is about learning how to find, diagnose, and fix bugs in a program. **Discussing bugs with other students is thus an academic integrity violation for this assignment.**

FINAL NOTE: If you earn at least 10 points (not including the README.txt and compilation) on Submitty by 11:59pm on Wednesday, Feb. 22, you may submit your assignment on Friday, Feb. 24th by 11:59pm without being charged a late day.