Valgrind's Memcheck

The C compiler will give you warnings and errors for things it can tell are problematic. However, what the compiler can do is limited to static analysis of the code *i.e.*, things it can do without running the code or knowing the inputs to the program. Consequently, there are many types of problems the compiler cannot detect. These types of problems are typically found by running the code on a test case where the problem occurs.

However, just because the problem occurs does not mean that it produces a useful symptom for the tester. Sometimes a problem can occur with no observable result, with an observable result that only manifests much later, or with an observable result that is hard to trace to the actual cause. All of these possibilities make testing and debugging more difficult.

Ideally, we would like to have any problem be immediately detected and reported with useful information about what happened. When we just run a C program directly, such things do not occur, as the compiler does not insert any extra checking or reporting for us. However, there are tools that can perform additional checking as we run our program to help us test and debug the program. One such tool is *Valgrind*, in particular, its *Memcheck* tool.

Valgrind is actually a collection of tools, which are designed so that more can be added if desired. However, we are primarily interested in Memcheck, which is the default tool and will do the checking we require. Whenever you are testing and debugging your code, you should run it in Valgrind(called "valgrinding your program"). While valgrinding your program is much slower than running the program directly, it will help your testing and debugging immensely. You should fix any errors that valgrind reports, even if they do not seem to be problems.

To "valgrind your program," run the**valgrind** command, and give it your program's name as an argument (Memcheck is the default tool). If your program takes command line arguments, simply pass them as additional arguments after your program's name. For example, if you would normally run**./myProgram hello 42**, instead run **valgrind ./myProgram hello 42**. For the most benefits, you should compile with debugging information in your program (pass the -g or -ggdb3 options to gcc).

When you run Valgrind, you will get some output that looks generally like this:

```
8

9

10

11
```

```
12

13

14

==11907==     in use at exit: 0 bytes in 0 blocks

==11907==   total heap usage: 2 allocs, 2 frees, 128 bytes allocated

==11907==

==11907== All heap blocks were freed -- no leaks are possible

==11907==

==11907== For counts of detected and suppressed errors, rerun with: -v

==11907== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Each line that starts with **==11907==** here is part of the output of Valgrind. Note that when you run it, the number you get will vary (it is the process ID, so it will change each time). Valgrind prints a message telling you what it is doing (including the command it is running), then it runs your program. In this case, the program did not produce any output on **stdout**; however, if it did, that output would be interspersed with Valgrind's output. There are also no Valgrind errors—if there were, they would be printed as they happen. At the end, Valgrind gives a summary of our dynamic allocations and errors. The last line shows that we "valgrinded cleanly"—we had no errors that Valgrind could detect. If we did not valgrind cleanly, we should fix our program even if the output appears to be correct.

Note that if our program has errors, Valgrind will report them and keep running (until a certain error limit is reached). Whenever you have multiple errors, you should start with the first one, fix it, and then move to the next one. Much like compiler errors, later problems may be the result of an earlier error.

One common misconception novice programmers often get is that they should run Valgrind only after they have debugged their program and otherwise think it works. However, you will find the debugging process much easier if you use Valgrind's Memcheck throughout your testing and debugging process. You may find that the odd or confusing bug you have not been able to figure out for hours is actually caused by a subtle problem you did not notice earlier, which Valgrind could have found for you. Note that Valgrind does not "play nice" with code that has been compiled with-fsanitize=address, so you should compile without that option to valgrind your code. Ensuring that both tools cannot find any problems with your code is likely a great idea, as they can detect different problems.

We will introduce the basics here but recommend further reading in the Valgrind user's manual: http://valgrind.org/docs/manual/manual.html for further information.

Completed