The University of Queensland
School of Information Technology and Electrical Engineering
Semester Two, 2019
CSSE2310 / CSSE7231 - Assignment 2
Due: 8:00pm Tuesday 3 September, 2019
Marks: 50
Weighting: 25% of your overall assignment mark (CSSE2310)

Introduction

The purpose of this assignment is to develop and test your debugging skills. You will be awarded marks for solving a collection of puzzles which involve different aspects of debugging. The puzzles are of two types (with up to 25 marks available from each): *binary bomb* and *quicksand*.

This is an *individual assignment*. This is different from programming assignments in that this assessment is about technique rather than writing code. **Do not discuss your methods for solving the specific problems given in the assignment however.** You should not actively help (or seek help from) other students with the exercises.

The Binary Bomb

The aim of this part of the assessment is to develop and test your skills using the gdb debugger. This exercise will solving puzzles (called phases) about the behaviour of a program called the *Binary Bomb*. You must "defuse" each phase by entering the correct passphrase. You will need to deduce this passphrase by gaining insights into the the bomb program's operation in `gdb`. You will be given marks for the number of phases you solve correctly and the number of attempts you take to do so. This assignment must be done on *moss*. The *Binary Bomb* will ***NOT*** run on any other system. The operation of the bomb will be demoed in lectures.

Bomb rules

1. You shall not D.O.S. moss (unlikely but it needs to be said).
2. You shall not discuss strategies for solving phases. Working out how to do that is part of the assignment.
3. You shall not allow anyone else to access your bomb or associated source files (or a copies or excerpts).
4. Your bomb will start a second program in the background. Any attempt to attack or compromise that program may be considered misconduct. (That is not something you can do by accident).
5. Subject to all of the above, analyse the bomb with whatever tools you like.

Important programs

- getbomb - retrieves *your* version of the bomb. If you run this command it will give you the same bomb (or an updated bomb with the same answers as the one you had).
- bomb - executes the bomb and prompts for the passphrase to defuse the current phase. It takes as a parameter, the name of the phase you wish to attempt:
  - demo1
  - demo2
  - demo3
  - demo4
  - phase1
  - phase2
  - phase3
  - phase4
  - phase5
  - phase6

  The `demo` phases do not affect your mark, so practice on them to ensure you understand what is happening. You may attempt the phases in any order and do not need to complete one before attempting another.

  You can also start the bomb with the `status` argument. This will report your history so far.

Your attempt will not be recorded until you answer `Y` to "Are you sure". You can quit/kill/restart the bomb at any time before this with no penalty. **We recommend that you run your bomb inside gdb.**

Marks

Your attempts will be recorded automatically when you run the bomb on *moss*. Each phase of the bomb is worth a maximum of 25/6 marks. The mark you receive for each phase depends on the number of attempts you make to defuse that phase. Specifically, if it takes n attempts to defuse a phase, then your mark will be:

$$25/6 \times 0.85^{n-1}$$

Bomb Tips

- Remember that you must enter input before a phase starts to run.
- You should be familiar with the commands `p`, `break`, `run`, `cont`, `next`, `step`, `up`, and conditional breakpoints.
- This is not a race. It is very easy to make silly mistakes due to lack of concentration.
- You may believe the comments written in the bomb.

Quicksand

Where as in the bomb you used a debugger as a problem solving tool, in this set of puzzles, you will be using the compiler. The aim of this part is to develop your ability to identify the origin of problems in code. Note that there is an important difference between the cause of a problem (as far as it can be determined) and the symptoms which result (possibly much later in the program).

The `quicksand` can provide you with a (buggy) program and will also allow you to compile and run it. Your task is to identify the line of source code from which the fault originates. You are not being asked to fix or explain the bugs, just to identify the problem line.

So how do you find out more about quicksand? How do you go about solving puzzles? To encourage use of system documentation, the rest of the details for quicksand can be found in quicksand's man page on moss.