Lab 6– Debugging with gdb

This lab is a **debugging** exercise. The goal is to learn how to use gdb for analyzing and debugging code. You can use the lab slides and online sources to help you on using gdb.

To complete the lab, you need to finish both assignments using gdb, and show your steps and explain methodology to one of the TAs.

Assignment 1 – gdb basics

You are given the code **factorial.cpp**, which finds the factorial of a user-entered value. However, the code is not properly working. Compile factorial.cpp **using a makefile**, create an executable named "factorial.out", and test it.

To understand what is happening in the code, you will be using gdb. Note that you need to **compile** factorial**.cpp with the debugging flag** (-g) to be able to use gdb. To run gdb, type "gdb logf" and then use run, break, continue, step, print, and other commands to debug your program.

Set breakpoints at lines 13 and 17, and print the value of **&fact** at these lines. Write down the sequence of commands you used in this step and the **&fact** values at these lines.

Explain why the address of a program variable, fact, is changing during the execution. Provide a way to fix the problem, and verify that you have fixed the problem by checking the value of **&fact**.

Extra Credit – backtracing

You are given the code **sum.cpp**, which sums up the numbers from a user-entered number down to 1 (e.g., if the user enters 4, the program sums up 4, 3, 2, 1). You need to compile it with c++11 support (add the argument `-std=c++0x` to g++). When you compile and run the code, you will get a segmentation fault. **Modify the makefile to compile the code**.

Run your binary with gdb. Enter "100" when you are asked to enter a number. Once you get a segmentation fault, use "backtrace" in gdb to track down the cause of the problem. Observe the backtrace output. You will see that main() calls printSum(), which then calls strcpy(). Everything called after strcpy() is not really under your control. So, set a *conditional breakpoint* to stall the code when the user enters "100" as input, at the line where strcpy() was called during the code's execution.
After setting the breakpoint, you need to re-run the program in gdb. Once you get to the breakpoint, print the contents of "line" and "buffer". Make note of the contents of both variables.

Can you tell the cause of the segmentation fault looking at the gdb output you have observed so far? What is the cause? Explain.

To pass this lab, show your notes and explain your methodology to one of the TAs.

Note: If you get a segmentation fault outside gdb, the program will dump the memory state in a file with the name “core.XXXXX” (e.g., core.13655). gdb can use these files to analyze the fault. You can delete those files once you are done with debugging.