**University of California at Berkeley**
**College of Engineering**
**Department of Electrical Engineering and Computer Science**

EECS 61C, Fall 2003

# Lab 2: Strings and pointers; the GDB debugger

**PRELIMINARY VERSION**

## Goals

To learn to use the gdb debugger to debug string and pointer programs in C.

## Reading

Sections 5.1-5.5, in K&R
GDB Reference Card (linked to class page under "resources.")
Optional: Complete GDB documentation (http://www.gnu.org/manual/gdb-5.1.1/gdb.html)

**Note:** GDB currently only works on the following machines:

- torus.cs.berkeley.edu
- rhombus.cs.berkeley.edu
- pentagon.cs.berkeley.edu

**Please ssh into one of these machines before starting the lab.**

## Basic tasks in GDB

There are two ways to start the debugger:

1. In EMACS, type *M-x gdb*, then type *gdb <filename>*
2. Run *gdb <filename>* from the command line

The following are fundamental operations in gdb. Please make sure you know the gdb commands for the following operations before you proceed.

1. How do you run a program in gdb?
2. How do you pass arguments to a program when using gdb?
3. How do you set a breakpoint in a program?
4. How do you set a breakpoint which which only occurs when a set of conditions is true (eg when certain variables are a certain value)?
5. How do you execute the next line of C code in the program after a break?

6. If the next line is a function call, you'll execute the call in one step. How do you execute the C code, line by line, inside the function call?
7. How do you continue running the program after breaking?
8. How can you see the value of a variable (or even an expression) in gdb?
9. How do you print a list of all variables and their values in the current function?
10. How do you exit out of gdb?

1. **Debugging a short C program**

   Consider the C program appendTest.c. You can copy it from ~cs61c/labs/lab02. Compile and run the program, and experiment with it. Try appending a few strings, and notice that it does not always produce the correct result. (Press Ctrl-C to exit). You will now use gdb to debug this program.

   First, the program must be compiled with debugging information. To do this, use the **-g** flag:

   gcc -g -o appendTest appendTest.c

   Now start *gdb* on the resulting program, either in emacs or on the command line. Set a breakpoint in the append function, and run the program. When the debugger returns at the breakpoint, step through the instructions in the append function line by line, and examine the values of the variables. In particular, examine the value of s1 just before the function returns? Why is this a bug? Hint: How does C represent strings? Fix this bug.

2. **The infamous "segmentation fault"**

   A *segmentation fault* or *bus error* is a common pointer error in C programs. In general, it is caused by an invalid pointer dereference, or by using an invalid address. In this part you will debug a program with such an error.

   Compile and run the program average.c (found in ~cs61c/labs/lab02). As its name suggests, the program is supposed to compute the average of a set of numbers. Notice that currently, the program seg faults after accepting more than one input.

   Make sure the program has been compiled with debug information, and load and run the program in gdb. Notice that gdb traps the segmentation fault, and brings you back to the debugger.

   First, verify the current location in the program. The command to do this is backtrace (bt for short), which will print a stack trace similar to the trace printed in Java programs

when an exception is not caught. Notice that the program is currently deep in several system function calls. Since the system code is correct (at least we hope!), use the frame command to move to the last function called in our code. Note that gdb can print the exact line of the segmentation fault. Examine this line carefully, and fix the error.

Try recompiling and rerunning. The program now reads values correctly, but returns an incorrect average. Use gdb to debug and fix the program by examining the output values of read_values. To do this, either set a breakpoint using the line number, or set a breakpoint in the read_values function, and then continue execution to the end of the function and view the return values. (To run until the end of the current function, use the finish command).