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

EECS 61C, Spring 2004

Lab 1

Goals

These lab exercises are intended to show you how to run a C program on the EECS instructional computers, to introduce you to the gdb debugger, and to get you thinking about the internal representations of numbers.

Initial preparation

Read sections 1.1-1.5 in K&R, and sections 4.1-4.2 in P&H.

If you do not yet have an account form, pick one up from your t.a. Login. Then ssh po to change your password using the passwd command. Logout from po—you should still be logged in on the workstation in 271 Soda—and copy the directory ~cs61c/labs/lab01 to your home directory.

Exercise 1: Simple C program.

Fill in the blank in the following C program, also in ~cs61c/labs/lab01/output0.c, so that its output is a line containing 0. Don't change anything else in the program.

```
#include <stdio.h>

int main ( ) {
   int n;
   n = _____;
   printf ("%c\n", n);
   return 0;
}
```

To verify your answer, compile the program using the gcc command, then run it with the command a.out.

Exercise 2: Debugger.

Compile your solution to exercise 1 with the "–g" option. This causes gcc to store information in the executable program for gdb to make sense of it. Then single-step through the whole program by

1. setting a breakpoint at main;
2. giving gdb's run command; and
3. using gdb's single-step command.

Type help from within gdb to find out the commands to do these things.

Exercise 3: Octal Dump.

The program ~cs61c/labs/lab01/mysteryout when run apparently produces a blank line as output. Find out what it really prints using the od (octal dump) command; man od will give you information about how it works.

Exercise 4: The Biggest Integer.

In class we discussed number representation. In particular, we discussed unsigned integers and two's complement, the almost ubiquitous format for signed integers. Look at biggestInt.c. You may wish to read through the comments but at this point it is not critical that you understand exactly how the program works. Basically, it is a C program that will tell you some useful information about certain C data types. It does this by exploiting the fact that C does not check for overflow and wrap around conditions. Compile and run the program and answer the following questions:

1. The first piece of information output by the program tells you the value of the most significant bit (MSB) of an unsigned int on your terminal. Based on this information, How many bits does the C data type unsigned int have?

2. The second piece of information tells you the value of the largest positive signed long on your terminal. Based on this information, how many bits does the C data type long have?

3. The third piece of information tells you the value of the most negative signed int on your machine. Based on this information, do the unsigned int and signed int have the same number of bits?

4. 2's Complement number representations have one more negative number than they do positive numbers. That is, the most negative number does not have a positive counter-part. The final piece of information printed is the signed value of what you get when you try to negate the most negative value. Can you explain why this happens?