Debugging

Overview

- Compilation
- Two key tools in the trade
  - gdb
  - valgrind
- Techniques
- Demo
- Reading

Compilation

- By default…
  - The compiler generates vanilla code
- To “debug”
  - You *could* add print statement everywhere (ouch!)
- Much better way…
  - Use a symbolic tool
  - Allows to “follow” the code symbolically
  - Allows to “inspect” the memory state
  - Allows to “stop” on all sorts of conditions

Requirements

- To enable symbolic debugging you must
  - Tell the compiler to generate extra information
  - Tell the linker to include that information in the executable.
- Downside
  - The executable is a little bigger.
  - The executable *may* be a little slower.
  - The presence of symbolic information *may* hide bugs.
- Upside
  - Debugging is much easier!

Compilation Change

- On UNIX Operating systems (OS X included), this is quite easy
  - With cc/gcc/clang
    - Add an option on the command-line: “-g ”
  - Withe the linker (again cc / gcc / clang, or directly with ld)
    - Add an option on the command-line: “ -g ”
- On Windows…. This is all done through the UI (difficult!)

Example

- Simplest version

```
src (master) $ cc -g hello.c
```

  - It compiles hello.c and links into an executable a.out with symbols

- Alternatively (decomposed commands)

```
src (master) $ cc -c -g hello.c  -o hello.o
src (master) $ cc -g hello.o -o hello
src (master) $ ./hello
Hello world!
```

  - Clearly, no visible difference yet….
  - However, debug information is present and usable in “hello”

The GNU Debugger (gdb)

- Available on Linux, OS X 10.1 through 10.12
- Features
  - A simple CLI (command-line interface) debugger
  - Meaning…. No ***graphical*** user interface
  - Yet, it is extremely powerful and works from any terminal
  - Lots of tutorials online (just a handful)
    - http://www.dirac.org/linux/gdb/
    - http://www.unknownroad.com/rtfm/gdbtut/gdbtoc.html
    - http://www.cs.cmu.edu/~gilpin/tutorial/

Starting up gdb

- Quite easy to do!
  - Simply run gdb passing the executable in argument

```
src (master) $ gdb ./hello
GNU gdb 6.3.50-20050815 (Apple version gdb-1824) (Wed Feb  6 22:51:23 UTC 2013)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "x86_64-apple-darwin"...Reading symbols for shared
libraries .. done

(gdb)
```

What can you do?

- ***Lots*** of commands available (only showing a handful here)
  - help [get some textual help]
  - run [run the program — go —]
  - list [show a listing of the source code]
  - print [print pretty much anything you want]
  - break [set a breakpoint]
  - next [go to next instruction in source code]
  - step [step into a function call]
  - cont [resume execution until next stop]
  - where [show the current backtrace — stack —]
  - up [navigate one frame up in the backtrace]
  - down [navigate one frame down in the backtrace]
  - thread [show the list of threads in existence]
  - set [modify a gdb or a program variable!]

Source Code

```c
#include <stdio.h>
#include <stdlib.h>
int fact(int n);
int main(int argc,char* argv[]) {
   if (argc < 2) {
      printf("usage is:…”);
      return 1;
   }
   int n = atoi(argv[1]);
   int r = fact(n);
   printf("Factorial(%d) = %d\n",n,r);
   return 0;
}

int fact(int n) {
   if (n == 0)
      return 1;
   else
      return n * fact(n-1);
}
```

Shell session transcript

```
src (master) $ cc -c -g fact.c -o fact.o
src (master) $ cc -g fact.o -o fact
src (master) $ gdb ./fact
```

Gdb Demo