Lab 8

Debugging

October 27th, 2010
James Marshall

Previous Labs

- Lab 7
  - Stacks
  - Assignment 3
- Lab 6
  - Command Line Arguments
  - Recursion and CLA example
- Lab 4
  - File I/O

Assignment 3

- You have all the tools you need to finish.
- If you haven't started yet, you are behind.

Remember Memory

- malloc
- free
- realloc
  - http://opengroup.org/onlinepubs/007908775/xsh/
  - Read this!

Motivation

- Debugging
  - This is why you need to start early!
  - Toughest part of coding
    - Most common too
  - Has potential to drive you insane
  - C is not friendly

Make Your Life Easier

- Small pieces of code.
  - Write small, modular piece of code (function)
  - Compile it
  - Debug
  - Test it
  - Debug
- Repeat until program is finished
- Keep code as simple as possible

Quote

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

-Brian Kernighan

- Unix contributor, Princeton professor

Warning

- DO NOT:
  - Write your entire program
  - Compile it
  - Test it
- Compiling and testing should be done throughout.

How to debug

- Compiler errors
- Testing
- Sanity Checks
- Seg faults!
- GDB

Compiler Error

- Compile code often
- Learn compiler errors
- Remember, only trust first error listed
  - Errors may hide errors
  - Later errors may be caused by earlier errors
- Pitfalls
  - Header files
  - Semi-colons
  - == vs. =

Testing

- Compiling does not mean working!
- Run-time errors
  - Exposed only when code is run, often only under certain conditions
- Functional
  - Does the program do what it is supposed to?

Testing cont.

- Test your functions!
  - Test each case, and corner cases
    - ie: if {} else {}, From field with 50 characters
  - Make sure the function does what you want it to
    - Delete a message, find all of the appropriate messages
- Then Test your whole program
  - This is why we give you example inputs.
  - But remember that the examples aren't exhaustive

Sanity Checks

- Program not behaving as expected? Don't know why you are getting a run-time error?
- Make sure it's doing what you think it is.
- printf() - variables
- Trace by hand, compare to printf statements
- Start by testing large sections, then narrowing down possibilities

Segmentation Faults

- Something is wrong with memory.
- printf() is no good
  - Statements may be reached, but will not finish executing before seg fault.
- Comment out possible offending lines
- Many functions return NULL when they fail
  - malloc
  - fopen

GDB

- GDB - GNU project DeBugger
  - GNU – GNU is Not Unix
- Very powerful
  - We'll look at a limited sub-set
- Can tell you what line causes a seg-fault

Example fopen_ex.c

```
#include "stdio.h"

int main(int argc, char * argv[]) {
  FILE * file;
  int n;

  file = fopen("input.txt", "r");
  fscanf(file, "%d", &n);
  printf("Number: %d\n", n);
}
```

GDB compiling

- Use `-g` flag
- C compiling typically obscures useful debugging information.
- `gcc -g -o fopen fopen_ex.c`

Run GDB

- `gdb fopen`
  - (gdb) `run`
  - (gdb) `backtrace`
- Program crashes after run
- `backtrace` shows the function call stack
  - Much less useful without `-g`

A Bit More GDB

- Run with arguments
  - `gdb fopen2`
  - (gdb) `run arg1 arg2 ...`
- Can set break points
- Print variables
- Examine memory address
- http://ace.cs.ohiou.edu/~bhumphre/gdb.html
- http://www.cs.cmu.edu/~gilpin/tutorial/

Review

- You are ready for assignment 3
- Start early, so you can ask questions early, and find bugs early
- Debugging
- Testing
- GDB