Debugging

Eric McCreath

Prevention better than cure

Aim to write correct and robust code from the very beginning. A few tips:

- be consistent with your coding style,
- focus on correctness and readability over performance,
- check return values of calls to libraries/system calls,
- using commonly used libraries when possible,
- unit test your code, and
- code reviews are great if you can get someone else to read over your code.

Inductively tracking down errors

Tracking down errors in your code is a little bit like the process of science. Come up with a theory that fits the facts about what is wrong with your program. Use this theory to direct you to the problem, or modify your code to test the theory.

To help track down problems you can:

- use printf to provide info about what your program is doing,
- use strace and ltrace to see what your program is up to,
- use a debugger,

Using a debugger

- If you compile your code with '-g' you can make use of the **'gdb'** debugger.
- Once you have started **gdb** with your program as the parameter. You can run and step through the program and see data within the program as it goes.
- A few helpfully commands with the debugger include:

```
run - runs the code.
start - start the code (but only take one step).
step - take one step.
bt - show the stack frames.
display <var> - show the value of a variable (this is very powerful).
break <line number> - add a break point.
quit - end running the debugger.
```

Exercises

- Track down the bugs with:

```
// same - test if any two elements in "array" are the same.
//        The array has "size" elements.
int same(int array[], int length) {
    int i,j;
    for (i=0;i<=length;i++) {
        for (j=0;j<=length;j++) {
            if (array[i] = array[j]) return 1;
        }
    }
    return 0;
}
```

- Make some unit tests for the above code.
- Use a debugger on a program you have written.