Introduction to Low-Level Programming

- Review Syllabus
- Class Grades Server
  - grades.cs.umd.edu
- Program #1A Handout
  - It's Due two weeks from today
- Discussion Sections
  - Will meet in 1120 AV Williams
- Reading
  - Chapters 1 & 2
    - Skip 1.1.4

What is “low-level” Programming

- Possible Definitions
  - Programming where quirks and warts of hardware are visible
- The C Language
  - lower level than Java (need to manage memory)
  - higher level than assembly language

Why Study Low-level Programming?

- Provides Understanding of How Things Work
  - Compilers, Processors, Data Structures
- Allows access to Hardware when required
  - Writing device drivers
  - Creating Operating Systems
- Can be faster
  - If you are careful and good
  - BUT can also be:
    - Slower – you write low level routines where faster ones exist
    - Dangerous – easier to have software that
      - Crashes program (or OS)
      - Fails in odd ways (memory corruption)
- It's Fun!
  - Or at least I think so

The “C” Programming Language

- Widely Used for System Programming
  - Writing Operating Systems and Compilers
- “Middle Aged” Language – created in late 1970’s
  - Reflects lessons learned from Fortran, COBOL
- Goals
  - Designed to allow low-level programming
  - Small language
    - Syntax is relatively simple
- Major Differences from JAVA
  - Explicit Memory Allocation and Deallocation
  - Procedural rather than object oriented
  - *Compiles* directly to machine code (rather than byte codes)

What Does a C Program Look Like?

```
#include <stdio.h>
int main(void)
{
   printf("Hello World\n");
}
```

- First thing executed is the main subroutine
- Return from main subroutine terminates the program

Preprocessor

- Not everything is in one file
  - Definitions of routines are in separate files
- Function *Prototypes*
  - Define name, parameters, and return types
  - Sort of like how methods are defined in class definitions
  - Example: int myFunc(int a, int b);
- #include
  - provides ability to include definitions from other files
  - Generally only include prototypes from external files
    - Actual code is kept in a separate file
    - Files are generally compiled seperately

Formatting Your program

- Comments
  - Must be surrounded by /* and */
  - May span multiple lines
    - Common bug is to forget to close a comment
    - Can not be nested
- White Space
  - Blank lines before a new function definition
  - Blank lines within a function to separate different steps
- Indentation
  - Generally corresponds to nesting level of program

Basic C Syntax

- Variable Declarations
  - Format:
    - type variableName1, variableName2;
  - Must be before any executable code
  - May be global (whole file) or specific to a function
- Looping
  - While loop
- Subroutine Calls
  - Name(param1, param2, …)
- Block of code
  - { }

Printf Function

- Stntax:
  - printf(formatString, variable1, variable2, variabe3, ..);
- Format String:
  - Defines constants to print, or format for variables:
    - %d Print an integer in decimal
    - %x Print an integer in hex
    - %f Print a floating point value
    - %c Print a character
    - %s Print a character string
    - \n Print a newline

Passing Parameters to Subroutines

- Comma separated list of parameters
- All parameters are passed by value
- May pass literal values
- Compiler will check parameter usage:
  - Match a given call to its prototype
  - Match a function definition to its prototype
  - Pitfall: Critical to have only one prototype for each function
    - Put in a common file (.h)
- Which of these is legal (given int a, b, c):
  - void Func1(int p1, int p2, intp3);
  - Func1(1, 2, 3);
  - Func2(a, b, c);
  - Func3(a + b, c, 3);
- Answer: All of Them!

Using Linux

- Logging in
  - Can login at console or remotely (ssh)
- Basic operations (command line)
  - ls — list directory
  - pwd — print current directory
  - mkdir <dir> — create a new file
  - cp <f1> <f2> — copy file <f1> into file <f2>
  - rm <f1> — delete file <f1>
  - logout — end session
- Section on Monday will cover using UNIX
- Demo of basic commands

Compiling & Running a Program

- Programs must be compiled to execute
- Compiler is invoked as:
  - gcc <options> <source files>
  - Common Options:
    - -g Enable Debugging
    - -c Only compile to object
    - -o <fileName> Where to put the compiled program
    - -Wall Warn about common errors
    - -fprofile-arcs Count how often statements run
    - -ftest-coverage Test program coverage
  - A simple command line:
    - gcc –g –Wall –o prog1 prog1.c tools.c
- Make
  - invokes gcc for you