CSE 351
Section 1: Intro to C

Nick Hunt
September 29, 2011
Misc. Tidbits

• Welcome to 351! It'll be fun...

• Sections once a week
  Alternatively led by Nick and Aryan

• Section should be interactive
  Please ask questions

• Other avenues for help
  Discussion boards, direct email, office hours
  With 4 TAs, shouldn't be difficult to find help
Announcements

- Subscribed to the mailing list?
  - Should've received a message from Luis yesterday

- Small change to HW0
  - Originally we said to increase array size from 2048x2048 to 8192x8192.
  - Instead, only go to 4096x4096
  - Attu doesn't have enough memory for Java versions
Who am I?

• 3rd year grad student working with Luis
• I did my undergrad here as well (7th year overall)
• Broadly interested in operating systems and computer architecture
• Research focused primarily on:
  Software reliability/debugging via determinism
  Power/energy efficiency
Who am I really?

- A climber!
- 7 years on technical rock, last two on glaciated terrain
- Major peaks this summer:
  - Rainier, Baker, Olympus, Daniel, Colchuck, Ingalls, Unicorn, SEWS and more!
- Always looking for new partners :)

Who are you?

- Ever used Linux?
- Ever programmed in C?
- Ever debugged with GDB?
- Ever written in ASM?
- Any interesting summer stories?
Today

- Pleasantries
- Overview of C
  - Mainly discuss a few differences from Java
  - Not a real tutorial; just not enough time
  - See the C book for a good introduction
- Overview of debugging C programs
- Introduction to pointers in C
- Touch on HW0?
Intro to C: Why C?

• It's awesome and ubiquitous
  2\textsuperscript{nd} most popular language today - TIOBE.com
• Modern languages are still implemented in C
  Java, Python, Perl, PHP, Ruby
• So are operating systems
• Affords great performance and more control
  “With great freedom comes great responsibility”
Intro to C: Hello World in Java

/* HelloWorld.java */

class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, " + "world!");
    }
}

Intro to C: Hello World

/* hello.c */
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Hello, world!\n");
    return 0;
}

/* hello.c */
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Hello, world!
    return 0;
}

Preamble of file includes headers, provides function declarations, useful comments, etc.

Common headers, see refs:
    stdio.h, stdlib.h,
    stdint.h, unistd.h,
    string.h
/* hello.c */
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Hello, world!\n");
    return 0;
}

main() is the program's entry point, just like Java, but is not contained in a class.
Intro to C: Compiling

• Previous program in hello.c

• To compile and run:

  $ gcc hello.c -o hello -Wall

  $ ./hello

  Hello, world!

• Options:

  –o – What to name the output file

  –Wall – Print all warnings
Intro to C: C and Java

• C is a weakly typed language
  • int, float, long int, double, etc.
• Syntax similar to Java
  • if/then/else, do/while, for, switch/case
• printf/scanf for console I/O
• open/read/write/close for file I/O
Intro to C: Differences from Java

- No classes! No objects!
  
  Class-like things though; check out structs
  
  Data only, no methods

- No garbage collection! Not managed!
  
  Must remember to allocate/deallocate on your own
  
  No built-in bounds checking

- No exceptions!
  
  Need to do own error checking / handling

- No virtual machine!
  
  Must recompile the code for different architectures
  
  Compiles to “real” op codes (as opposed to virtual)
Intro to C: References

- The C Programming Language
  Written by the authors of the language
  Concise and precise
  Excellent collection of practice problems

- Linux man pages
  Useful for looking up how to use a particular function, e.g.:
    $ man printf
Intro to C: Debugging

• You write a program, try to run it, and it crashes. What now?
Intro to C: Debugging

• One option: “printf debugging”
  • Add print statements to the code to see where/why it crashes

• Another idea: run it through a debugger
  • Monitor accesses to variables, see where the program crashes, verify loop invariants, etc.

• Depends on the situation; one may be easier than the other
Intro to C: `printf` Debugging

- `printf` allows you to print formatted strings
- Arguments include a *format string*, and data to display
- Format string is a literal string, containing special placeholders indicating how to display the data
- Ex:
  - `printf("Sum: %d + %d = %d\n", 1, 2, 1+2)`
  - `%d` displays an integer
  - Produces “Sum: 1 + 2 = 3”
- See “man printf” or the C book for more
/* Buggy program */
#include <stdio.h>

int main(int argc, char* argv[]) {
    int a = 5, *b = &a;
    printf("%d %d\n", a, *b);
    a ^= a; b = *b ^ a;
    printf("%d %d\n", a, *b);
    return 0;
}

Intro to C: Debugging with GDB

- Use `-ggdb` to compile w/ debugging symbols
  
  `gcc -o foo -Wall -ggdb foo.c`

- Invoke with `gdb`:
  
  `gdb ./foo`

- Important commands:
  
  - `run`
  - `break <line# / symbol>`
  - `step`
  - `continue`
  - `info <locals / frame / register>`
  - `print, x`
  - `backtrace`
  - `help`
/* Buggy program */
#include <stdio.h>

int main(int argc, char* argv[]) {
    int a = 5, *b = &a;
    printf("%d %d\n", a, *b);
    a ^= a; b = *b ^ a;
    printf("%d %d\n", a, *b);
    return 0;
}
Intro to C: Taste of Pointers

- Variables in C have types
  - `int`, `long`, `double`, `float`, `char`, etc.
- A *pointer* is just another type
  - Pointers store addresses of other variables
  - `int` is an integer, but `int*` is a pointer to an `int`
  - Same for `float` and `float*`, `char` and `char*`, etc.
- “NULL pointers” are pointers containing 0 (zero)
Intro to C: Taste of Pointers

• & is the address-of operator
  • Returns the address of a variable
• * is the value-of operator
  • Retrieves the value stored at the address in a pointer; “dereferencing”; NULL pointers cannot be dereferenced

Ex:

• int a = 5; int *ap;
• ap = &a; *ap = 10;
• printf("%d %d\n", a, *ap);
Intro to C: Debugging with GDB

/* Buggy program */
#include <stdio.h>

int main(int argc, char* argv[]) {
    int a = 5, *b = &a;
    printf("%d %d\n", a, *b);
    a ^= a; b = *b ^ a;
    printf("%d %d\n", a, *b);
    return 0;
}
/* Buggy program */
#include <stdio.h>

int main(int argc, char* argv[]) {
    int a = 5, *b = &a;
    printf("%d %d\n", a, *b);
    a ^= a; b = *b ^ a;
    printf("%d %d\n", a, *b);
    return 0;
} /* b becomes NULL, so dereferencing causes a crash */
Intro to C: Taste of Pointers

• Why are pointers useful?
Intro to C: Taste of Pointers

• Why are pointers useful?
• Some ideas:
  • Linked data structures
  • Passing by reference
  • Avoid copying large blocks of data
  • Any others?
• Don't need to know this stuff now; just wanted to whet your appetite!
HW0

• Has anyone started yet?
• Any questions?
Thanks!

Questions:

cse351-tas@cs.washington.edu