COS 217: Introduction to Programming Systems

Goals for Today’s Class

- Course overview
  - Introductions
  - Course goals
  - Resources
  - Grading
  - Policies

- Getting started with C
  - C programming language overview
Introductions

• Instructor-of-Record
  • Vivek Pai, Ph.D. (Professor)
    • vivek@cs.princeton.edu

• Preceptors (in alphabetical order)
  • Robert Dondero, Ph.D. (Lead Preceptor)
    • rdondero@cs.princeton.edu
  • Sunha Ahn
    • sahn@princeton.edu
  • Dushyant Arora
    • dushyant@cs.princeton.edu
  • Sasha Koruga
    • skoruga@cs.princeton.edu
  • Meng Zhang
    • mengz@princeton.edu

Course Goal 1: “Programming in the Large”

• Goal 1: “Programming in the large”
  • Help you learn how to write large computer programs

• Specifically, help you learn how to:
  • Write modular code
  • Hide information
  • Manage resources
  • Handle errors
  • Write portable code
  • Test and debug your code
  • Improve your code’s performance (and when to do so)
  • Use tools to support those activities
Course Goal 2: “Under the Hood”

- Goal 2: “Look under the hood”
  - Help you learn what happens “under the hood” of computer systems
- Specifically, two downward tours

```
C Language
   Assembly Language
   Machine Language

language levels tour
```

```
Application Program
   Operating System
   Hardware

service levels tour
```

- Goal 2 supports Goal 1
  - Reveals many examples of effective abstractions

Course Goals: Why C?

- Q: Why C instead of Java?

- A: C supports Goal 1 better
  - C is a lower-level language
    - C provides more opportunities to create abstractions
    - C has some flaws
      - C’s flaws motivate discussions of software engineering principles

- A: C supports Goal 2 better
  - C facilitates language levels tour
    - C is closely related to assembly language
    - C facilitates service levels tour
    - Linux is written in C
Course Goals: Why Linux?

• Q: Why Linux instead of Microsoft Windows?
  • A: Linux is good for education and research
    • Linux is open-source and well-specified

• A: Linux is good for programming
  • Linux is a variant of Unix
  • Unix has GNU, a rich open-source programming environment

Course Goals: Summary

• Help you to become a...

Power Programmer!!!
Resources: Lectures and Precepts

• Lectures
  • Describe concepts at a high level
  • Slides available online at course Web site
  • Stronger influence on **exams**

• Precepts
  • Support lectures by describing concepts at a lower level
  • Support your work on assignments
  • Builds practically on a **subset** of information

• Important: Precepts begin **TODAY**

Resources: Website and Listserv

• Website
  • Access from http://www.cs.princeton.edu
    • Academics → Course Schedule → COS 217

• Piazza
  • http://piazza.com/class#spring2012/cos217/
  • Instructions provided in first precept
Resources: Books

• Required book
    • Covers the C programming language and standard libraries

• Highly recommended books
  • *The Practice of Programming*, Kernighan and Pike, 1999.
    • Covers "programming in the large"
    • (Required for COS 333)
    • Covers "under the hood"
    • Some key sections are on electronic reserve
    • First edition is sufficient
    • Covers tools

• All books are on reserve in Engineering Library

Resources: Manuals

• Manuals (for reference only, available online)
  • *IA32 Intel Architecture Software Developer’s Manual, Volumes 1-3*
  • *Tool Interface Standard & Executable and Linking Format*
  • *Using as, the GNU Assembler*

• See also
  • Linux `man` command
    • `man` is short for “manual”
    • For more help, type `man man`
Resources: Programming Environment

**Option 1**
- hats.princeton.edu
- Friend Center 016 or 017 Computer
- SSH
- Lab TAs

**Option 2**
- hats.princeton.edu
- Your PC/Mac/Linux Computer
- SSH
Resources: Programming Environment

- Other options
  - Use your own PC/Mac/Linux computer; run GNU tools locally; run
    your programs locally
  - Use your own PC/Mac/Linux computer; run a non-GNU
development environment locally; run your programs locally
  - Etc.

- Notes
  - Other options cannot be used for some assignments (esp. timing
    studies)
  - Instructors cannot promise support of other options
  - Strong recommendation: Use Option 1 or 2 for all assignments
  - First precept provides setup instructions

Grading

- Seven programming assignments (50%)
  - Working code
  - Clean, readable, maintainable code
  - On time (penalties for late submission)
  - Final assignment counts double (12.5%)

- Exams (40%)
  - Midterm (15%)
  - Final (25%)

- Class participation (10%)
  - Lecture and precept attendance is mandatory
Programming Assignments

- Programming assignments
  1. A “de-comment” program
  2. A string module
  3. A symbol table module
  4. IA-32 assembly language programs
  5. A buffer overrun attack
  6. A heap manager module
  7. A Unix shell

- Key part of the course
- See course “Schedule” web page for due dates/times
- First assignment is available now
- Advice: Start early to allow time for debugging (especially in the background while you are doing other things)…

Why Debugging is Necessary…
Policies

Study the course “Policies” web page!

• Especially the assignment collaboration policies
  • Violation involves trial by Committee on Discipline
  • Typical penalty is suspension from University for 1 academic year

• Some highlights:
  • Don’t view anyone else’s work during, before, or after the assignment time period
  • Don’t allow anyone to view your work during, before, or after the assignment time period
  • In your assignment “readme” file, acknowledge all resources used

• Ask your preceptor for clarifications if necessary

Course Schedule

• Very generally…

<table>
<thead>
<tr>
<th>Weeks</th>
<th>Lectures</th>
<th>Precepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2</td>
<td>Intro to C (conceptual)</td>
<td>Intro to Linux/GNU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Intro to C (mechanical)</td>
</tr>
<tr>
<td>3-6</td>
<td>“Pgmming in the Large”</td>
<td>Advanced C</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Midterm Exam</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>Recess</td>
</tr>
<tr>
<td>8-13</td>
<td>“Under the Hood”</td>
<td>Assembly Language</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pgmming Assignments</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reading Period</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Final Exam</td>
</tr>
</tbody>
</table>

• See course “Schedule” web page for details
Any questions before we start?

C vs. Java: History

We will use

Not yet popular; our compiler supports only partially


BCPL B C K&R C ANSI C89 ISO C90 ISO/ANSI C99

LISP Smalltalk C++ Java
C vs. Java: Design Goals

• Java design goals
  • Support object-oriented programming
  • Allow same program to be executed on multiple operating systems
  • Support using computer networks
  • Execute code from remote sources securely
  • Adopt the good parts of other languages (esp. C and C++)

• Implications for Java
  • Good for application-level programming
  • High-level
    • Virtual machine insulates programmer from underlying assembly language, machine language, hardware
  • Portability over efficiency
  • Security over efficiency
  • Security over flexibility

C vs. Java: Design Goals

• C design goals
  • Support structured programming
  • Support development of the Unix OS and Unix tools
    • As Unix became popular, so did C

• Implications for C
  • Good for system-level programming
    • But often used for application-level programming – sometimes inappropriately
  • Low-level
    • Close to assembly language; close to machine language; close to hardware
  • Efficiency over portability
  • Efficiency over security
  • Flexibility over security
C vs. Java: Design Goals

- Differences in design goals explain many differences between the languages
- C’s design goal explains many of its eccentricities
  - We’ll see examples throughout the course

C vs. Java: Overview

- Dennis Ritchie on the nature of C:
  - “C has always been a language that never attempts to tie a programmer down.”
  - “C has always appealed to systems programmers who like the terse, concise manner in which powerful expressions can be coded.”
  - “C allowed programmers to (while sacrificing portability) have direct access to many machine-level features that would otherwise require the use of assembly language.”
  - “C is quirky, flawed, and an enormous success. While accidents of history surely helped, it evidently satisfied a need for a system implementation language efficient enough to displace assembly language, yet sufficiently abstract and fluent to describe algorithms and interactions in a wide variety of environments.”
**C vs. Java: Overview (cont.)**

- Bad things you **can** do in C that you **can’t** do in Java
  - Shoot yourself in the foot (safety)
  - Shoot others in the foot (security)
  - Ignore wounds (error handling)
- Dangerous things you **must** do in C that you **don’t** in Java
  - Explicitly manage memory via `malloc()` and `free()`
- Good things you **can** do in C, but (more or less) **must** do in Java
  - Program using the object-oriented style
- Good things you **can’t** do in C but **can** do in Java
  - Write completely portable code

**C vs. Java: Details**

- Remaining slides provide some details
  - Suggestion: Use for future reference
- Slides covered briefly now, as time allows…
### C vs. Java: Details (cont.)

<table>
<thead>
<tr>
<th>Overall Program Structure</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello.java:</td>
<td></td>
<td>hello.c:</td>
</tr>
<tr>
<td><code>public class Hello {</code></td>
<td></td>
<td><code>#include &lt;stdio.h&gt;</code></td>
</tr>
<tr>
<td><code>public static void</code></td>
<td></td>
<td><code>int main(void) {</code></td>
</tr>
<tr>
<td><code>main(String[] args) {</code></td>
<td></td>
<td><code>printf(&quot;Hello, world\n&quot;);</code></td>
</tr>
<tr>
<td><code>System.out.println(&quot;Hello, world&quot;);</code></td>
<td></td>
<td><code>return 0;</code></td>
</tr>
<tr>
<td><code>}</code></td>
<td></td>
<td><code>}</code></td>
</tr>
</tbody>
</table>

### Building

% javac Hello.java
% ls
Hello.class
Hello.java
%
% gcc217 hello.c
% ls
a.out
hello.c
%

### Running

% java Hello
Hello, world
%
% a.out
Hello, world
%

### C vs. Java: Details (cont.)

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character type</td>
<td>char // 16-bit unicode</td>
</tr>
<tr>
<td>Integral types</td>
<td>byte // 8 bits</td>
</tr>
<tr>
<td></td>
<td>short // 16 bits</td>
</tr>
<tr>
<td></td>
<td>int // 32 bits</td>
</tr>
<tr>
<td></td>
<td>long // 64 bits</td>
</tr>
<tr>
<td>Floating point types</td>
<td>float // 32 bits</td>
</tr>
<tr>
<td></td>
<td>double // 64 bits</td>
</tr>
<tr>
<td>Logical type</td>
<td>boolean</td>
</tr>
<tr>
<td>Generic pointer type</td>
<td>// no equivalent</td>
</tr>
<tr>
<td>Constants</td>
<td>final int MAX = 1000;</td>
</tr>
<tr>
<td></td>
<td>const int MAX = 1000;</td>
</tr>
<tr>
<td></td>
<td>enum (MAX - 1000);</td>
</tr>
</tbody>
</table>
### C vs. Java: Details (cont.)

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arrays</strong></td>
<td></td>
</tr>
<tr>
<td><code>int () a = new int [10];</code></td>
<td><code>int a[10];</code></td>
</tr>
<tr>
<td><code>float [][] b = new float [5][20];</code></td>
<td><code>float b[5][20];</code></td>
</tr>
<tr>
<td><strong>Array bound checking</strong></td>
<td></td>
</tr>
<tr>
<td>// run-time check</td>
<td>/* no run-time check */</td>
</tr>
<tr>
<td><strong>Pointer type</strong></td>
<td></td>
</tr>
<tr>
<td>// Object reference is an implicit pointer</td>
<td><code>int *p;</code></td>
</tr>
<tr>
<td><strong>Record type</strong></td>
<td></td>
</tr>
<tr>
<td><code>class Mine {</code></td>
<td><code>struct Mine {</code></td>
</tr>
<tr>
<td><code>    int x;</code></td>
<td><code>    int x;</code></td>
</tr>
<tr>
<td><code>    float y;</code></td>
<td><code>    float y;</code></td>
</tr>
<tr>
<td><code>}</code></td>
<td><code>}</code></td>
</tr>
</tbody>
</table>

#### Strings

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>String s1 = &quot;Hello&quot;;</code></td>
<td><code>char *s1 = &quot;Hello&quot;;</code></td>
</tr>
<tr>
<td><code>String s2 = new String(&quot;hello&quot;);</code></td>
<td><code>char s2[6];</code></td>
</tr>
<tr>
<td><code>s1 += s2</code></td>
<td><code>strcpy(s2, &quot;hello&quot;);</code></td>
</tr>
<tr>
<td><strong>String concatenation</strong></td>
<td><code>#include &lt;string.h&gt;</code></td>
</tr>
<tr>
<td><code>s1 += s2</code></td>
<td><code>strcat(s1, s2);</code></td>
</tr>
<tr>
<td><strong>Logical ops</strong></td>
<td></td>
</tr>
<tr>
<td>`&amp;&amp;,</td>
<td></td>
</tr>
<tr>
<td><strong>Relational ops</strong></td>
<td></td>
</tr>
<tr>
<td><code>==, !=, &gt;, &lt;, &gt;=, &lt;=</code></td>
<td><code>==, !=, &gt;, &lt;, &gt;=, &lt;=</code></td>
</tr>
<tr>
<td><strong>Arithmetic ops</strong></td>
<td></td>
</tr>
<tr>
<td><code>+, -, *, /, %, unary -</code></td>
<td><code>+, -, *, /, %, unary -</code></td>
</tr>
<tr>
<td><strong>Bitwise ops</strong></td>
<td></td>
</tr>
<tr>
<td>`&gt;&gt;, &lt;&lt;, &gt;&gt;&gt;, &amp;,</td>
<td>, ^`</td>
</tr>
<tr>
<td><strong>Assignment ops</strong></td>
<td></td>
</tr>
<tr>
<td>`=, *=, /=, +=, -=, &lt;&lt;=, &gt;&gt;=, &gt;&gt;&gt;=, ^=,</td>
<td>=, %=`</td>
</tr>
<tr>
<td></td>
<td>Java</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td><strong>if stmt</strong></td>
<td>if (i &lt; 0)</td>
</tr>
<tr>
<td></td>
<td>statement1;</td>
</tr>
<tr>
<td></td>
<td>else</td>
</tr>
<tr>
<td></td>
<td>statement2;</td>
</tr>
<tr>
<td><strong>switch stmt</strong></td>
<td>switch (i) {</td>
</tr>
<tr>
<td></td>
<td>case 1:</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>break;</td>
</tr>
<tr>
<td></td>
<td>case 2:</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>break;</td>
</tr>
<tr>
<td></td>
<td>default:</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td><strong>goto stmt</strong></td>
<td>// no equivalent</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>for stmt</strong></td>
<td>for (int i=0; i&lt;10; i++)</td>
<td>int i;</td>
</tr>
<tr>
<td></td>
<td>statement;</td>
<td>for (i=0; i&lt;10; i++)</td>
</tr>
<tr>
<td></td>
<td>statement;</td>
<td>statement;</td>
</tr>
<tr>
<td><strong>while stmt</strong></td>
<td>while (i &lt; 0)</td>
<td>while (i &lt; 0)</td>
</tr>
<tr>
<td></td>
<td>statement;</td>
<td>statement;</td>
</tr>
<tr>
<td><strong>do-while stmt</strong></td>
<td>do {</td>
<td>do {</td>
</tr>
<tr>
<td></td>
<td>statement;</td>
<td>statement;</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>} while (i &lt; 0)</td>
<td>} while (i &lt; 0);</td>
</tr>
<tr>
<td><strong>continue stmt</strong></td>
<td>continue;</td>
<td>continue;</td>
</tr>
<tr>
<td><strong>labeled continue stmt</strong></td>
<td>continue SomeLabel;</td>
<td>/* no equivalent */</td>
</tr>
<tr>
<td><strong>break stmt</strong></td>
<td>break;</td>
<td>break;</td>
</tr>
<tr>
<td><strong>labeled break stmt</strong></td>
<td>break SomeLabel;</td>
<td>/* no equivalent */</td>
</tr>
</tbody>
</table>
## C vs. Java: Details (cont.)

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>return stmt</strong></td>
<td><code>return 5;</code></td>
<td><code>return 5;</code></td>
</tr>
<tr>
<td></td>
<td><code>return;</code></td>
<td><code>return;</code></td>
</tr>
<tr>
<td><strong>Compound stmt (alias block)</strong></td>
<td><code>{ statement1; statement2; }</code></td>
<td><code>{ statement1; statement2; }</code></td>
</tr>
<tr>
<td><strong>Exceptions</strong></td>
<td><code>throw, try-catch-finally</code> /* no equivalent */</td>
<td>/* no equivalent */</td>
</tr>
<tr>
<td><strong>Comments</strong></td>
<td>/* comment */</td>
<td>/* comment */</td>
</tr>
<tr>
<td></td>
<td><code>// another kind</code></td>
<td></td>
</tr>
<tr>
<td><strong>Method / function call</strong></td>
<td><code>f(x, y, z);</code> someObject.f(x, y, z); SomeClass.f(x, y, z);`</td>
<td><code>f(x, y, z);</code></td>
</tr>
</tbody>
</table>

### Example C Program

```c
#include <stdio.h>
#include <stdlib.h>

const double KMETERS_PER_MILE = 1.609;

int main(void) {
    int miles;
    double kmeters;
    printf("miles: ");
    if (scanf("%d", &miles) != 1) {
        fprintf(stderr, "Error: Expect a number.\n");
        exit(EXIT_FAILURE);
    }
    kmeters = miles * KMETERS_PER_MILE;
    printf("%d miles is %f kilometers.\n", miles, kmeters);
    return 0;
}
```
Summary

• Course overview
  • Goals
    • Goal 1: Learn “programming in the large”
    • Goal 2: Look “under the hood”
    • Goal 2 supports Goal 1
    • Use of C and Linux supports both goals
  • Learning resources
    • Lectures, precepts, programming environment, course listserv, textbooks
    • Course Web site: access via http://www.cs.princeton.edu

Summary

• Getting started with C
  • C was designed for system programming
    • Differences in design goals of Java and C explain many differences between the languages
    • Knowing C design goals explains many of its eccentricities
    • Knowing Java gives you a head start at learning C
    • C is not object-oriented, but many aspects are similar
Getting Started

• Check out course Web site soon
  • Study “Policies” page
  • First assignment is available

• Establish a reasonable computing environment soon
  • Instructions given in first precept