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
  • Iasonas Petras, Ph.D. (Lead Preceptor)
    • ipetras@cs.princeton.edu
  • Margo Flynn
    • margof@princeton.edu
  • Akshay Mittal
    • akshay@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

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
Precept Etiquette

• Attend YOUR precept
  • 130 students in one precept is bad

• Want to CHANGE precepts?
  • Ask Collen Kenny-McGinley (2nd floor)
  • But she can’t move you into a FULL precept

Resources: Website and Listserv

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

• Piazza
  • http://piazza.com/class#spring2014/cos217/
  • Instructions provided in first precept
Resources: Books

- **Required book**
    - Covers the C programming language and standard libraries

- **Highly recommended books**
    - 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**

  - Nobel.princeton.edu
  - Friend Center 016 or 017 Computer
  - SSH
  - Lab TAs

- **Option 2**

  - Nobel.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 (30%)
  - Working code
  - Clean, readable, maintainable code
  - On time (penalties for late submission)
  - Final assignment counts double (7.5%)

- Exams (60%)
  - Midterm (30%)
  - Final (30%)

- Subjective (10%)
  - Lecture attendance is highly encouraged
  - Precept attendance is mandatory
  - Be considerate in your interactions
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, due Feb 16 @ 9:00pm
- Advice: Start early to allow time for debugging (especially in the background while you are doing other things!)

Policies

Study the course “Policies” web page!

- Especially the assignment collaboration policies
  - Violations often involve trial by Committee on Discipline
  - Typical penalty is suspension from University for 1 academic year
  - Default penalty for course policy violation is F

- Some highlights:
  - Don’t view anyone else’s work during, before, or after the assignment time period
  - In your assignment “readme” file, properly acknowledge all resources used
  - There are course policies and university policies, with different procedures and different penalties for violation

- Ask the professor for clarifications if necessary
  - Only the professor can waive any policies (and not verbally)
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 Intro to C (mechanical)</td>
</tr>
<tr>
<td>3-6</td>
<td>“Pgmmin 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 Pgmmin 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

Not yet popular; our compiler supports only partially

We will use

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

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><strong>Hello.java:</strong></td>
<td>public class Hello {</td>
<td>hello.c:</td>
</tr>
<tr>
<td></td>
<td>public static void</td>
<td>#include &lt;stdio.h&gt;</td>
</tr>
<tr>
<td></td>
<td>main(String[] args) {</td>
<td>int main(void) {</td>
</tr>
<tr>
<td></td>
<td>System.out.println(</td>
<td>printf(&quot;Hello, world\n&quot;);</td>
</tr>
<tr>
<td></td>
<td>&quot;Hello, world&quot;);</td>
<td>return 0;</td>
</tr>
<tr>
<td></td>
<td>}</td>
<td></td>
</tr>
<tr>
<td>% javac Hello.java</td>
<td>% gcc217 hello.c</td>
<td></td>
</tr>
<tr>
<td>% is Hello.class</td>
<td>% is a.out</td>
<td></td>
</tr>
<tr>
<td>% Hello.java</td>
<td>% hello.c</td>
<td></td>
</tr>
<tr>
<td>% Running % java Hello</td>
<td>% a.out</td>
<td></td>
</tr>
<tr>
<td>% Hello, world</td>
<td>% Hello, world</td>
<td></td>
</tr>
</tbody>
</table>
### C vs. Java: Details (cont.)

<table>
<thead>
<tr>
<th>Character type</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>char // 16-bit unicode</td>
<td>char /* 8 bits */</td>
</tr>
<tr>
<td>Integral types</td>
<td>Java</td>
<td>C</td>
</tr>
<tr>
<td>byte</td>
<td>// 8 bits</td>
<td>(unsigned) char</td>
</tr>
<tr>
<td>short</td>
<td>// 16 bits</td>
<td>(unsigned) short</td>
</tr>
<tr>
<td>int</td>
<td>// 32 bits</td>
<td>(unsigned) int</td>
</tr>
<tr>
<td>long</td>
<td>// 64 bits</td>
<td>(unsigned) long</td>
</tr>
<tr>
<td>Floating point types</td>
<td>Java</td>
<td>C</td>
</tr>
<tr>
<td>float</td>
<td>// 32 bits</td>
<td>float</td>
</tr>
<tr>
<td>double</td>
<td>// 64 bits</td>
<td>double</td>
</tr>
<tr>
<td>Logical type</td>
<td>Java</td>
<td>C</td>
</tr>
<tr>
<td>boolean</td>
<td></td>
<td>/* no equivalent */</td>
</tr>
<tr>
<td>Generic pointer type</td>
<td>Java</td>
<td>C</td>
</tr>
<tr>
<td>// no equivalent</td>
<td></td>
<td>void*</td>
</tr>
<tr>
<td>Constants</td>
<td>Java</td>
<td>C</td>
</tr>
<tr>
<td>final int MAX = 1000</td>
<td>#define MAX 1000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>const int MAX = 1000;</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>Arrays</td>
<td>Java</td>
</tr>
<tr>
<td>int [] a = new int [10];</td>
<td>int a[10];</td>
</tr>
<tr>
<td>float [][] b = new float [5][20];</td>
<td>float b[5][20];</td>
</tr>
<tr>
<td>Array bound checking</td>
<td>Java</td>
</tr>
<tr>
<td>// run-time check</td>
<td>/* no run-time check */</td>
</tr>
<tr>
<td>Pointer type</td>
<td>Java</td>
</tr>
<tr>
<td>// Object reference is an implicit pointer</td>
<td>int *p;</td>
</tr>
<tr>
<td>Record type</td>
<td>Java</td>
</tr>
<tr>
<td>class Mine {</td>
<td>struct Mine {</td>
</tr>
<tr>
<td>int x;</td>
<td>int x;</td>
</tr>
<tr>
<td>float y;</td>
<td>float y;</td>
</tr>
</tbody>
</table>
} | } |
### C vs. Java: Details (cont.)

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strings</strong></td>
<td></td>
</tr>
<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>strcpy(s2, &quot;hello&quot;);</code></td>
<td></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><code>=, *=, /=, %=, &lt;&lt;=, &gt;&gt;=, &gt;&gt;&gt;=</code></td>
<td><code>=, *=, /=, %=, &lt;&lt;=, &gt;&gt;=, &gt;&gt;&gt;=</code></td>
</tr>
</tbody>
</table>

#### Java

```java
String s1 = "Hello";
String s2 = new String("hello");
s1 += s2
```

#### C

```c
char *s1 = "Hello";
char s2[6];
strcpy(s2, "hello");
```

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

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>if stmt</strong></td>
<td></td>
</tr>
<tr>
<td><code>if (i &lt; 0) statement1;</code></td>
<td><code>if (i &lt; 0) statement1;</code></td>
</tr>
<tr>
<td><code>else statement2;</code></td>
<td><code>else statement2;</code></td>
</tr>
<tr>
<td><strong>switch stmt</strong></td>
<td></td>
</tr>
<tr>
<td><code>switch (i) {</code></td>
<td><code>switch (i) {</code></td>
</tr>
<tr>
<td><code>case 1: ... break;</code></td>
<td><code>case 1: ... break;</code></td>
</tr>
<tr>
<td><code>case 2: ... break;</code></td>
<td><code>case 2: ... break;</code></td>
</tr>
<tr>
<td><code>default: ...</code></td>
<td><code>default: ...</code></td>
</tr>
<tr>
<td><code>}</code></td>
<td><code>}</code></td>
</tr>
<tr>
<td><strong>goto stmt</strong></td>
<td></td>
</tr>
<tr>
<td><code>// no equivalent</code></td>
<td><code>goto SomeLabel;</code></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>for stmt</strong></td>
<td><code>for (int i=0; i&lt;10; i++) statement;</code></td>
<td><code>int i; for (i=0; i&lt;10; i++) statement;</code></td>
</tr>
<tr>
<td><strong>while stmt</strong></td>
<td><code>while (i &lt; 0) statement;</code></td>
<td><code>while (i &lt; 0) statement;</code></td>
</tr>
<tr>
<td><strong>do-while stmt</strong></td>
<td><code>do { statement; ... } while (i &lt; 0);</code></td>
<td><code>do { statement; ... } while (i &lt; 0);</code></td>
</tr>
<tr>
<td><strong>continue stmt</strong></td>
<td><code>continue;</code></td>
<td><code>continue;</code></td>
</tr>
<tr>
<td><strong>labeled continue stmt</strong></td>
<td><code>continue SomeLabel;</code></td>
<td>/* no equivalent */</td>
</tr>
<tr>
<td><strong>break stmt</strong></td>
<td><code>break;</code></td>
<td><code>break;</code></td>
</tr>
<tr>
<td><strong>labeled break stmt</strong></td>
<td><code>break SomeLabel;</code></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; return;</code></td>
<td><code>return 5; 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></td>
<td>/* no equivalent */</td>
</tr>
<tr>
<td><strong>Comments</strong></td>
<td>/* comment */ // another kind</td>
<td>/* comment */</td>
</tr>
<tr>
<td><strong>Method / function call</strong></td>
<td><code>f(x, y, z); someObject.f(x, y, z); Someclass.f(x, y, z);</code></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