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
  • Jaswinder Pal Singh
    • jps@cs.princeton.edu

• Preceptors (in alphabetical order)
  • Robert Dondero, Ph.D. (Lead Preceptor)
    • rdondero@cs.princeton.edu
  • Christopher Moretti, Ph.D.
    • cmoretti@cs.princeton.edu
  • Vivek Pai, Ph.D. (former Instructor-of-Record)
    • vivek@cs.princeton.edu
  • Cole Schlesinger
    • cschlesi@princeton.edu
  • Richard Wang
    • rwthree@princeton.edu

Course Goal 1: “Programming in the Large”

• Learn how to write large 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”

- Learn what happens “under the hood” of computer systems
- Specifically, two downward tours

<table>
<thead>
<tr>
<th>C Language</th>
<th>Assembly Language</th>
<th>Machine Language</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>language levels</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tour</td>
<td></td>
</tr>
<tr>
<td>Application Program</td>
<td>Operating System</td>
<td>Hardware</td>
</tr>
<tr>
<td>service levels tour</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 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
    - Provides more opportunities to create abstractions
  - C has some flaws
    - The flaws motivate discussions of software engineering principles
- A: C supports Goal 2 better
  - Facilitates language levels tour
    - C is closely related to assembly language
  - 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 has good open-source programming tools
  • 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
- Precepts
  - Support lectures by describing concepts at a lower level
  - Support your work on assignments
- Note: Precepts begin on Monday

Resources: Website and Listserv

- Website
    - Academics → Course Schedule → COS 217
- Listserv
  - [cos217@lists.cs.princeton.edu](mailto:cos217@lists.cs.princeton.edu)
  - Subscription is required
  - 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
  - Linux
  - GNU
  - Your Pgm
  - fez
  - fedora

- **Option 2**
  - hats.princeton.edu
  - Your PC/Mac/Linux Computer
  - Linux
  - GNU
  - Your Pgm
  - fez
  - fedora
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
  • Sign-up sheets will be used for both. Attendance is a key part of class participation.
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 ensure you understand the assignment fully and to allow time for debugging.

Why Debugging is Necessary...

Copyright 2003 Randy Glasbergen.  www.glasbergen.com
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

Study the course “Policies” web page

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>“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 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

- 1960: BCPL
- 1970: B
- 1972: C
- 1978: K&R C
- 1989: ANSI C89, ISO C90

- LISP
- Smalltalk
- C++
- Java

Not yet popular; our compiler supports only partially

We will use
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
  • 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.)

### Overall Program Structure

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

### Building

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>% javac Hello.java</td>
<td>% gcc217 hello.c</td>
</tr>
<tr>
<td>% ls Hello.class</td>
<td>% ls</td>
</tr>
<tr>
<td>Hello.java</td>
<td>a.out</td>
</tr>
<tr>
<td>%</td>
<td>hello.c</td>
</tr>
</tbody>
</table>

### Running

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>% java Hello HelloWorld</td>
<td>% a.out</td>
</tr>
<tr>
<td>%</td>
<td>Hello, world</td>
</tr>
</tbody>
</table>

### Character type

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>char /* 8 bits*/</td>
</tr>
</tbody>
</table>

### Integral types

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>(unsigned) char</td>
</tr>
<tr>
<td>short</td>
<td>(unsigned) short</td>
</tr>
<tr>
<td>int</td>
<td>(unsigned) int</td>
</tr>
<tr>
<td>long</td>
<td>(unsigned) long</td>
</tr>
</tbody>
</table>

### Floating point types

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>float</td>
</tr>
<tr>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>long double</td>
<td>long double</td>
</tr>
</tbody>
</table>

### Logical type

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>/* no equivalent */</td>
</tr>
<tr>
<td></td>
<td>/* use integral type */</td>
</tr>
</tbody>
</table>

### Generic pointer type

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>// no equivalent</td>
<td>void*</td>
</tr>
</tbody>
</table>

### Constants

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>final int MAX = 1000;</td>
<td>#define MAX 1000</td>
</tr>
<tr>
<td>const int MAX - 1000;</td>
<td>const MAX - 1000;</td>
</tr>
<tr>
<td>enum (MAX - 1000);</td>
<td></td>
</tr>
</tbody>
</table>
### C vs. Java: Details (cont.)

<table>
<thead>
<tr>
<th>Array Bound Checking</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>// run-time check</td>
<td>int a[10];</td>
<td>int a[10];</td>
</tr>
<tr>
<td>/* no run-time check */</td>
<td>float b[5][20];</td>
<td>float b[5][20];</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pointer Type</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>// Object reference is an implicit pointer</td>
<td>int *p;</td>
<td>int *p;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record Type</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>class Mine { int x; float y; }</td>
<td>struct Mine { int x; float y; }</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Strings</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>String s1 = &quot;Hello&quot;; String s2 = new String(&quot;hello&quot;);</td>
<td>char *s1 = &quot;Hello&quot;; char s2[6]; strcpy(s2, &quot;hello&quot;);</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String Concatenation</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1 + s2</td>
<td>#include &lt;string.h&gt; strcat(s1, s2);</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Logical Ops</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;,</td>
<td></td>
<td>, !</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relational Ops</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>=, !=, &gt;, &lt;, &gt;=, &lt;=</td>
<td>=, !=, &gt;, &lt;, &gt;=, &lt;=</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arithmetic Ops</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>+, -, *, /, %, unary -</td>
<td>+, -, *, /, %, unary -</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bitwise Ops</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt;, &lt;&lt;, &gt;&gt;&gt;, &amp;</td>
<td>&gt;&gt;, &lt;&lt;, &amp;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assignment Ops</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>=, *=, /=, +=, -=, &lt;&lt;=, &gt;&gt;=, &gt;&gt;&gt;=, &amp;=, ^=,</td>
<td>=, %=</td>
<td>=, *=, /=, +=, -=, &lt;&lt;=, &gt;&gt;=, &gt;&gt;&gt;=, &amp;=, ^=,</td>
</tr>
</tbody>
</table>
### C vs. Java: Details (cont.)

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
</table>
| **if stmt** | if (i < 0)  
statement1;  
else  
statement2; | if (i < 0)  
statement1;  
else  
statement2; |
| **switch stmt** | switch (i) {  
case 1:  
...  
break;  
case 2:  
...  
break;  
default:  
...  
} | switch (i) {  
case 1:  
...  
break;  
case 2:  
...  
break;  
default:  
...  
} |
| **goto stmt** | // no equivalent | goto SomeLabel; |

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

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

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