COS 217: Introduction to Programming Systems

Professor Jennifer Rexford
http://www.cs.princeton.edu/~jrex

Goals for Today’s Class

• Course overview
  • Introductions
  • Course goals
  • Resources
  • Grading
  • Policies

• Getting started with C
  • C programming language overview
Introductions

• Jennifer Rexford (professor)
  • Room 306 in Computer Science Building
  • jrex@cs.princeton.edu

• Robert Dondero (lead preceptor)
  • Room 206 in Computer Science Building
  • rdondero@cs.princeton.edu

• Jailu Huang (preceptor)
  • Room 213 in Computer Science Building
  • jailuh@cs.princeton.edu

• Donna O’Leary (administrator)
  • Room 210 in Computer Science Building
  • doleary@cs.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**
  - Language levels tour
    - High-level language (C) → assembly language (AT&T) → machine language (IA-32)
  - Service levels tour
    - High-level language (C) → standard libraries → operating system (Linux)

• **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
  - C facilitates language levels tour

• **A: C supports Goal 2 better**
  - 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
- Precepts
  - Support lectures by describing concepts at a lower level
  - Support your work on assignments

Resources: Website and Listserv

- Website
    - Course Information → COS 217
- Listserv
  - 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
    • First edition is not quite as good, but is sufficient

• 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 will be on electronic reserve
    • 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 – short for manual
    • For more help, type “man man”
Resources: Programming Environment

• Option 1

hats.princeton.edu

Friend Center 016 Mac
Friend Center 017 PC
Lab TAs

• Option 2

Your Own
PC/Mac/Linux Computer

hats.princeton.edu
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 detailed 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 “decomment” program
  2. A string module
  3. A symbol table module
  4. A heap manager module
  5. IA-32 assembly language programs
  6. A buffer overrun attack
  7. A UNIX shell

• Key aspect of the course
• Due (typically) Sundays at 9:00PM
• 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...

Copyright 2003 Randy Glasbergen. www.glasbergen.com
Policies

• From course Policies web page:
  “Programming in an individual creative process much like composition. You must reach your own understanding of the problem and discover a path to its solution. During this time, discussions with friends are encouraged. However, when the time comes to write code that solves the problem, such discussions are no longer appropriate - the program must be your own work. If you have a question about how to use some feature of C, UNIX, etc., you can certainly ask your friends or the teaching assistants, but do not, under any circumstances, copy another person's program. Letting someone copy your program or using someone else's code in any form is a violation of academic regulations. "Using someone else's code" includes using solutions or partial solutions to assignments provided by commercial web sites, instructors, preceptors, teaching assistants, friends, or students from any previous offering of this course or any other course.”

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</td>
<td>Intro to Linux/GNU</td>
</tr>
<tr>
<td>3-6</td>
<td>&quot;Pgmming in the large&quot;</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>&quot;Under the hood&quot;</td>
<td>Assembly language</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
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 objected-oriented style

• Good things that 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>public static void</td>
<td></td>
<td>#include &lt;stdio.h&gt;</td>
</tr>
<tr>
<td>main(String[] args) {</td>
<td>int main(void) {</td>
<td></td>
</tr>
<tr>
<td>System.out.println(</td>
<td>printf(&quot;Hello, world\n&quot;);</td>
<td></td>
</tr>
<tr>
<td>&quot;Hello, world&quot;); }</td>
<td>return 0; }</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Running</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>% java Hello</td>
<td>% a.out</td>
<td></td>
</tr>
<tr>
<td>Hello, world</td>
<td>Hello, world</td>
<td></td>
</tr>
<tr>
<td>%</td>
<td>%</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>char</td>
<td>char /* 8 bits */</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Integral types</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte // 8 bits</td>
<td>(unsigned) char</td>
<td></td>
</tr>
<tr>
<td>short // 16 bits</td>
<td>(unsigned) short</td>
<td></td>
</tr>
<tr>
<td>int // 32 bits</td>
<td>(unsigned) int</td>
<td></td>
</tr>
<tr>
<td>long // 64 bits</td>
<td>(unsigned) long</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Floating point types</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>float // 32 bits</td>
<td>float double</td>
<td></td>
</tr>
<tr>
<td>double // 64 bits</td>
<td>long double</td>
<td></td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Constants</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>final int MAX = 1000;</td>
<td>#define MAX 1000</td>
<td></td>
</tr>
<tr>
<td>const int MAX = 1000;</td>
<td>enum (MAX = 1000);</td>
<td></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>Arrays</strong></td>
<td><code>int [] a = new int [10];</code></td>
<td><code>int a[10];</code></td>
</tr>
<tr>
<td></td>
<td><code>float [][] b =</code></td>
<td><code>float b[5][20];</code></td>
</tr>
<tr>
<td>**Array bound</td>
<td><code>// run-time check</code></td>
<td><code>/* no run-time check */</code></td>
</tr>
<tr>
<td><strong>checking</strong></td>
<td><code>// Object reference is an</code></td>
<td><code>int *p;</code></td>
</tr>
<tr>
<td></td>
<td><code>// implicit pointer</code></td>
<td></td>
</tr>
<tr>
<td>**Pointer type</td>
<td><code>class Mine {</code></td>
<td><code>struct Mine {</code></td>
</tr>
<tr>
<td></td>
<td><code>  int x;</code></td>
<td><code>  int x;</code></td>
</tr>
<tr>
<td></td>
<td><code>  float y;</code></td>
<td><code>  float y;</code></td>
</tr>
<tr>
<td>**Record type</td>
<td><code>}</code></td>
<td><code>}</code></td>
</tr>
<tr>
<td><strong>Assignment ops</strong></td>
<td>`+=, *=, /=, %=, +=, -=, &lt;&lt;=, &gt;&gt;=, &gt;&gt;&gt;=, &amp;=, ^=,</td>
<td>=`</td>
</tr>
</tbody>
</table>

**Strings**

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>String s1 = &quot;Hello&quot;;</code></td>
<td><code>char *s1 = &quot;Hello&quot;;</code></td>
</tr>
<tr>
<td></td>
<td><code>String s2 = new String(&quot;hello&quot;);</code></td>
<td><code>char s2[6];</code></td>
</tr>
<tr>
<td></td>
<td><code>s1 += s2</code></td>
<td><code>strcpy(s2, &quot;hello&quot;);</code></td>
</tr>
<tr>
<td><strong>Logical ops</strong></td>
<td>`&amp;&amp;,</td>
<td></td>
</tr>
<tr>
<td><strong>Relational ops</strong></td>
<td><code>==, !=, &gt;, &gt;=, &lt;, &lt;, &lt;=</code></td>
<td><code>==, !=, &gt;, &lt;, &gt;, &lt;=</code></td>
</tr>
<tr>
<td><strong>Arithmetic ops</strong></td>
<td><code>+, -, *, /, %, unary -</code></td>
<td><code>+, -, *, /, %, unary -</code></td>
</tr>
<tr>
<td><strong>Bitwise ops</strong></td>
<td>`&gt;&gt;, &lt;&lt;, &gt;&gt;&gt;, &amp;,</td>
<td>, ^`</td>
</tr>
<tr>
<td><strong>Assignment ops</strong></td>
<td>`=, *=, /=, %=, -=, &lt;&lt;=, &gt;&gt;=, &gt;&gt;&gt;=, ^=,</td>
<td>=, &amp;=, ^=,</td>
</tr>
</tbody>
</table>
## C vs. Java: Details (cont.)

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

---

<table>
<thead>
<tr>
<th>C vs. Java: Details (cont.)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Java</strong></td>
</tr>
</tbody>
</table>
| **for stmt** | **for** (int i=0; i<10; i++)  
statement; |
| **while stmt** | **while** (i < 0)  
statement; |
| **do-while stmt** | **do** {  
statement;  
...  
} while (i < 0) |
| **continue stmt** | continue; |
| **labeled continue stmt** | continue SomeLabel;  
/* no equivalent */ |
| **break stmt** | break; |
| **labeled break stmt** | break SomeLabel;  
/* no equivalent */ |
## 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</strong> (alias block)</td>
<td><code>{ statement1; statement2; }</code></td>
<td><code>{ statement1; statement2; }</code></td>
</tr>
<tr>
<td><strong>Exceptions</strong></td>
<td>throw, try-catch-finally</td>
<td>/* no equivalent */</td>
</tr>
<tr>
<td><strong>Comments</strong></td>
<td>/* comment */</td>
<td>/* comment */</td>
</tr>
<tr>
<td></td>
<td>// another kind</td>
<td></td>
</tr>
<tr>
<td><strong>Method / function call</strong></td>
<td><code>f(x, y, z);</code></td>
<td><code>f(x, y, z);</code></td>
</tr>
<tr>
<td></td>
<td><code>someObject.f(x, y, z);</code></td>
<td><code>SomeClass.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
  • First assignment is available

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

• Reading
  • Required: C Programming (King) 1, 2, 3
  • Required: Computer Systems (Bryant & O’Hallaron) 1
  • Recommended: Programming with GNU Software (Loukides & Oram) 1, 2, 4