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
• Vivek Pai, Ph.D. (Professor)
  • vivek@cs.princeton.edu

• Robert Dondero, Ph.D. (Lead Preceptor)
  • rdondero@cs.princeton.edu

• Jack Tzu-Han Hung (Preceptor)
  • thhung@princeton.edu

• Richard Wang (Preceptor)
  • rwthree@princeton.edu
Course Goal 1: “Programming in the Large”

• Goal 1: “Programming in the large”
  • Help you learn how to write large computer programs
  • Abstraction; Interfaces and implementations
• 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
  • 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

• Start you on the path to understanding systems

Course Goals: Summary

• Start you on the path to understanding systems
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
  - Academics → Course Schedule → 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 so good, but is sufficient
- Highly recommended books
    - Covers "programming in the large"
    - (Required for COS 333)
    - Covers "under the hood"
    - Some key sections are on electronic reserve
    - Covers tools
- All books are on reserve in Engineering Library
Resources: Manuals

- Manuals (for reference only, available online)
  - 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
  - RTFM ➔ read the fine manual

Resources: Programming Environment

**Option 1**

hats.princeton.edu

Friend Center 016 or 017 Computer

Lab TAs

**Option 2**

hats.princeton.edu

Your PC/Mac/Linux Computer
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**
- **Due (typically) Sundays at 9:00PM**
- **First assignment is available now**
- **Advice: Start early to allow time for debugging …**
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 Intro to C (mechanical)</td>
</tr>
<tr>
<td>3-6</td>
<td>“Programming in the Large”</td>
<td>Advanced C</td>
</tr>
<tr>
<td>6</td>
<td>Midterm Exam</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Recess</td>
<td></td>
</tr>
<tr>
<td>8-13</td>
<td>“Under the Hood”</td>
<td>Assembly Language Programming Assignments</td>
</tr>
<tr>
<td></td>
<td>Reading Period</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Final Exam</td>
<td></td>
</tr>
</tbody>
</table>

- See course “Schedule” web page for details

Any questions before we start?
Putting C vs. Java In Context

- **C** Designed 1978 by AT&T(!)
  - Your parents were groovy teenagers
  - You did not exist
  - Computers were shared by dozens of people
  - Because they were incredibly expensive

- **Java** Designed in 1995 by Sun(!)
  - Most computers were cheap (except those made by Sun)
  - HTML "programmers" were making $150K/year
  - Greedy college students were entering CS instead of medicine
  - And most couldn't program if their lives depended on it

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

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;
}
```

About This Course

- Involves a lot of programming
  - You should already know 126-level material
  - Goal of the assignments: reinforce material, gain proficiency
  - But the assignments are not the entire course (only 50%)
  - Some time-flexibility: dropping final portions designed into system
- Testing
  - Two timed exams – tests proficiency
  - Combined, count almost as much (40%) as all assignments
  - Exams are not re-hashes of assignments
  - Open book and notes, but not as a first resort
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
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></th>
<th><strong>Java</strong></th>
<th><strong>C</strong></th>
</tr>
</thead>
</table>
| Overall Program Structure | ```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world");
    }
}
``` | ```
#include <stdio.h>

int main(void) {
    printf("Hello, world\n");
    return 0;
}
``` |
| Building | ```
$ javac Hello.java
$ ls
Hello.class
Hello.java
``` | ```
$ gcc 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>Character type</th>
<th><strong>Java</strong></th>
<th><strong>C</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Character type</td>
<td>char // 16-bit unicode</td>
<td>char /* 8 bits */</td>
</tr>
<tr>
<td>Integral types</td>
<td>byte // 8 bits, short // 16 bits, int // 32 bits, long // 64 bits</td>
<td>(unsigned) char, (unsigned) short, (unsigned) int, (unsigned) long</td>
</tr>
<tr>
<td>Floating point types</td>
<td>float // 32 bits, double // 64 bits</td>
<td>float, double</td>
</tr>
<tr>
<td>Logical type</td>
<td>boolean</td>
<td>// no equivalent &quot;true&quot;</td>
</tr>
<tr>
<td>General pointer type</td>
<td>// no equivalent &quot;void&quot;</td>
<td></td>
</tr>
<tr>
<td>Constants</td>
<td>final int MAX = 1000;</td>
<td>#define MAX 1000</td>
</tr>
</tbody>
</table>

C vs. Java: Details (cont.)

<table>
<thead>
<tr>
<th>Character type</th>
<th><strong>Java</strong></th>
<th><strong>C</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Character type</td>
<td>char // 16-bit unicode</td>
<td>char /* 8 bits */</td>
</tr>
<tr>
<td>Integral types</td>
<td>byte // 8 bits, short // 16 bits, int // 32 bits, long // 64 bits</td>
<td>(unsigned) char, (unsigned) short, (unsigned) int, (unsigned) long</td>
</tr>
<tr>
<td>Floating point types</td>
<td>float // 32 bits, double // 64 bits</td>
<td>float, double</td>
</tr>
<tr>
<td>Logical type</td>
<td>boolean</td>
<td>// no equivalent &quot;true&quot;</td>
</tr>
<tr>
<td>General pointer type</td>
<td>// no equivalent &quot;void&quot;</td>
<td></td>
</tr>
<tr>
<td>Constants</td>
<td>final int MAX = 1000;</td>
<td>#define 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><code>// run-time check</code></td>
<td><code>/* no run-time check */</code></td>
</tr>
<tr>
<td><strong>Pointer 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>
</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 sa = &quot;Hello&quot;;</code></td>
</tr>
<tr>
<td><code>String s2 = new String(&quot;hello&quot;);</code></td>
<td><code>String s2=</code></td>
</tr>
<tr>
<td><code>char *s1 = &quot;Hello&quot;;</code></td>
<td><code>strcpy(s2, &quot;hello&quot;);</code></td>
</tr>
<tr>
<td><code>char s2[6];</code></td>
<td><code>#include &lt;string.h&gt;</code></td>
</tr>
<tr>
<td><code>#include &lt;string.h&gt;</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><code>&gt;&gt;, &lt;&lt;, &gt;&gt;, &amp;</code>, `</td>
<td><code>, </code>^`</td>
</tr>
<tr>
<td><strong>Assignment ops</strong></td>
<td></td>
</tr>
<tr>
<td><code>x, te, /=, -=, *=, %=, /=</code></td>
<td><code>x, te, /=, -=, *=, %=, /=</code></td>
</tr>
</tbody>
</table>

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) {</code></td>
<td><code>if (i &lt; 0) {</code></td>
</tr>
<tr>
<td><code>  statement1;</code></td>
<td><code>  statement1;</code></td>
</tr>
<tr>
<td><code>  else</code></td>
<td><code>  else</code></td>
</tr>
<tr>
<td><code>  statement2;</code></td>
<td><code>  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:</code></td>
<td><code>  case 1:</code></td>
</tr>
<tr>
<td><code>    break;</code></td>
<td><code>    break;</code></td>
</tr>
<tr>
<td><code>  case 2:</code></td>
<td><code>  case 2:</code></td>
</tr>
<tr>
<td><code>    break;</code></td>
<td><code>    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><code>  ...</code></td>
<td><code>  ...</code></td>
</tr>
</tbody>
</table>

C vs. Java: Details (cont.)

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<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>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>for</code> stmt</td>
<td>for (int i=0; i&lt;10; i++) statement;</td>
</tr>
<tr>
<td><code>while</code> stmt</td>
<td>while (i &lt; 0) statement;</td>
</tr>
<tr>
<td><code>do-while</code> stmt</td>
<td>do statement; while (i &lt; 0);</td>
</tr>
<tr>
<td><code>continue</code> stmt</td>
<td>continue;</td>
</tr>
<tr>
<td><code>break</code> stmt</td>
<td>break;</td>
</tr>
<tr>
<td><code>labeled break</code> stmt</td>
<td>break SomeLabel;</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>return</code> stmt</td>
<td>return 5;</td>
</tr>
<tr>
<td>Compound stmt (alias block)</td>
<td>statement1; statement2;</td>
</tr>
<tr>
<td>Exceptions</td>
<td>throw, try-catch-finally</td>
</tr>
<tr>
<td>Comments</td>
<td>// comment */ // another kind</td>
</tr>
<tr>
<td>Method / function call</td>
<td>f(x, y, z); someClass.f(x, y, z); someFunction.f(x, y, z);</td>
</tr>
</tbody>
</table>