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

• Jaswinder Pal Singh, Ph.D. (Professor)
  • jps@cs.princeton.edu

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

Introductions (cont.)

• Muneeb Ali (Preceptor)
  • muneeb@princeton.edu

• Matvey Arye (Preceptor)
  • arye@princeton.edu

• Jialu Huang (Preceptor)
  • jialuh@princeton.edu

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

• Hanjun Kim (Preceptor)
  • hanjunk@princeton.edu

• Indraneel Mukherjee (Preceptor)
  • imukherj@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
  - C Language
  - Assembly Language
  - Machine Language
  - Application Program
  - Operating System
  - Hardware

- 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

- 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`

Resources: Programming Environment

- Option 1

  ![Diagram](image.png)

  - hats.princeton.edu
  - Linux
  - GNU
  - Your Pgm
  - fez
  - fedora

  Friend Center 016 or 017 Computer

  - SSH
  - Lab TAs
Resources: Programming Environment

• Option 2

hats.princeton.edu

Your PC/Mac/Linux Computer

- SSH

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 (45%)
  - Midterm (15%)
  - Final (30%)

- Class participation (5%)

- 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 …
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

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>
</table>
| **Hello.java:**           | public class Hello { public static void main(String[] args) { System.out.println("Hello, world"); } } | **hello.c:**
|                           | #include <stdio.h> int main(void) { printf("Hello, world\n"); return 0; } |   |

<table>
<thead>
<tr>
<th>Building</th>
<th>% javac Hello.java</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>% ls Hello.class</td>
</tr>
<tr>
<td></td>
<td>Hello.java</td>
</tr>
<tr>
<td></td>
<td>%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Running</th>
<th>% java Hello</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hello, world</td>
</tr>
<tr>
<td></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>Character type</strong></td>
<td>char // 16-bit unicode</td>
<td>char /* 8 bits */</td>
</tr>
<tr>
<td><strong>Integral types</strong></td>
<td>byte // 8 bits</td>
<td>(unsigned) char</td>
</tr>
<tr>
<td></td>
<td>short // 16 bits</td>
<td>(unsigned) short</td>
</tr>
<tr>
<td></td>
<td>int // 32 bits</td>
<td>(unsigned) int</td>
</tr>
<tr>
<td></td>
<td>long // 64 bits</td>
<td>(unsigned) long</td>
</tr>
<tr>
<td><strong>Floating point types</strong></td>
<td>float // 32 bits</td>
<td>float</td>
</tr>
<tr>
<td></td>
<td>double // 64 bits</td>
<td>double</td>
</tr>
<tr>
<td><strong>Logical type</strong></td>
<td>boolean</td>
<td>/* no equivalent */</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/* use integral type */</td>
</tr>
<tr>
<td><strong>Generic pointer type</strong></td>
<td>// no equivalent</td>
<td>void*</td>
</tr>
<tr>
<td><strong>Constants</strong></td>
<td>final int MAX = 1000;</td>
<td>#define MAX 1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>const int MAX = 1000;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enum {MAX = 1000};</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>int [] a = new int [10];</td>
<td>int a[10];</td>
</tr>
<tr>
<td></td>
<td>float [][] b = new float [5][20];</td>
<td>float b[5][20];</td>
</tr>
<tr>
<td><strong>Array bound checking</strong></td>
<td>// run-time check</td>
<td>/* no run-time check */</td>
</tr>
<tr>
<td><strong>Pointer type</strong></td>
<td>// Object reference is an implicit pointer</td>
<td>int *p;</td>
</tr>
<tr>
<td><strong>Record type</strong></td>
<td>class Mine {</td>
<td>struct Mine {</td>
</tr>
<tr>
<td></td>
<td>int x;</td>
<td>int x;</td>
</tr>
<tr>
<td></td>
<td>float y;</td>
<td>float y;</td>
</tr>
</tbody>
</table>
|                        | }                                        | }
<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strings</strong></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>String</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>concatenation</strong></td>
<td><code>s1 + s2</code></td>
<td><code>#include &lt;string.h&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>s1 += s2</code></td>
<td><code>strcat(s1, s2);</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;, &lt;, &gt;=, &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><code>&gt;&gt;, &lt;&lt;, &gt;&gt;&gt;, &amp;</code>, `</td>
<td><code>, </code>^`</td>
</tr>
<tr>
<td><strong>Assignment ops</strong></td>
<td><code>=, *=, /=, %=, unary -</code></td>
<td><code>=, *=, /=, %=, unary -</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>if stmt</strong></td>
<td><code>if (i &lt; 0)</code></td>
<td><code>if (i &lt; 0)</code></td>
</tr>
<tr>
<td></td>
<td><code>statement1;</code></td>
<td><code>statement1;</code></td>
</tr>
<tr>
<td></td>
<td><code>else</code></td>
<td><code>else</code></td>
</tr>
<tr>
<td></td>
<td><code>statement2;</code></td>
<td><code>statement2;</code></td>
</tr>
<tr>
<td><strong>switch stmt</strong></td>
<td><code>switch (i) {</code></td>
<td><code>switch (i) {</code></td>
</tr>
<tr>
<td></td>
<td><code>case 1:</code></td>
<td><code>case 1:</code></td>
</tr>
<tr>
<td></td>
<td><code>...</code></td>
<td><code>...</code></td>
</tr>
<tr>
<td></td>
<td><code>break;</code></td>
<td><code>break;</code></td>
</tr>
<tr>
<td></td>
<td><code>case 2:</code></td>
<td><code>case 2:</code></td>
</tr>
<tr>
<td></td>
<td><code>...</code></td>
<td><code>...</code></td>
</tr>
<tr>
<td></td>
<td><code>break;</code></td>
<td><code>break;</code></td>
</tr>
<tr>
<td></td>
<td><code>default: </code></td>
<td><code>default: </code></td>
</tr>
<tr>
<td></td>
<td><code>...</code></td>
<td><code>...</code></td>
</tr>
<tr>
<td></td>
<td><code>}</code></td>
<td><code>}</code></td>
</tr>
<tr>
<td><strong>goto stmt</strong></td>
<td><code>// no equivalent</code></td>
<td><code>goto SomeLabel;</code></td>
</tr>
</tbody>
</table>
### C vs. Java: Details (cont.)

<table>
<thead>
<tr>
<th>Statement Type</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++)</code> statement;</td>
<td><code>for (i=0; i&lt;10; i++)</code> statement;</td>
</tr>
<tr>
<td><strong>while stmt</strong></td>
<td><code>while (i &lt; 0)</code> statement;</td>
<td><code>while (i &lt; 0)</code> statement;</td>
</tr>
<tr>
<td><strong>do-while stmt</strong></td>
<td><code>do {</code> statement; <code>} while (i &lt; 0)</code></td>
<td><code>do {</code> statement; <code>} 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>
<tr>
<td><strong>return stmt</strong></td>
<td><code>return 5;</code> <code>return;</code></td>
<td><code>return 5;</code> <code>return;</code></td>
</tr>
<tr>
<td><strong>Compound stmt (alias block)</strong></td>
<td><code>{</code> <code>statement1;</code> <code>statement2;</code> <code>}</code></td>
<td><code>{</code> <code>statement1;</code> <code>statement2;</code> <code>}</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 */ // another kind</td>
<td>/* comment */</td>
</tr>
<tr>
<td><strong>Method / function call</strong></td>
<td><code>f(x, y, z);</code> <code>someObject.f(x, y, z);</code> <code>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