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

- Lecturer
  - Prof. Jaswinder Pal (J.P.) Singh

- Preceptors (in alphabetical order)
  - Dr. Robert Dondero (Lead Preceptor)
  - Mojgan Ghasemi
  - Madhuvanthi Jayakumar
  - Yi-Hsien (Stephen) Lin
  - Dr. Iasonas Petras
  - Raghav Sethi
  - Logan Stafman
  - Yannan Wang
  - KatieAnna Wolf

Course Goal 1: “Programming in the Large”

- How to write large programs

- Specifically, how to:
  - Break things down into modules
  - Use abstraction
  - Write modular code
  - Separate interface from implementation
  - Write code as part of a large team
  - Write portable code
  - Test and debug your code
  - Improve your code’s performance
  - Use tools to support these activities
**Course Goal 2: “Under the Hood”**

- What happens inside in computer systems?
- Specifically, two downward tours
  - We will cover some key aspects of both

![Diagram](image)

- Goal 2 supports Goal 1
  - Reveals many examples of effective abstractions

---

**Course Goals: Why C, not Java?**

- The course is not about a language. The language is merely a vehicle to convey the key concepts.
- C happens to better support the goals of the course.
- C supports Goal 1 better
  - C is a lower-level language
    - Forces you to create your own abstractions
  - C has some useful flaws
    - Motivates discussion of software engineering principles
- C supports Goal 2 better
  - C facilitates language levels tour
    - C is closely related to assembly language
  - C facilitates service levels tour
    - Linux operating system is written in C
Course Goals: Why Linux?

- Q: Why Linux?
- A: Good for education and research
  - Linux is open-source and well-specified
- A: Has good support 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...
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 Piazza

- **Website**
    - Academics → Course Schedule → COS 217

- **Piazza**
  - [https://piazza.com/login?#cos217](https://piazza.com/login?#cos217)
  - Subscription is required
  - 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

- 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**

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

**Option 2**

- `nobil.princeton.edu`
- `Your Windows/Mac/Linux computer`
- `SSH`
- `Lab TAs`
Resources: Programming Environment

- Other options
  - Use your own Windows/Mac/Linux computer; run GNU tools locally; run your programs locally
  - Use your own Windows/Mac/Linux computer; run a non-GNU development environment locally; run your programs locally
  - Build your own hardware, port Windows/Mac/Linux to it, …
  - Develop a new material, build hardware using it, port a new OS to it, …

- 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

- Exams (40%)
  - Midterm (15%)
  - Final (25%)

- Class participation (10%)
  - Lecture and precept attendance is mandatory
Programming Assignments

• Programming assignments
  1. A “de-comment” program (individual)
  2. A string module (individual)
  3. A symbol table module (individual)
  4. IA-32 assembly language programs (individual)
  5. A buffer overrun attack (teams-of-two)
  6. A heap manager module (teams-of-two)
  7. A Unix shell (individual)

• See course “Schedule” web page for due dates/times

• First assignment is available now

• Advice: Start early to allow time for
  • Understanding the assignment and how to get started
  • Debugging
  • Osmosis, background processes, eureka moments …

Leave lots of time for debugging …
Policies: EXTREMELY IMPORTANT

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

• You are responsible for reading the Policies page carefully and understanding it
  • Saying I didn’t know or understand will not be okay

• 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>“Prog. 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>Assignment Support 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


BCPL B C K&R C ANSI C89 ISO C90 ISO/ANSI C99

LISP Smalltalk C++ Java

We will use
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: Design Goals

- **Java design goals**
  - Application programming in the age of multiple operating systems that are highly networked
  - Support object-oriented programming
  - Allow same program to be executed on multiple operating systems
  - Support download-and-run over computer networks
  - Execute code from remote sources securely
  - Adopt the good parts of other languages (esp. C and C++)

- **Implications for Java**
  - High-level
    - Virtual machine insulates programmer from underlying assembly language, machine language, hardware
    - Protects you from shooting yourself in the foot
  - Portability over efficiency
  - Security over efficiency and over flexibility
C vs. Java: Design Goals

• C design goals
  • System-level programming with high efficiency
  • 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
  • And often used for application-level programming
  • Low-level
    • Close to assembly language; close to machine language; close to hardware
  • Efficiency over portability
  • Efficiency and flexibility over security
  • Shoot away (yourself, in the foot …)

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

Course Goals: Why C, not Java?

• The course is not about a language. The language is merely a vehicle to convey the key concepts.
• C happens to better support the goals of the course.
• C supports Goal 1 better
  • C is a lower-level language
    • Forces you to create your own abstractions
  • C has some flaws
    • Motivates discussion of software engineering principles
• C supports Goal 2 better
  • C facilitates language levels tour
    • C is closely related to assembly language
  • C facilitates service levels tour
    • Linux operating system is written in C
# Appendix

## C vs. Java: Details

### Read on your own

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></td>
<td><strong>hello.c:</strong></td>
</tr>
<tr>
<td>public class Hello {</td>
<td></td>
<td>#include &lt;stdio.h&gt;</td>
</tr>
<tr>
<td>public static void</td>
<td></td>
<td>int main(void) {</td>
</tr>
<tr>
<td>main(String[] args) {</td>
<td></td>
<td>printf(&quot;Hello, world\n&quot;);</td>
</tr>
<tr>
<td>System.out.println(</td>
<td></td>
<td>return 0;</td>
</tr>
<tr>
<td>&quot;Hello, world&quot;);</td>
<td></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></td>
<td>% gcc217 hello.c</td>
</tr>
<tr>
<td>% is</td>
<td></td>
<td>% is</td>
</tr>
<tr>
<td>Hello.class</td>
<td></td>
<td>a.out</td>
</tr>
<tr>
<td>Hello.java</td>
<td></td>
<td>hello.c</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></td>
<td>% a.out</td>
</tr>
<tr>
<td>Hello, world</td>
<td></td>
<td>Hello, world</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></td>
<td>char</td>
<td>(unsigned) char</td>
</tr>
<tr>
<td></td>
<td>/* 16-bit unicode */</td>
<td>/* 8 bits */</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Floating point types</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>float</td>
<td>float</td>
</tr>
<tr>
<td></td>
<td>double</td>
<td>double</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>/* use integral type */</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Generic pointer type</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>// no equivalent</td>
<td>void*</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>const int MAX = 1000;</td>
<td></td>
</tr>
<tr>
<td>enum MAX = 1000;</td>
<td>enum (MAX = 1000);</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Arrays</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>int [] a = new int [10];</td>
<td>int a[10];</td>
<td></td>
</tr>
<tr>
<td>float [][] b = new float [5][20];</td>
<td>float b[5][20];</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Array bound checking</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>// run-time check</td>
<td>/* no run-time check */</td>
<td></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></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></th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strings</strong></td>
<td><code>String s1 = &quot;Hello&quot;;</code> <code>String s2 = new</code></td>
<td><code>char *s1 = &quot;Hello&quot;;</code> <code>char s2[6];</code></td>
</tr>
<tr>
<td></td>
<td><code>String(&quot;hello&quot;);</code></td>
<td><code>strcpy(s2, &quot;hello&quot;);</code></td>
</tr>
<tr>
<td><strong>String</strong></td>
<td><code>s1 + s2</code></td>
<td><code>s1 += s2</code></td>
</tr>
<tr>
<td>Concatenation</td>
<td></td>
<td><code>#include &lt;string.h&gt;</code> <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>`&gt;&gt;, &lt;&lt;, &gt;&gt;&gt;, &amp;,</td>
<td>, ^`</td>
</tr>
<tr>
<td><strong>Assignment ops</strong></td>
<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>

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

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>if stmt</strong></td>
<td><code>if (i &lt; 0) statement1;</code> <code>else statement2;</code></td>
<td><code>if (i &lt; 0) statement1;</code> <code>else statement2;</code></td>
</tr>
</tbody>
</table>
| **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>
<tbody>
<tr>
<td><strong>for stmt</strong></td>
<td>for (int i=0; i&lt;10; i++) statement;</td>
<td>int i; for (i=0; i&lt;10; i++) statement;</td>
</tr>
<tr>
<td><strong>while stmt</strong></td>
<td>while (i &lt; 0) statement;</td>
<td>while (i &lt; 0) statement;</td>
</tr>
<tr>
<td><strong>do-while stmt</strong></td>
<td>do { statement; ... } while (i &lt; 0)</td>
<td>do { statement; ... } while (i &lt; 0)</td>
</tr>
<tr>
<td><strong>continue stmt</strong></td>
<td>continue;</td>
<td>continue;</td>
</tr>
<tr>
<td><strong>labeled continue stmt</strong></td>
<td>continue SomeLabel;</td>
<td>/* no equivalent */</td>
</tr>
<tr>
<td><strong>break stmt</strong></td>
<td>break;</td>
<td>break;</td>
</tr>
<tr>
<td><strong>labeled break stmt</strong></td>
<td>break SomeLabel;</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>return 5; return;</td>
<td>return 5; return;</td>
</tr>
<tr>
<td><strong>Compound stmt (alias block)</strong></td>
<td>{ statement1; statement2; }</td>
<td>{ statement1; statement2; }</td>
</tr>
<tr>
<td><strong>Exceptions</strong></td>
<td>throw, try-catch-finally /* no equivalent */</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>f(x, y, z); someObject.f(x, y, z); Someclass.f(x, y, z);</td>
<td>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”
    - Modularity, abstraction, separation of interface from implementation
  - 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, Piazza, 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