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)
  • Dushyant Arora
  • Stephen Beard
  • Jacopo Cesareo
  • Dr. Robert Dondero (Lead Preceptor)
  • Soumyadeep Ghosh
  • Diego Perez Botero
  • Prof. Jennifer Rexford (former Instructor-of-Record)

Course Goal 1: “Programming in the Large”

• How to write large programs

• Specifically, how to:
  • Use abstraction
  • Write modular code
    • Hide information, manage resources, handle errors
  • 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

```
C Language  | Application Program
-----------------------------------------------
Assembly Language | Operating System
Machine Language  | Hardware
```

- 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 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...

*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 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
  • *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
- SSH
- Lab TAs

**Option 2**

- hats.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
  • 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 (48%)
  • Working code
  • Clean, readable, maintainable code
  • On time (penalties for late submission)
  • Final assignment counts double (12%)

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

• Class participation (12%)
  • Lecture and precept attendance is mandatory
  • Will have attendance sheet for lectures; make sure you mark it every time
Programming Assignments

- Programming assignments
  1. A “de-comment” program (individual)
  2. A string module (individual)
  3. A symbol table module (individual)
  4. A primality tester program (large teams)
  5. IA-32 assembly language programs (individual)
  6. A buffer overrun attack (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 …

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 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

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 remote invocation and download over 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
    • Protects you from shooting yourself in the foot
  • 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
  • Low-level
    • Close to assembly language; close to machine language; close
      to hardware
    • Efficiency over portability
    • Efficiency over security
    • Flexibility over security
    • Shoot away (yourself in the foot …)
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

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
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>Java</th>
<th>C</th>
</tr>
</thead>
</table>
| **Overall Program Structure** | `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;
  }
| **Building**   | `javac Hello.java`
  `% is Hello.class`  
  `Hello.java`
  `%`
|                | `gcc217 hello.c`
  `% is 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>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>char /* 16-bit unicode */</td>
<td>char /* 8 bits */</td>
</tr>
<tr>
<td>Integral types</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>Floating point types</td>
<td>float /* 32 bits */</td>
<td>float</td>
</tr>
<tr>
<td></td>
<td>double /* 64 bits */</td>
<td>double</td>
</tr>
<tr>
<td>Logical type</td>
<td>boolean /* no equivalent */</td>
<td>boolean</td>
</tr>
<tr>
<td>Generic pointer type</td>
<td>// no equivalent</td>
<td>void*</td>
</tr>
<tr>
<td>Constants</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>Arrays</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrays</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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Array bound checking</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array bound checking</td>
<td>// run-time check</td>
<td>/* no run-time check */</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pointer type</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer type</td>
<td>// Object reference is an implicit pointer</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>Record type</td>
<td>class Mine { int x; float y; }</td>
<td>struct Mine { int x; float y; }</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>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>
</tr>
<tr>
<td><strong>String concatenation</strong></td>
<td>s1 + s2</td>
<td>#include &lt;string.h&gt; strcat(s1, s2);</td>
</tr>
<tr>
<td><strong>Logical ops</strong></td>
<td>&amp;&amp;,</td>
<td></td>
</tr>
<tr>
<td><strong>Relational ops</strong></td>
<td>=, !=, &gt;, &lt;, &gt;=, &lt;=</td>
<td>=, !=, &gt;, &lt;, &gt;=, &lt;=</td>
</tr>
<tr>
<td><strong>Arithmetic ops</strong></td>
<td>+, -, *, /, %, unary -</td>
<td>+, -, *, /, %, unary -</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>=, ^=</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>if (i &lt; 0) statement1; else statement2;</td>
<td>if (i &lt; 0) statement1; else statement2;</td>
</tr>
<tr>
<td><strong>switch stmt</strong></td>
<td>switch (i) { case 1: ... break; case 2: ... break; default: ... }</td>
<td>switch (i) { case 1: ... break; case 2: ... break; default: ... }</td>
</tr>
<tr>
<td><strong>goto stmt</strong></td>
<td>// no equivalent</td>
<td>goto SomeLabel;</td>
</tr>
</tbody>
</table>
### C vs. Java: Details (cont.)

<table>
<thead>
<tr>
<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>
</table>
| **return stmt** | `return 5;
   return;` | `return 5;
   return;` |
| **Compound stmt (alias block)** | `{
   statement1;
   statement2;
   }` | `{
   statement1;
   statement2;
   }` |
| **Exceptions** | `throw, try-catch-finally` | /* no equivalent */ |
| **Comments** | /* comment */
// another kind | /* comment */ |
| **Method / function call** | `f(x, y, z);
someObject.f(x, y, z);
SomeClass.f(x, y, z);` | `f(x, y, z);` |
Example C Program

#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