COS 217: Introduction to Programming Systems

Jennifer Rexford
Goals for Today’s Class

- Course overview
  - Introductions
  - Course goals
  - Resources
  - Grading
  - Policies

- Getting started with C
  - C programming language overview
Introductions

• Lectures
  • Jennifer Rexford (Professor)
    • jrex@cs.princeton.edu

• Preceptors
  • Christopher Moretti (Lead Preceptor)
    • cmoretti@cs.princeton.edu
  • Sibren Isaacman
    • isaacman@princeton.edu
Course Goal 1: “Programming in the Large”

- Help you learn how to write large computer programs
- Specifically:
  - Write modular code
  - 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”

- Help you learn what happens “under the hood” of computer systems
- Two downward tours

<table>
<thead>
<tr>
<th>C Language</th>
<th>Application Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly Language</td>
<td>Operating System</td>
</tr>
<tr>
<td>Machine Language</td>
<td>Hardware</td>
</tr>
</tbody>
</table>

- Goal 2 supports Goal 1
  - Reveals many examples of effective abstractions
Course Goals: 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?

• 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

• Note: Precepts begin on Monday (i.e., today)
  • P01: MW 1:30-2:20pm, in CS 102
  • P02: TTh 1:30-2:20pm, in CS 102
  • P03: TTh 7:30-8:20pm, in CS 102
Resources: Website and Piazzza

• Website
  • Access from http://www.cs.princeton.edu
    • Academics → Course Schedule → COS 217

• Discussion forum
  • Piazzza: http://www.piazzza.com
  • “Join or create your class now”
    • School: Princeton University
    • Class: COS 217
    • Fill in your Princeton e-mail address
  • Click “get started” link in your email to activate
    • Please use your real name when signing up
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,” key sections are on e-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

  Linux
  GNU
  Your Pgm

  fez
  fedora

  Friend Center 016 or 017 Computer

  SSH

  Lab TAs
• Option 2

hats.princeton.edu

Your PC/Mac/Linux Computer

Your Pgm

GNU

Linux

fez

fedora

SSH

SSH
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 programs locally

• **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

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

• Advice: Start early to allow time for debugging (especially in the background while you are doing other things!)…
Why Debugging is Necessary…

CAN’T YOU DO ANYTHING RIGHT?
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>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</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


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

LISP → Smalltalk → C++ → Java

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 runs on multiple operating systems
  • Support using computer networks
  • Execute code from remote sources securely
  • Adopt the good parts of other languages

• Implications for Java
  • Good for application-level programming
  • High-level (insulates from assembly 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
  • Low-level
  • 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><strong>Java</strong></th>
<th><strong>C</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello.java:</td>
<td><code>public class Hello {</code>&lt;br&gt;<code>public static void</code>&lt;br&gt;<code>main(String[] args) {</code>&lt;br&gt;<code>System.out.println(</code>&lt;br&gt;<code>&quot;Hello, world&quot;);&lt;br&gt;}</code></td>
<td><code>hello.c:</code>&lt;br&gt;include &lt;stdio.h&gt;&lt;br&gt;int main(void) {&lt;br&gt;printf(&quot;Hello, world\n&quot;);&lt;br&gt;return 0;`</td>
</tr>
</tbody>
</table>
| Building                  | `% javac Hello.java`
 `% ls`
 Hello.class
 Hello.java
 `%` | `% gcc217 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></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></td>
<td></td>
<td>long 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>const int MAX = 1000;</td>
<td>const int MAX = 1000;</td>
</tr>
<tr>
<td></td>
<td>enum {MAX = 1000};</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><code>int [] a = new int [10]; float [][] b = new float [5][20];</code></td>
<td><code>int a[10]; float b[5][20];</code></td>
</tr>
<tr>
<td><strong>Array bound checking</strong></td>
<td><code>// run-time check</code></td>
<td><code>/* no run-time check */</code></td>
</tr>
<tr>
<td><strong>Pointer type</strong></td>
<td><code>// Object reference is an implicit pointer</code></td>
<td><code>int *p;</code></td>
</tr>
<tr>
<td><strong>Record type</strong></td>
<td><code>class Mine { int x; float y; }</code></td>
<td><code>struct Mine { int x; float y; }</code></td>
</tr>
</tbody>
</table>
## C vs. Java: Details (cont.)

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
</table>
| **Strings**    | `String s1 = "Hello";`  
                 | `String s2 = new String("hello");`           | `char *s1 = "Hello";`  
                 |                                                | `char s2[6];`  
                 |                                                | `strcpy(s2, "hello");` |
| **String concatenation** | `s1 + s2`  
                 | `s1 += s2`                                     | `#include <string.h>`  
                 |                                                | `strcat(s1, s2);` |
| **Logical ops**| `&&, ||, !`                                     | `&&, ||, !`               |
| **Relational ops** | `=, !=, >, <, >=, <=`                        | `=, !=, >, <, >=, <=` |
| **Arithmetic ops** | `+, -, *, /, %, unary -`                  | `+, -, *, /, %, unary -` |
| **Bitwise ops** | `>>, <<, >>>>, &, |, ^`                       | `>>, <<, &, |, ^`         |
| **Assignment ops** | `=, *=-, /=, +=, -=, <<=, >>>=, >>>>, =, ^=, |=, %=` | `=, *=, /=, +=, -=, <<=, >>>=, =, ^=, |=, %=` |
### C vs. Java: Details (cont.)

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
</table>
| **if stmt** | if (i < 0)  
statement1;  
else  
statement2; | if (i < 0)  
statement1;  
else  
statement2; |
|        | 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>Statement Type</th>
<th>Java Code</th>
<th>C Code</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></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>
</tbody>
</table>
| **Compound stmt**       | `{ statement1;  
    statement2;  
  }`                                      | `{ statement1;  
    statement2;  
  }`                          |
| **(alias block)**       |                                          |                            |
| **Exceptions**          | `throw, try-catch-finally`               | /* no equivalent */         |
| **Comments**            | `/* comment */`                          | `/* comment */`             |
|                         | `// another kind`                        |                            |
| **Method / function call** | `f(x, y, z);`                             | `f(x, y, z);`               |
|                         | `someObject.f(x, y, z);`                 |                            |
|                         | `SomeClass.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;
}
```
Conclusions

• Getting started with C
  • C was designed for system programming
    • Different design goals from Java
    • Explains many of C’s eccentricities
  • Knowing Java gives you a head start at learning C
    • C is not object-oriented, but many aspects are similar

• Getting started in the course
  • Check out course Web site soon
    • Study “Policies” page
    • First assignment
  • Establish a reasonable computing environment soon
    • Instructions given in first precept
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