Hello to COS 217
Introduction to Programming Systems
Fall 2019

Hello to COS 217
Importance of Programming Systems
Fall 2019

Goal 1: Programming in the Large
Learn how to compose large computer programs

Topics
- Modularity/abstraction, information hiding, resource management,
  error handling, testing, debugging, performance improvement,
  tool support

Goal 2: Under the Hood
Learn what happens "under the hood" of computer systems

Downward tours
C Language
Assembly Language
Machine Language
Application Program
Operating System
Hardware

Lead Instructor
- Jennifer Rexford

Lead Preceptors
- Xiaoyan Li
- Christopher Moretti

Graduate Student Preceptors
- Alberto Benmamun
- Greg Chan
- John Li
- Ethan Tseng
- Josh Zhang

Introductions

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

Getting started with C
- History of C
- Building and running C programs
- Characteristics of C
- C details (if time)
Modularity!

Goals: Summary

Help you to become a...

Power Programmer!!!

Specific Goal: Learn C

Question: Why C instead of Java?

Answer 1: A primary language for “under the hood” programming

Answer 2: Knowing a variety of approaches helps you “program in the large”

Specific Goal: Learn Linux

Question: Why use the Linux operating system?

Answer 1: Linux is the industry standard for servers, embedded devices, education, and research

Answer 2: Linux (with GNU tools) is good for programming (which helps explain answer 1)

Agenda

Course overview
• Introductions
• Course goals
• Resources
• Grading
• Policies
• Schedule

Getting started with C
• History of C
• Building and running C programs
• Characteristics of C
• C details (if time)

Lectures

Lectures
• Describe material at conceptual (high) level
• Slides available via course website

Etiquette
• Use electronic devices only for taking notes or annotating slides (but consider taking notes by hand – research shows it works better!)
• No SnapFaceNewsBookInstaGoo, please

iClicker
• Register in Blackboard (not with iClicker – they’ll charge you)
• Occasional questions in class, graded on participation (with a generous allowance for not being able to attend)
iClicker Question

Q: Do you have an iClicker with you today?

A. Yes
B. No, but I've been practicing my mental electrotelekinesis and the response is being registered anyway
C. I'm not here, but someone is iClicking for me (don't do this – it's a violation of our course policies!)

Precepts

Precepts
- Describe material at the "practical" (low) level
- Support your work on assignments
- Hand copy handouts distributed during precepts
- Handouts available via course website

Etiquette
- Attend your precept – attendance will be taken
  - Must miss your precept? ⇒ inform preceptors & attend another
  - Use TigerHub to move to another precept
  - Trouble ⇒ See Colleen Kenny (CS Bldg 210)
  - But Colleen can’t move you into a full precept

Precepts begin next week!

Website

https://www.cs.princeton.edu/courses/archive/fall19/cos217/
- Home page, schedule page, assignment page, policies page

Piazza

Piazza
- Instructions provided in first precept

Piazza etiquette
- Study provided material before posting question
  - Lecture slides, precept handouts, required readings
  - Read / search all (recent) Piazza threads before posting question
  - Don’t reveal your code!
  - See course policies

Books

- King
- C programming language and standard libraries

ARM 64-bit Assembly Language (required)
- Pyeatt & Uighetta
- Book or preprint will be made available later in the term

The Practice of Programming (recommended)
- Kernighan & Pike
- "Programming in the large"

- Bryant & O’Hallaron
- "Under the hood"

Manuals

Manuals (for reference only, available online)
- ARMv8 Instruction Set Overview
- Using as, the GNU Assembler

See also
- Linux man command
Programming Environment

Server
ArmLab Cluster
- Linux OS
- Your Program
- armlab01
- armlab02

Client
Your Computer

On-campus or off-campus

Agenda
Course overview
- Introductions
- Course goals
- Resources

Getting started with C
- History of C
- Building and running C programs
- Characteristics of C
- C details (if time)

Grading
* Final assignment counts double; penalties for lateness
** Closed book, closed notes, no electronic devices
*** Did your involvement benefit the course as a whole?
  - Lecture/precept attendance and participation counts

Programming Assignments
Regular (not-quite-weekly) assignments
0. Introductory survey
1. *De-comment* program
2. String module
3. Symbol table module
4. Assembly language programs
5. Buffer overrun attack
6. Heap manager module
7. Unix shell
*(some individual, some done with a partner from your precept)*

Assignments 0 and 1 are available now
Start early!!!

Policies
Learning is a collaborative activity!
- Discussions with others that help you understand concepts from class are encouraged

But programming assignments are graded!
- Everything that gets submitted for a grade must be exclusively your own work
- Don’t look at code from someone else, the web, Github, etc. – see the course "Policies" web page
- Don’t reveal your code or design decisions to anyone except course staff – see the course "Policies" web page

Violations of course policies
- Typical course-level penalty is 0 on the assignment
- Typical University-level penalty is suspension from University for 1 academic year
### Assignment Related Policies

**Some highlights:**
- You may not reveal any of your assignment solutions (products, descriptions of products, design decisions) on Piazza.
- **Getting help:** To help you compose an assignment solution you may use only authorized sources of information, may consult with other people only via the course's Piazza account or via interactions that might legitimately appear on the course's Piazza account, and must declare your sources in your readme file for the assignment.
- **Giving help:** You may help other students with assignments only via the course's Piazza account or interactions that might legitimately appear on the course's Piazza account, and you may not share your assignment solutions with anyone, ever (including after the semester is over), in any form.

**Ask the instructor for clarifications**
- Permission to deviate from policies must be obtained in writing.

### Questions?

### Agenda

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

#### Getting started with C
- History of C
- Building and running C programs
- Characteristics of C
- C details (if time)

### Course Schedule

<table>
<thead>
<tr>
<th>Weeks</th>
<th>Lectures</th>
<th>Precepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2</td>
<td>C (conceptual) Number Systems</td>
<td>C (pragmatic) Linux/GNU</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>Fall break!</td>
<td></td>
</tr>
<tr>
<td>8-13</td>
<td>&quot;Under the Hood&quot; (conceptual)</td>
<td>&quot;Under the Hood&quot; (assignment how-to)</td>
</tr>
<tr>
<td></td>
<td>Reading Period</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Final Exam</td>
<td></td>
</tr>
</tbody>
</table>

### The C Programming Language

**Who?** Dennis Ritchie  
**When?** ~1972  
**Where?** Bell Labs  
**Why?** Build the Unix OS
Java vs. C: History

<table>
<thead>
<tr>
<th>Year</th>
<th>Language</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1960</td>
<td>BCPL</td>
<td></td>
</tr>
<tr>
<td>1970</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>1972</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>1978</td>
<td>K&amp;R C</td>
<td></td>
</tr>
<tr>
<td>1989</td>
<td>ANSI C89</td>
<td></td>
</tr>
<tr>
<td>1999</td>
<td>ISO C99</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>ISO C11</td>
<td></td>
</tr>
</tbody>
</table>

C vs. Java: Design Goals

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Build the Unix OS</td>
<td>Language of the Internet</td>
</tr>
<tr>
<td>Low-level; close to HW and OS</td>
<td>High-level; insulated from hardware and OS</td>
</tr>
<tr>
<td>Good for system-level programming</td>
<td>Good for application-level programming</td>
</tr>
<tr>
<td>Support structured programming</td>
<td>Support object-oriented programming</td>
</tr>
<tr>
<td>Unsafe: don’t get in the programmer’s way</td>
<td>Safe: can’t step “outside the sandbox”</td>
</tr>
<tr>
<td>Look like C!</td>
<td></td>
</tr>
</tbody>
</table>

Agenda

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

Getting started with C
- History of C
- Building and running C programs
- Characteristics of C
- C details (if time)

Building Java Programs

```
$ javac MyProg.java
```

Java compiler (machine lang code)

```
HW (ArmLab)  OS (Linux)  MyProg.java (Java code)  MyProg.class (bytecode)
```

Running Java Programs

```
$ java MyProg
```

Java interpreter / “virtual machine” (machine lang code)

```
HW (ArmLab)  OS (Linux)  java  MyProg.class (bytecode)
```

Building C Programs

```
$ gcc217 myprog.c –o myprog
```

C “Compiler driver” (machine lang code)

```
HW (ArmLab)  OS (Linux)  gcc217  myprog (machine lang code)
```

Getting started with C
- History of C
- Building and running C programs
- Characteristics of C
- C details (if time)
Running C Programs

$ ./myprog

HW (ArmLab)

OS (Linux)

myprog

(data)

(data)

Agenda

Course overview
• Introductions
• Course goals
• Resources
• Grading
• Policies
• Schedule

Getting started with C
• History of C
• Building and running C programs
• Characteristics of C
• C details (if time)

Java vs. C: Portability

<table>
<thead>
<tr>
<th>Program</th>
<th>Code Type</th>
<th>Portable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyProg.java</td>
<td>Java source code</td>
<td>Yes</td>
</tr>
<tr>
<td>myprog.c</td>
<td>C source code</td>
<td>Mostly</td>
</tr>
<tr>
<td>MyProg.class</td>
<td>Bytecode</td>
<td>Yes</td>
</tr>
<tr>
<td>myprog</td>
<td>Machine lang code</td>
<td>No</td>
</tr>
</tbody>
</table>

Conclusion: Java programs are more portable

(In particular, last semester we moved from the x86_64-based "courselab" to the ARM64-based "armlab", and all of the programs had to be recompiled!)

Java vs. C: Safety & Efficiency

Java
• Automatic array-bounds checking,
• NULL pointer checking,
• Automatic memory management (garbage collection)
• Other safety features

C
• Manual bounds checking
• NULL pointer checking,
• Manual memory management

Conclusion 1: Java is often safer than C
Conclusion 2: Java is often slower than C

Java vs. C: Characteristics

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portability</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Efficiency</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Safety</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

iClicker Question

Q: Which corresponds to the C programming language?

A.

B.

C.
Agenda

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

Getting started with C
- History of C
- Building and running C programs
- Characteristics of C
- C details (if time)

Java vs. C: Details

Remaining slides provide some details

Use for future reference

Slides covered now, as time allows...

Java vs. C: Details

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello.java</td>
<td>hello.c</td>
</tr>
<tr>
<td>public class Hello</td>
<td>include&lt;stdio.h&gt;</td>
</tr>
<tr>
<td>{</td>
<td>int main(void)</td>
</tr>
<tr>
<td>System.out.println(&quot;hello, world&quot;);</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>return 0;</td>
</tr>
</tbody>
</table>

Building

$ javac Hello.java
$ gcc hello.c -o hello

Running

$ java Hello
hello, world
$ ./hello
hello, world

Java vs. C: Details

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

Array bound checking

// run-time check /* no run-time check */

Pointer type

// Object reference is an implicit pointer
int *p;

Record type

class Mine
{  int x;
  float y;
};

struct Mine
{  int x;
  float y;
};

Java vs. C: Details

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>String x1 = &quot;Hello&quot;;</td>
<td>char *x1 = &quot;Hello&quot;;</td>
</tr>
<tr>
<td>String x2 = new String(&quot;Hello&quot;);</td>
<td>#include &lt;string.h&gt;</td>
</tr>
<tr>
<td>x1 += x2</td>
<td>strcat(x1, x2);</td>
</tr>
</tbody>
</table>

Logical ops *

&&, ||, !

Relational ops *

=, !=, <, >, <=, >=

Arithmetic ops *

+, -, *, /, %

<, >>, >>>, &", ", -

Assignment ops

=, +=, -=, *=, /=, %=,

* Essentially the same in the two languages
Java vs. C: Details

### If stmt

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<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></td>
</tr>
<tr>
<td><code>statement2;</code></td>
<td><code>statement2;</code></td>
</tr>
</tbody>
</table>

### Switch stmt

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>switch (i)</code></td>
<td><code>switch (i)</code></td>
</tr>
<tr>
<td>`{</td>
<td></td>
</tr>
<tr>
<td><code>case 1:</code></td>
<td></td>
</tr>
<tr>
<td><code>break;</code></td>
<td></td>
</tr>
<tr>
<td><code>case 2:</code></td>
<td></td>
</tr>
<tr>
<td><code>break;</code></td>
<td></td>
</tr>
<tr>
<td><code>...</code></td>
<td></td>
</tr>
<tr>
<td><code>default:</code></td>
<td></td>
</tr>
<tr>
<td><code>break;</code></td>
<td></td>
</tr>
<tr>
<td><code>...</code></td>
<td></td>
</tr>
<tr>
<td><code>}</code></td>
<td><code>}</code></td>
</tr>
</tbody>
</table>

### Goto stmt

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>goto</code></td>
<td><code>goto</code></td>
</tr>
<tr>
<td><code>statement;</code></td>
<td></td>
</tr>
</tbody>
</table>

* Essentially the same in the two languages

### For stmt

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>for (int i=0; i&lt;10; i++)</code></td>
<td><code>for (i=0; i&lt;10; i++)</code></td>
</tr>
<tr>
<td><code>statement;</code></td>
<td></td>
</tr>
</tbody>
</table>

### While stmt

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>while (i &lt; 0)</code></td>
<td></td>
</tr>
<tr>
<td><code>statement;</code></td>
<td></td>
</tr>
</tbody>
</table>

### Do-while stmt

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>do</code></td>
<td><code>while</code></td>
</tr>
<tr>
<td><code>statement;</code></td>
<td></td>
</tr>
<tr>
<td><code>while (i &lt; 0)</code></td>
<td></td>
</tr>
</tbody>
</table>

### Continue stmt

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>continue;</code></td>
<td><code>continue;</code></td>
</tr>
</tbody>
</table>

### Break stmt

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>break;</code></td>
<td><code>break;</code></td>
</tr>
</tbody>
</table>

### Compound stmt

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{</code></td>
<td></td>
</tr>
<tr>
<td><code>statement1;</code></td>
<td></td>
</tr>
<tr>
<td><code>statement2;</code></td>
<td></td>
</tr>
<tr>
<td><code>}</code></td>
<td><code>}</code></td>
</tr>
</tbody>
</table>

### Exceptions

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>throw, try-catch-finally</code></td>
<td><code>/* no equivalent */</code></td>
</tr>
</tbody>
</table>

### Comments

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>// another kind</code></td>
<td></td>
</tr>
<tr>
<td><code>/* comment */</code></td>
<td><code>/* comment */</code></td>
</tr>
</tbody>
</table>

### Method / function

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>f(x, y, z);</code></td>
<td><code>f(x, y, z);</code></td>
</tr>
<tr>
<td><code>someObject.f(x, y, z);</code></td>
<td><code>SomeClass.f(x, y, z);</code></td>
</tr>
<tr>
<td><code>f(x, y, z);</code></td>
<td><code>f(x, y, z);</code></td>
</tr>
</tbody>
</table>

* Essentially the same in the two languages

Example C Program

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  const double KMETERS_PER_MILE = 1.609;
  int miles;
  double kMeters;
  printf("miles: ");
  if (scanf("%d", &miles) != 1)
  {
    fprintf(stderr, "Error: Expected a number.\n");
    exit(EXIT_FAILURE);
  }
  kMeters = (double)miles * KMETERS_PER_MILE;
  printf("%d miles is %f kilometers.\n", miles, kMeters);
  return 0;
}
```

Summary

**Course overview**
- Introductions
- Course goals
  - Goal 1: Learn "programming in the large"
  - Goal 2: Look "under the hood" and learn low-level programming
- Use of C and Linux supports both goals
- Resources
  - Lectures, precepts, programming environment, Piazza, textbooks
  - Course website: access via http://www.cs.princeton.edu
- Grading
- Policies
- Schedule

**Getting started with C**
- History of C
- Building and running C programs
- Characteristics of C
- Details of C
  - Java and C are similar
  - Knowing Java gives you a head start at learning C
Getting Started

Check out course website soon
- Study "Policies" page
- First assignment is available

Establish a reasonable computing environment soon
- Instructions given in first precept