$ cat welcome.c
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("COS 217\n");
    printf("Introduction to Programming Systems\n\n");
    printf("Fall, 2018\n");
    return 0;
}

$ gcc217 welcome.c –o welcome
$ ./welcome
COS 217
Introduction to Programming Systems
Fall, 2018

---

### 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)

---

### Introductions

#### Lead Instructor
- Andrew Appel
  - appel@princeton.edu

#### Lead Preceptor
- Xiaoyan Li
  - xiaoyan@cs.princeton.edu

#### Faculty Preceptor
- Donna Gabai
  - dgabai@cs.princeton.edu

#### Preceptors
- Seo Young Kyung
  - skyung@princeton.edu
- Austin Le
  - austinle@princeton.edu
- Logan Stafman
  - stafman@princeton.edu
- Alberto Mizrahi Benmamna
  - albertob@princeton.edu
- Jiashuo Zhang
  - jiashuoz@princeton.edu

---

### Goal 1: Programming in the Large

#### Goal 1: “Programming in the large”
- Help you 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

#### Downward tours

- C Language
- Assembly Language
- Machine Language
- Operating System
- Hardware
Modularity!

Goals: Summary

Help you to become a...

Power Programmer!!!
Lectures

iClicker

- Please obtain one and 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 academic fraud)

Precepts

Precepts

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

Precept etiquette

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

Precepts begin next week! (No precept this week)

Website

Website

  - Princeton CS → Courses → Course Schedule → COS 217
  - Home page, schedule page, assignment page, policies page

Piazza

Piazza

- http://piazza.com/class#spr2018/cos217/
- Instructions provided in first precept

Piazza etiquette

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

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

Computer Systems: A Programmer’s Perspective (Third Edition) (recommended)*
- Bryant & O’Hallaron
- “Under the hood”

- King
- C programming language and standard libraries

* out of stock until Oct 16th, a few used copies at Labyrinth, readings available in Blackboard

Manuals

Manuals (for reference only, available online)
- Intel 64 and IA-32 Architectures Software Developer’s Manual, Volumes 1-3
- Intel 64 and IA-32 Architectures Optimization Reference Manual
- Using as, the GNU Assembler

See also
- Linux man command

Programming Environment

Server

CourseLab Cluster

Client

Your Computer

On-campus or off-campus

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)

Grading

<table>
<thead>
<tr>
<th>Course Component</th>
<th>Percentage of Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignments *</td>
<td>50</td>
</tr>
<tr>
<td>Midterm Exam **</td>
<td>15</td>
</tr>
<tr>
<td>Final Exam **</td>
<td>25</td>
</tr>
<tr>
<td>Participation ***</td>
<td>10</td>
</tr>
</tbody>
</table>

These percentages are approximate

* 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

Programing Assignments

Programming assignments
(some individual, some done with a partner from your precept)
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. Game referee

Assignments 0 and 1 are available now
Start early!
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)

Policies

Study the course “Policies” web page!

Especially the assignment collaboration policies
• Violations often involve trial by Committee on Discipline
• Typical course-level penalty is F for course
• 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, in any form.

Ask the instructor for clarifications
• Permission to deviate from policies must be obtained in writing

Course Schedule

<table>
<thead>
<tr>
<th>Weeks</th>
<th>Lectures</th>
<th>Precepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2</td>
<td>Number Systems C (conceptual)</td>
<td>Linux/GNU C (pragmatic)</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>Midterm break!</td>
<td></td>
</tr>
<tr>
<td>8-13</td>
<td>“Under the Hood” (conceptual)</td>
<td>“Under the Hood” (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>

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)

**The C Programming Language**

Who? Dennis Ritchie
When? ~1972
Where? Bell Labs
Why? Build the Unix OS

**Java vs. C: History**

**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>

**Building Java Programs**

$ javac MyPgm.java

Java compiler (machine lang code)

MyPgm.java (Java code) -> javac -> MyPgm.class (bytecode)
Running Java Programs

$ java MyPgm

Java interpreter (Java virtual machine) (machine lang code)

$ gcc217 mypgm.c –o mypgm

C “compiler driver” (machine lang code)

Running C Programs

$ ./mypgm

$ gcc217 mypgm.c –o mypgm

Agenda

Getting started with C

Java vs. C: Portability

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

Conclusion: Java programs are more portable

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>Characteristics</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></th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall</td>
<td>Hello.java:</td>
<td>hello.c:</td>
</tr>
<tr>
<td>Program</td>
<td>public class</td>
<td>#include&lt;stdio.h&gt;</td>
</tr>
<tr>
<td>Structure</td>
<td>Hello</td>
<td>int main(void)</td>
</tr>
<tr>
<td></td>
<td>{</td>
<td>printf(&quot;hello, world\n&quot;);</td>
</tr>
<tr>
<td></td>
<td>public static</td>
<td>return 0;</td>
</tr>
<tr>
<td></td>
<td>void main(String[] args)</td>
<td>}</td>
</tr>
<tr>
<td>Building</td>
<td>$ javac Hello.java</td>
<td>$ gcc hello.c -o hello</td>
</tr>
<tr>
<td>Running</td>
<td>$ java Hello</td>
<td>$ ./hello hello, world</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character Type</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 double</td>
</tr>
<tr>
<td></td>
<td>double // 64 bits</td>
<td>double long double</td>
</tr>
<tr>
<td>Logical Type</td>
<td>boolean /* no equivalent */</td>
<td>/* use int */</td>
</tr>
<tr>
<td>Generic Pointer Type</td>
<td>Object</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>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>
Java vs. C: Details

### Java vs. C

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arrays</strong></td>
<td></td>
</tr>
<tr>
<td>int [] a = new int [10]; float [][] b = new float [5][20];</td>
<td>int a[10]; float b[5][20];</td>
</tr>
<tr>
<td><strong>Array bound checking</strong></td>
<td>/* no run-time check */</td>
</tr>
<tr>
<td>// run-time check</td>
<td></td>
</tr>
<tr>
<td><strong>Pointer type</strong></td>
<td></td>
</tr>
<tr>
<td>// Object reference is an implicit pointer</td>
<td>int *p;</td>
</tr>
<tr>
<td>// Check for NULL, throw exception</td>
<td>/* no run-time check */</td>
</tr>
<tr>
<td><strong>Nullpointer checking</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Record type</strong></td>
<td></td>
</tr>
<tr>
<td>class Mine { int x; float y; }</td>
<td>struct Mine { int x; float y; }</td>
</tr>
<tr>
<td><strong>Strings</strong></td>
<td></td>
</tr>
<tr>
<td>String s1 = &quot;Hello&quot;; char *s1 = &quot;Hello&quot;; String s2 = new String(&quot;hello&quot;); char s2[6]; strcpy(s2, &quot;hello&quot;);</td>
<td></td>
</tr>
<tr>
<td><strong>String concatenation</strong></td>
<td></td>
</tr>
<tr>
<td>s1 + s2</td>
<td>$include &lt;string.h&gt; strcat(s1, s2);</td>
</tr>
<tr>
<td>s1 += s2</td>
<td></td>
</tr>
<tr>
<td><strong>Logical ops</strong></td>
<td></td>
</tr>
<tr>
<td>&amp;&amp;,</td>
<td></td>
</tr>
<tr>
<td><strong>Relational ops</strong></td>
<td></td>
</tr>
<tr>
<td>=, !=, &lt;, &gt;, &gt;=, &lt;=</td>
<td>=, !=, &lt;, &gt;, &gt;=, &lt;=</td>
</tr>
<tr>
<td><strong>Arithmetic ops</strong></td>
<td></td>
</tr>
<tr>
<td>+, -, *, /, %, unary -</td>
<td>+, -, *, /, %, unary -</td>
</tr>
<tr>
<td><strong>Bitwise ops</strong></td>
<td></td>
</tr>
<tr>
<td>&gt;&gt;, &lt;&lt;, &gt;&gt;&gt;, &amp;,</td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>^</td>
</tr>
<tr>
<td><strong>Assignment ops</strong></td>
<td></td>
</tr>
<tr>
<td>=, *=, /=, +=, -=, &lt;&lt;=, &gt;&gt;=, &gt;&gt;&gt;=, &amp;=, ^=,</td>
<td>=, %=</td>
</tr>
</tbody>
</table>

* Essentially the same in the two languages

### Java vs. C

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>if stmt</strong></td>
<td></td>
</tr>
<tr>
<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></td>
</tr>
<tr>
<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></td>
</tr>
<tr>
<td>// no equivalent</td>
<td>goto someLabel;</td>
</tr>
<tr>
<td><strong>for stmt</strong></td>
<td></td>
</tr>
<tr>
<td>for (int i=0; i&lt;10; i++) statement;</td>
<td>for (i=0; i&lt;10; i++) statement;</td>
</tr>
<tr>
<td><strong>while stmt</strong></td>
<td></td>
</tr>
<tr>
<td>while (i &lt; 0) statement;</td>
<td>while (i &lt; 0) statement;</td>
</tr>
<tr>
<td><strong>do-while stmt</strong></td>
<td></td>
</tr>
<tr>
<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></td>
</tr>
<tr>
<td>continue;</td>
<td>continue;</td>
</tr>
<tr>
<td><strong>labeled continue stmt</strong></td>
<td></td>
</tr>
<tr>
<td>continue someLabel;</td>
<td>/* no equivalent */</td>
</tr>
<tr>
<td><strong>break stmt</strong></td>
<td></td>
</tr>
<tr>
<td>break;</td>
<td>break;</td>
</tr>
<tr>
<td><strong>labeled break stmt</strong></td>
<td></td>
</tr>
<tr>
<td>break someLabel;</td>
<td>/* no equivalent */</td>
</tr>
</tbody>
</table>

* Essentially the same in the two languages

### Java vs. C

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>return stmt</strong></td>
<td></td>
</tr>
<tr>
<td>return 5;</td>
<td>return 5;</td>
</tr>
<tr>
<td><strong>Compound stmt (alias block)</strong></td>
<td></td>
</tr>
<tr>
<td>{ statement1; statement2; }</td>
<td>{ statement1; statement2; }</td>
</tr>
<tr>
<td><strong>Exceptions</strong></td>
<td></td>
</tr>
<tr>
<td>throw, try-catch-finally</td>
<td>/* no equivalent */</td>
</tr>
<tr>
<td><strong>Comments</strong></td>
<td></td>
</tr>
<tr>
<td>/* comment */</td>
<td>/* comment */</td>
</tr>
<tr>
<td>// another kind</td>
<td></td>
</tr>
<tr>
<td><strong>Method / function call</strong></td>
<td></td>
</tr>
<tr>
<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>

* 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;
}
```
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