```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("Welcome to COS 217\n");
    printf("Introduction to Programming Systems\n\n");
    printf("%s %d\n", "Spring", 2020);
    return 0;
}
```

```
$ cat Makefile
CC=gcc217
welcome: welcome.o

$ make
$ make -c -o welcome.o welcome.c
gcc217 welcome.o -o welcome
$ ./welcome
Welcome to COS 217
Introduction to Programming Systems
Spring 2020
```
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
• Christopher Moretti cmoretti@cs.princeton.edu

Lead Preceptor
• Xiaoyan Li xiaoyan@cs.princeton.edu

Preceptors
• Donna Gabai dgabai@cs.princeton.edu
• Alberto Mizrahi albertob@princeton.edu
• Josh Zhang jiashuoz@princeton.edu
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)
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

Learn “how to be a client of an operating system”

Downward tours

- C Language
- Assembly Language
- Machine Language

- Application Program
- Operating System
- Hardware

language levels tour

service levels tour
Goals: Summary

Help you to gain ...

Programming Maturity
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

• 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)
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

• Describe material at the “practical” (low) level
• Support your work on assignments
• Hard 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 today!
Website

https://www.cs.princeton.edu/courses/archive/spr20/cos217/

- Home page, schedule page, assignment page, policies page
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 & Ughetta

**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”
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
- GNU tools
- Your Program

Client

Your Computer

SSH

On-campus or off-campus

Server:
- ArmLab Cluster
- Server nodes: armlab01, armlab02

Client:
- Your computer
- SSH connection

Your Program runs on the server, and you access it through SSH from your computer, whether 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>

* 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 (~every 10-14 days) 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

*(partnered assignment)*

Assignments 0 and 1 are available now

Start early! Assignment 1 is due in 10 days!
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

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
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.
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>3-5</td>
<td>Programming 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>Fall break!</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></td>
<td>Reading Period</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Final Exam</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

This is what we’re using
## 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><strong>Unsafe: don’t get in the programmer’s way</strong></td>
<td><strong>Safe: can’t step “outside the sandbox”</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Look like C!</strong></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)

javac

$ javac MyProg.java
Java compiler
(machine lang code)
Running Java Programs

$ java MyProg

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

data

HW (ArmLab)

OS (Linux)

java

MyProg.class (bytecode)

data
Building C Programs

$ gcc217 myprog.c -o myprog

C “Compiler driver” (machine lang code)
Running C Programs

$ ./myprog

myprog (machine lang code)

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 spring we moved from the x86_64-based “courselab” to the ARM64-based “armlab”, and all our 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>
Q: Which corresponds to the C programming language?

A.  

B.  

C.  

iClicker Question
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>Overall Program Structure</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
</table>
| 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; 
} |

<table>
<thead>
<tr>
<th>Building</th>
<th>$ javac Hello.java</th>
<th>$ gcc217 hello.c -o hello</th>
</tr>
</thead>
</table>
| Running                   | $ java Hello 
hello, world 
$ | $ ./hello 
hello, world 
$ |
# Java vs. C: Details

<table>
<thead>
<tr>
<th>Character type</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>char /* 16-bit Unicode</td>
<td>char /* 8 bits */</td>
</tr>
<tr>
<td>Integral types</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>byte /* 8 bits */</td>
<td>(unsigned, signed) char</td>
</tr>
<tr>
<td></td>
<td>short /* 16 bits */</td>
<td>(unsigned, signed) short</td>
</tr>
<tr>
<td></td>
<td>int /* 32 bits */</td>
<td>(unsigned, signed) int</td>
</tr>
<tr>
<td></td>
<td>long /* 64 bits */</td>
<td>(unsigned, signed) 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></td>
<td>long double</td>
<td>long double</td>
</tr>
<tr>
<td>Logical type</td>
<td>boolean</td>
<td>/* no equivalent */</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/* use 0 and non-0 */</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></td>
<td>const int MAX = 1000;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enum {MAX = 1000};</td>
</tr>
</tbody>
</table>
## Java vs. C: Details

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arrays</strong></td>
<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>// run-time check</td>
<td>/* no run-time check */</td>
</tr>
<tr>
<td><strong>Pointer type</strong></td>
<td>// Object reference is an implicit pointer</td>
<td>int *p;</td>
</tr>
<tr>
<td><strong>Record type</strong></td>
<td>class Mine { int x; float y; }</td>
<td>struct Mine { int x; float y; }</td>
</tr>
</tbody>
</table>
## Java vs. C: Details

<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></td>
<td>s1 += s2</td>
<td></td>
</tr>
<tr>
<td>**Logical ops *</td>
<td>&amp;&amp;,</td>
<td></td>
</tr>
<tr>
<td>**Relational ops *</td>
<td>=, !=, &lt;, &gt;, &lt;=, &gt;=</td>
<td>=, !=, &lt;, &gt;, &lt;=, &gt;=</td>
</tr>
<tr>
<td>**Arithmetic ops *</td>
<td>+, -, *, /, %, unary -</td>
<td>+, -, *, /, %, unary -</td>
</tr>
<tr>
<td>**Bitwise ops</td>
<td>&lt;&lt;, &gt;&gt;, &gt;&gt;&gt;, &amp;, ^,</td>
<td>&lt;&lt;, &gt;&gt;, &amp;^, ^,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Assignment ops</strong></td>
<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: Details

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>if stmt *</td>
<td><code>if (i &lt; 0) statement1; else statement2;</code></td>
<td><code>if (i &lt; 0) statement1; else statement2;</code></td>
</tr>
<tr>
<td>switch stmt *</td>
<td><code>switch (i) { case 1: ... break; case 2: ... break; default: ... }</code></td>
<td><code>switch (i) { case 1: ... break; case 2: ... break; default: ... }</code></td>
</tr>
<tr>
<td>goto stmt</td>
<td><code>// no equivalent</code></td>
<td><code>goto someLabel;</code></td>
</tr>
</tbody>
</table>

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

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

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

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

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