$ cat welcome.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", 2022);
    return 0;
}

$ cat Makefile
CC=gcc217
welcome: welcome.o

$ make
gcc217 -c -o welcome.o welcome.c
gcc217 welcome.o -o welcome

$ ./welcome
Welcome to COS 217
Introduction to Programming Systems

Spring 2022
Agenda

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

A taste of C
- History of C
- Building and running C programs
- Characteristics of C
- Java vs C
Introductions

Lead Faculty

• Aarti Gupta  aartig@cs.princeton.edu

Lead Preceptor

• Christopher Moretti  cmoretti@cs.princeton.edu

Preceptors

• Cedrick Argueta  argueta@princeton.edu
• Huihan Li  huihanl@princeton.edu
• Maxine Perroni-Scharf  mp4215@princeton.edu
Agenda

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

A taste of C
- History of C
- Building and running C programs
- Characteristics of C
- Java vs C
Goal 1: Programming in the Large

Learn how to compose large(r) computer programs

Topics

- Modularity/abstraction, information hiding, resource management, error handling, testing, debugging, performance improvement
- Tools: ssh, bash, shell utilities, emacs, git, gcc, make, gdb, gprof, valgrind
Modularity!
Goal 2: Lower-level Languages

```c
int main(void) {
    while ((iChar = getchar()) != EOF) {
        lCharCount++;
        if (isspace(iChar)) {
            if (iInWord) {
                lWordCount++;
                iInWord = FALSE;
            }
        }
    }
}
```

**THE C PROGRAMMING LANGUAGE**

```
main:
.LFB0:
.cfi_startproc
stp x29, x30, [sp, -16]!
.cfi_def_cfa_offset 16
.cfi_offset 29, -16
.cfi_offset 30, -8
add x29, sp, 0
.cfi_def_cfa_register 29
b .L2

RELOCATION RECORDS FOR [.eh_frame]:
OFFSET TYPE VALUE
000000000000001c R_AARCH64_PREL32 .text

Contents of section .text:
0000 fd7bbfa9 fd030091 39000014
00000090 .{......9.......}
```
Goals: Summary

Help you to gain ...

Programming Maturity

Jungwoo Hong
Specific Goal: Learn C

Question: Why C instead of Java?

Answer 1: A primary language for “under the hood” programming in real code bases.

Answer 2: A variety of experience 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)
Programming Environment

Server

ArmLab Cluster

Linux OS

GNU tools

Your Program

armlab01

armlab02

Client

Your Computer

SSH
Agenda

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

A taste of C
- History of C
- Building and running C programs
- Characteristics of C
- Java vs C
Lectures

Describe material at a mix of levels
• Some conceptual (high) overview
• Some digging into details

Slides on course website

Recordings of live lectures will be posted on course website

Videos from last year (Spring, 2021) available on Youtube

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

Occasional questions in class, graded on participation (with a generous allowance for not being able to attend)

- Can use either a physical remote, an app on your phone, or web

- Create account / register at iclicker.com
  - If asked, we’re using “iClicker Cloud” and “Canvas”
Q: Do you have an iClicker (remote or app) with you today?

A. Yes

B. No, but I’ve been practicing my mental electrotelekenisis 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 in precept
• 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

Precepts begin today!
Websites

https://www.cs.princeton.edu/~cos217 (Course website)
  • Home page, schedule page, assignment page, policies page

https://princeton.instructure.com/courses/5922 (Canvas)
  • Links to Ed, Library reserves and other readings, NameCoach
https://us.edstem.org/us/courses/19718/discussion/

• Also available as a Canvas link
• Q&A – post here instead of emailing staff

Etiquette

• Study provided material before posting question
  • Lecture slides, precept handouts, required readings
• Read / search all (recent) Ed threads before posting question
• Don’t reveal your code!
  • See course policies
  • Click “private” if in doubt
Books

- King
- C programming language and standard libraries

ARM 64-bit Assembly Language (online)
- Pyeatt with Ughetta

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

Manuals (for reference only, available online)

• ARMv8 Instruction Set Overview
• ARM Architecture Reference Manual
• Using as, the GNU Assembler

See also

• Linux man command
Help!

Office Hours

- Preceptors: 2+ hours scheduled every weekday + Sunday, in-person and Zoom
- Me: after lecture
- Schedule is on the course website
- Zoom form / links are on Canvas

Lab TAs

- Your peers are available 4+ hours per day, every single day
- These are specific to debugging your assignments. For conceptual help with course materials, go to office hours.
- [https://labta.cs.princeton.edu/](https://labta.cs.princeton.edu/)
Agenda

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

A taste of C
• History of C
• Building and running C programs
• Characteristics of C
• Java vs C
Grading

<table>
<thead>
<tr>
<th>Course Component</th>
<th>Percentage of Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignments *</td>
<td>60</td>
</tr>
<tr>
<td>Midterm Exam **</td>
<td>10</td>
</tr>
<tr>
<td>Final Exam **</td>
<td>20</td>
</tr>
<tr>
<td>Participation ***</td>
<td>10</td>
</tr>
</tbody>
</table>

* 6 assignments × 10% each. Late assignments 20% off per day; 4 late days free.

** During midterms week and final exam period, respectively. Closed book/notes.

*** Did your involvement benefit the course?
  • Lecture/precept attendance and precept/Ed participation
Programming Assignments

Regular (every 1.5-2.5 weeks) assignments

0. Introductory survey
1. “De-comment” program
2. String module
3. Symbol table module
4. Debugging directory and file trees *
5. Assembly language programming *
6. Buffer overrun attack *

*(partnered assignment)

Assignments 0 and 1 are available now: start early!!
Agenda

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

A taste of C
- History of C
- Building and running C programs
- Characteristics of C
- Java vs C
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
• Typical University-level penalty is suspension
COS 1xx/2xx courses are hard under the best of circumstances
  • Information-dense
  • Programming is a new skill, or “craft”: not like writing essays or doing problem sets

These are not the best of circumstances
  • We are all worried about ourselves, friends, family
  • We all feel stressed, anxious, uncertain – but when these veer into panic or depression...

Say something, and get help
  • Reach out to CPS, your residential college dean, course staff
  • No judgment – the rest of us are feeling it too
Questions?
Agenda

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

A taste of C
- History of C
- Building and running C programs
- Characteristics of C
- Java vs C
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>Unsafe: don’t get in the programmer’s way</td>
<td>Safe: can’t step “outside the sandbox”</td>
</tr>
<tr>
<td></td>
<td>Look like C!</td>
</tr>
</tbody>
</table>
Agenda

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

A taste of C
- History of C
- Building and running C programs
- Characteristics of C
- Java vs C
Building Java Programs

$ javac MyProg.java

Java compiler
(machine lang code)

HW (ArmLab)

OS (Linux)

MyProg.java
(Java code)

javac

MyProg.class
(bytecode)
Running Java Programs

$ java MyProg

Java interpreter / “virtual machine” (machine lang code)
Building C Programs

$ gcc217 myprog.c –o myprog

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

$ ./myprog

myprog (machine lang code)
Agenda

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

A taste of C
• History of C
• Building and running C programs
• Characteristics of C
• Java vs C
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

(For example, COS 217 has used many architectures over the years, and every time we switched, 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
Q: Which corresponds to the C programming language?
#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;
}
| Agenda |
|------------------|------------------|
| **Course overview** | **A taste of C** |
| • Introductions | • History of C |
| • Course goals | • Building and running C programs |
| • Resources | • Characteristics of C |
| • Grading | • Java vs C |
| • Policies | |

43
Java vs. C: Details

Remaining slides provide some details

Use for future reference

Slides covered now, as time allows...
<table>
<thead>
<tr>
<th>Overall Program Structure</th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello.java:</td>
<td>public class Hello</td>
<td>hello.c:</td>
</tr>
<tr>
<td></td>
<td>{ public static void main</td>
<td>#include &lt;stdio.h&gt;</td>
</tr>
<tr>
<td></td>
<td>(String[] args)</td>
<td>int main(void)</td>
</tr>
<tr>
<td></td>
<td>{ System.out.println(</td>
<td>{ printf(&quot;hello, world\n&quot;);</td>
</tr>
<tr>
<td></td>
<td>&quot;hello, world&quot;);</td>
<td>return 0;</td>
</tr>
<tr>
<td></td>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td>Building</td>
<td>$ javac Hello.java</td>
<td>$ gcc217 hello.c -o hello</td>
</tr>
<tr>
<td>Running</td>
<td>$ java Hello</td>
<td>$ ./hello</td>
</tr>
<tr>
<td></td>
<td>hello, world $</td>
<td>hello, world $</td>
</tr>
</tbody>
</table>
# 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>
</tbody>
</table>

### Integral types

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>byte // 8 bits</td>
<td>(unsigned, signed) char</td>
</tr>
<tr>
<td>short // 16 bits</td>
<td>short // 16 bits</td>
<td>(unsigned, signed) short</td>
</tr>
<tr>
<td>int // 32 bits</td>
<td>int // 32 bits</td>
<td>(unsigned, signed) int</td>
</tr>
<tr>
<td>long // 64 bits</td>
<td>long // 64 bits</td>
<td>(unsigned, signed) long</td>
</tr>
</tbody>
</table>

### Floating point types

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>float // 32 bits</td>
<td>float</td>
</tr>
<tr>
<td>double // 64 bits</td>
<td>double // 64 bits</td>
<td>double</td>
</tr>
<tr>
<td></td>
<td></td>
<td>long double</td>
</tr>
</tbody>
</table>

### Logical type

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>boolean</td>
<td>/* no equivalent */</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/* use 0 and non-0 */</td>
</tr>
</tbody>
</table>

### Generic pointer type

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Object</td>
<td>void*</td>
</tr>
</tbody>
</table>

### Constants

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></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>

**Arrays**: In Java, arrays are created using the `new` keyword, and in C, arrays are declared without `new`. Java provides Array bound checking, whereas C does not check bounds at runtime.

**Pointer type**: In Java, an object reference is an implicit pointer, whereas in C, pointers are explicitly declared using `*`.

**Record type**: Java uses classes to define record types, whereas C uses structures.
# 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><strong>Logical ops</strong></td>
<td>&amp;&amp;,</td>
<td></td>
</tr>
<tr>
<td><strong>Relational ops</strong></td>
<td>==, !=, &lt;, &gt;, &lt;=, &gt;=</td>
<td>==, !=, &lt;, &gt;, &lt;=, &gt;=</td>
</tr>
<tr>
<td><strong>Arithmetic ops</strong></td>
<td>+, -, *, /, %, unary -</td>
<td>+, -, *, /, %, unary -</td>
</tr>
<tr>
<td><strong>Bitwise ops</strong></td>
<td>&lt;&lt;, &gt;&gt;, &gt;&gt;&gt;, &amp;, ^,</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>
</table>
| **if stmt** | if (i < 0) 
  statement1; 
  else 
  statement2; | if (i < 0) 
  statement1; 
  else 
  statement2; |
|          | **switch stmt** | switch (i) 
  {  case 1: 
    ...  
    break; 
    case 2: 
    ...  
    break; 
    default: 
    ...  
  } | switch (i) 
  {  case 1: 
    ...  
    break; 
    case 2: 
    ...  
    break; 
    default: 
    ...  
  } |
|          | **goto stmt** | // no equivalent | goto someLabel; |

* 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>while stmt *</td>
<td>while (i &lt; 0) statement;</td>
<td>while (i &lt; 0) statement;</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>continue stmt *</td>
<td>continue;</td>
<td>continue;</td>
</tr>
<tr>
<td>labeled continue stmt</td>
<td>continue someLabel;</td>
<td>/* no equivalent */</td>
</tr>
<tr>
<td>break stmt *</td>
<td>break;</td>
<td>break;</td>
</tr>
<tr>
<td>labeled break stmt</td>
<td>break someLabel;</td>
<td>/* no equivalent */</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;</td>
<td>return 5;</td>
</tr>
<tr>
<td></td>
<td>return;</td>
<td>return;</td>
</tr>
<tr>
<td>Compound stmt (alias block) *</td>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>statement1;</td>
<td>statement1;</td>
</tr>
<tr>
<td></td>
<td>statement2;</td>
<td>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 call</td>
<td>f(x, y, z);</td>
<td>f(x, y, z);</td>
</tr>
<tr>
<td></td>
<td>someObject.f(x, y, z);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SomeClass.f(x, y, z);</td>
<td></td>
</tr>
</tbody>
</table>

* Essentially the same in the two languages*
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, Ed, textbooks
  • Course website: access via https://www.cs.princeton.edu/~cos217
• Grading
• Policies
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

On Wednesday: computing environment
  • In preparation for assignments 0 and 1