```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", 2019);
    return 0;
}
```

```makefile
CC=gcc217
welcome: welcome.o

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

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

Spring, 2019
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
• Prof. Szymon Rusinkiewicz: smr@princeton.edu

Lead Preceptors
• Robert Dondero, Ph.D.: rdondero@cs.princeton.edu
• Xiaoyan Li: xiaoyan@cs.princeton.edu

Graduate Student Preceptors
• James Heppenstall: jwmh@princeton.edu
• Seo Young Kyung: skyung@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
  - language levels tour

- Application Program
  - Operating System
    - Hardware
  - service levels tour
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

• 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 electrotelekiniesis 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 SCORE 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 and tomorrow!
Website

http://www.cs.princeton.edu/~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

*C Programming: A Modern Approach (Second Edition)* (required)

- King
- C programming language and standard libraries

**ARM 64-bit Assembly Language** (required)

- Pyeatt & Ughetta
- Preprint will be made available through Pequod

*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

Manuals (for reference only, available online)

- *ARMv8 Instruction Set Overview*
- *ARM Architecture Reference Manual*
- *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
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 (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!!!
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)
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 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 (including after the semester is over), in any form.

Ask the instructor for clarifications

• Permission to deviate from policies must be obtained in writing.
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><strong>Spring break!</strong></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

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>
</tbody>
</table>

| Look like 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)
$ javac MyProg.java

Java compiler (machine lang code)

HW (ArmLab)

OS (Linux)

MyProg.java (Java code)

javac

MyProg.class (bytecode)
$ java MyProg

Java interpreter / “virtual machine” (machine lang code)
$ gcc217 myprog.c –o myprog

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

$ ./myprog

myprog
(machine lang code)

HW (ArmLab)

OS (Linux)

data

myprog

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, this semester we’re moving from the x86_64-based “courselab” to the ARM64-based “armlab”, and all of the programs must 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.
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>
<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></td>
</tr>
<tr>
<td></td>
<td>{ System.out.println(&quot;</td>
<td>int main(void)</td>
</tr>
<tr>
<td></td>
<td>hello, world&quot;);</td>
<td>{ printf(&quot;hello, world\n&quot;);</td>
</tr>
<tr>
<td></td>
<td>}</td>
<td>return 0;</td>
</tr>
<tr>
<td></td>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

### Building

<table>
<thead>
<tr>
<th></th>
<th>$ javac Hello.java</th>
<th>$ gcc217 hello.c –o hello</th>
</tr>
</thead>
</table>

### Running

<table>
<thead>
<tr>
<th></th>
<th>$ java Hello</th>
<th>$ ./hello</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello, world</td>
<td>$</td>
<td>hello, world</td>
</tr>
<tr>
<td>$</td>
<td>$</td>
<td>$</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>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, 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><strong>Floating point</strong></td>
<td>float // 32 bits</td>
<td>float</td>
</tr>
<tr>
<td>types</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 0 and non-0 */</td>
</tr>
<tr>
<td><strong>Generic pointer</strong></td>
<td>Object</td>
<td>void*</td>
</tr>
<tr>
<td>type</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constants</strong></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>Arrays</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>Array bound checking</td>
<td>// run-time check</td>
<td>/* no run-time check */</td>
</tr>
<tr>
<td>Pointer type</td>
<td>// Object reference is an implicit pointer</td>
<td>int *p;</td>
</tr>
<tr>
<td>Record type</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;;</td>
<td>char *s1 = &quot;Hello&quot;;</td>
</tr>
<tr>
<td></td>
<td>String s2 = new String(&quot;hello&quot;);</td>
<td>char s2[6];</td>
</tr>
<tr>
<td></td>
<td></td>
<td>strcpy(s2, &quot;hello&quot;);</td>
</tr>
<tr>
<td><strong>String concatenation</strong></td>
<td>s1 + s2</td>
<td>#include &lt;string.h&gt;</td>
</tr>
<tr>
<td></td>
<td>s1 += s2</td>
<td>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;, &lt;&lt;&lt;, &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>
<tbody>
<tr>
<td><strong>if stmt</strong></td>
<td>if (i &lt; 0)</td>
<td>if (i &lt; 0)</td>
</tr>
<tr>
<td></td>
<td>statement1;</td>
<td>statement1;</td>
</tr>
<tr>
<td></td>
<td>else</td>
<td>else</td>
</tr>
<tr>
<td></td>
<td>statement2;</td>
<td>statement2;</td>
</tr>
<tr>
<td><strong>switch stmt</strong></td>
<td>switch (i)</td>
<td>switch (i)</td>
</tr>
<tr>
<td></td>
<td>{  case 1:</td>
<td>{  case 1:</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>break;</td>
<td>break;</td>
</tr>
<tr>
<td></td>
<td>case 2:</td>
<td>case 2:</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>break;</td>
<td>break;</td>
</tr>
<tr>
<td></td>
<td>default:</td>
<td>default:</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>}</td>
<td>}</td>
</tr>
<tr>
<td><strong>goto stmt</strong></td>
<td>// no equivalent</td>
<td>goto someLabel;</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Statement Type</th>
<th>Java</th>
<th>C</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;
|-------------------|---------------------------------------------|
| **labeled continue stmt** | continue *someLabel*;
| **break stmt**    | break;
| **labeled break stmt** | break *someLabel*;

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

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

* Essentially the same in the two languages*
#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