



# COS 217: Introduction to Programming Systems





# Agenda

## Course overview

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

## Getting started with C

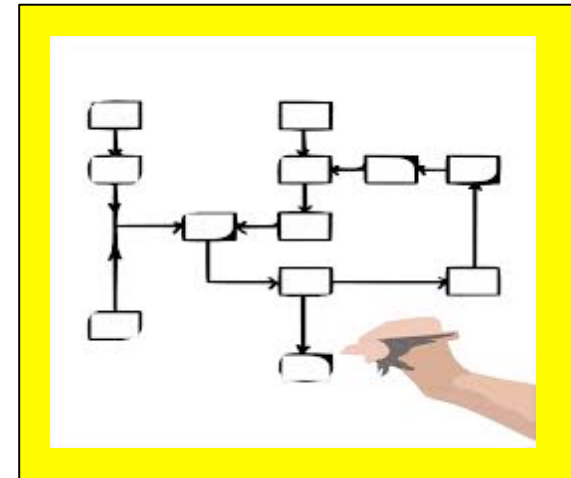
- History of C
- Building and running C programs
- Characteristics of C
- C details (if time)



# Goal 1: “Pgmming 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

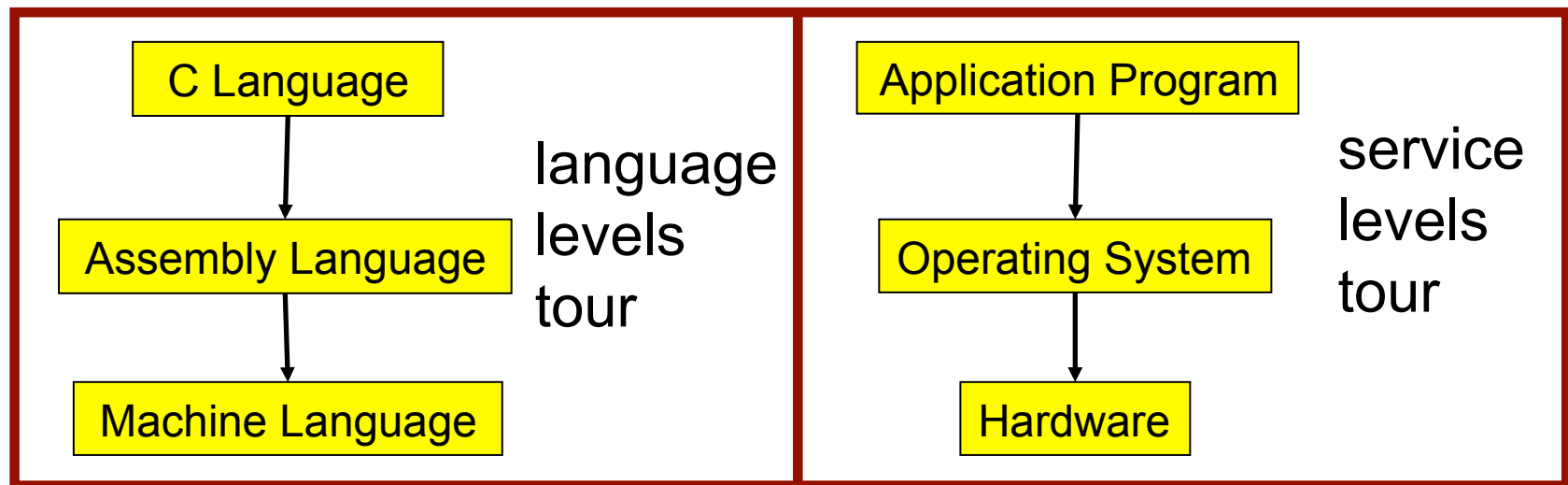


# Goal 2: “Under the Hood”

## Goal 2: “Look under the hood”

- Help you learn what happens “under the hood” of computer systems

### Downward tours



# Goals: Summary

Help you to become a...



***Power Programmer!!!***



# Goals: Why C?

**Question:** Why C instead of Java?

**Answer 1:** C supports Goal 2 better

**Answer 2:** C supports Goal 1 better

THE  
C  
PROGRAMMING  
LANGUAGE



# 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?** Compose the Unix OS



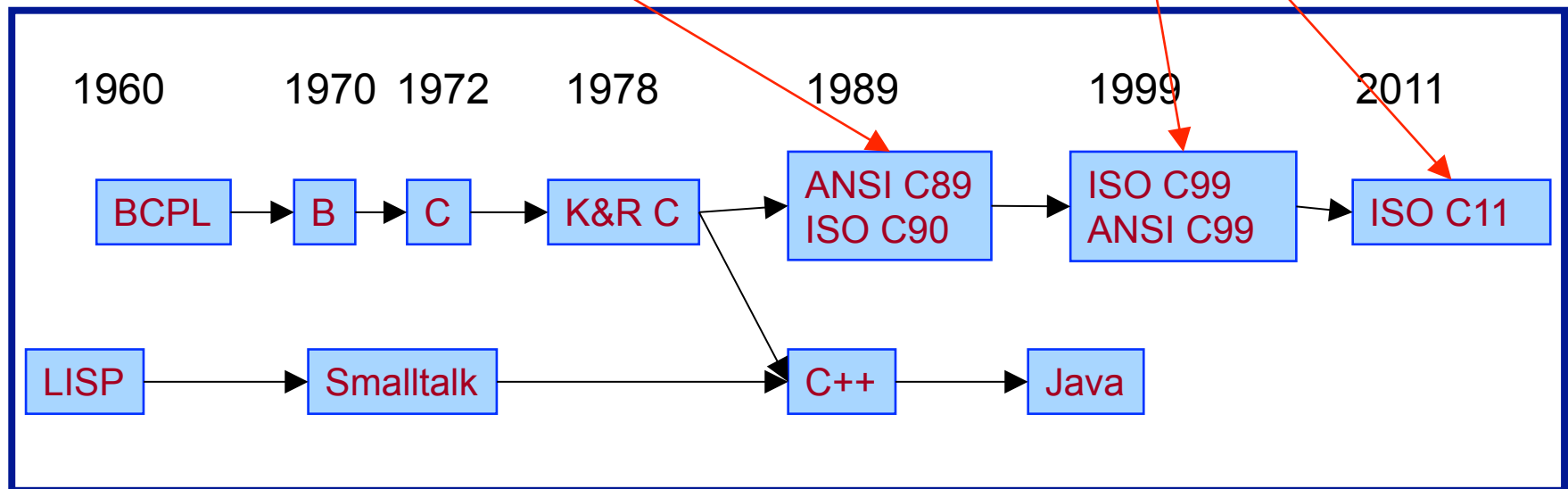




# Java vs. C: History

We will use

Not (yet?) popular;  
our compiler  
supports only  
partially





# Java vs. C: Design Goals

Java Design Goals	C Design Goals
Language of the Internet	Compose Unix
High-level; insulated from hardware and OS	Low-level; close to HW and OS
Good for application-level programming	Good for system-level programming
Support object-oriented programming	Support structured programming
Look like C!	



# Agenda

## Course overview

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

## Getting started with C

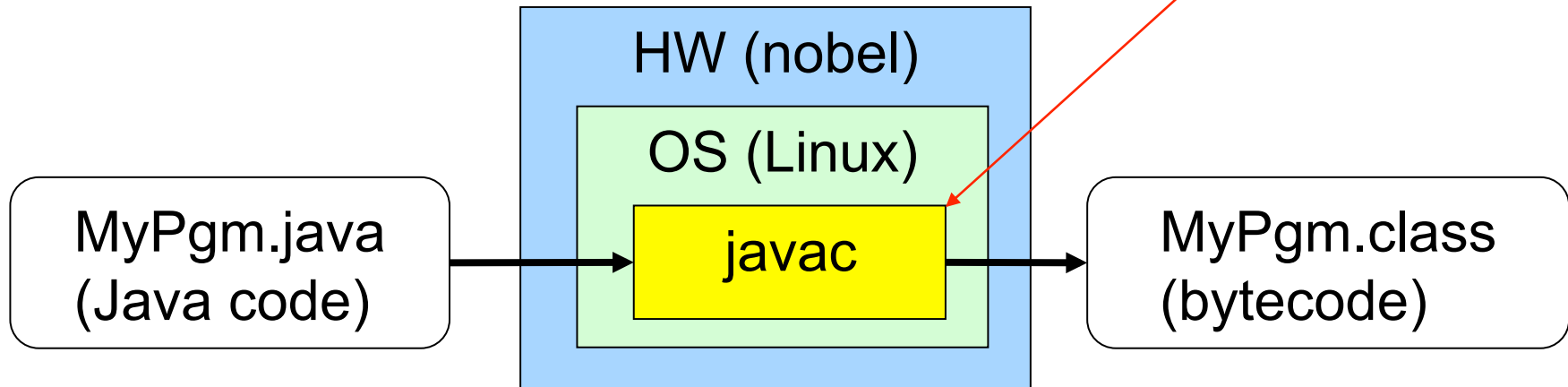
- History of C
- **Building and running C programs**
- Characteristics of C
- C details (if time)



# Building Java Programs

**\$ javac MyPgm.java**

Java compiler  
(machine lang code)

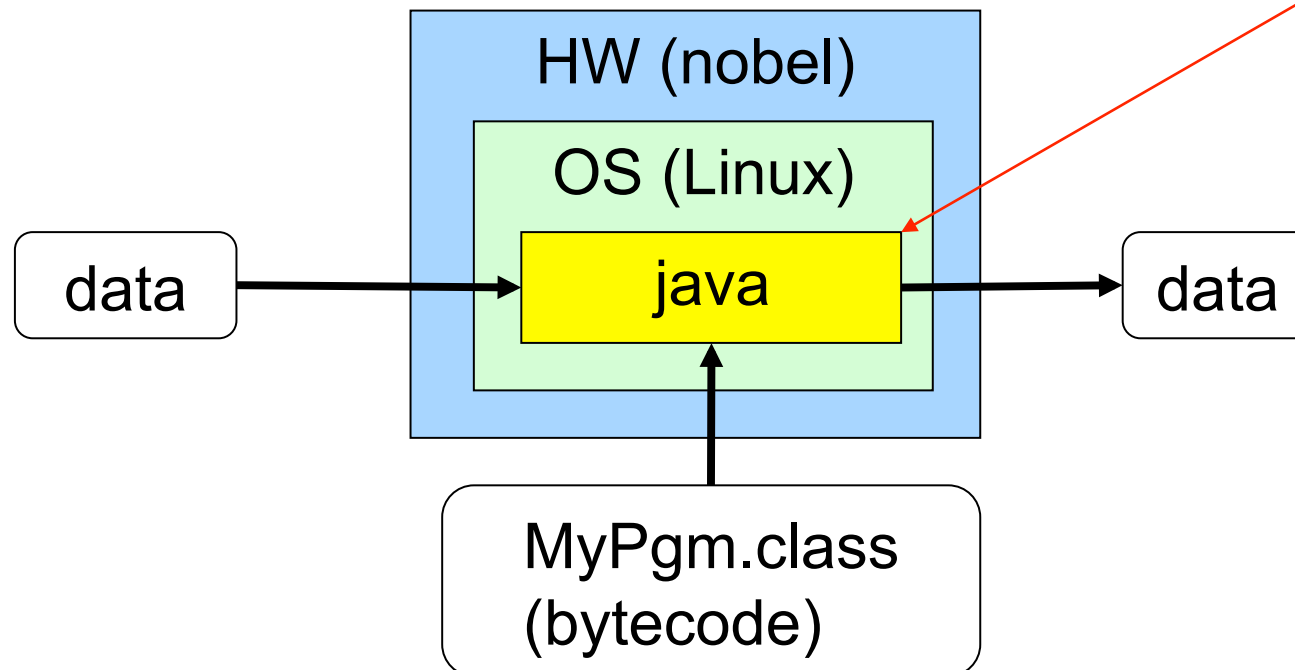




# Running Java Programs

**\$ java MyPgm**

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

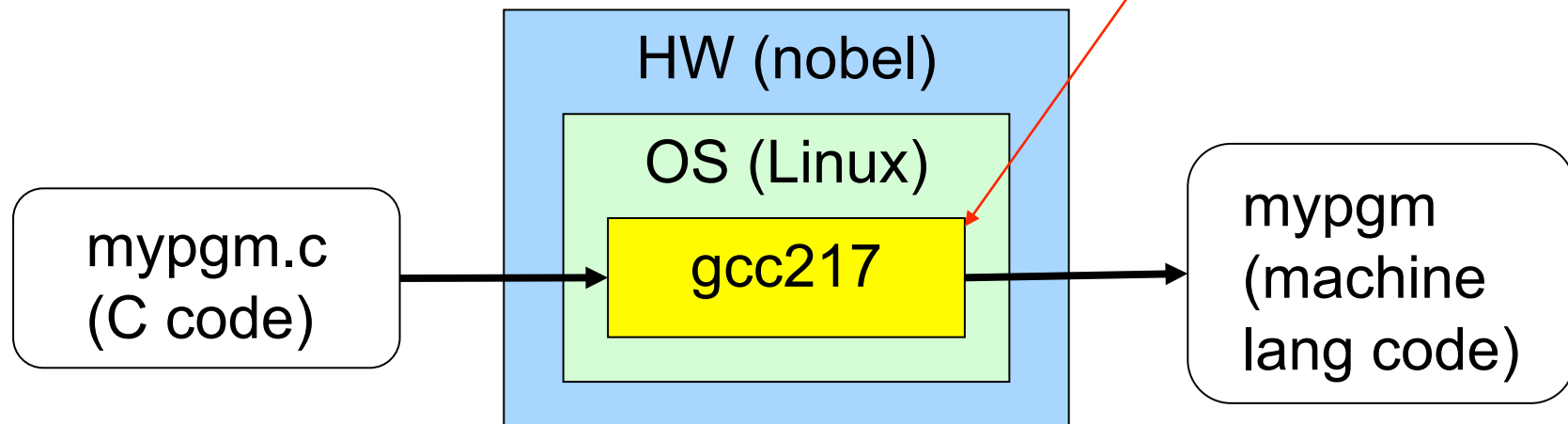




# Building C Programs

```
$ gcc217 mypgm.c -o mypgm
```

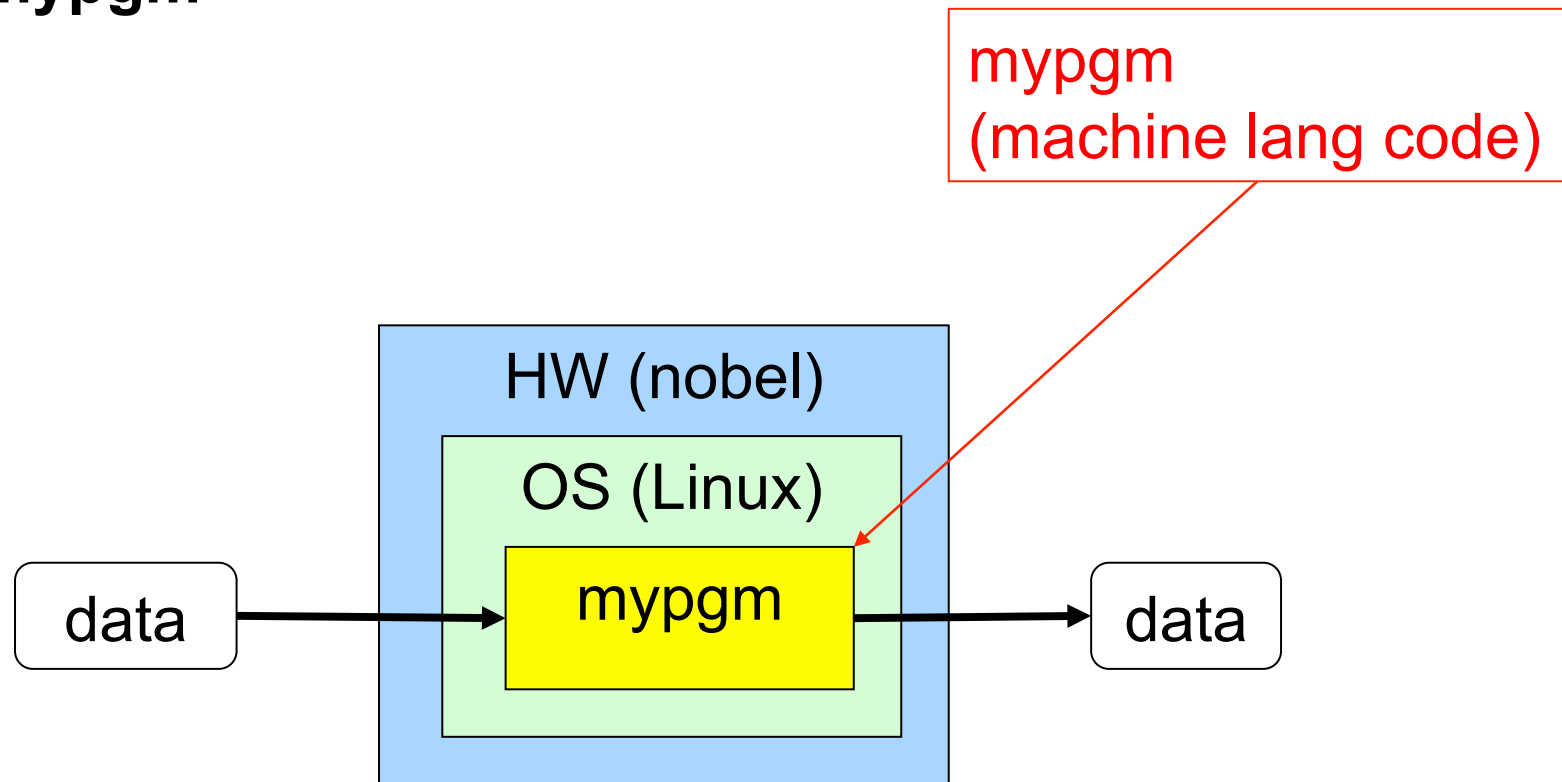
C “compiler driver”  
(machine lang code)





# Running C Programs

**\$ mypgm**





# 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)





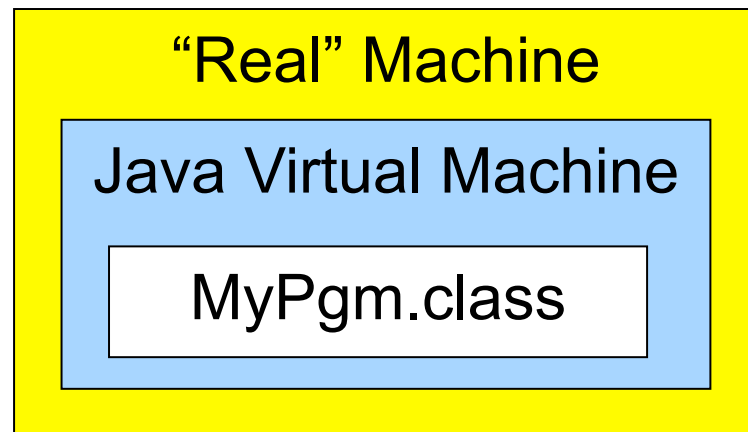
# Java vs. C: Portability

Program	Code Type	Portable?
MyPgm.java	Java source code	Yes
mypgm.c	C source code	Mostly
MyPgm.class	Bytecode	Yes
mypgm	Machine lang code	No
javac (Java compiler)	Machine lang code	No
java (Java interpreter)	Machine lang code	No
gcc217 (C compiler driver)	Machine lang code	No

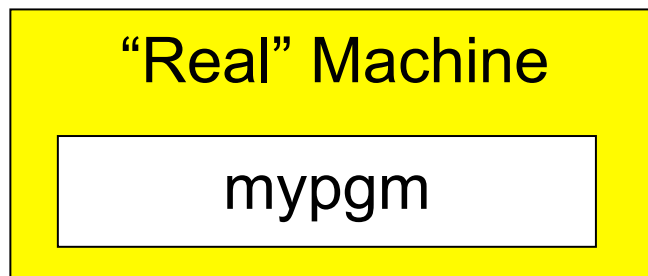
**Conclusion:** Java programs are more portable



# Java vs. C: Efficiency



Java programs run on "virtual" machine which runs on "real" machine

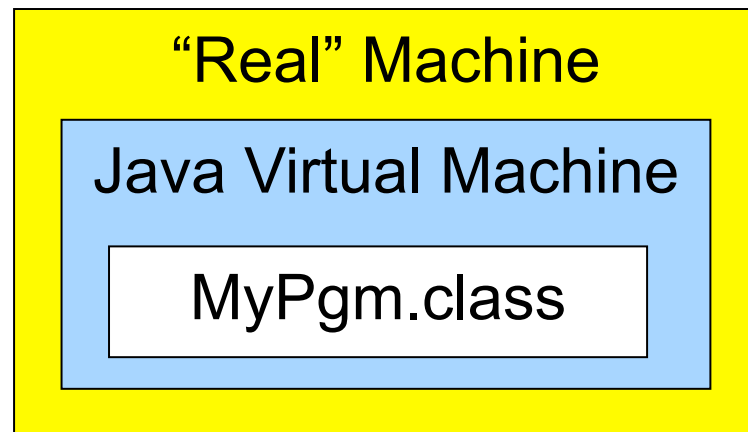


C programs run on "real" machine

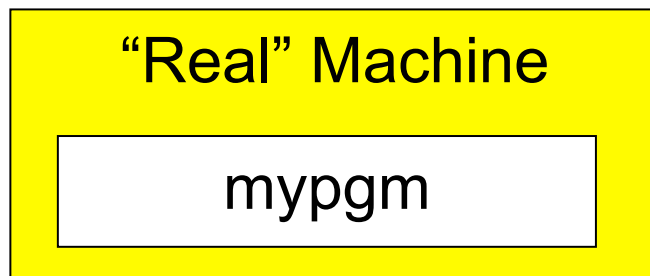
**Conclusion:** C programs are faster



# Java vs. C: Safety



Java programs run on  
"virtual" machine defined by  
interpreter; can provide safe  
environment  
(e.g. array bounds checks)



C programs run directly  
on "real" machine

**Conclusion:** Java programs are safer



# Java vs. C: Characteristics

	Java	C
Portability	+	-
Efficiency	-	+
Safety	+	-



# Java vs. C: Characteristics



If this is Java...



# Java vs. C: Characteristics



Then this is 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

	Java	C
Overall Program Structure	<pre>Hello.java:  public class Hello {   public static void main     (String[] args)     {   System.out.println(         "hello, world");     } }</pre>	<pre>hello.c:  #include &lt;stdio.h&gt;  int main(void) {   printf("hello, world\n");     return 0; }</pre>
Building	<pre>\$ javac Hello.java</pre>	<pre>\$ gcc217 hello.c -o hello</pre>
Running	<pre>\$ java Hello hello, world \$</pre>	<pre>\$ hello hello, world \$</pre>



# Java vs. C: Details

	Java	C
Character type	<code>char // 16-bit Unicode</code>	<code>char /* 8 bits */</code>
Integral types	<code>byte // 8 bits</code> <code>short // 16 bits</code> <code>int // 32 bits</code> <code>long // 64 bits</code>	<code>(unsigned) char</code> <code>(unsigned) short</code> <code>(unsigned) int</code> <code>(unsigned) long</code>
Floating point types	<code>float // 32 bits</code> <code>double // 64 bits</code>	<code>float</code> <code>double</code> <code>long double</code>
Logical type	<code>boolean</code>	<code>/* no equivalent */</code> <code>/* use integral type */</code>
Generic pointer type	<code>// no equivalent</code>	<code>void*</code>
Constants	<code>final int MAX = 1000;</code>	<code>#define MAX 1000</code> <code>const int MAX = 1000;</code> <code>enum {MAX = 1000};</code>



# Java vs. C: Details

	Java	C
Arrays	<pre>int [] a = new int [10]; float [][] b =     new float [5][20];</pre>	<pre>int a[10]; float b[5][20];</pre>
Array bound checking	<pre>// run-time check</pre>	<pre>/* no run-time check */</pre>
Pointer type	<pre>// Object reference is an // implicit pointer</pre>	<pre>int *p;</pre>
Record type	<pre>class Mine {   int x;     float y; }</pre>	<pre>struct Mine {   int x;     float y; };</pre>



# Java vs. C: Details

	Java	C
Strings	<code>String s1 = "Hello";</code> <code>String s2 = new</code> <code>String("hello");</code>	<code>char *s1 = "Hello";</code> <code>char s2[6];</code> <code>strcpy(s2, "hello");</code>
String concatenation	<code>s1 + s2</code> <code>s1 += s2</code>	<code>#include &lt;string.h&gt;</code> <code>strcat(s1, s2);</code>
Logical ops *	<code>&amp;&amp;</code> , <code>  </code> , <code>!</code>	<code>&amp;&amp;</code> , <code>  </code> , <code>!</code>
Relational ops *	<code>=</code> , <code>!=</code> , <code>&gt;</code> , <code>&lt;</code> , <code>&gt;=</code> , <code>&lt;=</code>	<code>=</code> , <code>!=</code> , <code>&gt;</code> , <code>&lt;</code> , <code>&gt;=</code> , <code>&lt;=</code>
Arithmetic ops *	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code> , unary <code>-</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code> , unary <code>-</code>
Bitwise ops	<code>&gt;&gt;</code> , <code>&lt;&lt;</code> , <code>&gt;&gt;&gt;</code> , <code>&amp;</code> , <code> </code> , <code>^</code>	<code>&gt;&gt;</code> , <code>&lt;&lt;</code> , <code>&amp;</code> , <code> </code> , <code>^</code>
Assignment ops	<code>=</code> , <code>*=</code> , <code>/=</code> , <code>+=</code> , <code>-=</code> , <code>&lt;&lt;=</code> , <code>&gt;&gt;=</code> , <code>&gt;&gt;&gt;=</code> , <code>=</code> , <code>&amp;=</code> , <code>^=</code> , <code> =</code> , <code>%=</code>	<code>=</code> , <code>*=</code> , <code>/=</code> , <code>+=</code> , <code>-=</code> , <code>&lt;&lt;=</code> , <code>&gt;&gt;=</code> , <code>=</code> , <code>&amp;=</code> , <code>^=</code> , <code> =</code> , <code>%=</code>

\* Essentially the same in the two languages



# Java vs. C: Details

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

\* Essentially the same in the two languages



# Java vs. C: Details

	Java	C
for stmt	<pre>for (int i=0; i&lt;10; i++)     statement;</pre>	<pre>int i; for (i=0; i&lt;10; i++)     statement;</pre>
while stmt *	<pre>while (i &lt; 0)     statement;</pre>	<pre>while (i &lt; 0)     statement;</pre>
do-while stmt *	<pre>do     statement; while (i &lt; 0)</pre>	<pre>do     statement; while (i &lt; 0);</pre>
continue stmt *	<pre>continue;</pre>	<pre>continue;</pre>
labeled continue stmt	<pre>continue someLabel;</pre>	<pre>/* no equivalent */</pre>
break stmt *	<pre>break;</pre>	<pre>break;</pre>
labeled break stmt	<pre>break someLabel;</pre>	<pre>/* no equivalent */</pre>

\* Essentially the same in the two languages



# Java vs. C: Details

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

\* Essentially the same in the two languages



# Example C Program

```
#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”
  - 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