EE 209: Programming Structures for Electrical Engineering

(Many Slides Borrowed from Princeton COS 217)

Goals for Today's Class

- Course overview
  - Introduction
  - Course goals
  - Resources
  - Grading
  - Policies
- Getting started with C
  - C programming language overview

Introduction

- Lecturer: KyoungSoo Park, Ph.D.
- TAs:
  - Younghwan Go (yhwan@ndsl.kaist.edu)
  - Byungsoo Kim (kevinzzang@kaist.ac.kr)
  - Asim Jamshed (ajamshed@ndsl.kaist.edu)

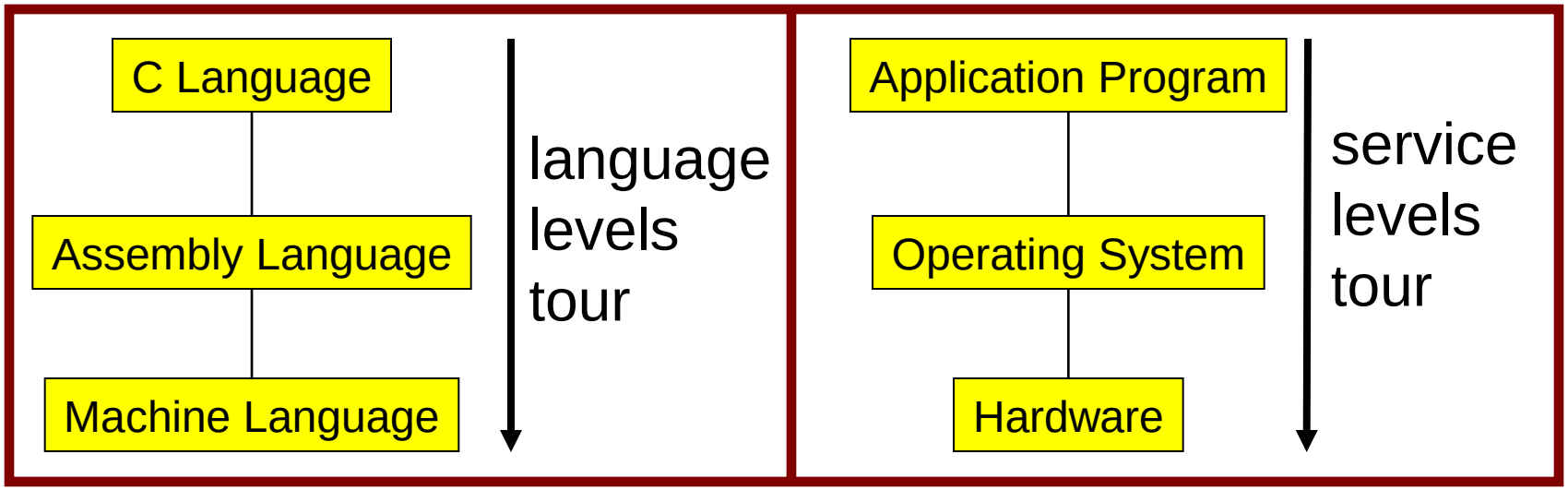
- Modeled around Princeton COS 217
  - We borrow many slides and programming assignments from Princeton COS 217
  - Got permission to use the materials

Course Goal 1: "Programming in the Large"

- Goal 1: "Programming in the large"
  - How to write large computer programs
  - Abstraction; Interfaces and implementations
- Specifically, help you learn how to:
  - Write modular code
    - Hide information
    - Manage resources
    - Handle errors
  - Write portable code
  - Test and debug your code
  - Improve your code's performance (and when to do so)
  - Use tools to support those activities

Course Goal 2: "Under the Hood"

- Goal 2: "Look under the hood"
  - Help you learn what happens "under the hood" of computer systems
- Specifically, two downward tours



- Goal 2 supports Goal 1
  - Reveals many examples of effective abstractions

Course Goals: Why C?

Q: Why C?

A: C supports Goal 1 better

- C is a lower-level language
  - C provides more opportunities to create abstractions

A: C supports Goal 2 better

- C facilitates language levels tour
  - C is closely related to assembly language
- C facilitates service levels tour
  - Linux is written in C

Course Goals: Why Linux?

Q: Why Linux instead of Microsoft Windows?

A: Linux is good for education and research

- Linux is open-source and well-specified

A: Linux is good for programming

- Linux is a variant of Unix
- Unix has a rich open-source programming environment

Lectures and Precepts

- Lectures
  - Describe concepts at a high level
  - Slides available online at course Web site
- Precepts
  - Once every week (**Wed 7-8:15pm**, this place(2220))
  - Attendance is **required**
  - Support lectures by describing concepts at a lower level
  - Support your work on assignments

Homepage and Mailing List

- Course Website
  - http://www.ndsl.kaist.edu/~kyoungsoo/ee209/
- Course mailing list (Important!)
  - ee209@list.ndsl.kaist.edu
  - Subscription is required (Did you receive my email?)
  - Q&A and announcements (e.g., cancelling class)
- KLMS (KAIST Learning Management System)
  - Linked to the course website
  - To submit your programming assignments
  - To check your score on each assignment

Textbooks

- Required books
  - *C Programming: A Modern Approach (Second Edition)*, King, 2008.
    - Covers the C programming language and standard libraries
  - *Computer Systems: A Programmer's Perspective*, Bryant and O'Hallaron, 2010.
    - Covers "under the hood"
- Highly recommended books
  - *The C Programming Language*, Kernighan and Ritchie, 1988.
    - Covers the C programming language
  - *The Practice of Programming*, Kernighan and Pike, 1999.
    - Covers "programming in the large"
  - *Programming with GNU Software*, Loukides and Oram, 1997.
    - Covers tools
- All books are in the Library

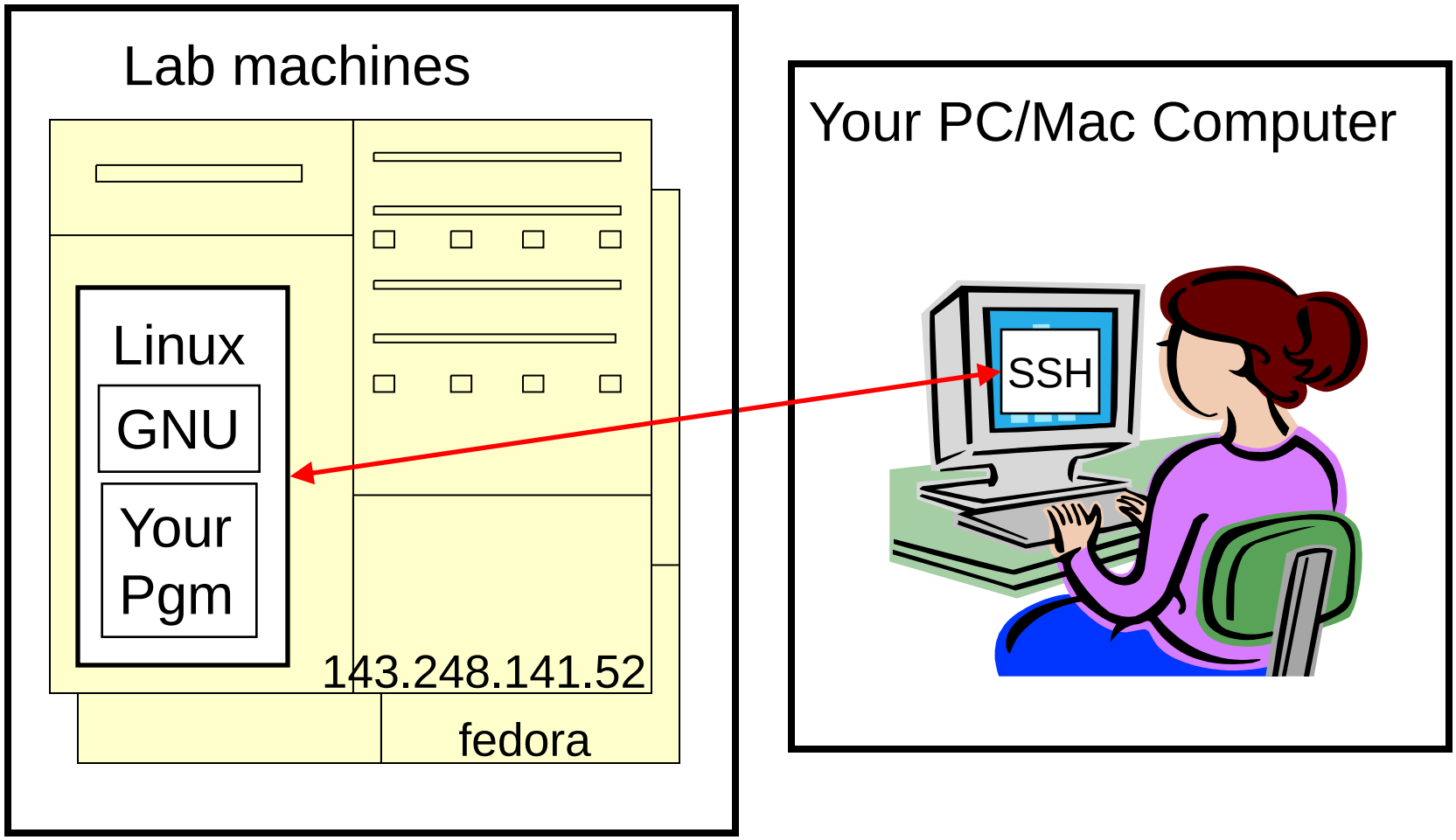
Manuals

- Manuals (for reference only, available online)
  - *Intel Architecture Software Developer's Manual, Volumes 1-3*
  - *Tool Interface Standard & Executable and Linking Format*
  - *Using as, the GNU Assembler*

- See also
  - Linux **man** command
    - **man** is short for "manual"
    - For more help, type **man man**

Programming Environment

12 Lab machines: 143.248.141.52 ~ 143.248.141.63

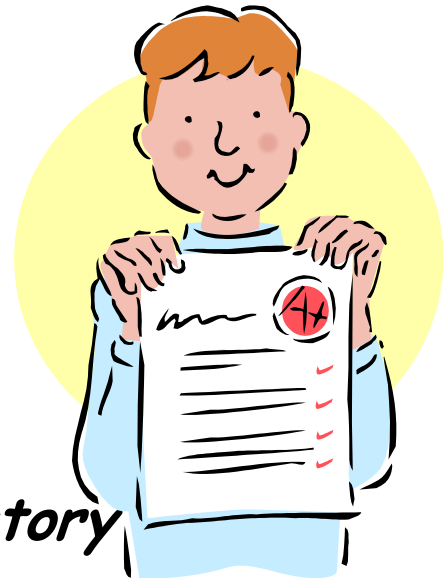


Programming Environment

- Other options
  - Use your own computer; run GNU tools run your programs locally
    - e.g., Install Linux (FYI, I use Linux as the main OS for my desktop)
    - e.g., Install Cygwin (http://www.cygwin.com/) on Windows
    - e.g., Install Linux on VMWare Player on Windows (Free)
  - Use your own computer; run a non-GNU development environment locally; run your programs locally
    - e.g., Visual C++
- Notes
  - We test your program on our Lab machines.
    - Cannot give grade if your program works on your local machine but does not run on our Lab machines.
  - **My recommendation**: use local environment for coding and lab environment for testing & debugging
  - First precept provided setup instructions

Grading

- Six programming assignments (50%)
  - Working code
  - Clean, readable, maintainable code
  - On time (penalties for late submission)
  - Final assignment counts more (12.5%)
- Exams (40%)
  - Midterm (20%)
  - Final (20%)
- Class participation (10%)
  - Lecture and precept attendance is ***mandatory***
  - Attendance + participation (+)
  - Evil(?) behavior (-)
    - e.g., moving around in class, cell phone noise, etc.
    - Unintentional(?) sleeping in class is fine, but let's not do harm to other students



Programming Assignments

- Tentative programming assignments
  1. A "de-comment" program
  2. A regular expression module
  3. A symbol table module
  4. IA-32 assembly language programs
  5. A heap manager module
  6. A Unix shell
- Key part of the course
- Due (typically) Sundays at 9:00PM
- First assignment is available now
- My advice:
  - Start early to allow time for debugging (important!!)
  - Study the class materials/books before each assignment
  - Think before you write code

Why Debugging is Necessary…

Course Policy

Study the course "Policy" web page!!!

- Especially the assignment and exam Policy
  - Violation is automatic failure (F) of this course.
  - We'll use MOSS to check plagiarism
- Some highlights:
  - Don't look at anyone else's work during, before, or after the assignment time period
  - Don't allow anyone to view your work during, before, or after the assignment time period
  - In your assignment "readme" file, acknowledge all resources used
- Ask your preceptor for clarifications if necessary

Course Schedule

- Tentatively…

Weeks	Lectures	Precepts
1-2	Intro to C (conceptual)	Intro to Linux/GNU Intro to C (mechanical)
3-7	“Pgmming in the Large”	Advanced C
8	Midterm Exam	
9-15	“Under the Hood”	Assembly Language Pgmming Assignments
16	Final Exam	

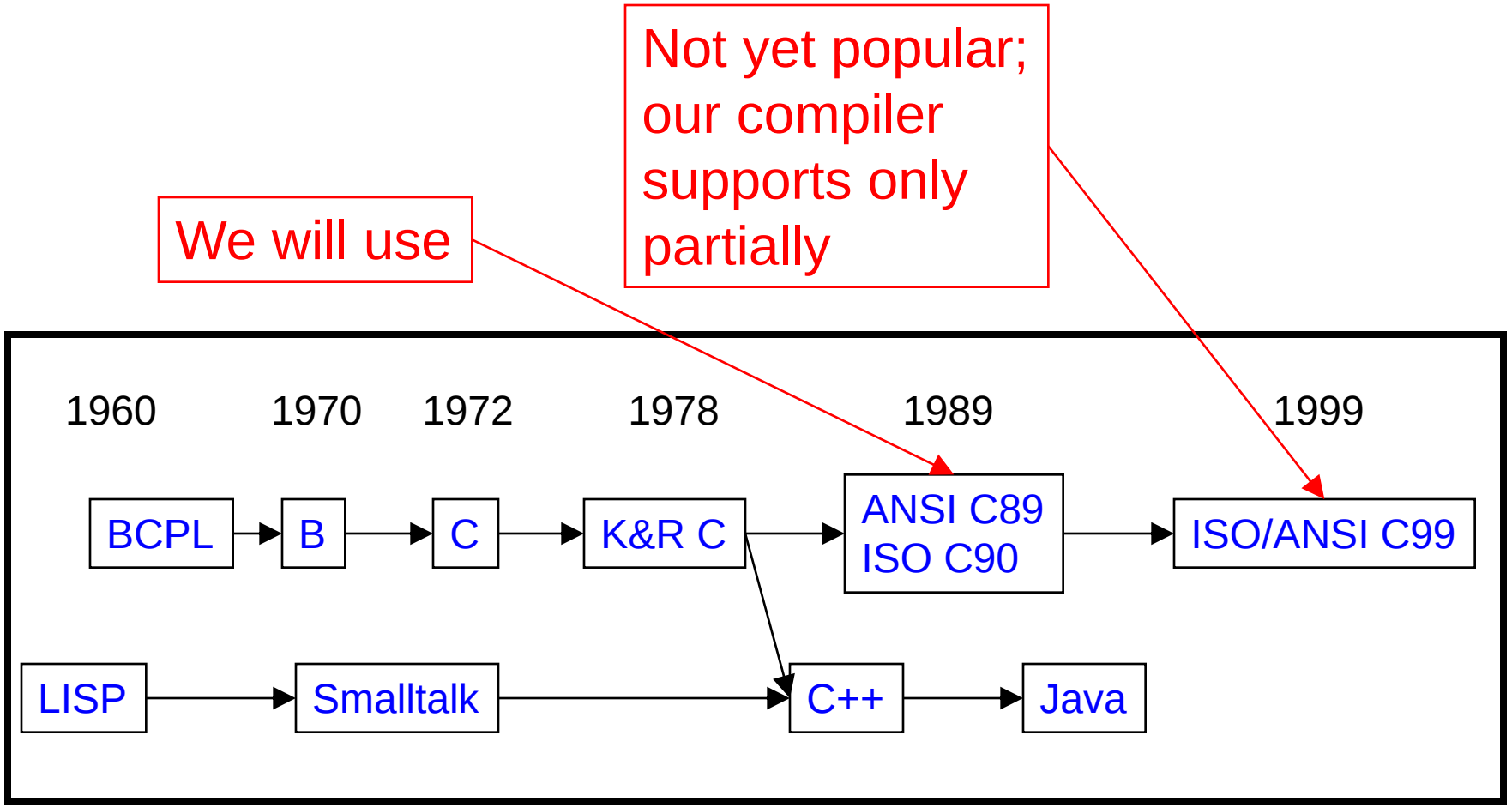
- See course “Schedule” web page for details

Getting a Good Grade for EE209

- Programming is really fun!
  - If you know how to do it
  - Often time, it is painful to reach the threshold
- Internalize the basic stuff first
  - Know the basic grammar: C types, loops, structures, arrays, strings, pointers, static functions, C runtime library functions, etc.
  - Finish the reading assignment before each class
- Allocate some time for EE209
  - 7-10 hours per week on EE209
  - Systematic approach would dramatically reduce debugging time
- Take sister class: EE205 Data Structure for EE (this semester)
  - Understanding of data structure is essential for intelligent programming

Any questions before we start?

C : History


Not yet popular; our compiler supports only partially
We will use
1960
1970
1972
1978
1989
1999
BCPL
B
C
K&R C
ANSI C89
ISO C90
ISO/ANSI C99
LISP
Smalltalk
C++
Java

C vs. Java: Design Goals

- *C* design goals
  - Support structured programming
  - Support development of the Unix OS and Unix tools
    - As Unix became popular, so did *C*
- Implications for *C*
  - Good for system-level programming
    - But often used for application-level programming
  - Low-level
    - Close to assembly language; close to machine language; close to hardware
  - Efficiency over portability
  - Efficiency over security
  - Flexibility over security

C vs. Java: Design Goals

- Java design goals
  - Support object-oriented programming
  - Allow same program to be executed on multiple operating systems
  - Support using computer networks
  - Execute code from remote sources securely
  - Adopt the good parts of other languages (esp. C and C++)
- Implications for Java
  - Good for application-level programming
  - High-level
    - Virtual machine insulates programmer from underlying assembly language, machine language, hardware
  - Portability over efficiency
  - Security over efficiency
  - Security over flexibility

C vs. Java: Design Goals

- Differences in design goals explain many differences between the languages
- C's design goal explains many of its eccentricities

  – We'll see examples throughout the course

C vs. Java: Overview

- Dennis Ritchie on the nature of C:
  - "C has always been a language that never attempts to tie a programmer down."
  - "C has always appealed to systems programmers who like the terse, concise manner in which powerful expressions can be coded."
  - "C allowed programmers to (while sacrificing portability) have direct access to many machine-level features that would otherwise require the use of assembly language."
  - "C is quirky, flawed, and an enormous success. While accidents of history surely helped, it evidently satisfied a need for a system implementation language efficient enough to displace assembly language, yet sufficiently abstract and fluent to describe algorithms and interactions in a wide variety of environments."

C vs. Java: Overview (cont.)

- Bad things you **can** do in C that you **can't** do in Java
  - Shoot yourself in the foot (safety)
  - Shoot others in the foot (security)
  - Ignore wounds (error handling)
- Dangerous things you **must** do in C that you **don't** in Java
  - Explicitly manage memory via `malloc()` and `free()`
- Good things you **can** do in C, but (more or less) **must** do in Java
  - Program using the object-oriented style
- Good things you **can't** do in C but **can** do in Java
  - Write completely portable code

C vs. Java: Details

- Remaining slides provide some details
  - Suggestion: Use for future reference

- Slides covered briefly now, as time allows…

C vs. Java: Details (cont.)

	Java	C

Overall Program Structure	**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;`
		`}`

Building	`% javac Hello.java`
	`% ls`
	`Hello.class`
	`Hello.java`
	`%`
		`% gcc209 hello.c`
		`% ls`
		`a.out`
		`hello.c`
		`%`

Running	`% java Hello`
	`Hello, world`
	`%`
		`% ./a.out`
		`Hello, world`
		`%`

C vs. Java: Details (cont.)

	Java	C
Character type	`char // 16-bit unicode`	`char /* 8 bits */`

Integral types	`byte // 8 bits`
	`short // 16 bits`
	`int // 32 bits`
	`long // 64 bits`
		`(unsigned) char`
		`(unsigned) short`
		`(unsigned) int`
		`(unsigned) long`

Floating point types	`float // 32 bits`
	`double // 64 bits`
		`float`
		`double`
		`long double`

Logical type	`boolean`	`/* no equivalent */` `/* use integral type */`
Generic pointer type	`// no equivalent`	`void*`

Constants	`final int MAX = 1000;`	`#define MAX 1000`
		`const int MAX = 1000;`
		`enum {MAX = 1000};`

C vs. Java: Details (cont.)

	Java	C

Arrays	`int [] a = new int [10];`
	`float [][] b =`
	`  new float [5][20];`
		`int a[10];`
		`float b[5][20];`

Array bound checking	`// run-time check`	`/* no run-time check */`
Pointer type	`// Object reference is an` `// implicit pointer`	`int *p;`

Record type	`class Mine {`
	`  int x;`
	`  float y;`
	`}`
		`struct Mine {`
		`  int x;`
		`  float y;`
		`}`

C vs. Java: Details (cont.)

	Java	C

Strings	`String s1 = "Hello";`
	`String s2 = new String("hello");`
		`char *s1 = "Hello";`
		`char s2[6];`
		`strcpy(s2, "hello");`

String concatenation	`s1 + s2` `s1 += s2`	`#include <string.h>` `strcat(s1, s2);`
Logical ops	`&&, ||, !`	`&&, ||, !`
Relational ops	`=, !=, >, <, >=, <=`	`=, !=, >, <, >=, <=`
Arithmetic ops	`+, -, *, /, %, unary -`	`+, -, *, /, %, unary -`
Bitwise ops	`>>, <<, >>>, &, |, ^`	`>>, <<, &, |, ^`
Assignment ops	`=, *=, /=, +=, -=, <<=, >>=, >>>=, =, ^=, |=, %=`	`=, *=, /=, +=, -=, <<=, >>=, =, ^=, |=, %=`

C vs. Java: Details (cont.)

	Java	C

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;`

C vs. Java: Details (cont.)

	Java	C

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;`	`continue;`
labeled continue stmt	`continue SomeLabel;`	`/* no equivalent */`
break stmt	`break;`	`break;`
labeled break stmt	`break SomeLabel;`	`/* no equivalent */`

C vs. Java: Details (cont.)

	Java	C
return stmt	`return 5;` `return;`	`return 5;` `return;`

Compound stmt (alias block)	`{`
	`  statement1;`
	`  statement2;`
	`}`
		`{`
		`  statement1;`
		`  statement2;`
		`}`

Exceptions	`throw, try-catch-finally`	`/* no equivalent */`
Comments	`/* comment */` `// another kind`	`/* comment */`

Method / function call	`f(x, y, z);`
	`someObject.f(x, y, z);`
	`SomeClass.f(x, y, z);`
		`f(x, y, z);`

Example C Program

```
#include <stdio.h>
#include <stdlib.h>

const double KMETERS_PER_MILE = 1.609;

int main(void)
{
   int miles;
   double kmeters;
   printf("miles: ");
   if (scanf("%d", &miles) != 1) {
      fprintf(stderr, "Error: Expect a number.\n");
      exit(EXIT_FAILURE);
   }
   kmeters = miles * KMETERS_PER_MILE;
   printf("%d miles is %f kilometers.\n",
            miles, kmeters);
   return 0;
}
```

Summary

- Course overview
  - Goals
    - Goal 1: Learn "programming in the large"
    - Goal 2: Look "under the hood"
    - Goal 2 supports Goal 1
    - Use of C and Linux supports both goals
  - Learning resources
    - Lectures, precepts, programming environment, course mailing list, textbooks
    - Course Web site: access via http://www.ndsl.kaist.edu/~kyoungsoo/ee209/

Summary

- Getting started with C
  - C was designed for system programming
    - Differences in design goals of Java and C explain many differences between the languages
    - Knowing C design goals explains many of its eccentricities
  - Knowing Java gives you a head start at learning C
    - C is not object-oriented, but many aspects are similar

Getting Started

- Check out course Web site **soon**
  - Study "Policy" page
  - First assignment is available

- Establish a reasonable computing environment **soon**
  - Instructions given in first precept