



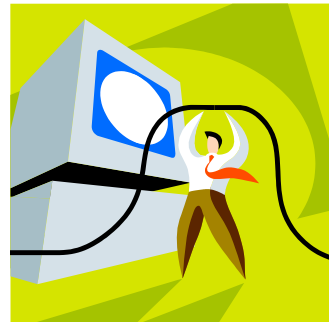
# COS 217: Introduction to Programming Systems

1

## Goals for Today's Class



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



2

# Introductions



- **Instructor-of-Record**
  - Vivek Pai, Ph.D. (Professor)
    - vivek@cs.princeton.edu
- **Preceptors (in alphabetical order)**
  - Robert Dondero, Ph.D. (Lead Preceptor)
    - rdondero@cs.princeton.edu
  - Sunha Ahn
    - sahn@princeton.edu
  - Dushyant Arora
    - dushyant@cs.princeton.edu
  - Sasha Koruga
    - skoruga@cs.princeton.edu
  - Meng Zhang
    - mengz@princeton.edu

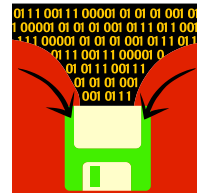


3

# Course Goal 1: “Programming in the Large”



- **Goal 1: “Programming in the large”**
  - Help you learn how to write large computer programs
- **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

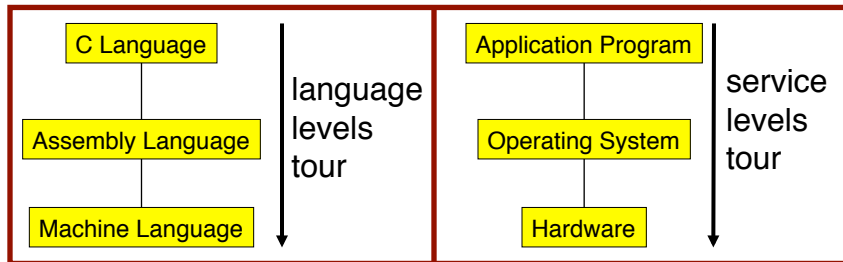


4

## 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

5

## Course Goals: Why C?



- Q: Why C instead of Java?
- A: C supports Goal 1 better
  - C is a lower-level language
    - C provides more opportunities to create abstractions
  - C has some flaws
    - C’s flaws motivate discussions of software engineering principles
- 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

6

## 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 GNU, a rich open-source programming environment

7

## Course Goals: Summary



- Help you to become a...

***Power Programmer!!!***

8

## Resources: Lectures and Precepts



- Lectures
  - Describe concepts at a high level
  - Slides available online at course Web site
  - Stronger influence on **exams**
- Precepts
  - Support lectures by describing concepts at a lower level
  - Support your work on assignments
  - Builds practically on a **subset** of information
- Important: Precepts begin **TODAY**

9

## Resources: Website and Listserv



- Website
  - Access from <http://www.cs.princeton.edu>
    - Academics → Course Schedule → COS 217
- Piazza
  - <http://piazza.com/class#spring2012/cos217/>
  - Instructions provided in first precept

10

## Resources: Books



- Required book
  - *C Programming: A Modern Approach (Second Edition)*, King, 2008.
    - Covers the C programming language and standard libraries
- Highly recommended books
  - *The Practice of Programming*, Kernighan and Pike, 1999.
    - Covers “programming in the large”
    - (Required for COS 333)
  - *Computer Systems: A Programmer's Perspective (Second Edition)*, Bryant and O'Hallaron, 2010.
    - Covers “under the hood”
    - Some key sections are on electronic reserve
    - First edition is sufficient
  - *Programming with GNU Software*, Loukides and Oram, 1997.
    - Covers tools
- *All books are on reserve in Engineering Library*

11

## Resources: Manuals



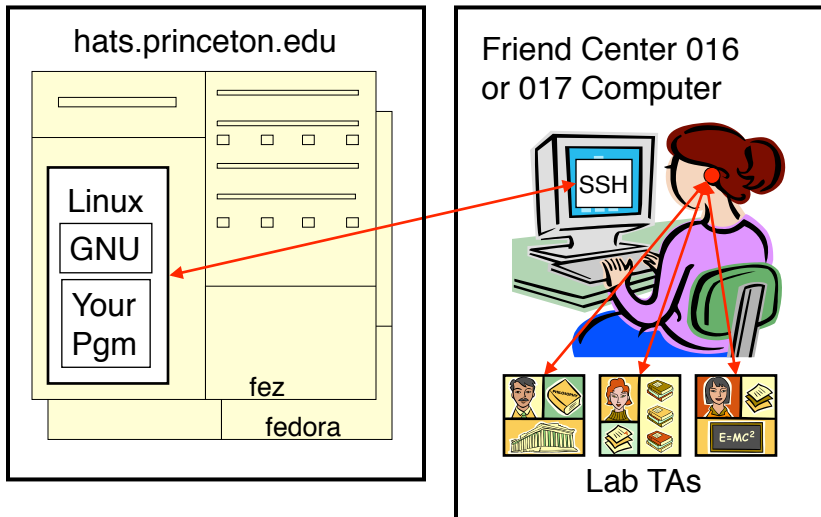
- Manuals (for reference only, available online)
  - *IA32 Intel Architecture Software Developer's Manual, Volumes 1-3*
  - *Tool Interface Standard & Executable and Linking Format*
  - *Using as, the GNU Assembler*
- See also
  - Linux **man** command
    - **man** is short for “manual”
    - For more help, type **man man**

12

## Resources: Programming Environment



### • Option 1

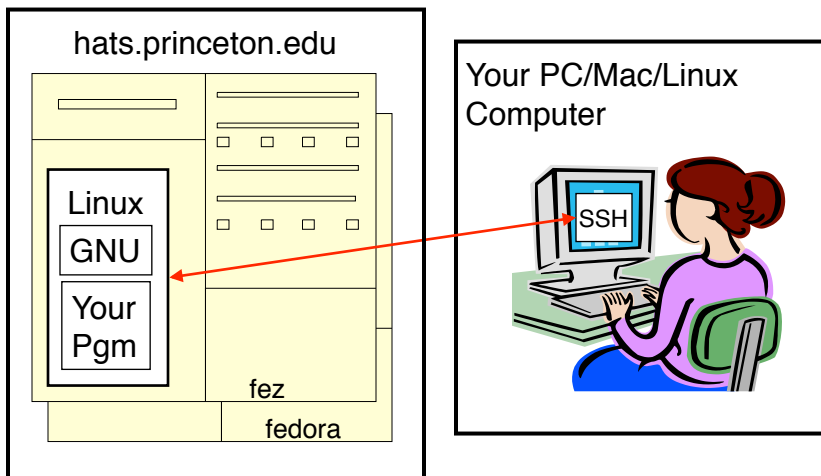


13

## Resources: Programming Environment



### • Option 2



14

## Resources: Programming Environment



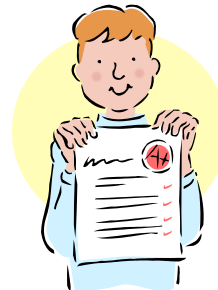
- **Other options**
  - Use your own PC/Mac/Linux computer; run GNU tools locally; run your programs locally
  - Use your own PC/Mac/Linux computer; run a non-GNU development environment locally; run your programs locally
  - Etc.
- **Notes**
  - Other options cannot be used for some assignments (esp. timing studies)
  - Instructors cannot promise support of other options
  - Strong recommendation: Use Option 1 or 2 for **all** assignments
  - First precept provides setup instructions

15

## Grading



- **Seven programming assignments (50%)**
  - Working code
  - Clean, readable, maintainable code
  - On time (penalties for late submission)
  - Final assignment counts double (12.5%)
- **Exams (40%)**
  - Midterm (15%)
  - Final (25%)
- **Class participation (10%)**
  - Lecture and precept attendance is **mandatory**



16



## Programming Assignments



- Programming assignments
  1. A “de-comment” program
  2. A string module
  3. A symbol table module
  4. IA-32 assembly language programs
  5. A buffer overrun attack
  6. A heap manager module
  7. A Unix shell
- Key part of the course
- See course “Schedule” web page for due dates/times
- **First assignment is available now**
- Advice: Start early to allow time for debugging (especially in the background while you are doing other things!)

17

## Why Debugging is Necessary...



18

## Policies



### Study the course “Policies” web page!

- Especially the assignment collaboration policies
  - Violation involves **trial by Committee on Discipline**
  - Typical penalty is **suspension from University** for 1 academic year
- Some highlights:
  - Don't view 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

19

## Course Schedule



- Very generally...

Weeks	Lectures	Precepts
1-2	Intro to C (conceptual)	Intro to Linux/GNU Intro to C (mechanical)
3-6	“Pgmning in the Large”	Advanced C
6	Midterm Exam	
7	Recess	
8-13	“Under the Hood”	Assembly Language Pgmning Assignments
	Reading Period	
	Final Exam	

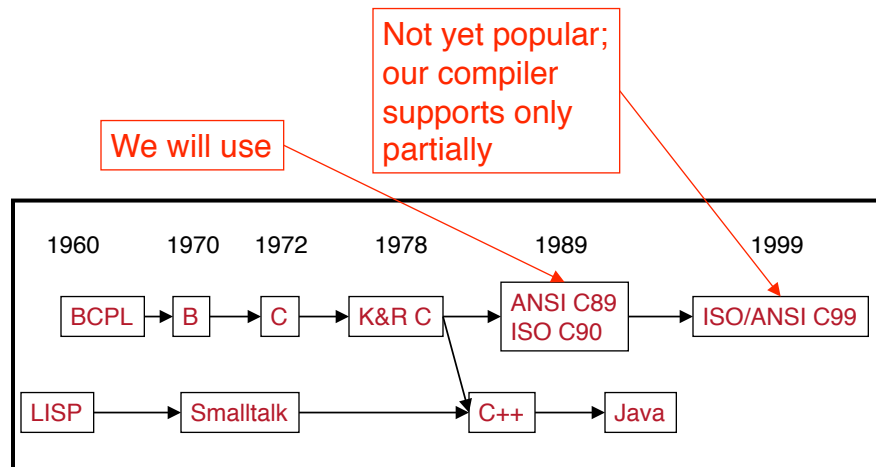
- See course “Schedule” web page for details

20



Any questions before we start?

## C vs. Java: History



## 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**

23

## 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 – sometimes inappropriately
  - **Low-level**
    - Close to assembly language; close to machine language; close to hardware
  - **Efficiency over portability**
  - **Efficiency over security**
  - **Flexibility over security**

24

## 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

25

## 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.”

26

## 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

27

## C vs. Java: Details



- Remaining slides provide some details
  - Suggestion: Use for future reference
- Slides covered briefly now, as time allows...

28

## C vs. Java: Details (cont.)



	Java	C
<b>Overall Program Structure</b>	<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>
<b>Building</b>	<pre> % javac Hello.java % ls Hello.class Hello.java %                     </pre>	<pre> % gcc217 hello.c % ls a.out hello.c %                     </pre>
<b>Running</b>	<pre> % java Hello Hello, world %                     </pre>	<pre> % a.out Hello, world %                     </pre>

29

## C vs. Java: Details (cont.)



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

30

## C vs. Java: Details (cont.)



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

31

## C vs. Java: Details (cont.)



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

32



## C vs. Java: Details (cont.)



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

33

## C vs. Java: Details (cont.)



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

## C vs. Java: Details (cont.)



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

35

## 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;
}
```

36

## 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 listserv, textbooks
    - Course Web site: access via <http://www.cs.princeton.edu>

37

## 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

38

# Getting Started



- Check out course **Web site** [soon](#)
  - Study “Policies” page
  - First assignment is available
- Establish a reasonable **computing environment** [soon](#)
  - Instructions given in first precept