# Introduction to Programming Systems

CS 217, Spring 2004

Kai Li
Princeton University





## Goals

- Master the art of programming
  - Learn how to be “good” programmers
  - Introduction to software engineering
- Learn languages for systems programming
  - C is the systems language of choice
  - Assembly is required for low-level system programming
- Introduction to computer systems
  - Machine architecture
  - Operating systems
  - Software tools



## Outline

- First three weeks
  - C programming language
- Next two weeks
  - Software engineering
- Next two weeks
  - Machine architecture
- Next two weeks
  - Software tools
- Next three weeks
  - Unix operating system services



## Coursework

- Six programming assignments (60%)
  - Un-comment filter
  - String library
  - Hash table ADT
  - IA32 assembly
  - Profiler
  - Shell
- Exams (30%)
  - Midterm
  - Final
- Class participation (10%)

# Materials

- Required textbooks
  - *C Programming: A Modern Approach,* King, 1996.
  - *The Practice of Programming,* Kernighan and Pike, 1999.
  - *Programming from the Ground Up (online)*, Bartlett 2004.
- Recommended textbooks
  - Programming with GNU Software. Loukides & Oram
- Other textbooks (on reserve)
  - IA32 Intel Architecture Software Developer's Manual (online)
  - The C Programming Language, Kernighan & Ritchie
  - C: A Reference Manual. Harbison & Steele
  - C Interfaces and Implementations. Hanson
  - The UNIX Programming Environment. Kernighan & Pike
- Web pages
  - **`www.cs.princeton.edu/courses/cs217/`**



# Facilities

- Unix machines
  - CIT's **`arizona (phoenix)`** cluster (Sparc)
  - OIT's **`hats`** cluster (Linux)
- Your own laptop
  - **`ssh`** access to **`arizona (or phoenix) and hats`**
  - run GNU tools on Windows
  - run GNU tools on Linux



# Logistics

- Lectures
  - Introduce concepts
  - Work through programming examples
    - M,W 10-10:50am CS105
- Precepts
  - Review concepts
  - Demonstrate tools (gdb, makefiles, emacs, …)
  - Work through programming examples
    - Precept 1: T,Th 12:30-1:30, room TBD
    - Precept 2: M,W 1:30-2:30, room TBD

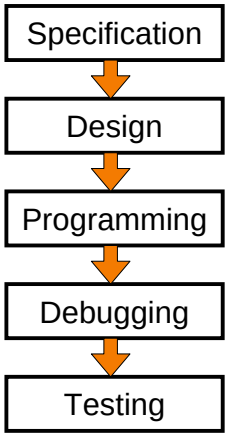


# Software is Hard

"What were the lessons I learned from so many years of intensive work on the practical problem of setting type by computer? One of the most important lessons, perhaps, is the fact that SOFTWARE IS HARD. From now on I shall have significantly greater respect for every successful software tool that I encounter. During the past decade I was surprised to learn that the writing of programs for TeX and Metafont proved to be much more difficult than all the other things I had done (like proving theorems or writing books). The creation of good software demands a significantly higher standard of accuracy than those other things do, and it requires a longer attention span than other intellectual tasks."

Donald Knuth, 1989

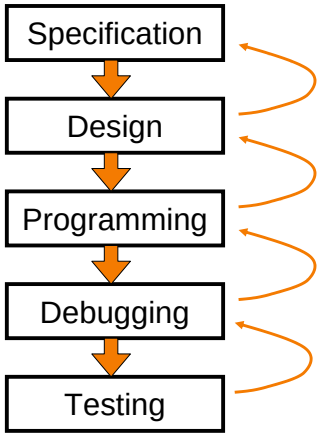
## Software in COS126




1 Person
$10^2$ Lines of Code
1 Type of Machine
0 Modifications
1 Week



## Software in the Real World




Lots of People
$10^6$ Lines of Code
Lots of Machines
Lots of Modifications
1 Decade or more



## Good Software in the Real World

- Understandable
  - Well-designed
  - Consistent
  - Documented

  ← Write code in modules with well-defined interfaces
- Robust
  - Works for any input
  - Tested

  ← Write code in modules and test them separately
- Reusable
  - Components

  ← Write code in modules that can be used elsewhere
- Efficient
  - Only matters for 1%

  ← Write code in modules and optimize the slow ones



## Good Software in the Real World

- Understandable
  - Well-designed
  - Consistent
  - Documented

  ← Write code in modules with well-defined interfaces
- Robust
  - Works for any input
  - Tested

  ← Write code in modules and test them separately
- Reusable
  - Components

  ← Write code in modules that can be used elsewhere
- Efficient
  - Only matters for 1%

  ← Write code in modules and optimize the slow ones

# The C Programming Language

- Systems programming language
  - Originally used to write Unix and Unix tools
  - Data types and control structures close to most machines
  - Now also a popular application programming language
- Notable features
  - All functions are call-by-value
  - Pointer (address) arithmetic
  - Simple scope structure
  - I/O and memory mgmt facilities provided by libraries
- History
  - BCPL (1960) → B (1970) → C (1972) → K&R C (1978) → ANSI C (1988)
  - LISP → Smalltalk → C++ → Java



# Java vs. C

| | JAVA | C |
|---|---|---|
| **Program** | **hello.java:**<br>`public class hello {`<br>`  public static void`<br>`  main(String[] args) {`<br>`  System.out.println(`<br>`  "Hello, world");`<br>`  }`<br>`}` | **hello.c:**<br>`#include <stdio.h>`<br>`main() {`<br>`  printf("Hello, world\n");`<br>`}` |
| **Compile** | `% javac hello.java`<br>`% ls`<br>`% hello.java  hello.class`<br>`%` | `% gcc hello.c`<br>`% ls`<br>`% a.out  hello.c`<br>`%` |
| **Run** | `% java hello`<br>`% Hello, world`<br>`%` | `% a.out`<br>`% Hello, world`<br>`%` |



# Java vs. C, cont'd

| | JAVA | C |
|---|---|---|
| **Boolean** | `boolean` | `int` |
| **Char type** | `char  // 16-bit unicode` | `char /* 8 bits */` |
| **Void type** | `// no equivalent` | `void` |
| **Integer types** | `byte  // 8 bits`<br>`short  // 16 bits`<br>`int  // 32 bits`<br>`long  // 64 bits` | `short`<br>`int`<br>`long` |
| **Floating point types** | `float  // 32 bits`<br>`double  // 64 bits` | `float`<br>`double` |
| **Constant** | `final int MAX = 1000;` | `#define MAX 1000` |
| **Arrays** | `int [] A = new int [10];`<br>`float [][] B =`<br>`  new float [5][20];` | `int A[10];`<br>`float B[5][20];` |
| **Bound check** | `// run-time checking` | `/* no run-time check */` |



# Java vs. C, cont'd

| | JAVA | C |
|---|---|---|
| **Pointer type** | `// no pointer` | `int *p;` |
| **Record type** | `class r {`<br>`  int x;`<br>`  float y;`<br>`}` | `struct r {`<br>`  int x;`<br>`  float y;`<br>`}` |
| **String type** | `String s1 = "Hello";`<br>`String s2 = new`<br>`  String( "hello" );` | `char *s1 = "Hello";`<br>`char s2[6];`<br>`strcpy( s2, "hello" );` |
| **String concatenate** | `s1 + s2` | `#include <string.h>`<br>`strcat( s1, s2 );` |
| **Logical** | `&&, ||, !` | `&&, ||, !` |
| **Compare** | `=, !=, >, <, >=, <=` | `=, !=, >, <, >=, <=` |
| **Arithmetic** | `+, -, *, /, %, unary -` | `+, -, *, /, %, unary -` |
| **Bit-wise ops** | `>>, <<, >>>, &, |, ^` | `>>, <<, &, |, ^` |

# Java vs. C, cont'd

| | JAVA | C |
|---|---|---|
| **Comments** | `/* comments */`<br>`// another kind` | `/* comments */` |
| **Block** | `{`<br>`  statement1;`<br>`  statement2;`<br>`}` | `{`<br>`  statement1;`<br>`  statement2;`<br>`}` |
| **Assignments** | `=, *=, /=, +=, -=, <<=, >>=, >>>=, =, ^=, \|=, %=` | `=, *=, /=, +=, -=, <<=, >>=, =, ^=, \|=, %=` |
| **Function / procedure call** | `Foo( x, y, z );` | `Foo( x, y, z );` |
| **Function return** | **`return`** `5;` | **`return`** `5;` |
| **Procedure return** | **`return`**`;` | **`return`**`;` |



# Java vs. C, cont'd

| | JAVA | C |
|---|---|---|
| **Conditional** | `if (expression)`<br>`  statement1`<br>`else`<br>`  statement2;` | `if (expression)`<br>`  statement1`<br>`else`<br>`  statement2;` |
| **Switch** | `switch (n) {`<br>`  case 1:`<br>`  ...`<br>`  break;`<br>`  case 2:`<br>`  ...`<br>`  break;`<br>`  default:`<br>`  ...`<br>`}` | `switch (n) {`<br>`  case 1:`<br>`  ...`<br>`  break;`<br>`  case 2:`<br>`  ...`<br>`  break;`<br>`  default:`<br>`  ...`<br>`}` |
| **Exception** | `Throw`<br>`try-catch-finally` | `/* no equivalent */` |



# Java vs. C, cont'd

| | JAVA | C |
|---|---|---|
| **"for" loop** | **`for`** `(int i=0;i<10;i++)`<br>`  statement;` | **`int`** `i;`<br>**`for`** `(i=0; i<10; i++)`<br>`  statement;` |
| **"while" loop** | **`while`** `(expression)`<br>`  statement;` | **`while`** `(expression)`<br>`  statement;` |
| **"do- while" loop** | **`do`** `{`<br>`  statement;`<br>`  …`<br>`}` **`while`** `(expression)` | **`do`** `{`<br>`  statement;`<br>`  …`<br>`}` **`while`** `(expression)` |
| **Terminate a loop body** | **`continue`**`;` | **`continue`**`;` |
| **Terminate a loop** | **`break`**`;` | **`break`**`;` |



# Standard I/O in C

- Three standard I/O streams
  - **`stdin`**
  - **`stdout`**
  - **`stderr`**
- Basic calls for standard I/O
  - **`int getchar(void);`**
  - **`int putchar(int c);`**
  - **`int puts(const char *s);`**
  - **`char *gets(char *s);`**
- Use "man" pages

```
% man getchar
```

copyfile.c:

```
#include <stdio.h>

main() {
    int c;
    c = getchar();
    while (c != EOF) {
        putchar(c);
        c = getchar();
    }
}
```

```
% a.out < file1 > file2
```

# Formatted Output: printf

- `int printf(char *format, ...);`
  - Translate arguments into characters according to “format”
  - Output the formatted string to stdout
- Conversions (read “man printf” for more)
  - %d – integer
  - %f – float
  - %lf – double
  - %3f – float with 3 decimal places
  - %% –percent
- Examples
  - ```
    int x = 217;
    printf( “Course number is: %d”, x );
    ```



# Formatted Input: scanf

- `int scanf(const char *format, ...);`
  - Read characters from stdin
  - Interpret them according to “format” and put them into the arguments
- Conversions (read “man scanf” for more)
  - %d – integer
  - %f – float
  - %lf – double
  - %% – literal %
- Example
  - ```
    double v;
    scanf( “%lf”, &v );
    ```
  - ```
    int day, month, year;
    scanf( “%d/%d/%d”, &month, &day, &year);
    ```



# Standard Error Handing: stderr

- **`stderr`** is the second output stream for output errors
- Some functions to use **`stderr`**
  - `int fprintf(FILE *stream, const char *format, ...);`
    - Same as `printf` except the file stream
  - `int fputc(int c, FILE *stream);`
    - `putc()` is the same as `fputc()`
  - `int fgetc(FILE *stream);`
    - `getc()` is the same as `fgetc()`
- **`Example`**
  - `fprintf( stderr, “This is an error.\n” );`
  - `fprintf( stdout, “This is correct.\n” );`
  - `printf( “This is correct.\n” );`



# Example

```
#include <stdio.h>

const float KMETERS_PER_MILE = 1.609;

int main(void) {
    int miles;
    float kmeters;

    printf(“miles: ”);
    if ( scanf(“%d”, &miles) != 1 ) {
        fprintf( stderr, “Error: Expect a number.\n”);
        exit(1);
    }
    kmeters = miles * KMETERS_PER_MILE;
    printf(“= %f kilometers.\n”, kmeters );
}
```
```

# Summary

- The goal of this course:
  - Master the art of programming
  - Learn C and assembly languages for systems programming
  - Introduction to computer systems
- It is easy to learn C by knowing Java
  - C is not object oriented, but many structures are similar
  - Standard I/O functions are quite different from Java's input and output