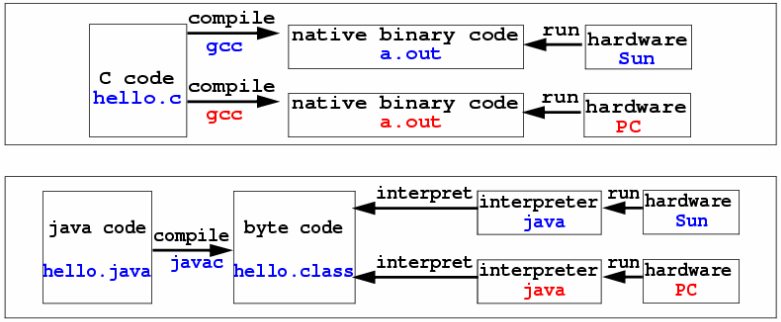
Outline

- Administrative trivia
- Goals of the class
- Introduction to C

Compiling, interpreting, and running



- Interpreter: a level of abstraction: the "virtual machine"
- The advantage of interpreting is beyond portability
- A convenient place to exercise all sorts of control
- Disadvantage: slower

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
- Pros and cons
  - Can do whatever you want: flexible and efficient
  - Can do whatever you want: can shoot yourself in the foot
- Notable features
  - All functions are call-by-value
  - Pointer (address) arithmetic
  - Simple scope structure
  - I/O and memory management 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

- Abstraction
  - C exposes the raw machine
  - Java hides a lot of it
- Bad things you **can** do in C that you **can't** do in Java
  - Shoot yourself in the foot (safety)
  - Others shoot you in the foot (security)
  - Ignoring wounds (error handling)
- Dangerous things you **have to** do in C that you **don't** in Java
  - memory management: malloc/free
- Good things that you **can** do in C but you might not; Java **makes** you
  - objected-oriented methodology
- Good things that you **can't** do in C but you **can** in Java
  - portability

Java vs. C

	JAVA	C

Program	**hello.java:**
	`public class hello {`
	`  public static void`
	`  main(String[] args) {`
	`  System.out.println(`
	`  “Hello, world”);`
	`  }`
	`}`
		hello.c:
		`#include <stdio.h>`
		`main() {`
		`  printf(“Hello, world\n”);`
		`}`

Compile	`% javac hello.java`
	`% ls`
	`% hello.java  hello.class`
	`%`
		`% gcc hello.c`
		`% ls`
		`% a.out  hello.c`
		`%`

Run	`% java hello`
	`% Hello, world`
	`%`
		`% a.out`
		`% Hello, world`
		`%`

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`
	`short  // 16 bits`
	`int  // 32 bits`
	`long  // 64 bits`
		`short`
		`int`
		`long`

Floating point types	`float  // 32 bits` `double  // 64 bits`	`float` `double`
Constant	`final int MAX = 1000;`	`#define MAX 1000`

Arrays	`int [] A = new int [10];`
	`float [][] B =`
	`  new float [5][20];`
		`int A[10];`
		`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 {`
	`  int x;`
	`  float y;`
	`}`
		`struct r {`
		`  int x;`
		`  float y;`
		`}`

String type	`String s1 = “Hello”;`
	`String s2 = new`
	`  String( “hello” );`
		`char *s1 = “Hello”;`
		`char s2[6];`
		`strcpy( s2, “hello” );`

String concatenate	`s1 + s2`	`#include <string.h>` `strcat( s1, s2 );`
Logical	`&&, ||, !`	`&&, ||, !`
Compare	`=, !=, >, <, >=, <=`	`=, !=, >, <, >=, <=`
Arithmetic	`+, -, *, /, %, unary -`	`+, -, *, /, %, unary -`
Bit-wise ops	`>>, <<, >>>, &, |, ^`	`>>, <<, &, |, ^`

Java vs. C, cont’d

	JAVA	C
Comments	`/* comments */` `// another kind`	`/* comments */`

Block	`{`
	`  statement1;`
	`  statement2;`
	`}`
		`{`
		`  statement1;`
		`  statement2;`
		`}`

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)`
	`  statement1`
	`else`
	`  statement2;`
		`if (expression)`
		`  statement1`
		`else`
		`  statement2;`

Switch	`switch (n) {`
	`  case 1:`
	`  ...`
	`  break;`
	`  case 2:`
	`  ...`
	`  break;`
	`  default:`
	`  ...`
	`}`
		`switch (n) {`
		`  case 1:`
		`  ...`
		`  break;`
		`  case 2:`
		`  ...`
		`  break;`
		`  default:`
		`  ...`
		`}`

"goto"	`// no equivalent`	`goto L;`
Exception	`throw, try-catch-finally`	`/* no equivalent */`

Java vs. C, cont'd

	JAVA	C

"for" loop	`for (int i=0;i<10;i++)`
	`  statement;`
		`int i;`
		`for (i=0; i<10; i++)`
		`  statement;`

"while" loop	`while (expression)` `  statement;`	`while (expression)` `  statement;`

"do- while" loop	`do {`
	`  statement;`
	`  …`
	`} while (expression)`
		`do {`
		`  statement;`
		`  …`
		`} while (expression)`

Terminate a loop body	`continue;`	`continue;`
Terminate a loop	`break;`	`break;`

Standard I/O

- Three standard I/O streams
  - **stdin**
  - **stdout**
  - **Stderr**
- **Binding**
  - **Flexible/dynamic binding of streams to actual devices or files**
  - **Default binding**
    - **stdin bound to keyboard**
    - **stdout and stderr bound to the terminal screen**

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`

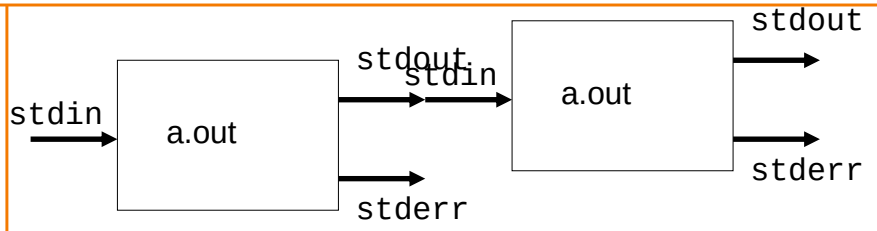
copyfile.c:

```
#include <stdio.h>

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

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

pipe

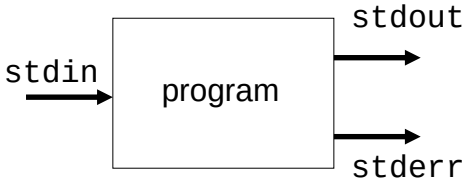



```
% a.out < file1 | a.out > file2
```


What's all this good for?

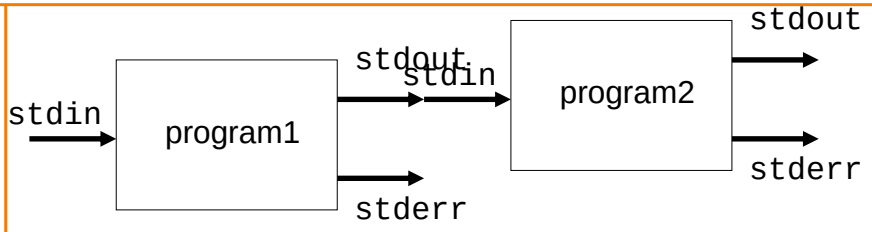
- In the old days: hard-code input/output devices into programs
- Hard to program
- and very hard to port to different input/output devices

What's all this good for?



- Along came Unix (early 1970s)
- First OS to have complete features of standard I/O redirection and pipes
- Standard I/O redirection
  - Write program once
  - Same program can be made to work for different input/output devices at run time
- Good practice of modularity

What's all this good for?



- Pipes
  - Write small programs that specialize in very simple tasks
  - Connect lots of smaller programs to make bigger programs
  - Makes bigger programs easier to write
  - Earliest and best success story of programming with components
- Standard I/O redirection and pipes: big part of Unix success
- Good practice of modularity is a learned art

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