Week 1 - Friday

CS222

Last time

- What did we talk about last time?
- Course overview
- Policies
- Schedule
- History of C, Unix, and Linux

Questions?

Quotes

In place of a dark lord you would have a queen! Beautiful and terrible as the dawn, treacherous as C! Stronger than the foundations of the earth! All shall love me and despair!

Galadriel
from *The Lord of the Rings* by J. R. R. Tolkien
edited by Dennis Brylow

Types in C

- Basic types in C are similar to those in Java, but there are fewer

Type	Meaning	Size
`char`	Smallest addressable chunk of memory	Usually 1 byte
`short`	Short signed integer type	At least 2 bytes
`int`	Signed integer type	At least 2 bytes, usually 4 bytes
`long`	Long signed integer type	At least 4 bytes
`float`	Single precision floating point type	Usually 4 bytes
`double`	Double precision floating point type	Usually 8 bytes

- No built-in boolean type!

But, wait, it gets worse…

- Unlike Java, C has signed and unsigned versions of all of its integer types
  - Perhaps even worse, there's more than one way to specify their names

Type	Equivalent Types
`char`	`signed char`
`unsigned char`	

`short`	`signed short`
	`short int`
	`signed short int`

`unsigned short`	`unsigned short int`
`int`	`signed int`
`unsigned int`	`unsigned`

`long`	`signed long`
	`long int`
	`signed long int`

`unsigned long`	`unsigned long int`

And yet again worse than that…

- There are also types that are officially supported in C99 but may or may not be supported by compilers in C89

Type	Meaning	Size
`long long`	Very long signed integer type	At least 8 bytes
`long double`	Extended precision floating point type	Usually 10 bytes or 16 bytes

- Naturally, a **`long long`** can also be written as a **`long long int`**, a **`signed long long int`** and has siblings **`unsigned long long`** and **`unsigned long long int`**

Derived types

- From these basic types, a number of types can be derived
- Structs
  - Collections of a fixed set of named items
  - Similar to a class with no methods and all public members
- Unions
  - A set of possible items, but only one of them is stored at a time
  - Used to conserve memory (but hard to program with)
- Arrays
  - Lists of items of with the same type
  - Can be indexed with integers
- Pointers
  - Types that point at other variables
  - Contain addresses
  - Pointer arithmetic is allowed, meaning that you can point at a variable, and then see what value exists 38 bytes later in memory

File organization

- In Java, all code and data is in a class
  - The class can optionally be in a package
  - The name of the class must match the name of the file it's in
- In C, every file is a list of functions and global variables
  - That's it
  - No classes, no requirements for naming anything any particular way
  - To use other files, you use the **`#include`** directive which literally copies and pastes those files into the code being compiled

Low level language

- You get operators for:
  - Basic math
  - Bitwise operations
  - Pointer manipulation
- There are no built-in operators or language features for composite data
  - No way to deal with strings, arrays, lists, sets, etc.
  - Instead of having language features for these things, C has a standard library that helps with some of these tasks

Other features

- It's a small language
  - You can expect to use all of it regularly
- I/O is painful and library driven
  - Like Java, unlike Pascal
- There's no garbage collection
  - In Java, create as many objects as you want with the **`new`** keyword and they will magically disappear when you no longer need them
  - In C, you can allocate chunks of memory using the **`malloc()`** function, but then you have to destroy them yourself using **`free()`**
- **Remember:** Java was designed, C was implemented

Why study C?

- Automotive mechanic vs. automotive engineer
  - Coding Java is like being a mechanic (though perhaps a fantastic one)
  - You're building applications out of nice building blocks
  - Coding C allows you to become an engineer
  - The JVM itself was written in C and C++
- Many parts of OS's, performance critical systems, virtual machines, and most embedded code is written in C

C's success

- It's close to what's actually happening in the machine
  - Fast and predictable
- It's sort of like Latin
  - Informs English, French, Italian, Spanish, etc.
  - The language of classical literature, church history, scientific nomenclature

> *You can argue about which language is best; C does not care, because it still rules the world.*
>
> Dennis Brylow

Hello, World

- The standard Hello World program is simpler in C, since no surrounding class is needed

```
#include <stdio.h>

int main()
{
    printf("Hello, World!");
    return 0;
}
```

Includes

- Libraries written by other people (and eventually code you've written yourself) can be used in your program using the **`#include`** directive
  - Always include header files (**`.h`** extension)
  - **`stdio.h`** is the header for basic input and output methods
- Standard libraries are specified in angle brackets: **`<stdio.h>`**
- Local files are specified in quotes: **`"mycode.h"`**
- It is legal to put **`#include`** directives anywhere in the code, but it is good style to put them at the top

`main()` function

- Executable code in C is inside of **functions**
  - Functions are similar to methods in Java
  - Think of them as static methods, since none of them are in an object
- Execution starts at the **`main()`** function
- Traditionally, the **`main()`** function has the **`int`** return type and returns `0` at the end
  - A value of `0` tells the OS that the program exited without error
  - Some people prefer a **`main()`** with **`void`** as its return type

`printf()` function

- The **`printf()`** function is the classic console output function in C
- It always prints out a string
- The string can have special control characters inside of it that are used to print numbers or other strings, all with specified formatting
- Any number of arguments can be given after the initial string, provided that there is a format specifier for each one

```
printf("%d fish, %f fish", 1, 2.0);
printf("%s in socks", "fox");
```

Format specifiers

- These specifiers can be used in a **`printf()`** format string
- They are preceded by a percent sign (`%`)
- You can also specify a minimum width (after the `%`) and a specific precision (after a **`.`** and before the specifier)

Specifier	Output
`d`, **`i`**	Integer
`u`	Unsigned integer
`f`	Floating point number
`e`	Floating-point number with exponent
`g`	Floating-point number in standard or scientific notation depending on size
`x`	Unsigned integer in hexadecimal
`o`	Unsigned integer in octal
`s`	Null-terminated string
`c`	Character

```
printf("You owe me $%.2f in cash!", 50.0/3);
```

Declaration syntax

- Another gotcha!
- Can't declare a variable in the header of a **`for`** loop
- Doesn't work:

```
for( int i = 0; i < 100; i++ )
{
    printf("%d ", i);
}
```

- You have to declare **`int i`** **before** the loop

Text editors

- You're used to using Eclipse for editing all your code
- In the Linux world, compilers are often separate from editors
- You can pick whichever text editor you like
- Ubuntu always provides **`gedit`**
- **`vim`** and **`emacs`** are two editors that run from the command line and do not require a GUI
  - They take some getting used to but are very powerful

Navigating with the command line

- Click on the Ubuntu logo in the upper left and type in "terminal"
- A command line will open up
- Type **`ls`** to list the current directory contents
- Type **`cd`** to change to another directory
  - **`cd ..`** changes to the parent directory

```
> cd stuff
> |
```

Compiling

- Navigate to whichever directory you saved your `.c` file
- Type **`gcc`** followed by the name of the file

```
> gcc hello.c
```

- By default, the executable will be called **`a.out`**
- To run your code type **`./a.out`**
  - The `./` specifies the current directory

```
> ./a.out
```

Credits

- Much of the structure and content of these lectures is based on lecture notes from Dennis Brylow from his version of CS240 taught at Purdue University

Java compilation model

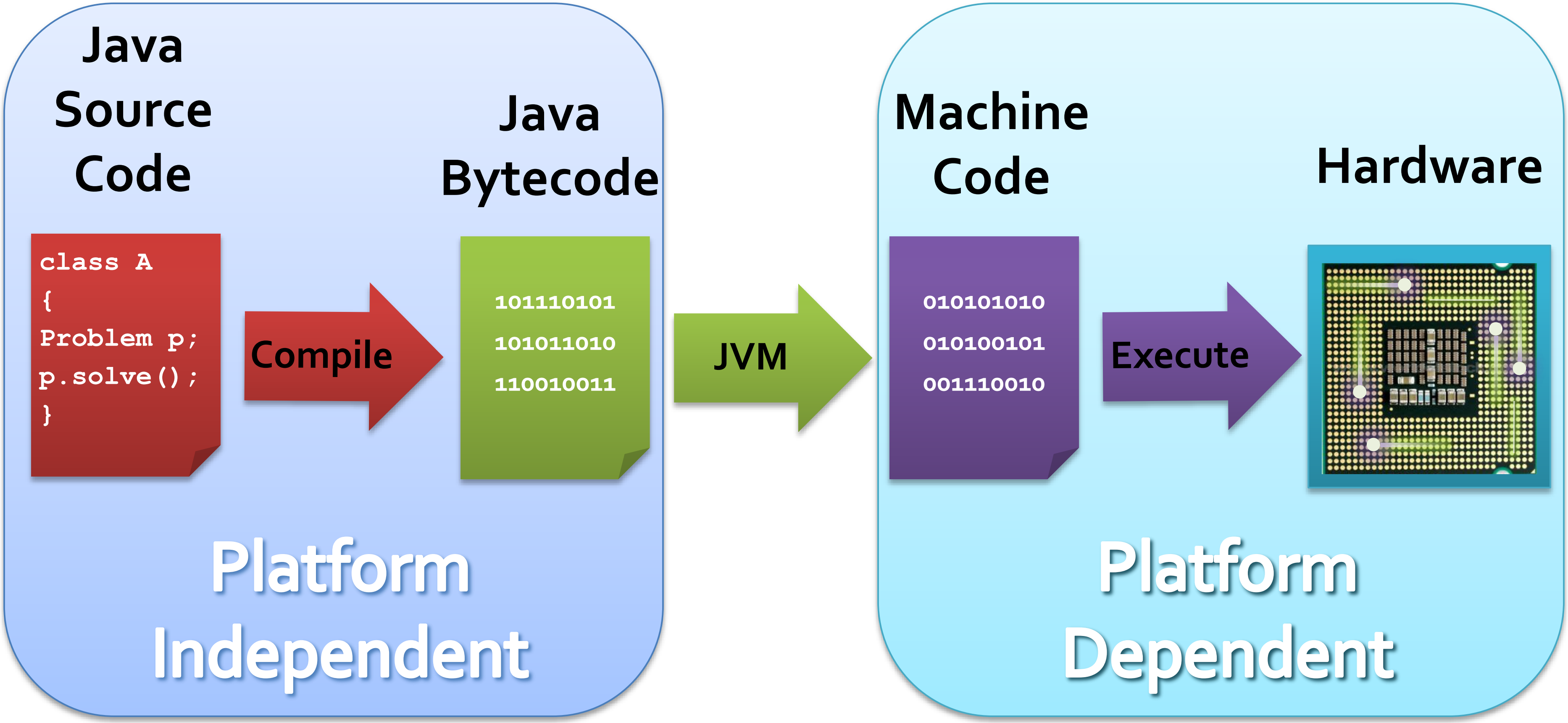
- You might not have thought too closely about this when using Eclipse
- When you compile Java from the command line, it looks like the following:

```
> javac Hello.java
```

- Doing so creates `.class` files
- You run a `.class` file by invoking the JVM

```
> java Hello
```

Compilation and execution for Java



Java Source Code

```
class A
{
Problem p;
p.solve();
}
```

Compile

Java Bytecode

101110101
101011010
110010011

JVM

Machine Code

010101010
010100101
001110010

Execute

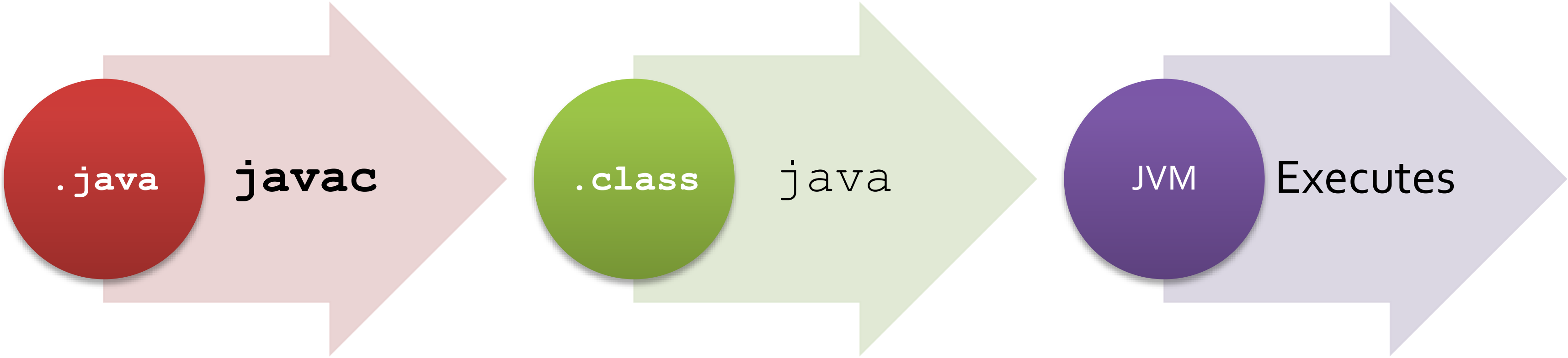
Hardware

Platform Independent

Platform Dependent

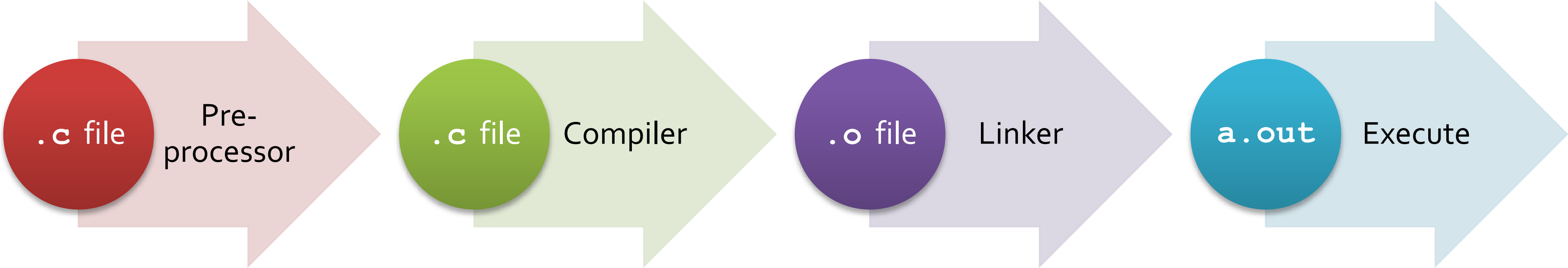
Java compilation details

- When you invoke the JVM, you specify which class you want to start with
  - If many classes in the same directory have a **`main()`** method, it doesn't matter
  - It starts the **`main()`** for the class you pick
- Java is smart
  - If you try to compile **`A.java`**, which depends on **`B.java`** and **`C.java`**, it will find those files and compile them too



C compilation model

- When you invoke **`gcc`**
  - It takes a **`.c`** file, preprocesses it to resolve **`#include`** and **`#define`** directives
  - The updated **`.c`** file is compiled into a **`.o`** object file
  - If needed, the linker links together multiple **`.o`** files into a single executable



C compilation details

- The C compiler is bare bones
- It doesn't include any other files that you might need
- You have to include and compile files in the right order
- What happens if file **`thing1.c`** wants to use functions from **`thing2.c`** and **`thing2.c`** also wants to use functions from **`thing1.c`**?
  - Which do you compile first?
  - Header files for each will eventually be the answer

Makefiles

- The order of compilation matters
- You have to compile all necessary files yourself to make your program work
- To make these issues easier to deal with, the **`make`** utility is used
- This utility uses makefiles
  - Each makefile has a list of targets
  - Each target is followed by a colon and a list of dependencies
  - After the list of dependencies, on a new line, preceded by a **tab**, is the command needed to create the target from the dependencies

Sample makefile

- Makefiles are called **`makefile`** or **`Makefile`**

```
all:    hello

hello: hello.c
    gcc -o hello hello.c

clean:
    rm -f *.o hello
```

Lab 1

Upcoming

Next time…

- More C basics
- Math library
- Data representation

Reminders

- Review the notes
- Play around with a C compiler if you can