Week 1 - Wednesday

COMP 2400

Last time

- What did we talk about last time?
- Course overview
- Policies
- Schedule
- C basics

Questions?

Project 1

C Basics

Control flow

- You're already a better C programmer than you think you are!
- For selection, C supports:
  - `if` statements
  - `switch` statements
- For repetition, C supports:
  - `for` loops
  - `while` loops
  - `do-while` loops
- Try to implement code the way you would in Java and see what happens …

Conditionals

- One big difference from Java is that C uses integer values for conditions
  - 0 (zero) is false
  - Anything non-zero is true

```
if (6)              if (0)               if (3 < 4)
{                   {                    {
    // Yep!             // Nope!             // Yep!
}                   }                    }
```

Type safety

- Java is a **strongly-typed** language
  - Types really mean something
- C is much looser

```
double a = 3.4;
int b = 27;
a = b; // Legal in Java and C
b = a; // Illegal in Java,
       // might give a warning in C
```

Precision

- The C standard makes floating-point precision compiler dependent
- Even so, it will usually work just like in Java
- Just a reminder about the odd floating-point problems you can have:

```
#include <stdio.h>

int main()
{
      float a = 4.0 / 3.0;
      float b = a - 1;
      float c = b + b + b;
      float d = c - 1;
      printf("%e\n", d);
      return 0;
}
```

- Just like in Java, you almost always want to use **`double`** to store floating-point values, since it has more precision than **`float`**

Java compilation model

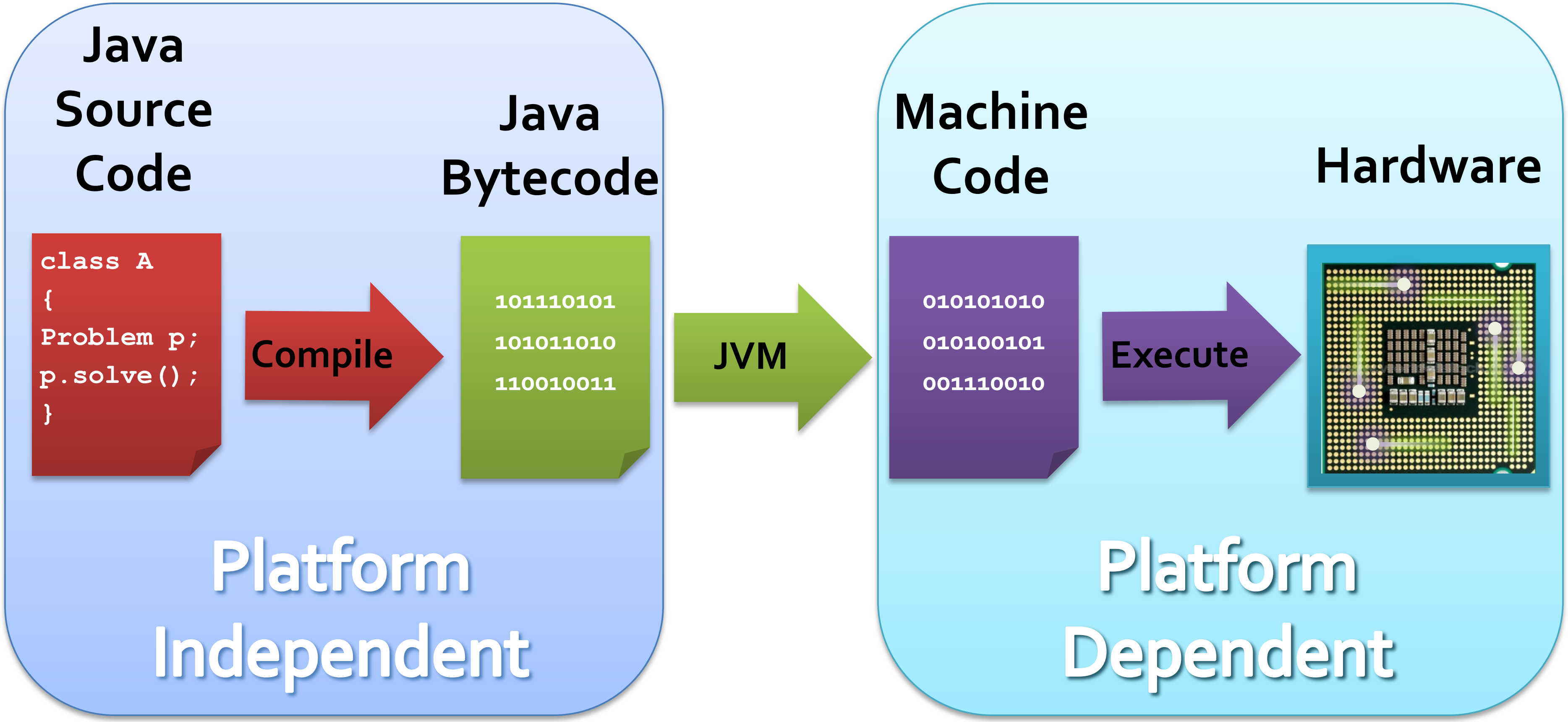
- You might not have thought too closely about this when using IntelliJ
- When you compile Java from the command line, it looks like the following:

```
> 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

Platform Independent

JVM

Machine Code

010101010
010100101
001110010

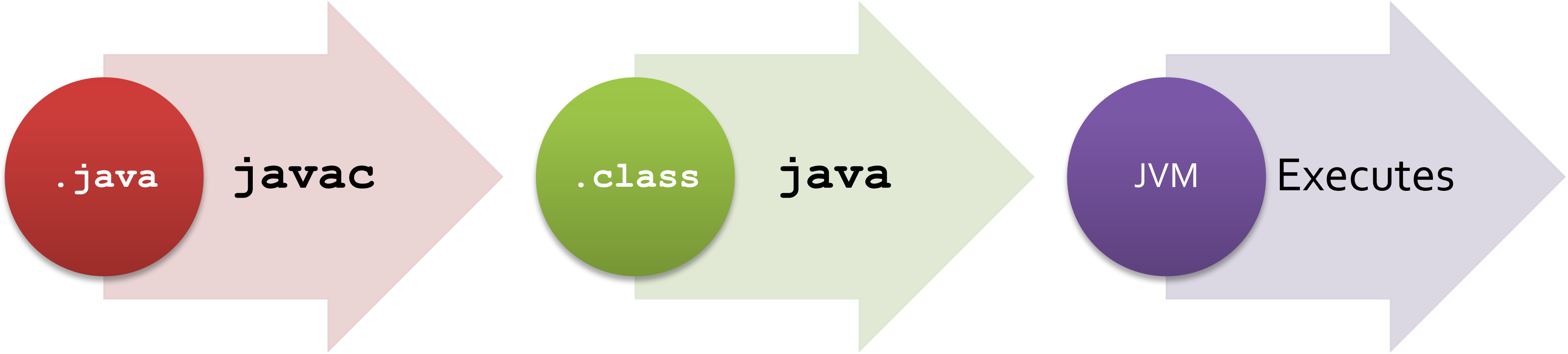
Execute

Hardware

Platform Dependent

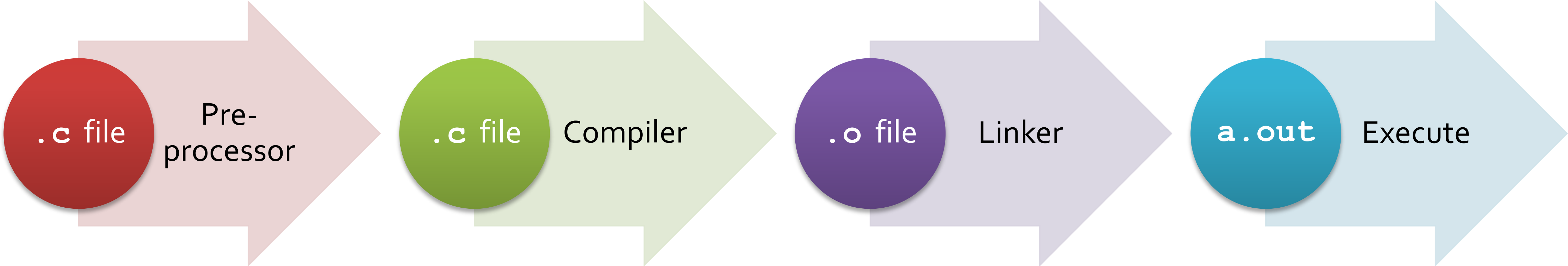
Java compilation details

- 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

Basic compilation

- To compile a file called **`hello.c`** into an executable called **`hello`**

```
> gcc hello.c -o hello
```

- To run **`hello`**, type **`./hello`**

```
> ./hello
```

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
```

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 OSes, performance critical systems, virtual machines, and most embedded code is still 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

Declaration syntax standards

- You couldn't declare a variable in the header of a **`for`** loop in C89
- The following line of code *used* to cause a compiler error:

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

- The version of **`gcc`** in this lab uses the C99 standard by default, which allows it
- For fully compliant C89 compilers, you actually have to declare **all** of your variables at the top of a block
- These older versions shouldn't be an issue, but you never know when you might have to use an older compiler for an older system

C standards

- Most programming languages have multiple versions
  - C is no exception
- The original, unstandardized version of the language used at Bell Labs from 1969 onward is sometimes called K&R C
  - It's similar to what we use now but allowed weird function definition syntax and didn't have function prototypes
- Most of what we talk about is ANSI C89 which is virtually identical to ISO C90
- We'll use a few features from C99
  - Declaring variables anywhere (including in **`for`**-loop headers)
  - Single-line comments (originally, only **`/* comment */`** was allowed)
  - **`stdbool.h`**
  - I encourage you *not* to use variable-length arrays, since they mostly cause trouble
- There's even a C11 (2011) standard, but it doesn't add anything we care about

History of Unix, Linux, and C

What does UNIX even mean?

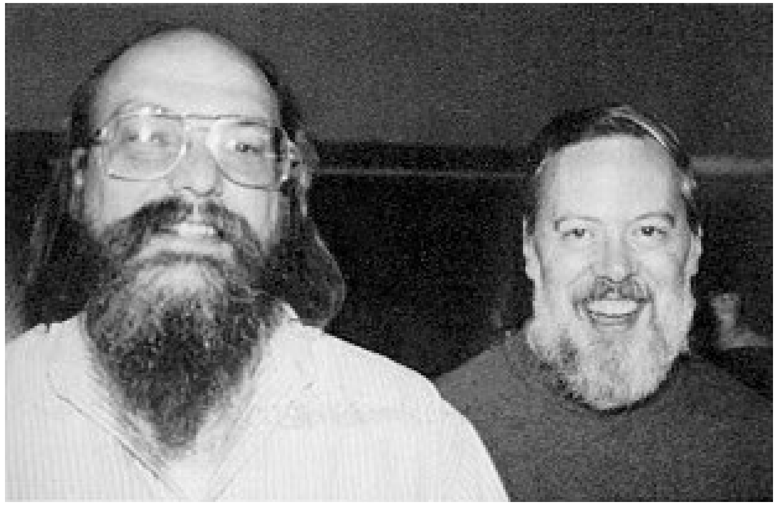
- It was originally called **Unics** (**UN**iplexed **I**nformation and **C**omputing **S**ervice)
  - A pun on another OS, Multics (**MULT**iplexed **I**nformation and **C**omputer **S**ervices)
  - After it starting supporting multiple simultaneous users, it was renamed Unix
- So, it doesn't stand for anything anymore (sort of like CERN)

What is Unix?

- It's a standard for operating systems based on a long, complex history with many companies and innovators
- The Open Group has the trademark on the term "UNIX," and you're only allowed to call your OS Unix if it meets their Single UNIX Specification
- Linux and FreeBSD and other free implementations of Unix do **not** meet this specification

Development

- Ken Thompson started working on Unix in 1969 at Bell Laboratories, a division of AT&T
- It was written in assembly language for the PDP-7 and PDP-11 minicomputers
  - Made by Digital Equipment Corporation (DEC), a giant of that era that was bought by Compaq (which was bought by HP)
- Meanwhile, Dennis Ritchie developed the C programming language
- It was mature enough in 1973 that most of Unix could be implemented in it
- This connection has established C as the pre-eminent systems programming language



Distribution

- Unix was originally only used within AT&T
- Because AT&T has a monopoly on telephone service, they weren't allowed to sell software
- They started giving Unix to universities for a distribution fee
- While spending a year at Berkeley, Thompson worked on BSD (Berkeley Software Distribution), a version of Unix that was widely used in academia
- AT&T's monopoly broke up, allowing them to sell Unix, eventually leading to the famous System V Unix in 1983

Ports

- System V was used as the basis of Unix systems on lots of different kinds of hardware
  - Sun: SunOS and Solaris
  - DEC: Ultrix and OSF/1 (which became HP Tru64 UNIX)
  - IBM: AIX
  - HP: HP-UX,
  - Apple: NeXTStep, A/UX
  - Intel: XENIX

GNU

- Richard Stallman (RMS) is the father of open source software
- He started in the GNU (GNU's Not Unix) project in 1984
  - This created the GPL (GNU Public License)
- The focus is on the ability to run, copy, and improve software
- Lots of useful programming tools that have been incorporated into Linux came out of GNU:
  - **`emacs`**
  - **`gcc`**
  - **`bash`**
  - The glibc

Linux

- Linus Torvalds started working in 1991 to make a Unix kernel to run on an Intel 386
- He put Linus's Unix (Linux) under the GNU GPL
- The BSD distributions also gave rise to free BSD implementations (notably FreeBSD), but their usage is much less widespread than Linux
- Linux kernel version numbers are ***x*.*y*.*z*** where ***x*** is a major version, ***y*** is a minor version, and ***z*** is a minor revision
  - Current stable release is 6.12.9

Distributions

- Linux is just the **kernel**, the part of the OS that manages resources and schedules processes
- To put Linux on your computer, you need a **distribution**
- A distribution includes a whole OS:
  - The kernel (of course)
  - Windowing system
    - GNOME
    - KDE
  - Package management
    - dpkg
    - APT
    - pacman
    - rpm
    - YUM
  - Tools and utilities
  - Hardware drivers
- Distributions can be big or small and are often customized for a particular purpose
  - Desktop
  - Server
  - Phone
  - Embedded software

Popular distributions

Family	Distribution	Notes
Debian	Debian	Stable but slow release cycle
	Ubuntu	▪ Managed by for-profit company Canonical ▪ Has long-term-support (LTS) versions
	Mint	Based on Ubuntu
	MX	Good driver support
Arch	Arch	Rolling release and highly customizable
Gentoo	Gentoo	Very customizable but hard for beginners to set up
Slackware	Slackware	Stable and been around forever
Red Hat	Red Hat Enterprise	For-profit, with commercial support
	Fedora	Open-source version of Red Hat
SUSE	openSUSE	Good installer, weak performance

Details from https://distrowatch.com/

Upcoming

Next time…

- More C basics
- Math library
- Data representation

Reminders

- Lab 1 is tomorrow
- Keep reading K&R Chapter 1