ALGONQUIN COLLEGE

MODULE 1 :
C - AN OVERVIEW

Professor : Dave Houtman

Office: T323

Office Hrs: Monday 15:30 – 16:00

Wednesday 15:30 – 16:00

Friday 15:30 – 16:00

Email: houtmad@algonquincollege.com

A Note About the Slides

- The ⌘ symbol in the top right corner of a slide means that the information on the slide is provided for context only – you won't be tested on this *specific* information. Use this when studying for exams to determine whether or not you can ignore the *details* of the slide.
- Note that, while you will not be *directly* tested on the material contained on these slides, this material is often essential to your overall understanding of the information that follows. So while you *don't need to memorize the specific contents of these slides* in preparation for an exam, you're nonetheless responsible for understanding how the material fits into the overall context of the course. You can skip the *details* of such slides; that doesn't mean you can skip these slides altogether.

A Note About the Slides

- The first time a term is used in a slide it will usually be highlighted in ***dark blue text***. This also includes terms which you should have seen in previous courses, in particular your Java courses (e.g. ***class***, ***abstraction***, ***overloading***, etc.) or sometimes, those which you'll be seeing in your upcoming C++ course. Terminology which you should already have encountered will *not* be defined in the slides, on the assumption that you already understand them. But if you have any questions, ask for clarification. Regardless of whether you've seen a term before or not, if its related to this course, then you're responsible for knowing what it means on a test, as well as knowing how to use it intelligently in conversation—or during a job interview.

A Note About the Slides

Java is a C-based language; C existed for 20 years before Java was invented, and almost all of Java's syntax and structure—except for the OOP parts—comes directly from C. However, Java's designers made certain decisions about the language which are not fully consistent with proper C usage. These differences, when they occur, will be indicated in red. For example:

In C, the && symbol corresponds to the AND operation on Boolean values, whereas & signals a bitwise AND.

(Note: In Java, && and & act on Boolean operands in almost exactly the same way, except that that && ***short-circuits***, while & does not. In C, you don't have a choice: Boolean operators in C, like && and ||, *always* short-circuit the evaluation of their operands.)

A Note About the Slides

Additionally, because of the relationship between C and C++, differences between these languages will be highlighted in red typeface as well, since even though you don't know C++ yet, its good to know where the differences lie. Most of these slides will contain the ⌘ symbol in the top right corner, since you're not going to be tested on this material on exams.

A Note About the Slides

- In order to explain why code works, it is sometimes necessary to provide examples of code that is contains bugs, and is therefore explicitly designed *not to work*. The following symbol is used to prevent you from accidentally referencing these deliberately-faulty code fragments:

Potential naming conflict

—where the words under the 'X' indicate the source of the error. When an unintended error is *possible*, the following symbol is used to caution you against using the code without understanding the nature of the *potential* bug:

Possible unintended rounding error

A Note About the Slides

- Because Kernighan and Ritchie's book is (1) a classic, and (2) the textbook for this course, I'll include references to relevant pages from K&R throughout the slides, to help you locate the material. Whenever you see the following:

K&R 123

this indicates the page in K&R to search out further information on the subject under discussion.

Additionally, I frequently borrow code from other sources (which may be harder to track down). The source of the code will be footnoted at the bottom of the slide in which the material first appears, such as (a favorite):

A Book on C, A. Kelley, I. Pohl, Addison Wesley, 1998, pg. 507

A Note About the Slides

- Code and keywords in C are indicated in `Courier New`
- When there's something in a code fragment or sentence that you need to focus on, like a new function, command, etc., it is usually written in **boldface** type, to help distinguish it from the rest of the writing.
- It is frequently necessary to shorten code to fit it on a slide. For convenience, I use the ellipsis (…) to indicate 'continued from before' or 'and so on.' This, of course, is usually not valid code (although the ellipsis can legitimately be used in function headings). Generally speaking, if you want to test code with a '…' in it, you'll need to figure out what the '…' stands for.
- I test all the code with the **gcc** compiler beforehand (time permitting), so it should work as indicated. However, code sometimes needs to be trimmed to fit it onto a slide, and this is where errors are most likely to occur. If you spot an error, please point it out, and I'll correct the slides ASAP.

A Note About the Slides

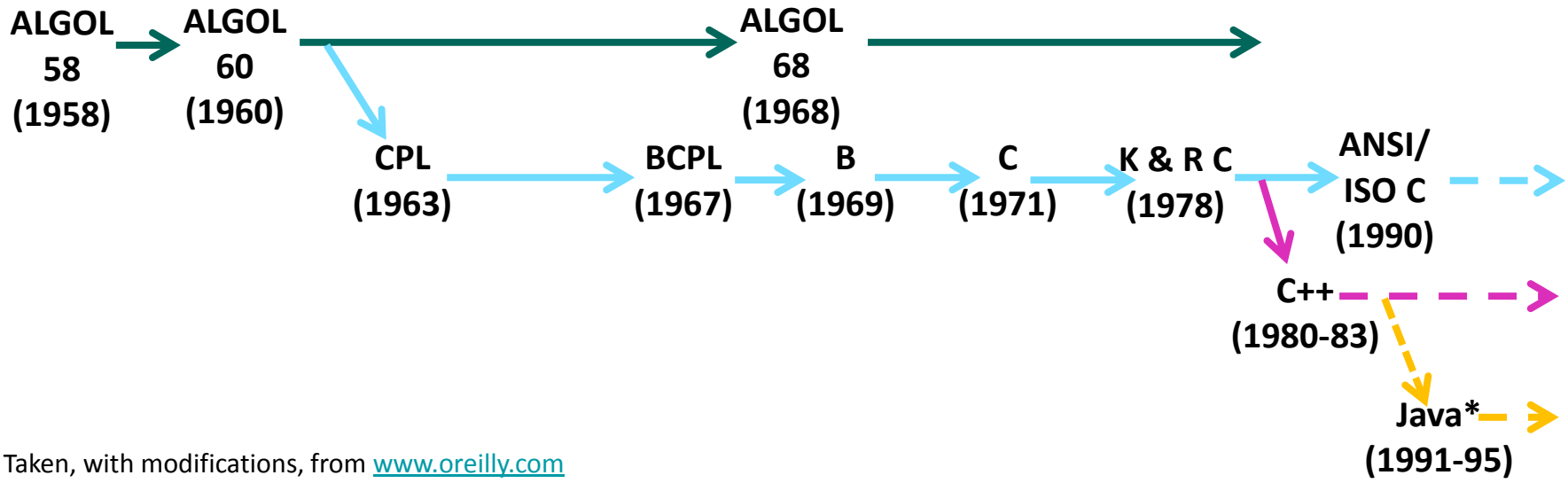
- Microsoft products have the annoying tendency of inserting angular ‵quotes″ in place of normal upright ones, like "this". C (and Java, C++, JavaScript, etc.) do not recognize ‵ and ″, and will flag an error if you try to compile code containing them (Note: they are not in the ASCII character set). If you want to run the sample code found in these slides, make sure it contains only "upright" quotes, and not ‵slanted″ quotes.

- More generally, code copied from .ppt and .pdf documents often contains characters which do not show up in text editors but which are forbidden in C. Oftentimes, the trouble and frustration spent finding these hidden, uncompilable spaces, tabs and carriage returns causes more wasted time than just typing the code in manually.

- Finally, code presented in lectures is almost always pared down in order to fit on a slide. Your programs need to be much better documented than I have space for on the slides. See the 'CST8234 Documentation Standard', available on Canvas, for an indication of what I expect in your assignments.

1.0 C – A Brief History

- A language descended from ALGOL (a 1950s era mainframe language), C is closer to PL/1, Pascal and Ada than FORTRAN, BASIC, or Lisp. In rough terms, this means that the designers of C put more emphasis on formal language structure, and less on human-readability.

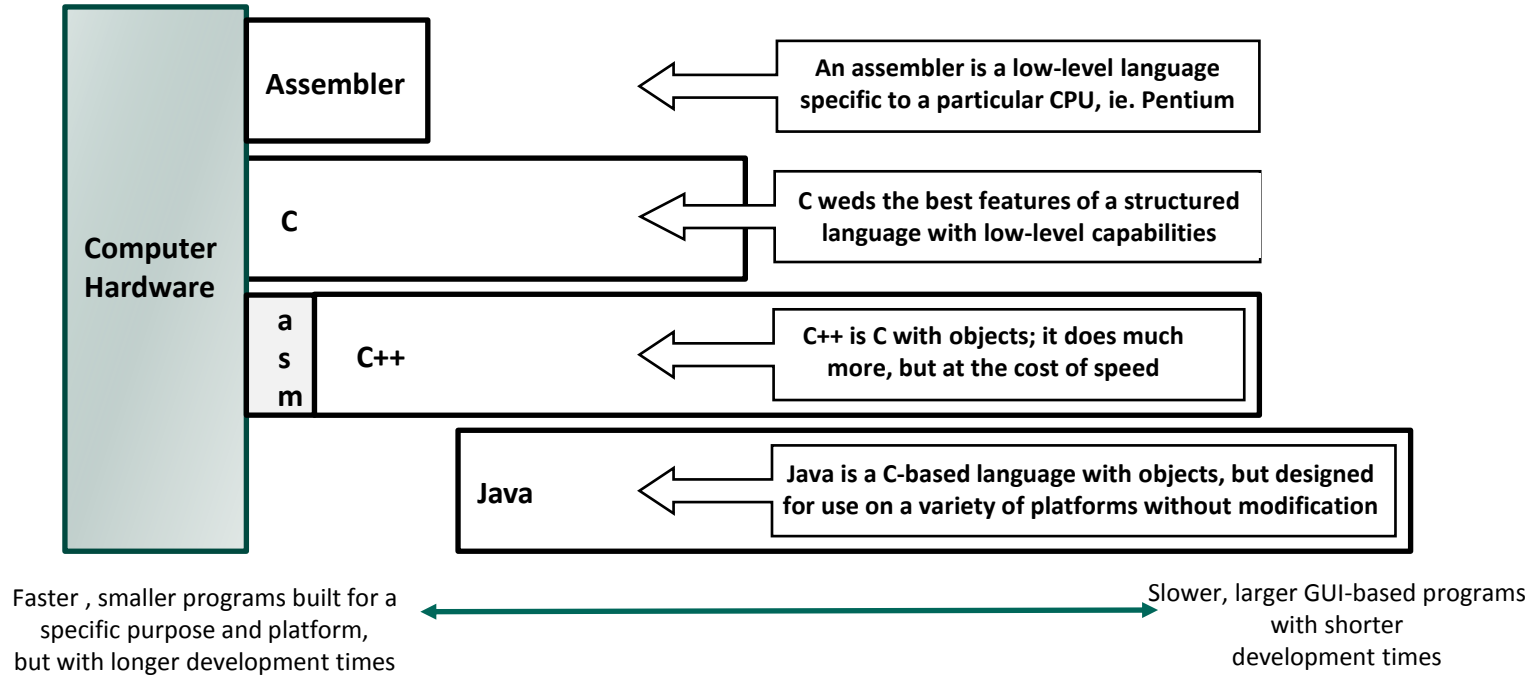


Taken, with modifications, from www.oreilly.com

*Java borrows elements from a half-dozen older languages, including Ada, Smalltalk and C++, although most of its influence comes from the latter, which traces its lineage directly back to C.

1.1 C Compared to Other languages

- Some languages operate 'closer to the silicon'—they allow programmers to exercise more direct control over the computer hardware. These languages generate extremely fast code, but it usually takes much longer to develop, and it is platform-specific (i.e. the code is written to run on a specific chip/IC, like the Pentium, and it cannot be easily ported to another hardware platform.) By contrast, Java was designed with fast application development and **portability** in mind, but at the cost of execution speed.



1.2 Why C? – Reason #1: Unix

- C was originally designed to allow the UNIX operating system to be implemented on a PDP-11. Prior to that time, operating systems were written in assembly language for the hardware on which they were intended to run. After C, many (perhaps most) new operating systems were written (almost entirely) in C. And so, as UNIX grew in popularity, so did C.

"C comes from the same culture as Unix—an elegant tool, but one designed with the skilled professional in mind. It is a language that resides fairly 'close to the machine'—its abstractions not too far removed from the understanding and operations of the computer."

Steve Lohr, *Go To*, Basic Books (2001), pg. 79

1.2 Why C? – Reason #1: Unix

- Largely as a result of its success with UNIX, C is *the* language for writing operating systems. C has been used to develop many different operating system, including Linux, the early versions of Windows and lately, Android. (Recent Mac OSs, such as iOS, tend to be written in Objective-C, although all early Mac OSs, like those associated with the original 'Mac Classic', were written mainly in C.)

- Other computer languages and compilers have been written in C. For example, the Sun JVM (Java Virtual Machine) is implemented in C, along with the native methods for most Java classes.

1.2 Why C? – Reason #2: Speed

- C was designed with instructions that reflect the underlying operation of a processor, but without the inherent complexity of any specific assembler language. It enjoys the benefits of being both a standardized, (fairly) high-level language, but one that is easily translated into the assembly code of any processor with little loss in execution speed along the way. C has been described as being essentially a "low-level high-level language." Thus it is *extremely* powerful: C represents the best of both worlds.

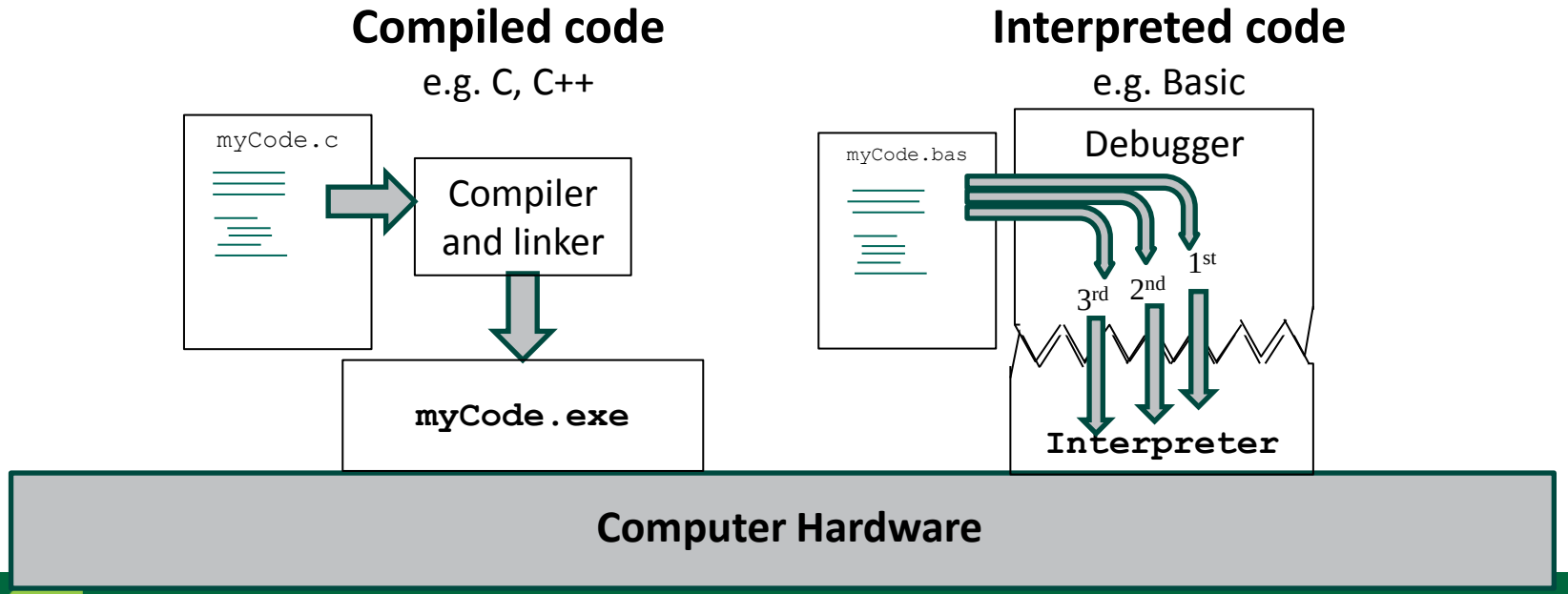
"…C is a higher-level language in that it is not wedded to a specific machine. Yet it was designed to allow programmers to do heavy-duty "systems" programming, working on such basic software plumbing as operating systems, compilers, and the like. Before C, systems programming was not really done in higher-level languages. C did for systems programming what FORTRAN did for scientific and engineering computation—it made it easier to communicate with the machine, lowering a barrier to computing in that discipline."

Steve Lohr, *Go To*, Basic Books (2001), pg. 79

Review – Compilers and Interpreters

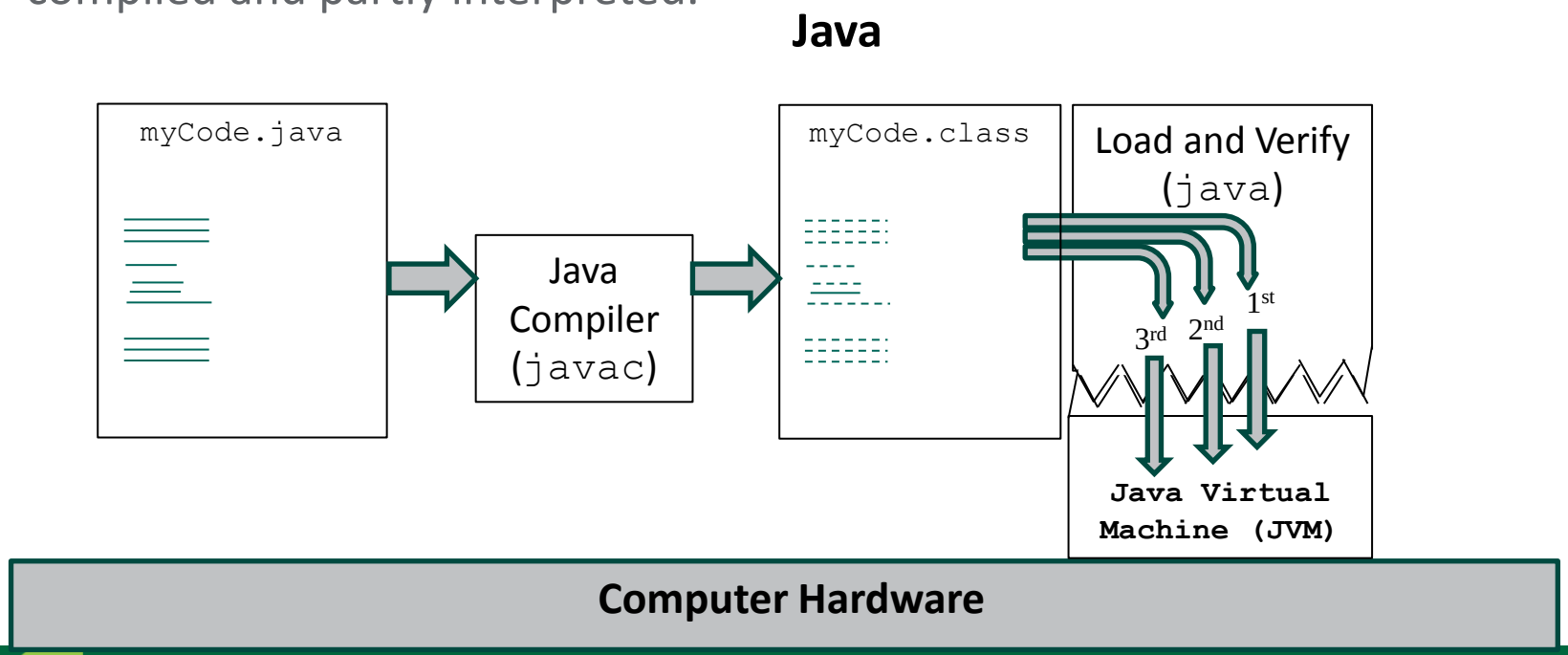
- C is a ***compiled*** language, meaning that your code is converted to machine-specific instructions that execute directly off the processor—no need to translate instructions one line at a time, as required by an ***interpreted*** language. The difference in speed is typically a factor of ~10.

NOTE: Traditionally, languages are classified according to whether they are compiled (e.g. C, C++) or interpreted (BASIC, PHP)



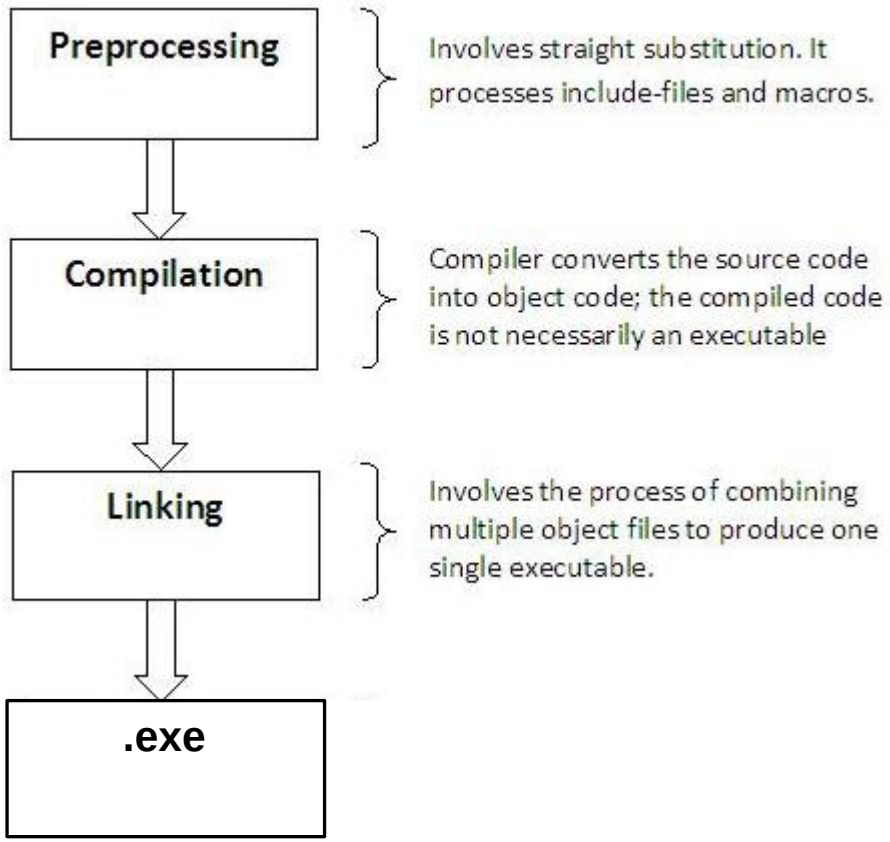
Review – Compilers and Interpreters

- Java is different; it's first compiled to ***bytecode***, and the bytecode is then interpreted by the Java interpreter (the ***JVM***). But parts of the bytecode may be further compiled for speed on the local processor, and then executed *directly* off the processor. (Note: *Every computer language is a compromise between different design goals; Java is no exception*). In the end, Java doesn't really fall into either of the two traditional categories: its partly compiled and partly interpreted.



1.2 Why C? – Reason #2: Speed

- C is a compiled language, and as such, it goes through a series of steps before the executable (.exe) file is created. These steps are shown at right. (More on this subject shortly)



Review – Java v. C

- C and Java are the products of very different eras in computer history

1970's

- Integrated circuits only just invented;
- Wide variety of CPUs/MPUs
- Multiple non-compatible architectures present (IBM, Tektronix, Honeywell…)
- Small RAM (kBytes) & hard drive size
- Slow speed (~MHz)

→ C Language

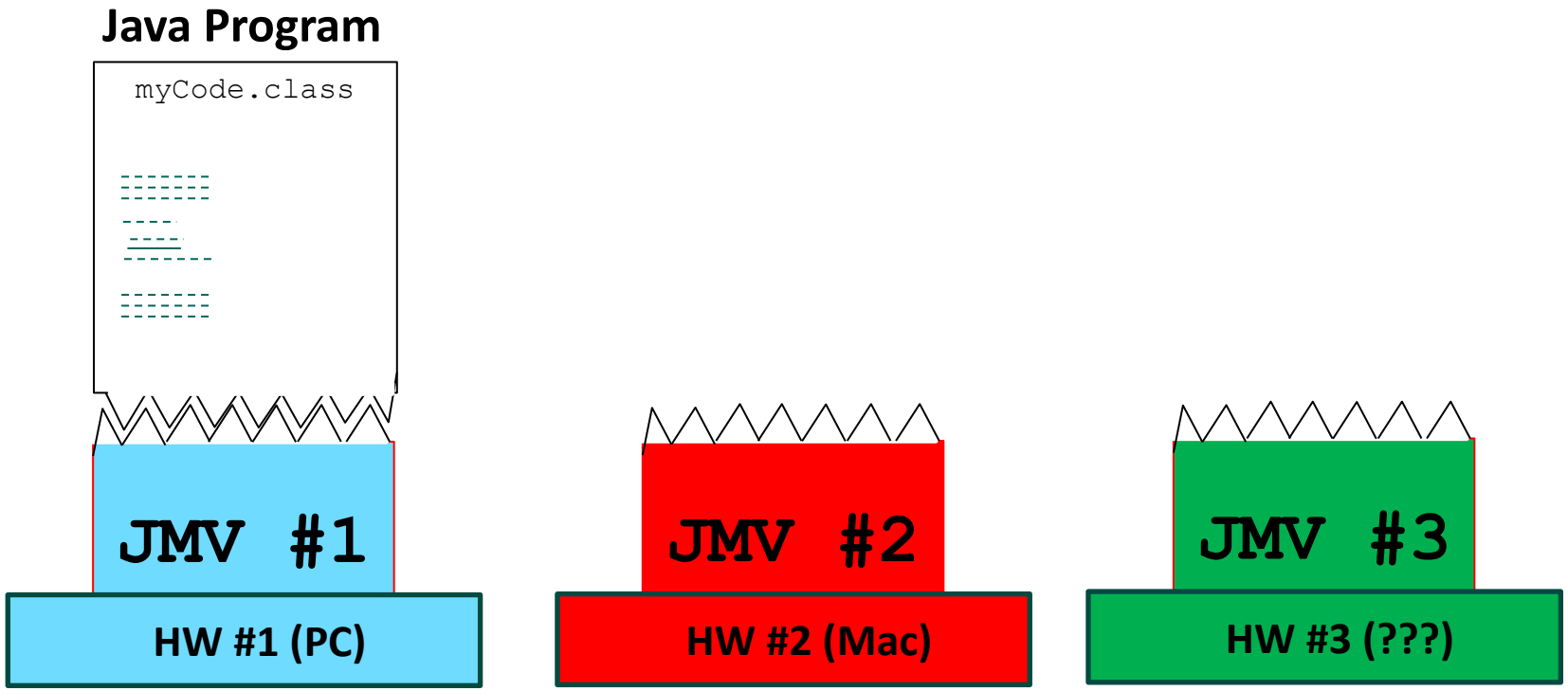
mid 1990's - 2000's

- Large projects & large # programmers
- Three main OSs: Windows, Mac, Unix
- Large RAM (GBs) and HDs (TBs)
- High speed chips and buses, multi-core architecture (several GHz)
- Internet connectivity is a given

→ Java

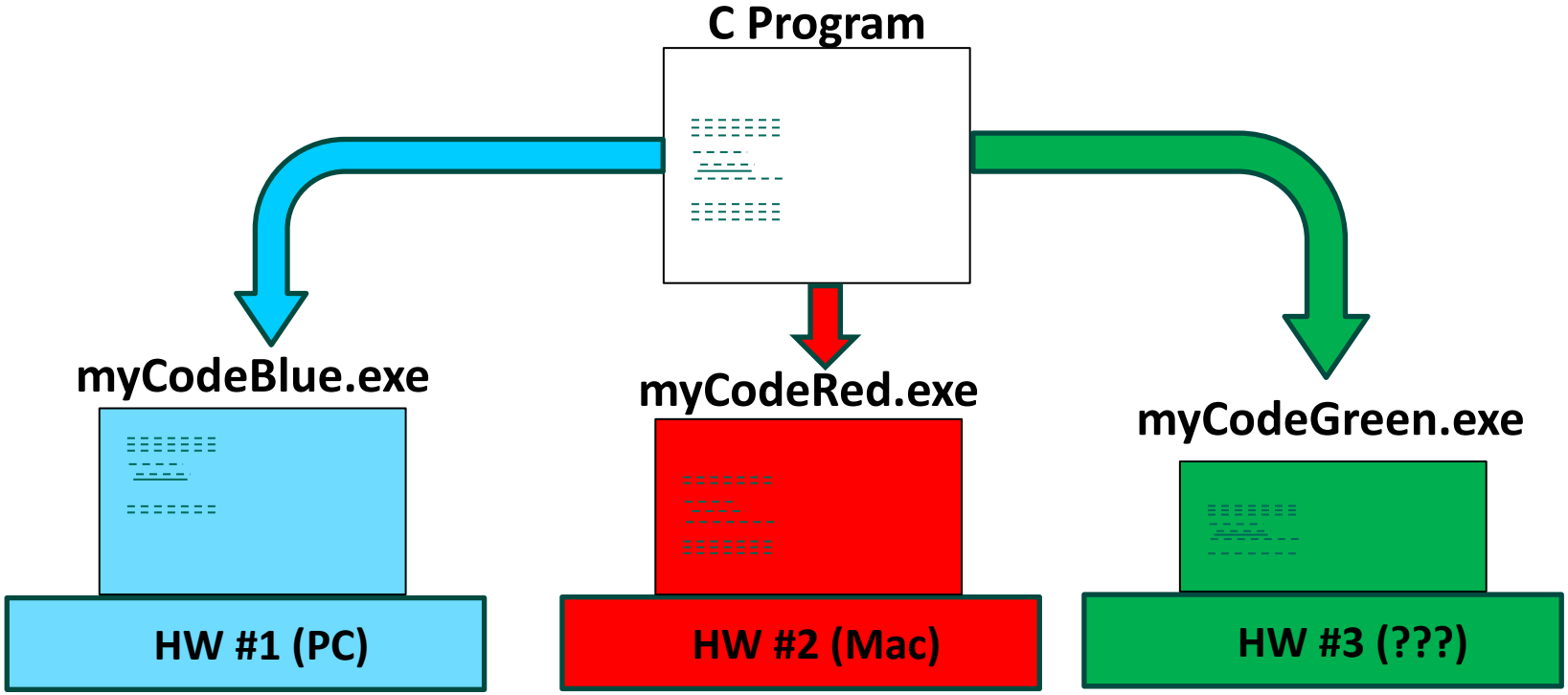
Review – Java v. C

- Java's solution to hardware diversity is to ignore it; as long as you can run the JVM on a piece of HW, you can (in principle) run *any* piece of Java code. Thus the JVM designer needs to follow a very tight set of specifications to ensure compliance with every Java program that might be executed off of it. The Java language itself is fairly restrictive (compared with C); it doesn't allow for a great deal of flexibility in the interpretation of the language itself.



Review – Java v. C

- C's solution is much different. The ANSI C Language standard is flexible enough to allow code to be tailored to different architectures. To a Java programmer, this flexibility can look a bit flakey—as if the standard itself was poorly written, and therefore allows for bugs—but in fact this plasticity was deliberate, and reflects the demands placed on the C language given the purposes for which it was designed and constraints imposed on it by the variety of platforms on which it ran—and still runs.

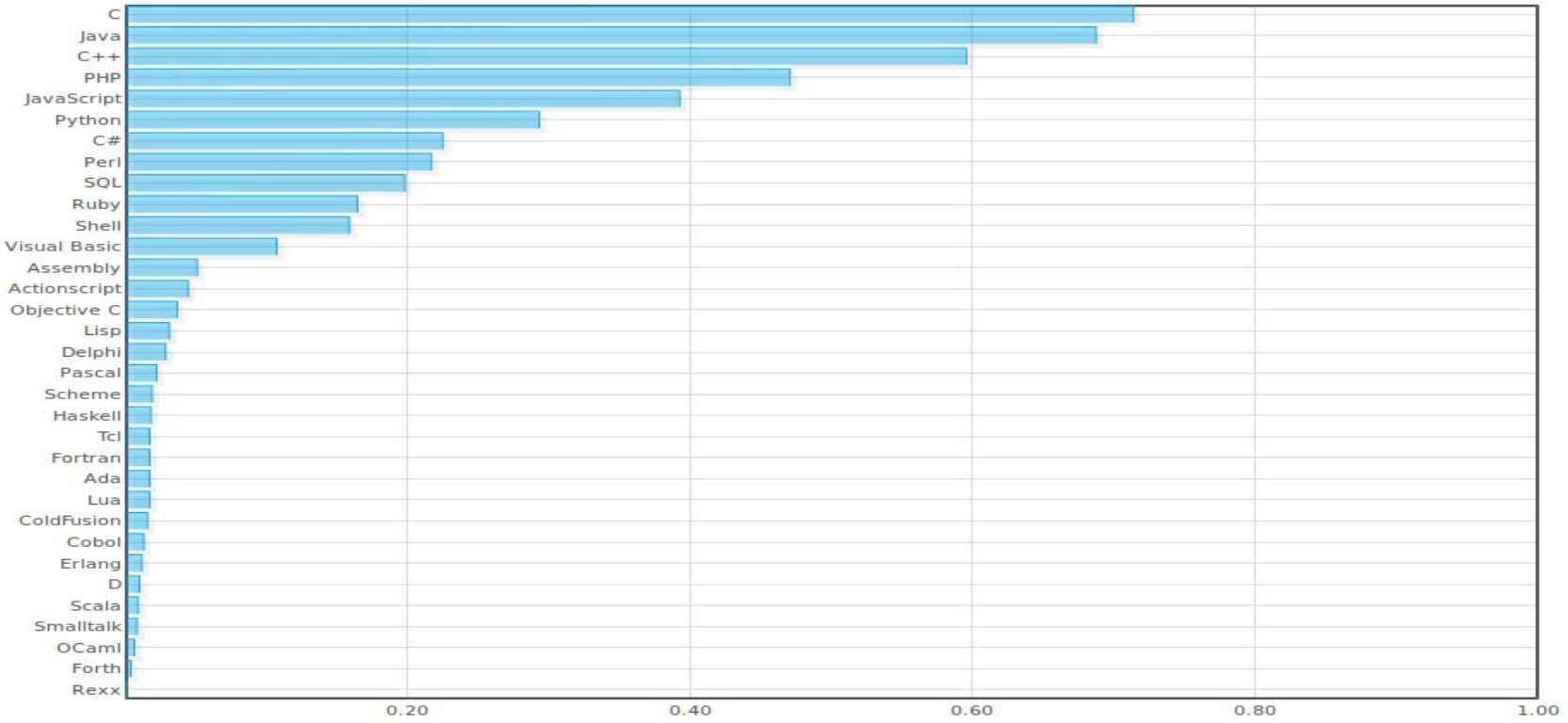


1.2 Why C? – Reason #3: Embedded Development

- Because of its 'best-of-both-worlds' combination a fast execution speed and *relatively* fast development speed, C gained a reputation as a very good language for low-level driver development. It is still used for this purpose, along with other hardware-related activities such as embedded development. (Additionally, because it is closer to assembler than other high-level languages, C produces compact executable code—a great benefit on systems with only a small amount of onboard memory.) Although C++ is gaining in popularity for this purpose, C is still the preferred language for programming the ICs that control everything from heart monitors to microwave ovens.

1.2 Why C? – Reason #4: Popularity

- Despite its antiquity and the fact it is not an OOPL, C remains one of the most popular programming languages, surpassing, by some estimates, C++, Java, Ruby, and Python. C is still used in the development of operating systems, browsers, game engines, device drivers, and word processors.



http://www.langpop.com

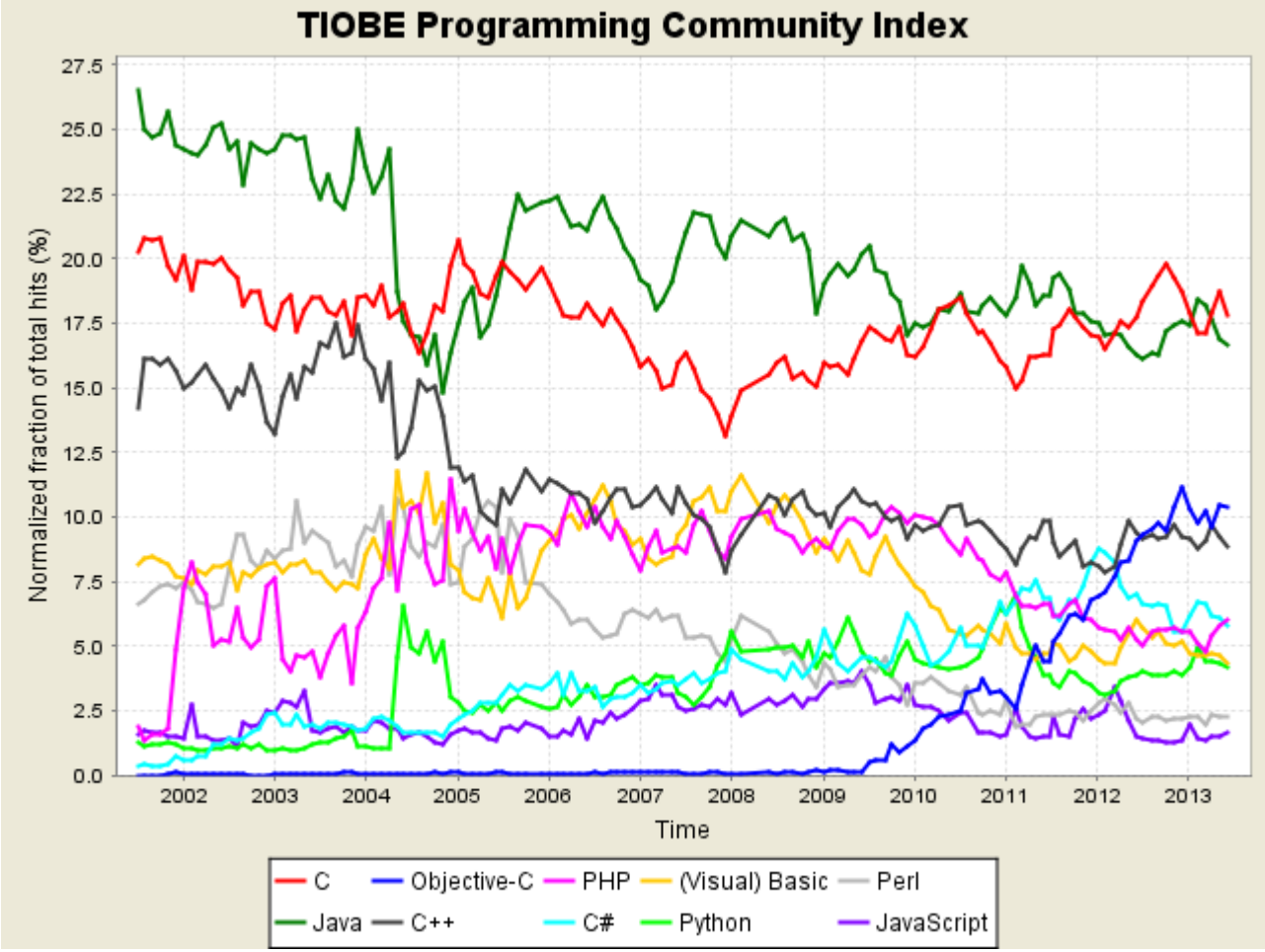
1.2 Why C? – Reason #4: Popularity

- The popularity of languages changes with time; while different sites use different metrics to determine 'language popularity', C (at least by some measures) has continued to remain on top for more than 25 years.

Programming Language	Position August 2012	Position August 2007	Position August 1997	Position August 1987
C	1	2	1	1
Java	2	1	5	-
Objective-C	3	47	-	-
C++	4	4	2	6
C#	5	7	-	-

http://www.langpop.com

1.2 Why C? – Reason #4: Popularity



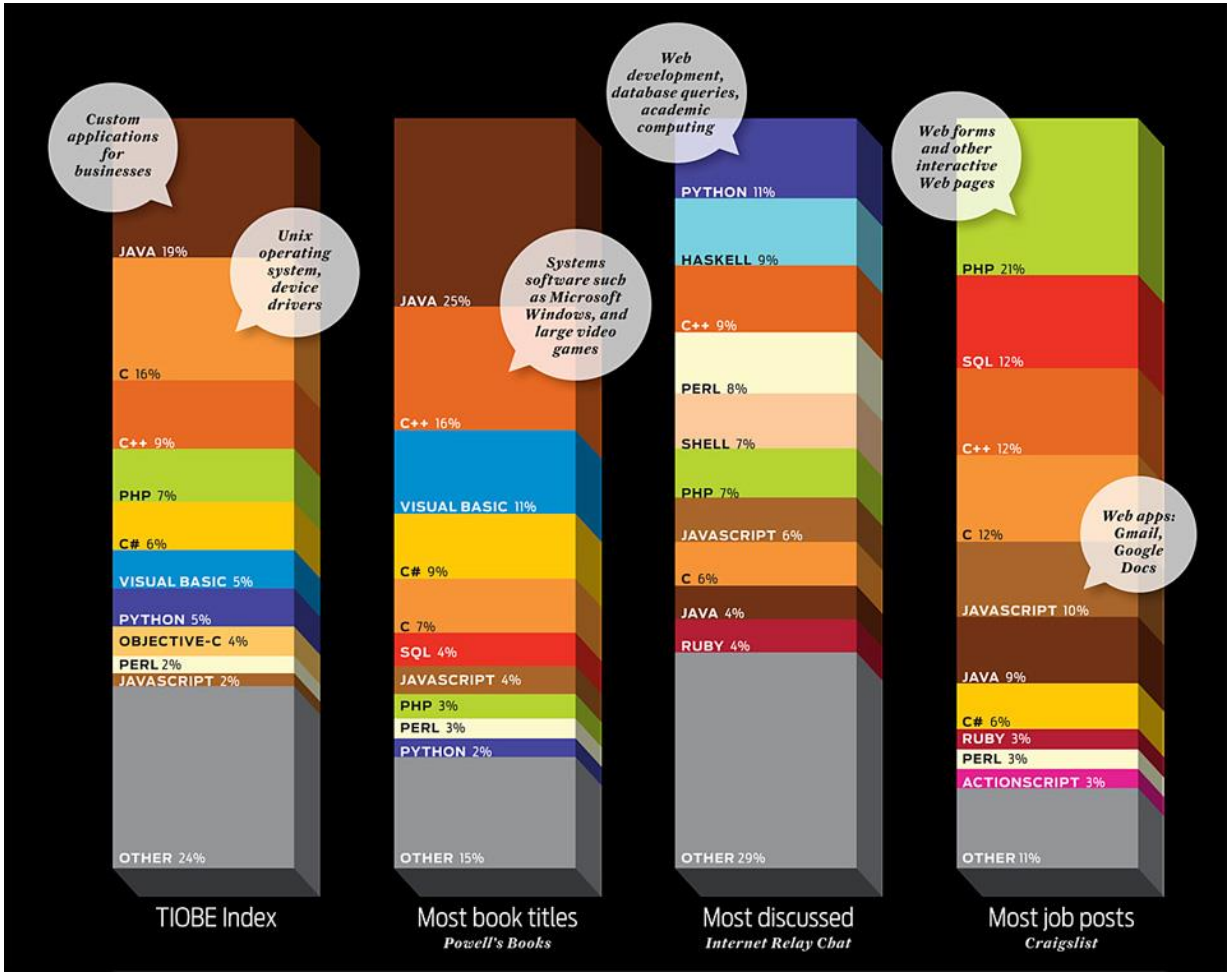
http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (from Sept. 2013)

1.2 Why C? – Reason #4: Popularity

Position Aug 2013	Position Aug 2012	Delta in Position	Programming Language	Ratings Aug 2013	Delta Aug 2012	Status
1	2	↑	Java	15.978%	-0.37%	A
2	1	↓	C	15.974%	-2.96%	A
3	4	↑	C++	9.371%	+0.04%	A
4	3	↓	Objective-C	8.082%	-1.46%	A
5	6	↑	PHP	6.694%	+1.17%	A
6	5	↓	C#	6.117%	-0.47%	A
7	7	=	(Visual) Basic	3.873%	-1.46%	A
8	8	=	Python	3.603%	-0.27%	A
9	11	↑↑	JavaScript	2.093%	+0.73%	A
10	10	=	Ruby	2.067%	+0.38%	A
11	9	↓↓	Perl	2.041%	-0.23%	A
12	15	↑↑↑	Transact-SQL	1.393%	+0.54%	A
13	14	↑	Visual Basic .NET	1.320%	+0.44%	A
14	12	↓↓	Delphi/Object Pascal	0.918%	-0.09%	A--
15	20	↑↑↑↑↑	MATLAB	0.841%	+0.31%	A--
16	13	↓↓↓	Lisp	0.752%	-0.22%	A
17	19	↑↑	PL/SQL	0.751%	+0.14%	A
18	16	↓↓	Pascal	0.620%	-0.17%	A-
19	23	↑↑↑↑	Assembly	0.616%	+0.11%	B
20	22	↑↑	SAS	0.580%	+0.06%	B

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (from Sept. 2013)

1.2 Why C? – Reason #4: Popularity



http://spectrum.ieee.org/at-work/tech-careers/the-top-10-programming-languages (from Sept. 2011)

1.2 Why C? – Reason #4: Popularity

- C is the progenitor of a whole family of C-based languages. While many people never learn C directly, anyone who learns any of the following languages has (to a considerable extent) already learned a substantial portion of the C language—whether they knew it or not:
    - C++
    - Java
    - Objective-C
    - JavaScript
    - PHP
    - Perl
    - C#

1.3 C vs. Java Programming – Fundamental Differences

Java is a C-based language and so shares most of its syntax, rules of precedence, and most other conventions of the C language. However because of its origins and purpose, there are two general features of C which are often confusing for Java programmers

1.3 C vs. Java Programming – Fundamental Differences

1. C was designed to be adapted to different architectures, and thus the language contains *several* features which allow it to be tailored to suit the needs of a programmer working on a specific platform; unlike Java, it is very much a 'hands-on' language that operates 'close to the silicon'. The single *most* confusing result of this hands-on aspect of the language is that:

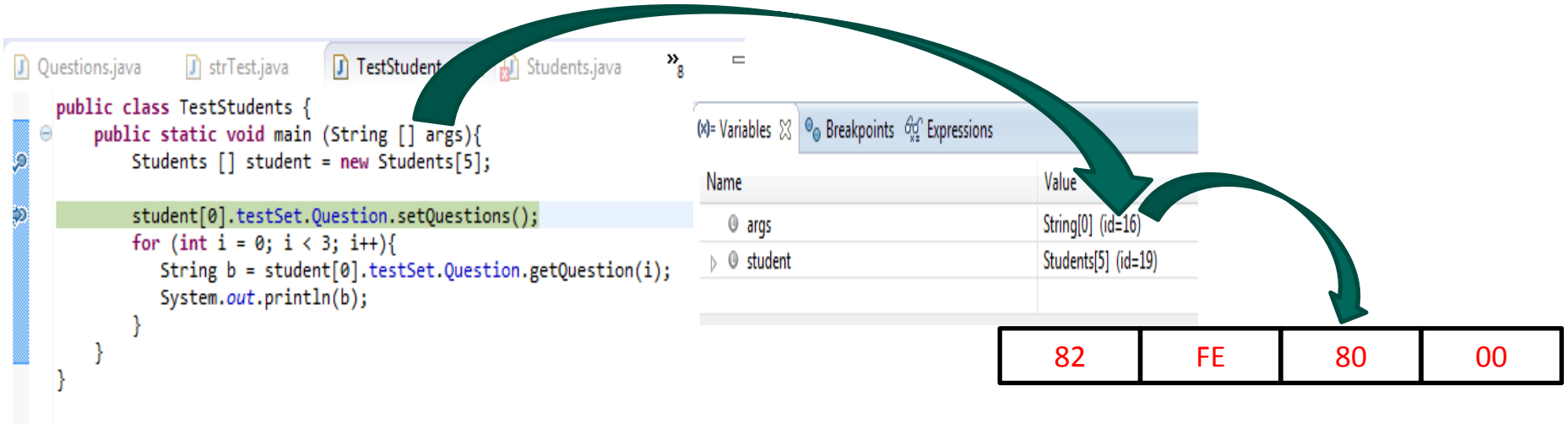
C uses pointers

1.3 C vs. Java Programming – Fundamental Differences

- A pointer is, fundamentally, a **reference data type**, rather than a **primitive data type**, like an `int`, `float` or `char`. Why use reference data types at all? Why not just pass all your information 'by value.' Remember, C was built for speed and small executable size, so the 'C Language Philosophy' on this subject might be summarized as follows:
  - When information occupies a few bytes of memory (e.g. primitive data types), it is easy to copy this information quickly to a new location. Hence this information can be passed '***by value***' – the value stored in memory is copied directly from one location to another.
  - When information occupies a large block of memory (e.g. arrays, files), then it is slow, inconvenient and wasteful to copy all the information over to a new location; it is easiest just to copy the address or ***handle*** (as its called in Windows programming). This is known as passing the information '***by reference***'. The original data stays in the same place in memory; only the location of the *pointer to the memory, i.e. its address,* is copied.

1.3 C vs. Java Programming – Fundamental Diffs ⌘

- This explanation applies to C and C++; with Java, there's an extra step. Java assigns IDs to referenced data types, and these refer to the actual address of the object being 'pointed' to. So in Java, there's an *extra* level of indirection not found in other languages.



1.3 C vs. Java Programming – Fundamental Differences

2. C was designed to allow individual programmers or small groups of programmers to build specific projects, where a knowledge of the underlying hardware was essential. This is far removed from the requirements of today's programming community, in which large projects involving dozens or hundreds of programmers are the norm, and the hardware is largely abstracted away from the programmer. Because of its origins in the 1970s, most of the programming practices and utilities you are used to (e.g. IDEs, classes, enterprise development…) do not exist in the world of C. Most particularly,

C is not an Object-Oriented Programming Language (OOPL)

1.3 C vs. Java Programming – Fundamental Differences

- You may think, at first glance, that this fact will make your life easier: it doesn't. Objects hide a great deal of the underlying complexity of your code; without them, you are responsible for managing that complexity.
- The fact that C is OOP-free has a number of additional consequences, which may be non-obvious at this point:

1. While you can make something that looks and acts like a class to organize your code—called a ***structure***—you cannot implement common true OOP features, like **operator overloading** and **abstract classes**.
2. Similarly, there is no such thing as **inheritance**, **polymorphism**, **implementing an interface**, **constructors**, etc.
3. Since there are no classes, you can't call on the **String class** for your I/O. You need to deal with arrays of characters directly. Also, because there is no array class (e.g. `int a[]=new int[12]`), you are responsible for keeping track of how large an array is: there's no `.length` property to keep you from running off the end of a `for` loop when processing a string of characters. There are no objects, so, in fact, strictly speaking there are no properties *whatsoever*.

1.3 C vs. Java Programming – Fundamental Differences

(And these are the least of your problems…). Therefore, in C,

You are responsible for ensuring the safety of your code.

1.4 C: Advantages & Disadvantages

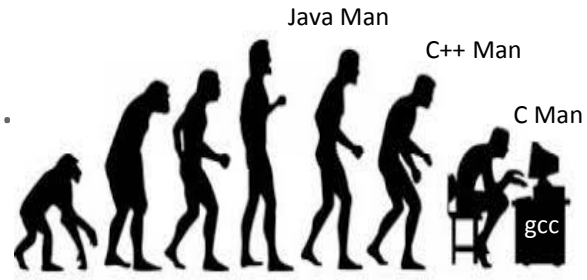
Disadvantages

- C creates extremely compact, powerful code, which can at times appear to be somewhat cryptic, especially for novices;
- C requires an understanding of pointers and addressing;
- C is a more hardware-focused language than some programmers are used to, hence there's more overhead getting started with C than with an OOPL, where the hardware is largely abstracted away from the programmer;
- C is less-structured than most object-oriented languages, so C programmers need to be more disciplined than OOP programmers—the C compiler isn't going to rescue you from potential mistakes the way Eclipse will…;
- No checks on array size; the programmer is responsible for not running off the end of arrays, such as when assigning strings;

1.4 C: Advantages & Disadvantages

Disadvantages (con't)

- The C standard is accommodating enough to be adaptable to different hardware platforms. However, this variability leads to apparent inconsistencies in the way the language is implemented on different compilers. Therefore, a frequently heard excuse when C code works differently than expected is: "it's compiler dependent." This sometimes makes the language look a bit 'flakey', when what's really being exposed is genuine *flexibility*. That being said, *no manufacturer* implements a language—any language—perfectly, exactly according standards on each release; every computer language, no matter how well documented, will vary slightly between implementations. And C, having many manufacturers over the many decades, is no different.
- C is not an OOP language, so many of the features of an OOPL that make them easy to use are missing. For modern programmers, this can feel like a step backward to a more primitive time…



1.4 C: Advantages & Disadvantages

Advantages

- C is a small, portable, and efficient language designed to run on, and be tailored to, a variety of platforms;
- Creates small-footprint executables with execution rates comparable to assembly code, but without the language-specific complexity of assembly: C code is *extremely* fast compared to OOP languages;
- Ideal for embedded applications, including systems level programming, device drivers, and anything related to UNIX/LINUX;
- Has wide applicability in a number of other languages like C++, C# and Java. When you understand C, you understand a large chunk of many other programming languages as well;
- Both an ANSI and ISO standard, recognized world-wide;
- An extremely concise and elegant language, C is as close to a *basic* universal computer language as exists on the planet.