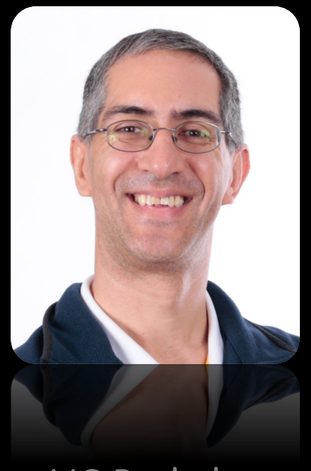


UC Berkeley
Teaching Professor
Dan Garcia

CS61C

Great Ideas in Computer Architecture

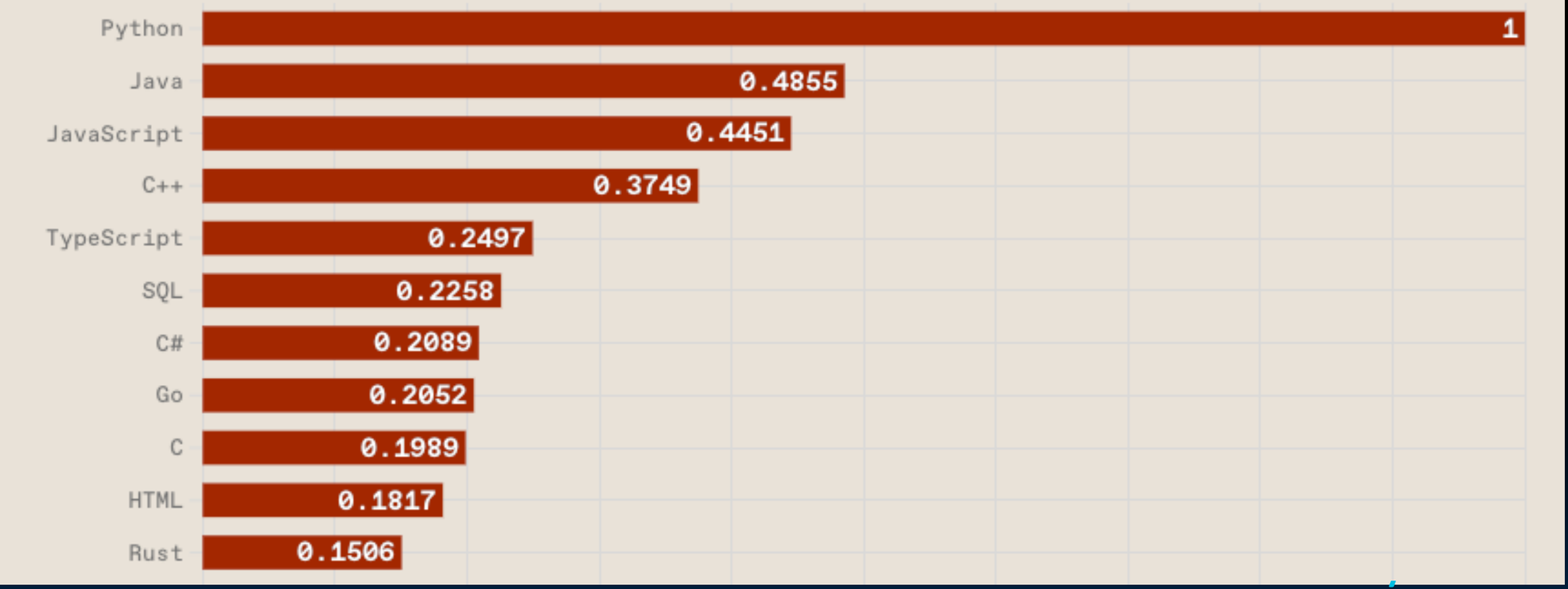
(a.k.a. Machine Structures)

Introduction to the C Programming Language

Top Programming Languages 2024

Click a button to see a differently weighted ranking

Spectrum | Trending | Jobs



IEEE Advancing Technology for Humanity

IEEE Spectrum 2023-08-23

Normalized relative popularity (0 to 1.0). Popularity is drawn from multiple internet sources; popularity of an active language is defined by whether software engineers use it (IEEE Spectrum), whether employers are asking for it on the job, and how trending it is on the internet. More about methodology here.

https://spectrum.ieee.org/the-top-programming-languages-2024

Great Idea #1: Abstraction (Levels of Representation/Interpretation)

High Level Language Program (e.g., C)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

Compiler

Assembly Language Program (e.g., RISC-V)

```
lw    x3, 0(x10)
lw    x4, 4(x10)
sw    x4, 0(x10)
sw    x3, 4(x10)
```

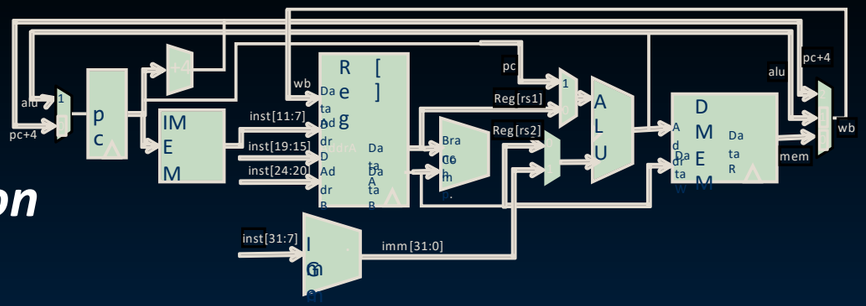
Assembler

Machine Language Program (RISC-V)

```
1000 1101 1110 0010 0000 0000 0000 0000
1000 1110 0001 0000 0000 0000 0000 0100
1010 1110 0001 0010 0000 0000 0000 0000
1010 1101 1110 0010 0000 0000 0000 0100
```

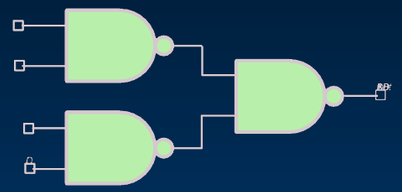
Anything can be represented as a number, i.e., data or instructions

Hardware Architecture Description (e.g., block diagrams)



Architecture Implementation

Logic Circuit Description (Circuit Schematic Diagrams)



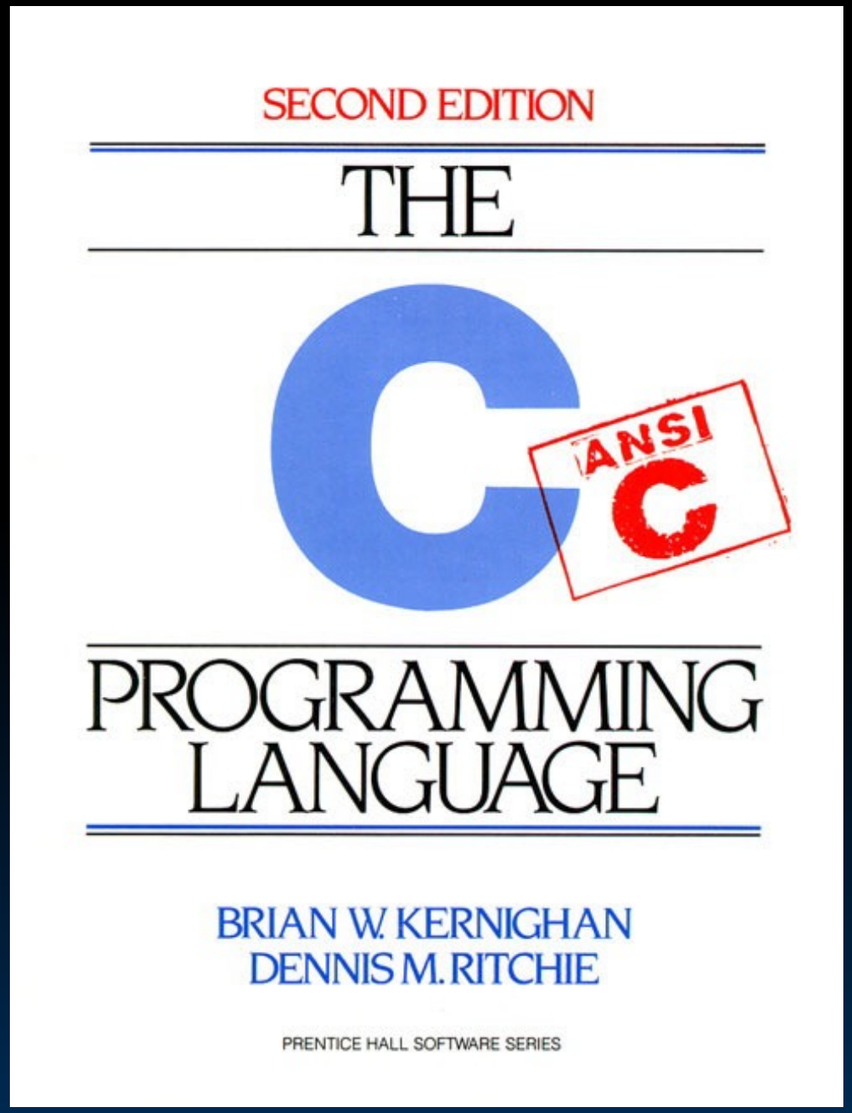
Agenda

Intro to C

- Introduction to C
- Hello World
- Compilation
- C vs. Java: Syntax
- Binary, Decimal, and Hex
- C Variables and Basic Types
- More C Features
- [Extra] Reference Slides

Why C? (1/2)

- Kernighan and Ritchie (K&R)
  - *C is not a "very high-level" language, nor a "big" one, and is not specialized to any particular area of application. But its absence of restrictions and its generality make it more convenient and effective for many tasks than supposedly more powerful languages.*



Why C? (2/2)

- Why C?
  - We can write programs that allow us to exploit underlying features of the architecture
    - memory management, special instructions, parallelism
- Enabled first operating system not written in assembly language!
  - UNIX - A portable OS!
- C and derivatives (C++/Obj-C/C#) still **one of the most popular** programming languages after >40 years!

The C Language Is Constantly Evolving

- The C programming language standard has had several significant revisions since its inception in 1972.
  - Just like Python2 vs Python 3 – same language, but slightly different syntax/features
- From StackOverflow (link):
  - Pre-1989: K&R C (note K&R 1st ed 1978, 2nd ed 1988)
  - 1989/1990: ANSI C
  - 1999: C99
  - 2011: C11
  - 2017: C17
  - 2024 (?): "C23"
    - New functions that are memory-safe, legacy-compatible
    - Syntax changes improve C++ compatibility.

We're teaching the current C17 standard in this course (plus some C23 later).

C23 is the informal name for what will likely become **ISO/IEC 9899:2024**, the next standard for the C programming language, which will replace C17 (standard ISO/IEC 9899:2018).[1] It was started in 2016 informally as C2x,[2] and expected to be published in 2024.[3] The most recent publicly available working draft of C23 was released on July 2, 2023.[4]

Disclaimer(s)

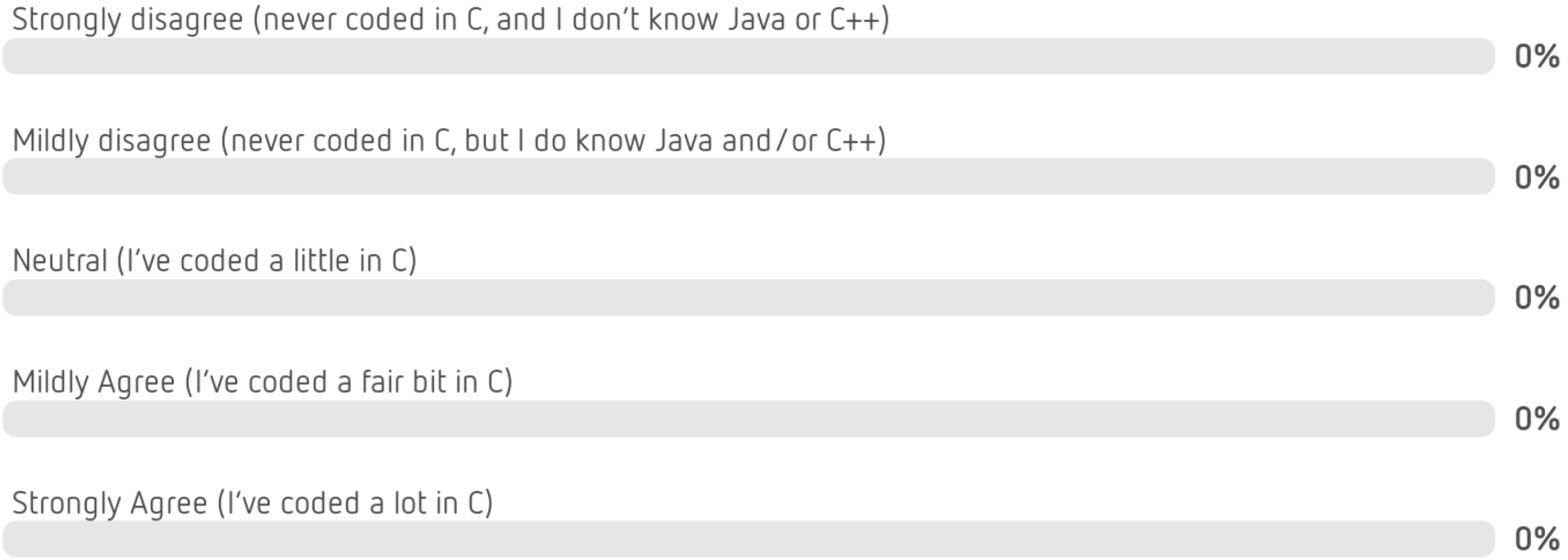
- **You will not learn how to fully code in C in these lectures!**
  You'll still need your C reference.
  - K&R is a ***must-have!*** More references at the end of these slides
- CS61C will teach **key C concepts**: Pointers, Arrays, Implications for Memory management
  - Key security concept: All of the above are ***unsafe***: If your CS61C program contains an error, it might not crash immediately but instead leave the program in an inconsistent (and often exploitable) state.
  - Take CS161/CS162 for more!
- **2024 Note**: If you are starting a new project where **performance** matters use either Go or Rust.
  - **Rust**, "C-but-safe": By the time your C is (theoretically) correct w/all necessary checks it should be no faster than Rust.
  - **Go**, "Concurrency": Practical concurrent programming takes advantage of modern multi-core microprocessors.

Agenda

Hello World

- Introduction to C
- Hello World
- Compilation
- C vs. Java: Syntax
- Binary, Decimal, and Hex
- C Variables and Basic Types
- More C Features
- [Extra] Reference Slides

L03 "Before this class, I (student) would say I am a solid C programmer"



Our very first C program

hello_world.c

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
  printf("Hello World!\n");

  return 0;
}
```

HelloWorld.java (for reference)

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

- Import library uses `#include`
  - For `printf()`
- Main **function**
  - Unlike Java, not an object method!
    - C is **function-oriented**.
  - Function return type is **integer**, not void.
    - **0 on success??**
- Data types seem similar enough
  - But why are there **two** command-line arguments, `argc` and `argv`?

Our very first C program: Let's run it!

hello_world.c

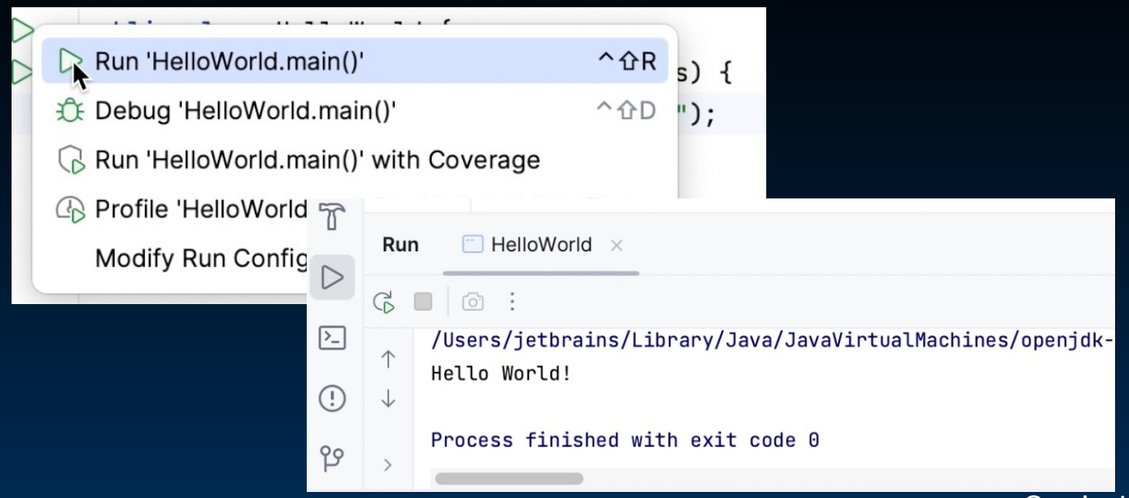
```c
#include <stdio.h>

int main(int argc, char *argv[]) {
  printf("Hello World!\n");

  return 0;
}
```

(demo; lecture code in Drive)

```
unix% gcc hello_world.c
unix% ./a.out
Hello World!
unix% gcc -o hello_world hello_world.c
unix% ./hello_world
```

HelloWorld.java (for reference)

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```



Source: IntelliJ [link]

Agenda

Compilation

- Introduction to C
- Hello World
- Compilation
- C vs. Java: Syntax
- Binary, Decimal, and Hex
- C Variables and Basic Types
- More C Features
- [Extra] Reference Slides

Compilation, an Overview

- C **compilers** map C programs directly into **architecture-specific** machine code (string of 1s and 0s).
  - Unlike Java, which converts to **architecture-independent** bytecode that may then be compiled by a just-in-time compiler (JIT)
  - Unlike Python environments, which converts to a byte code at runtime
    - These differ mainly in exactly **when** your program is converted to low-level machine instructions ("levels of interpretation")
- For C, **compiling** is **colloquially** the full process of using a compiler to translate C programs into executables.
  - But actually multiple steps: (more later)
    compiling .c files to .o files,
    automatic assembling,
    then linking the .o files into executables.

Our compiler in this class is the command-line program gcc.

```
~/lec02 $ gcc hello_world.c
~/lec02 $ ./a.out
Hello World!
```

Compilation: Disadvantages

- Compiled files, including the executable, are architecture-specific, depending on processor type (e.g., MIPS vs. x86 vs. RISC-V) and the operating system (e.g., Windows vs. Linux vs. MacOS)
- Executable must be rebuilt on each new system.
  - "Porting your code" to a new architecture means copying the .c file, then recompiling using gcc
- "Change → Compile → Run [repeat]" iteration cycle can be slow during development.
  - but make only rebuilds changed pieces, and can compile in parallel: `make -j`
  - linker is sequential though → Amdahl's Law

Compilation: Advantages

- Reasonable compilation time: enhancements in compilation procedure (`Makefiles`) allow only modified files to be recompiled
  - (See Project 1 for more information)
- Generally much faster run-time performance vs. Java for comparable code, because compilation optimizes for a given **architecture**.
- **2024 Note**: Plenty of people do scientific computation in **Python!?!**
  - They have good libraries for accessing GPU-specific resources.
  - Python can call low-level C code to do work: Cython
  - Pytorch, a popular Python library for machine learning, uses C++
  - Spark can manage many other machines in parallel. (more later)
  - The Python interpreter is written in C.

Agenda

C vs. Java: Syntax

- Introduction to C
- Hello World
- Compilation
- C vs. Java: Syntax
- Binary, Decimal, and Hex
- C Variables and Basic Types
- More C Features
- [Extra] Reference Slides

C vs. Java, so far

See more:
http://www.cs.princeton.edu/introcs/faq/c2java.html

	C	Java
Type of Language	Function Oriented	Object Oriented
Programming Unit	Function	Class = Abstract Data Type
Compilation	`gcc hello.c` creates machine language code	`javac Hello.java` creates Java virtual machine language bytecode
Execution	`a.out` loads and executes program	`java Hello` interprets bytecodes
Storage	Manual (`malloc`, `free`)	New allocates & initializes, Automatic (garbage collection) frees

(more next week)

C, Java syntaxes are similar!

See more: http://www.cs.princeton.edu/introcs/faq/c2java.html

	C	Java
Accessing a library	`#include <stdio.h>`	`import java.io.File;`
Comments	`/* … */` or `//` … end of line	
Variable declaration	Before you use it	
Many operators	Arithmetic, Assignment: `+, -, *, /, %, =, +=, -=, …` Boolean: `!, &&, ||` Comparators: `==, !=, <, <=, >, >=` increment and decrement: `++` and `–` Bitwise operators: `<<, >>, ~, &, |, ^` Subgroup expressions: `( )`	
Constants	`#define, const`	`final`
Variable naming conventions	`sum_of_squares`	`sumOfSquares`

What are bitwise operations? Stay tuned!

C Control flow syntax: Use your curly braces

- In C, function definition have curly braces.

```
int number_of_people () { return 3;     }
void print_something() { printf("something\n"); }
```

  - void: “no value will be returned,” just like in Java

- For control flow, single-line statement bodies **can omit** curly braces.
  - This is acceptable syntax; same as in Java (but we didn’t tell you)

```
while (expression) {
  statement;
}
```

```
for (initialize; check; update) {
  statement;
}
```

```
if (x == 0) { y++; }
```

```
while (expression) statement;
for (initialize; check; update)
  statement;
if (x == 0) y++;
```

- ⚠ However, this only applies to single-line statements.
  - Subsequent lines are considered outside of the body!
  - Leads to many debugging errors (StackOverflow link)
  - “Just because you can do it, doesn’t mean it’s a good idea.”

```
if (x = 0) {
  y++;
  j = j + y;
}
```

```
if (x = 0)
  y++;
  j = j + y;
```

executed outside of conditional!

C Syntax: `main()` command-line argments

- What do the arguments mean?
  - Combined, **`argc`** and **`argv`** get the **`main`** function to accept arguments.
  - `argc` will contain the number of strings on the command line (the executable counts as one, plus one for each argument). Here `argc` is 2:
    - `$ ./a.out myFile`
  - `argv` is a pointer to an array containing the arguments as strings (more later re: pointers and strings).

```
1  #include <stdio.h>
2
3  int main(int argc, char *argv[]) {
4    printf("Hello World!\n");
5
6    return 0;
7  }
```

Agenda

Binary, Decimal, & Hex

- Introduction to C
- Hello World
- Compilation
- C vs. Java: Syntax
- Binary, Decimal, and Hex
- C Variables and Basic Types
- More C Features
- [Extra] Reference Slides

Def: Number vs Numeral

Numeral

A symbol or name that stands for a number

e.g., 4 , four , quatro , IV , IIII , …

…and Digits are symbols that make numerals

Above the abstraction line

Abstraction Line

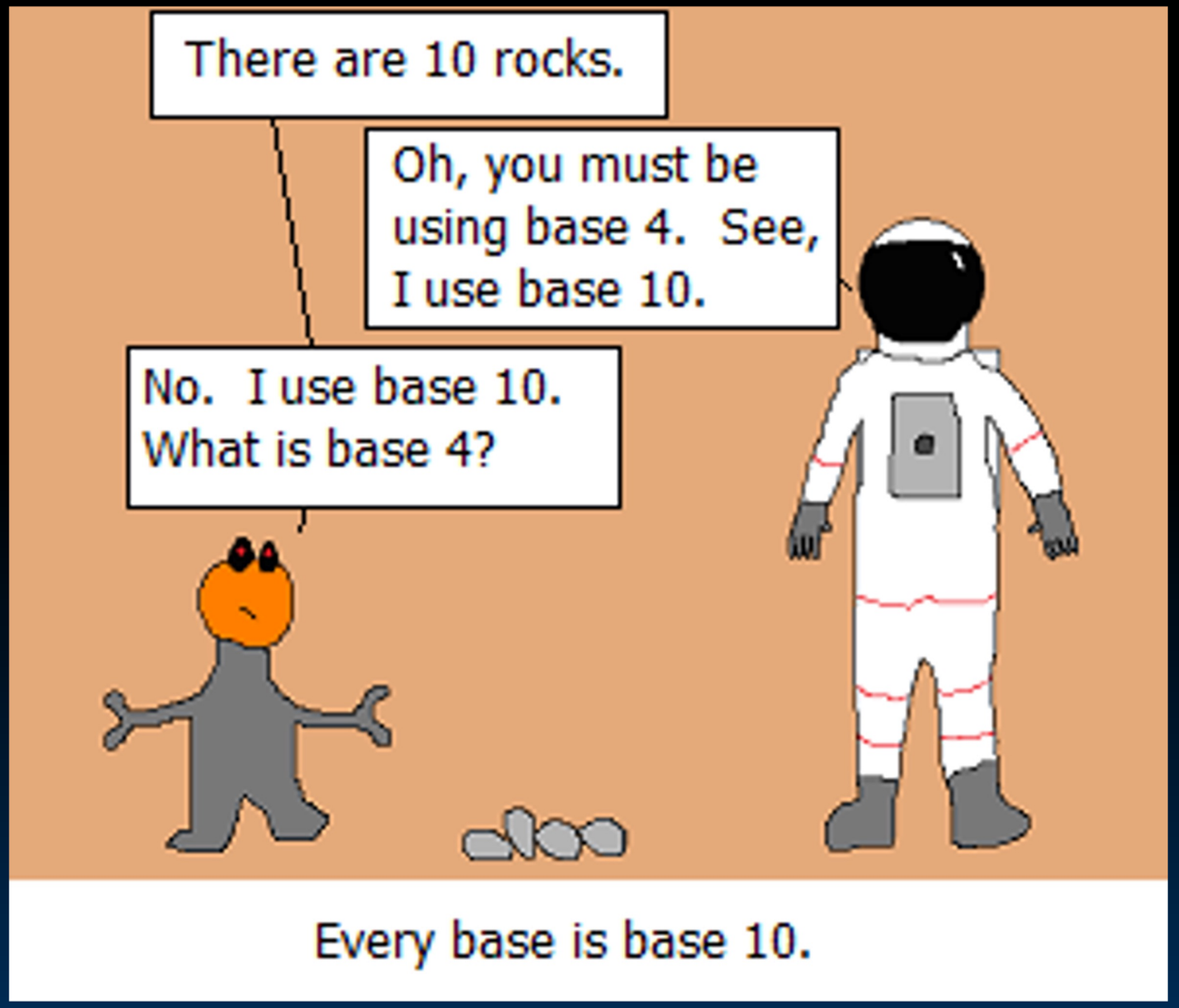
Below the abstraction line

Number

The “idea” in our minds…there is only ONE of these

e.g., the concept of “4”

Joke: Every Base is Base 10…



Relevant StackOverflow post

printf() string formats

hello_numbers.c

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    printf("Hello %s!\n", "Numbers!");

    int num = 1234;
    printf("Decimal: %d\n", num);


    return 0;
}
```

- %s Placeholder for first arg
  - s: format value as **string**
- %d Placeholder for first arg
  - d: format value as **decimal numeral**

```
~/lec02 $ gcc hello_numbers.c
~/lec02 $ ./a.out
Hello Numbers!
Decimal: 1234
```

https://en.wikipedia.org/wiki/Printf

The same number, but as a base-16 numeral!

hello_numbers.c

```
1  #include <stdio.h>
2  int main(int argc, char *argv[]) {
3      printf("Hello %s!\n", "Numbers!");
4
5      int num = 1234;
6      printf("Decimal: %d\n", num);
7
8      printf("Hex:     %x\n", num);
9      return 0;
10 }
```

- %s Placeholder for first arg
  - s: format value as **string**
- %d Placeholder for first arg
  - d: format value as **decimal numeral**
- %x Placeholder for first arg
  - x: format value as **hexadecimal numeral**

```
~/lec02 $ gcc hello_numbers.c
~/lec02 $ ./a.out
Hello Numbers!
Decimal: 1234
Hex:     4D2
```

https://en.wikipedia.org/wiki/Printf

Agenda

C Variables and Basic Types

- Introduction to C
- Hello World
- Compilation
- C vs. Java: Syntax
- Binary, Decimal, and Hex
- C Variables and Basic Types
- More C Features
- [Extra] Reference Slides

A Cautionary Note: Undefined Behavior…

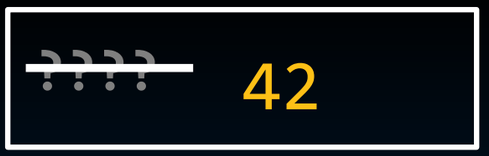
- A lot of C has "Undefined Behavior"
  - This means it is often unpredictable behavior
    - It will run one way on one computer…
    - But some other way on another
    - Or even just be different each time the program is executed!
- Often characterized as "Heisenbugs"
  - Bugs that seem random/hard to reproduce, and seem to disappear or change when debugging
  - Cf. "Bohrbugs" which are repeatable

Declaring a C Variable Does Not Also Initialize It

- Variables are not automatically initialized to default values!
- If a variable is not initialized in its declaration, ***it holds garbage!***
  - The contents are undefined…
  - …but C still lets you use uninitialized variables!
  - Danger: Bugs may only manifest after you've built other parts of your program.

```
int y = 38;
```


```
int x;
…
x = 42;
```



C Basic Types

- The C standard does not define the absolute size of integer types, other than char!
- C: `int` size depends on computer; what integer type is most efficient w/processor
- The C standard only guarantees* relative sizes:

```
sizeof(long long)
  ≥ sizeof(long)
  ≥ sizeof(int)
  ≥ sizeof(short)
```

  - Could all be 64 bits!

Type	Description	Example
`int`	Integer Numbers (including negatives) At least 16 bits, can be larger	`0, 78, -217, 0x7337`
`unsigned int`	Unsigned Integers (i.e., non-negatives)	`0, 6, 35102`
`float`	Floating point decimal	`0.0, 3.14159, 6.02e23`
`double`	Equal or higher precision floating point	`0.0, 3.14159, 6.02e23`
`char`	Single character	`'a', 'D', '\n'`
`long`	Longer `int`, Size >= `sizeof(int)`, at least 32b	`0, 78, -217, 301720971`
`long long`	Even longer `int`, size >= `sizeof(long)`, at least 64b	`317051927210925l`

We encourage you to instead use `intN_t` and `uintN_t`!!
`#include <stdint.h>`

`sizeof()`: the storage size of a datatype, in bytes.

Garcia, Kao

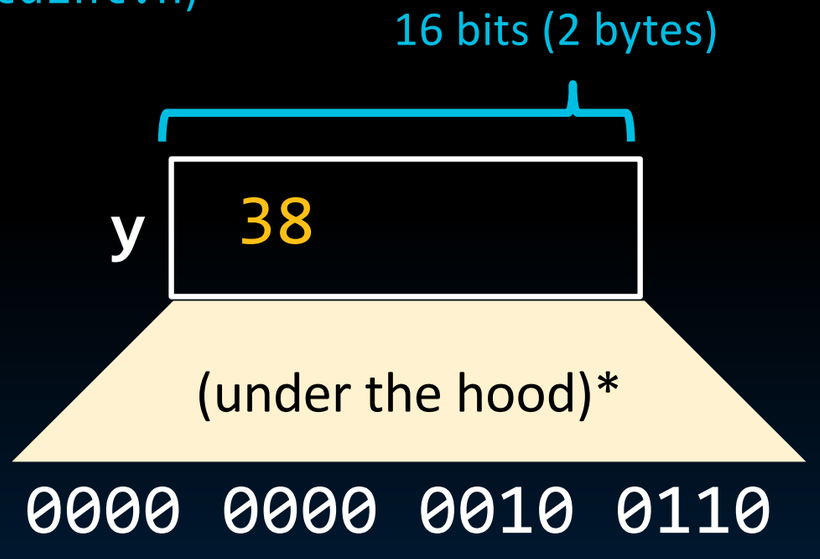
*Slightly up for debate [StackOverflow]

Why are variables typed?

- C variables are typed.
  - Types of ***variables*** can't change, e.g., y cannot now store a float.
  - However, you can cast ***value***s (more later)
- A variable's type helps the compiler determine how to translate the program to machine code designed for the computer's architecture, such as
  - How many bytes the variable takes up in memory, and
  - What operators the variable supports.

```
uint16_t y = 38;
```

(unsigned 16-bit integer, see `stdint.h`)



*possibly implementation-dependent [StackOverflow]

And In Conclusion, …

- C chosen to exploit underlying features of HW
- Key C concepts
  - Pointers, arrays, implications for Mem management
  - We'll discuss this in a LOT more detail next time!!
- C compiled and linked to make executables
  - Pros (speed) and Cons (slow edit-compile cycle)
- C looks mostly like Java except
  - no OOP, ADTs defined through structs
  - 0 (and NULL) FALSE, all else TRUE (C99 bool types)
  - Use `intN_t` and `uintN_t` for portable code!
  - Unitialized variables contain garbage
    - "Bohrbugs" (repeatable) vs "Heisenbugs" (random)

Agenda

for next time

More C Features

- Introduction to C
- Hello World
- Compilation
- C vs. Java: Syntax
- Binary, Decimal, and Hex
- C Variables and Basic Types
- More C Features
- [Extra] Reference Slides

The C bool: True or False?

for next time

- Boolean is **not** a primitive type in C! Instead:
- FALSE:
  - `0` (integer, i.e., all bits are 0)
  - `NULL` (pointer) (more later)
- TRUE:
  - **Everything else!**
  - Note: Same is true in Python.

```
if (42) {
  printf("meaning of life\n");
}
```

…what is printed?

A. `meaning of life`

B. (nothing)

- Nowadays: true and false are provided by `stdbool.h`.

Consts, Enums, #define in C

for next time

- Constant, **`const`**, is assigned a typed value once in the declaration.
  - Value can't change during entire execution of program.

```
const float  golden_ratio = 1.618;
const int    days_in_week = 7;
const double the_law      = 2.99792458e8;
```

  - You can have a constant version of any of the standard C variable types.
- `#define PI (3.14159)` is a CPP (C Preprocessor) Macro.
  - Prior to compilation, preprocess by performing string replacement in the program based on all #define macros.
  - Replace all `PI` with `(3.14159)` → In effect, makes **`PI`** a "constant"
- Enums: a group of related integer constants. E.g.,

```
enum cardsuit {CLUBS,DIAMONDS,HEARTS,SPADES};
enum color {RED, GREEN, BLUE};
```

More Typing: Typedefs and Structs

for next time

- typedef allows you to define new types.

```
typedef uint8_t BYTE;
BYTE b1, b2;
```

- structs are structured groups of variables, e.g.,

```
typedef struct {
  int length_in_seconds;
  int year_recorded;
} SONG;

SONG song1;
song1.length_in_seconds  =  213;
song1.year_recorded      = 1994;

SONG song2;
song2.length_in_seconds  =  248;
song2.year_recorded      = 1988;
```

SONG is an alias for `typedef struct {int length_in_seconds; int year_recorded; }`

Dot notation: `x.y = value`

song1

length_in_seconds	213	year_recorded	1994

Structs are **not** objects! The . operator is **not** a method call! (more later)

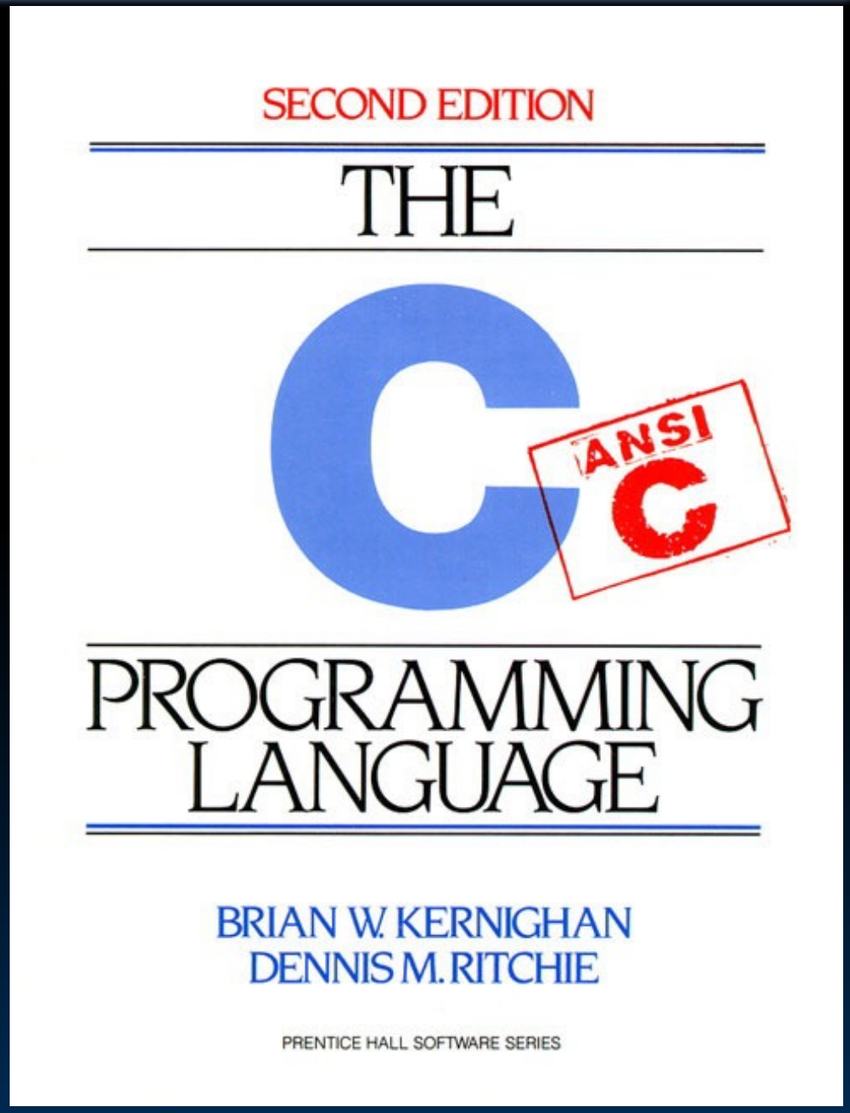
Agenda

[Extra] C Reference Slides

- Introduction to C
- Hello World
- Compilation
- C vs. Java: Syntax
- Binary, Decimal, and Hex
- C Variables and Basic Types
- More C Features
- [Extra] Reference Slides

C References

- K&R is a must-have!
- Useful Reference: "JAVA in a Nutshell," O'Reilly
  - Chapter 2, "How Java Differs from C"
- Brian Harvey's helpful transition notes
  - http://inst.eecs.berkeley.edu/~cs61c/resources/HarveyNotesC1-3.pdf



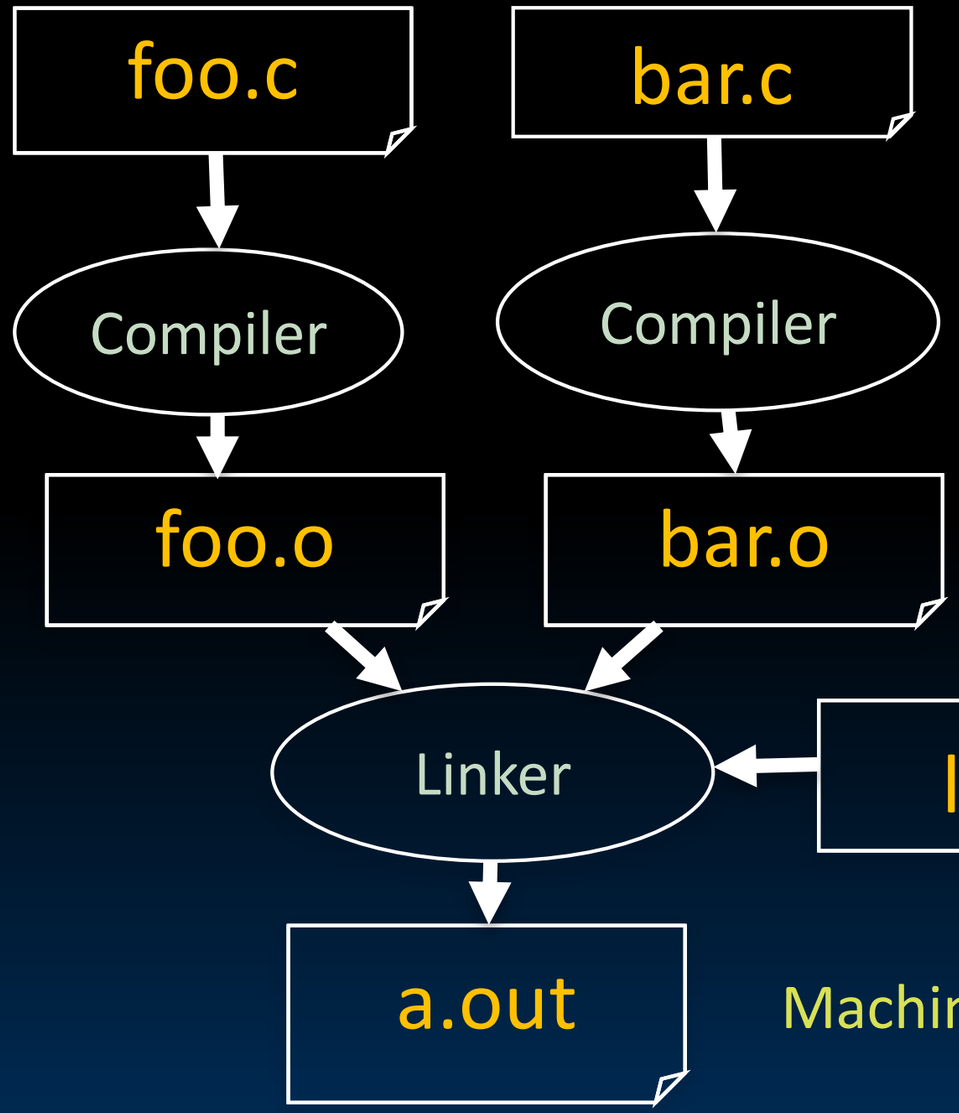
Has there been an update to ANSI C?

- Yes! It's called the "C99" or "C9x" std
  - To be safe: "`gcc -std=c99`" to compile
  - `printf("%ld\n", __STDC_VERSION__);` ➔ `199901`
- References
  - `en.wikipedia.org/wiki/C99`
- Highlights
  - Declarations in `for` loops, like Java
  - Java-like `//` comments (to end of line)
  - Variable-length non-global arrays
  - `<inttypes.h>`: explicit integer types
  - `<stdbool.h>` for boolean logic def's

Has there been an update to C99?

- Yes! It's called the "C11" (C18 fixes bugs…)
  - You need "`gcc -std=c11`" (or `c17`) to compile
  - `printf("%ld\n", __STDC_VERSION__);` ➔ `201112L`
  - `printf("%ld\n", __STDC_VERSION__);` ➔ `201710L`
- References
  - `en.wikipedia.org/wiki/C11_(C_standard_revision)`
- Highlights
  - Multi-threading support!
  - Unicode strings and constants
  - Removal of `gets()`
  - Type-generic Macros (dispatch based on type)
  - Support for complex values
  - Static assertions, Exclusive create-and-open, …

C Compilation Simplified Overview (more later)



C source files (text)

Compiler/assembler combined here

Machine code object files

Pre-built object file libraries

Machine code executable file

C Pre-Processor (CPP)

- C source files first pass through macro processor, CPP, before compiler sees code
- CPP replaces comments with a single space
- CPP commands begin with "#"
  - `#include "file.h"` /* Inserts `file.h` into output */
  - `#include <stdio.h>` /* Looks for file in standard location, but no actual difference! */
  - `#define PI (3.14159)` /* Define constant */
  - `#if/#endif` /* Conditionally include text */
- Use `-save-temps` option to gcc to see result of preprocessing
  - Full documentation at: `http://gcc.gnu.org/onlinedocs/cpp/`

C vs. Java…operators nearly identical

- arithmetic: `+, -, *, /, %`
- assignment: `=`
- augmented assignment: `+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=`
- bitwise logic: `~, &, |, ^`
- bitwise shifts: `<<, >>`
- boolean logic: `!, &&, ||`
- equality testing: `==, !=`
- subexpression grouping: `()`
- order relations: `<, <=, >, >=`
- increment and decrement: `++` and `--`
- member selection: `., ->`
  - Slightly different than Java because there are both structures and pointers to structures, more later
- conditional evaluation: `? :`

Typed Functions in C

- You have to declare the type of data you plan to return from a function
- Return type can be any C variable type, and is placed to the left of the function name
- You can also specify the return type as `void`
  - Just think of this as saying that no value will be returned
- Also need to declare types for values passed into a function
- Variables and functions MUST be declared before used

```
int number_of_people    () {  return 3;     }
float dollars_and_cents () {  return 10.33; }
```

C Syntax : Control Flow (1/2)

- Within a function, remarkably close to Java constructs (shows Java's legacy) for control flow
  - A statement can be a { } of code or just a standalone statement
- if-else
  - `if (expression) statement`

```
if (x == 0) y++;
if (x == 0) {y++;}
if (x == 0) {y++; j = j + y;}
```

  - `if (expression) statement1 else statement2`
    - There is an ambiguity in a series of if/else if/else if you don't use { }s, so use { }s to block the code
    - In fact, it is a bad C habit to not always have the statement in { }s, it has resulted in some amusing errors...
- while
  - `while (expression) statement`
  - `do statement while (expression);`

C Syntax : Control Flow (2/2)

- for

```
for (initialize; check; update) statement
```

- switch

```
switch (expression){
  case const1:    statements
  case const2:    statements
  default:        statements
}
break;
```

  - Note: until you do a `break` statement things keep executing in the **switch** statement

- C also has `goto`
  - But it can result in spectacularly bad code if you use it, so don't!

First Big C Program: Compute Sines table

```c
#include <stdio.h>
#include <math.h>
int main(void)
{
    int    angle_degree;
    double angle_radian, pi, value;

    printf("Compute a table of the sine function\n\n");
    pi = 4.0*atan(1.0); /* could also just use pi = M_PI */
    printf("Value of PI = %f \n\n", pi);
    printf("Angle\tSine\n");
    angle_degree = 0;/* initial angle value */
    while (angle_degree <= 360) { /* loop til angle_degree > 360 */
        angle_radian = pi * angle_degree / 180.0;
        value = sin(angle_radian);
        printf ("%3d\t%f\n ", angle_degree, value);
        angle_degree += 10; /* increment the loop index */
    }
 return 0;
}
```

```
PI = 3.141593

Angle  Sine
  0    0.000000
  10   0.173648
  20   0.342020
  30   0.500000
  40   0.642788
  50   0.766044
  60   0.866025
  70   0.939693
  80   0.984808
  90   1.000000
… etc …
```

C Syntax: Variable Declarations

- Similar to Java, but with a few minor but important differences
  - All variable declarations must appear before they are used
  - ANSI C: All must be at the beginning of a block.
  - A variable may be initialized in its declaration; ***if not, it holds garbage!***
    - the contents are undefined…
- Examples of declarations:
  - Correct: `{ int a = 0, b = 10; …`
  - Incorrect in ANSI C: `for (int i=0; …`
  - Correct in C99 (and beyond): `for (int i=0;…`

Integers: Python vs. Java vs. C

- C: `int` should be integer type that target processor works with most efficiently
- Only guarantee:
  - `sizeof(long long)`
    `≥ sizeof(long) ≥ sizeof(int) ≥ sizeof(short)`
  - Also, **`short`** >= 16 bits, **`long`** >= 32 bits
  - All could be 64 bits
  - This is why we encourage you to use `intN_t` and `uintN_t`!!

Language	`sizeof(int)`
Python	>=32 bits (plain `int`s), infinite (long `int`s)
Java	32 bits
C	Depends on computer; 16 or 32 or 64

CPP Macros: A Warning...

- You often see C preprocessor s defined to create small "functions"
  - But they aren't actual functions, instead it just changes the ***text*** of the program
  - In fact, all `#define` does is ***string replacement***
  - `#define min(X,Y) ((X)<(Y)?(X):(Y))`
- This can produce, umm, interesting errors with macros, if `foo(z)` has a side-effect
  - `next = min(w, foo(z));`
  - `next = ((w)<(foo(z))?(w):(foo(z)));`