What is C?

- The language of UNIX
- Procedural language (no classes)
- Low-level access to memory
- Easy to map to machine language
- Not much run-time stuff needed
- Surprisingly cross-platform

Why teach it now?
To transition from basic programming to Operating Systems (CSC246), Software Engineering (CSC326), etc.
Hey, do you want to build a system that will become the gold standard of OS design for this century? We can call it UNIX.

Okay, but only if we also invent a language to write it in, and only if that language becomes the default for all systems programming basically forever. We’ll call it C!
Cool, it worked!

Told ya.
What were they thinking?

• Main design considerations:
  – Compiler size: needed to run on PDP-11 with 24KB of memory (Algol60 was too big to fit)
  – Code size: needed to implement the whole OS and applications with little memory
  – Performance
  – Portability

• Little (if any consideration):
  – Security, robustness, maintainability
  – Legacy Code
## C vs. other languages

<table>
<thead>
<tr>
<th>Most modern languages</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Develop applications</td>
<td>Develop system code (and applications) (the two used to be the same thing)</td>
</tr>
<tr>
<td>Computer is an abstract logic engine</td>
<td>Near-direct control of the hardware</td>
</tr>
<tr>
<td>Prevent unintended behavior, reduce impact of simple mistakes</td>
<td>Never doubts the programmer, subtle bugs can have crazy effects</td>
</tr>
<tr>
<td>Runs on magic! (e.g. garbage collection)</td>
<td>Nothing happens without developer intent</td>
</tr>
<tr>
<td>May run via VM or interpreter</td>
<td>Compiles to native machine code</td>
</tr>
<tr>
<td>Smart, integrated toolchain (press button, receive EXE)</td>
<td>Discrete, UNIX-style toolchain</td>
</tr>
<tr>
<td></td>
<td>make → gcc (compilation) → gcc (linking) (even more discrete steps behind this)</td>
</tr>
</tbody>
</table>

```
$ make
gcc -o thing.o thing.c
gcc -o thing thing.o
```
Why C?

• It’s a “portable assembly language”
• Useful in...
  – Systems development: OS & Embedded
  – Optimized routines for use with other languages
  – Need for speed, size, or predictability
• Notable pure C software:
  – UNIX and Linux – kernel and most utilities
  – NetApp Data ONTAP (most common storage OS)
  – Python, Perl, PHP, Java*, Ruby*
  – A bajillion applications:

* With some C++ as well
Example C superpowers

Task: Export a list of coordinates in memory to disk

Most languages
• Develop file format
• Build routine to serialize data out to disk
• Build routine to read & parse data in
• Benchmark if performance is a concern

C
• Read/write memory to disk directly
### Example C superpowers

#### Task: Blink an LED

**Atmel ATTINY4 microcontroller:**
Entire computer (CPU, RAM, & storage)!
1024 bytes storage, 32 bytes RAM.

```python
led = 0
while (true):
    led = NOT led
    set_led(led)
    delay for 1 sec
```

<table>
<thead>
<tr>
<th>Language</th>
<th>Size of executable</th>
<th>Size of runtime (ignoring libraries)</th>
<th>Total size</th>
<th>RAM used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>410 B</td>
<td>13 MB</td>
<td>13 MB</td>
<td>14 MB</td>
</tr>
<tr>
<td>Python</td>
<td>60 B (source code)</td>
<td>2.9 MB</td>
<td>2.9 MB</td>
<td>5.4 MB</td>
</tr>
<tr>
<td>Desktop C</td>
<td>8376 B</td>
<td>None</td>
<td>8376 B</td>
<td>352 kB</td>
</tr>
<tr>
<td>Embedded C (Arduino)</td>
<td>838 B</td>
<td>None</td>
<td>838 B</td>
<td>~16 B</td>
</tr>
</tbody>
</table>

Max: 1024 B
Max: 32 B
What about C++?

• Originally called “C with Classes” (because that’s all it is)
• All C programs are C++ programs, as C++ is an extension to C
• Adds stuff you might recognize from Java (only uglier):
  – Classes (incl. abstract classes & virtual functions)
  – Operator overloading
  – Inheritance (incl. multiple inheritance)
  – Exceptions

Bjarne Stroustrup developed C++ in 1979 at Bell Labs

OUT OF SCOPE
C and Java: A comparison

### C

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, const char* argv[]) {
    int i;

    printf("Hello, world.\n");
    for (i=0; i<3; i++) {
        printf("%d\n", i);
    }
    return EXIT_SUCCESS;
}
```

### Java

```java
class Thing {
    static public void main (String[] args) {
        int i;

        System.out.printf("Hello, world.\n");
        for (i=0; i<3; i++) {
            System.out.printf("%d\n", i);
        }
    }
}
```

### Compilation and Execution

- **C:**
  ```
  $ gcc -o thing thing.c && ./thing
  Hello, world.
  0
  1
  2
  ```

- **Java:**
  ```
  $ javac Thing.java && java Thing
  Hello, world.
  0
  1
  2
  ```
• Different platforms have different conventions for end of line, end of file, tabs, compiler output, ...

• Solution (for this class): compile and run all programs consistently on one platform

• Our common platform: 

NCSU Linux Machines!

Don’t you gimme no “it worked on my box” nonsense!
### Your Choices

<table>
<thead>
<tr>
<th>Option</th>
<th>Use GUI-based Editor?</th>
<th>Access to your unity Filespace?</th>
<th>Matches grading environment?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Unity Lab Computer</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>ssh to VCL (linux)</td>
<td>N**</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>ssh to remote-linux.eos.ncsu.edu</td>
<td>N**</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Use Mac OS X (+developer tools)</td>
<td>Y</td>
<td>sftp*</td>
<td>N</td>
</tr>
<tr>
<td>Use MS Windows + cygwin</td>
<td>Y</td>
<td>sftp*</td>
<td>N</td>
</tr>
<tr>
<td>Use Linux on your PC (dual boot or virtualized)</td>
<td>Y</td>
<td>sftp*</td>
<td>N</td>
</tr>
</tbody>
</table>

* direct if you install realm kit
** Yes if you run X windows server on your computer
### Hello world

#### C

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, const char* argv[]) {
    int i;

    printf("Hello, world.\n");

    return EXIT_SUCCESS;
}
```

- **File with library function declarations**
- **Entry point of the program, with command line arguments**
- **Exit program and indicate successful completion**
- **Standard library function, with message argument**

---

```
$ gcc -Wall -std=c99 -o hello hello.c
$ ./hello
Hello, world.
```