CS 240 Programming in C

Guest Lecture: Real-world Applications

March 29, 2023

Where and *why* do people use C in 2023?

C as a Common Interface ("Lingua Franca")

- Code written 30+ years ago runs on today's hardware
  - Across multiple new processor architectures (`x86_64`, ARM, ...)
- Almost every language has a way to call C functions / libraries
  - **FFI**: Foreign Function Interface
  - Python's `cffi`, Java's JNI, Javascript's `node-ffi`, ...
- Example: `sqlite3` is a popular C library that nearly every language has bindings for

Embedded Systems

- Microcontrollers, single-purpose
- Small processors, limited memory (RAM)
  - 32 **kilo**bytes vs. 32 **giga**bytes
- Often no operating system
- Used in consumer electronics, robotics, automotive

Real-time Systems

- Applications where the program must perform an action **within a deadline** to be considered "correct" or "operational"
  - Robotics, telecommunications, automotive
  - Financial systems, high-frequency trading
  - Video games, online multiplayer
- C doesn't "solve" these problems, but it's frequently used when programmers want to control the program execution details

…but C has competition in 2023!

- **Rust** is gaining adoption in embedded and real-time systems
- **Go** is frequently used for modern "systems programming"
  - Databases, network services, command-line tools
- **C++** gets an "honorable mention", but has its own niches

C Development Tools

What are header files for? (`#includes`)

- Header files create *abstractions*
  - Control which parts of the code are "public" (visible to other files)
- Languages like Java use keywords like "`public`" instead
- Case study: `stdio.h`
  - `FILE *fopen(const char * pathname, const char * mode)`
  - "`FILE`" is a struct, but `stdio.h` doesn't expose its internals

Making friends with the compiler ❤️

- Warnings
  - Recommended flags: **`-Werror -Wall -Wextra -Wconversion`**
  - The compiler needs to support ~30 year old code, warnings can point to parts of the code that are likely buggy even if it "usually" works.
- Optimization levels
  - Flags: **`-O0`**, `-O1`, **`-O2`**, `-O3`, `-Og`, `-Os`, ...
  - If your code has *undefined behavior*, more likely it may break at higher optimization levels!

Why did my program crash?

- **`gdb`** – Debugger (or look for a debugger in your favorite IDE)
- **`valgrind`** memcheck – Detects *some* memory errors
  - Mistakes around `malloc` + `free`
- **`kcov`** – Code coverage: which lines of code actually ran?
- Compiler flags to enable *debug symbols*: **`-ggdb3`** (or **`-g`**)
  - If the output seems unreadable, add one of these flags

Why is my program slow?

- 4 main limits: **CPU, memory, disk, and network**
- Algorithms can help when **CPU**-bound (generally)
- C gives you a lot of control to optimize **memory** layout & usage
- `perf` (Linux) – Sampling profiler
  - "What functions is my program spending time in?"

What is going on?

```
SYNOPSIS      top

       #include <stdlib.h>

       void *malloc(size_t size);
       void free(void *ptr);
       void *calloc(size_t nmemb, size_t size);
       void *realloc(void *ptr, size_t size);
       void *reallocarray(void *ptr, size_t nmemb, size_t size);
```

- Manpages (e.g. `man malloc`)
- cdecl.org – "C gibberish ↔ English"
- *Modern C*, by Jens Gustedt (PDF)
- godbolt.org – Compiler Explorer (*advanced*)