# The C Programming Language

Peter Chapin

CIS-2730, IoT

Vermont State University

# A Brief History of C (part 1)

- Dates to the early 1970s (about)
  - Created for *systems programming* (operating systems, device drivers, machine control applications, etc.)
  - The Unix kernel was written in C
- Defacto standard programming language on Unix
  - Unix utilities written in C. Most Unix servers are written in C (Apache, MySQL, DNS, etc.)
  - Spread to other operating systems due to its high efficiency (fast, small executables)
  - Became the industry-standard language for writing operating systems and device drives

# A Brief History of C (part 2)

- First ISO standard published in 1990 (aka “C90”)
  - Modern C is still pretty much the same!
- Updated standards in:
  - 1999 (“C99”)
  - 2011 (“C11”)
  - 2017 (“C17”). No new features, just “bug fixes” to the standard
  - 2023 (“C23”). It has not been published yet (as of January 2024), but any day now!
- Adoption of new standards is slow. The community isn’t as fixated on using the latest standard as in other communities. **C99** is a good base

# Embedded Systems

- An *embedded system* is a computer system that is embedded in something else
  - Controller for a microwave
  - Software that runs a TV
  - Controller for automotive systems (e.g., anti-lock breaks)
  - Medical devices (blood pressure measurements, glucose measurements)
  - The keyboard for a desktop computer
  - *Many others!*
- The total number of embedded processors *far* exceeds the number of laptops, desktops, servers, and phones combined!

# Embedded Programming

- The range of systems is extreme
  - From highly constrained systems with tiny memories (16 KB) and slow processors (1 MHz)…
  - … to systems running high-end, multi-core CPUs with huge memories (many GB)
- The range of operating systems is also extreme
  - From no operating system at all…
  - … to specialized operating systems that can run in a highly constrained environment…
  - … to full-scale operating systems like Linux and Windows

# C for Embedded Systems

- C dominates the embedded systems market
  - About 80% of all embedded software is in C
  - Because of its efficiency, C is hard to beat when programming devices on the very constrained end of the spectrum.
- If you ever work in the (gigantic) embedded systems market *you must know C!*

# Examples

- Here are some examples of C usage
  - The Linux kernel. Rust is being used experimentally; otherwise, the kernel is entirely in C.
  - Oracle's Java Virtual Machine. The software that interprets and executes Java programs is itself a C program.
  - The Python interpreter. Again, it is a C program that ultimately executes Python code.
  - Internet servers. Many are in C, although not all.
  - Device drivers for the various hardware components you add to your system are mostly all in C.
  - NASA uses C99 for the flight software on all their recent missions.

# (Dis)Advantages of C

- C is very low level
  - The language does almost nothing automatically
  - “If you don’t write it, it doesn’t happen.”
  - No automatic checking for errors (you have to write the checks yourself)
  - No automatic memory management (you have to manage memory yourself)
  - No automatic initialization of objects (you have to initialize things yourself)
- PRO: You have complete control. *The program does what you say and only what you say*
- CON: **It is easy to forget to do important things!**

# Unfortunately…

- Because C puts so much responsibility on the programmer, most C programs have many errors due to programmer oversight
  - This leads to buggy software that crashes
  - This leads to security vulnerabilities
- What to do?
  - **Be careful**. Easier said than done
  - **Use tools**. Many tools analyze C programs, looking for potential problems. *Use them aggressively*.
  - For example, use `–Wall` with GCC to generate “all” warnings. *Treat the warnings as if they are errors*. **Do not ignore warnings!**