A Crash Course in C

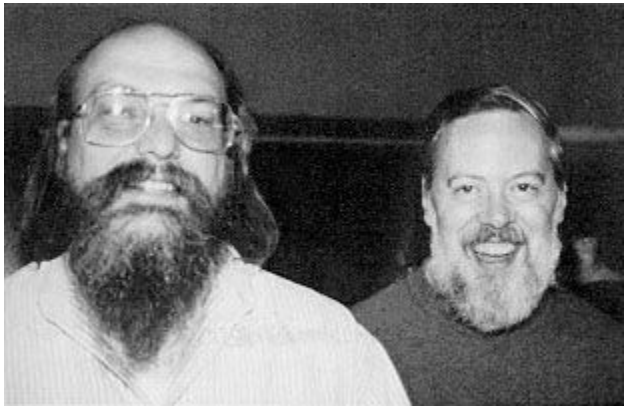
GDSC Reading Week Workshop 3

Admin Stuff

- A CSC209 disclaimer
- Getting the demo code
  - SSH into the DH2020 Lab machines using the Remote-SSH VSCode extension
  - Open terminal (Ctrl + `) in VSCode, and run:
    - `git clone https://github.com/utmgdsc/2023-c-workshop`
  - This should download the GitHub repository, and you should have access to the files

What is C?

- A statically-typed, compiled programming language
- Developed by Ken Thompson and Dennis Ritchie for UNIX



The Impact of C

- Implementation language of numerous software tools
  - All modern operating systems (Linux, Windows, MacOS)
  - The Python interpreter
  - Java Virtual Machine
  - Anything low-level, really
- Influenced numerous other programming languages

Influenced
Numerous: AMPL, AWK, csh, C++, C--, C#, Objective-C, D, Go, Java, JavaScript, JS++, Julia, Limbo, LPC, Perl, PHP, Pike, Processing, Python, Rust, Seed7, Vala, Verilog (HDL),[5] Nim, Zig

Python vs. C

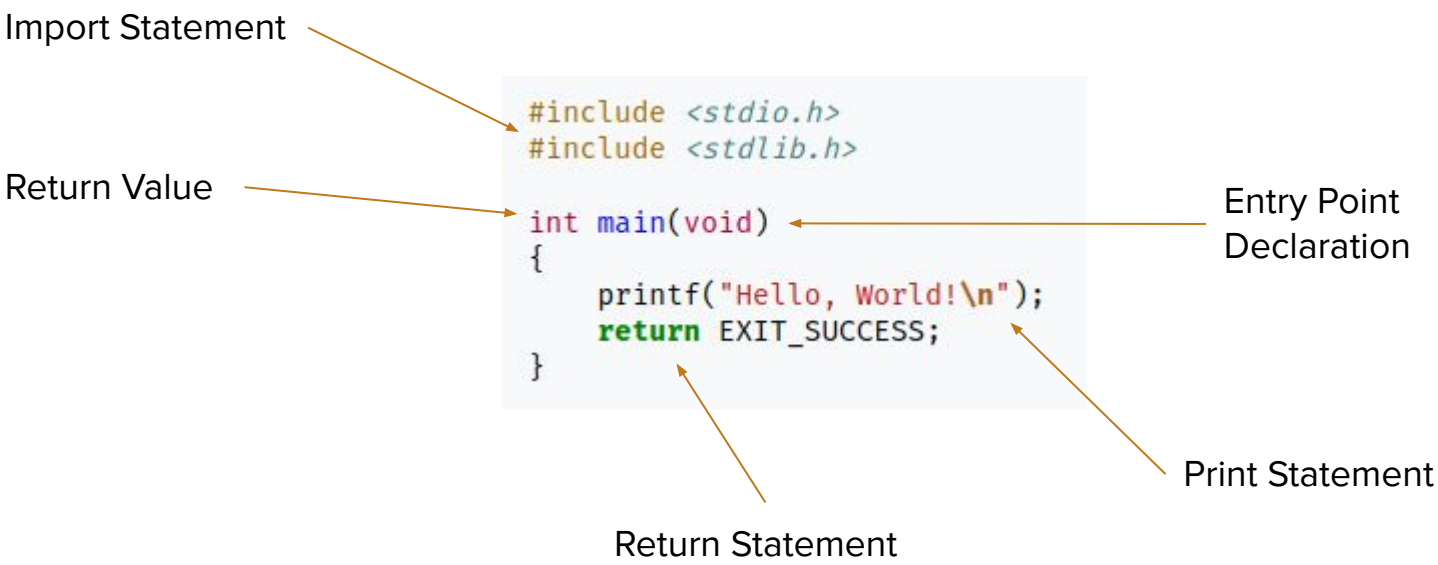
Python

- Runtime:
  - Interpreted
- Type System:
  - Dynamic, strong
- Paradigm:
  - Object-oriented, functional
- Easy to use, beginner-friendly
- Slooooooow

C

- Runtime:
  - Compiled
- Type System:
  - Static, weak
- Paradigm:
  - Structural/Imperative
- Difficult to master, memory management is a pain
- Blazing fast

Anatomy of a C Program



Control Flow and Data Types

- Conditional Statements
  - If-Else, Switch-Case
- Loops
  - For and While Loops
- Function Calls
- Type Casting

- Various Flavors of Integers
  - Int, unsigned int, long, long long
- Floats and Doubles
- Char (characters)
- Booleans (with a caveat!)
- Arrays
- Structs (create your own type!)
- Pointers
- Strings...?

C is a very simple language,
but very hard to master!

Pointers: A Motivating Example

How many bytes do data types use?

- Integers
  - Default 4 bytes (32 bits), longs are 8 bytes (64 bits)
- Characters
  - Always 1 byte (8 bits)
- Booleans
  - Always 1 byte
- Arrays
  - However big the programmer declares them to be
  - Size is known at compile time

How Big is a String?

- “Hello World!” has 12 characters, so 12 bytes
- “I like Dan Zingaro’s cows” has 25 characters, so 25 bytes
- A string can be 1 byte, 20 bytes, 420 bytes, 32,000 bytes...

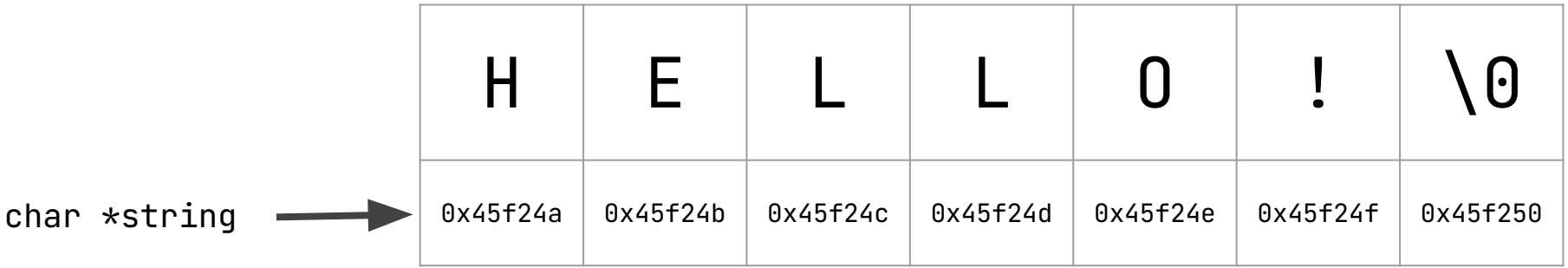
We don’t know how big a string is!

(Rather, strings have arbitrary size that is not always known at compile time)

Pointers (for real this time)

- Stores the memory address of an object
  - Rather than the object itself
  - “Points” to the object
- Closest analogue: object IDs in Python
- Pointers are 4 bytes (32 bits) or 8 bytes (64 bits), depending on the system
- Indicated by an asterisk * in C
  - e.g. `int *c` declares a pointer (named c) pointing to an integer
- Pointer operators
  - ‘&’ (The reference operator)
  - ‘*’ (The dereference operator)
- How strings are stored in C!
  - `char *`, or `char[]`

Strings in Memory



C’s Memory Model

The Stack (a.k.a the call stack)

- Most sized variables live here
  - Integers
  - Pointers
  - Predefined arrays
  - Etc etc.
- Most items get allocated and deallocated automatically when a function returns
- Size of stack variables cannot be changed

The Heap

- Used for storing variables of unknown size
  - e.g. strings
- Size of heap variables can be changed
- Always interacted with through pointers
- Must be managed manually by the programmer
  - The reason why memory management is so important!

Memory Management

Memory Allocation (`malloc`)

- Allocates a block of memory on the heap
  - Returns a pointer to the allocated block
- Takes one argument specifying how many bytes the programmer wants
- Comes in various flavors:
  - `calloc` - zeroes out the memory before being used
  - `realloc` - resizes an existing block of memory
- If allocation fails, returns NULL

Memory Deallocation (`free`)

- Deallocates the memory being pointed to by a given pointer and frees it up for other use
- Takes one argument, a pointer that points to the block of memory to be deallocated
  - The block to be deallocated must have been previously allocated, if not bad things happen!

Memory Management Best Practices

- The GOLDEN RULE of Memory Management:
  - Anything allocated must always be freed after use.
  - For every malloc there must exist one (and only one) free!
- Avoid aliasing
- Avoid dangling pointers
  - Any dangling pointers should be set to NULL
- Watch out for out-of-bounds errors!

What can go wrong?

- Segfaults
  - Out-of-bounds errors
- Use-after-free
  - Usually results from aliasing
- Buffer Overflow
  - Usually from writing to memory that is not allocated
- Security Vulnerabilities!

Structs and Typedefs

Structs

- Allow programmers to create custom data types
- Provide ways of grouping related data under a single name
- This is done by defining fields within a struct

Typedefs

- Allow programmers to create type aliases for existing types
- Can be used to redefine existing types to a more relevant name, or define a struct to use more succinct syntax

Pointers and Structs

- Pointers are often used with structs
  - C is a pass-by-value language, so pointers allow pass-by-reference
- Common syntax includes:
  - `struct some_struct *var_name;`
  - `typedef struct somestruct *structptr;`
  - `structptr ptr = malloc(sizeof(some_struct));`
- Same memory management rules apply!

Putting It All Together

A Linked List!

- What we will implement:
  - Append
  - Push
  - Remove
  - Index

Homework!

- Figure out recursion
  - Implement linked list operations recursively
  - Implement a recursive fibonacci number calculator
- Implement a dynamic string read function
- Implement a binary search tree
- Implement one of your existing (smaller) projects in C
- Do some reading
  - History of C and UNIX
  - How a C program is compiled
  - How `malloc` and `free` work
  - “The C Programming Language” by Brian Kernighan

Thank You!

Any Questions?