CS 360 Perspective

* Programmer centric: show that by knowing more about the underlining system, one can be more effective as a programmer
* Write programs that are more reliable and efficient
* Incorporate features that require hooks into os, ex: concurrency, scheduler
* Cover material that you wont see elsewhere

You need to access WSU onedrive to download materials

There are 4 assignments that are individual and each has 9 days to complete

What is C good for?

* One of the oldest languages
* Close to the physical hardware
* We can use for: embedded systems, building compilers (Haskell, c++), interpreters (perl, python), building OS kernels, device drivers, debuggers

How popular is C?

* Check slide for grach of popularity over time

Pointers

* Allow the programmer to manipulate memory directly
* Theyre variables that contain a memory address
* The data contained at the address is inferred by the pointer type. Ex: int\* (ptr to an int), int\*\* (ptr to a ptr to an int)
* Be very careful when declaring multiple pointers on the same line. Ex: int\* foo, bar foobar; // only foo is a pointer
* We can initialize a ptr by setting it to the address of a variable ex: int\* pFoo = &foo
* More syntax on slides
* Pointers contain an address, not a value. To get the value you have to dereference it with the \*.
* Pointers are useful for passing by reference instead of by value. This allows the programmer to modify a variable from within a function, and retain the change without returning the modified variable
* Useful for passing structs. Structs are passed by value by default, passing by reference is more efficient. **Check slide 27** for right and wrong syntax.

Memory

* Programs typically use 3 areas of memory:
  + Stack: stores local variables determined at compile time
  + Heap: dynamically allocated memory at run time
  + Global data: values loaded from the binary

Stack

* + The code in slide 29 is wrong because int foobar is declared in the foo1 function so it will go out of scope when the function call ends, so the stack memory of foobar will be deallocated (dangling pointer).
  + The other code in slide 29 is wrong only because foo1 is still wrong lol

Heap

* + Sometimes we don’t know at compile tie how much memory we will need, so dynamic memory comes in handy
  + Dynamic memory is issued from the heap which is managed by the OS
  + The heap is typically much larger than the stack by default
  + Storage is requested form the heap using **malloc system call** which accepts size in bytes and returns a pointer to a block of heap memory
  + Malloc returns a void pointer, so if you want a pointer to int memory, you need to cast it. Ex: **(int\*)malloc(sizeof(int))**. Sizeof(int) returns 4 for 4 bytes.
  + The example in slide 29 is good because \*bar was allocated on the heap using malloc which is global so it does not go out of scope when the function call ends. Stack is local to your function, heap is global.

Command line arguments

* These are data sent to the program when you run it. Programs execute with different data. You do not need to change the program
* They are passed to main. Two arguments: the number of command line arguments, an array of character strings with the command-line arguments
* These formal parameters are argc, argv. The name of the program is in argv[0]. Ex: int main(int argc, char \*argv[])
* **Slide 33** for example on using command line arguments that are passed into the program. We subtract 1 in the print statement in the example because the first arg is the program’s name so we don’t want to include it in our calculations.

Compilation

* Compiling a C code: **gcc -o hello hello.c**. this will give a binary file that you can run using ./hello
* A better way is to use the make program in linux/unix. Reads a makefile and performs automatic compilation. A makefile contains: target (files to be created), dependency list (source files), and rules (commands).
* **Slide 35 - 38** for makefile examples. in example 3 you just type all and everything below that is printed and ran automatically (bold lines are what was typed by the user).