C - Week. 5 Notes

L1 Why Functions ?

- compiler output – disassembly
  ① conti. block of instruc. in memory
  ② difficult to follow struc.
  ③ difficult to reuse code
- struc. programming
  ① restrict the subset of allowed prog. constructs, easier to understand
  ② reusability of secⁿˢ of code.
- Core ideas → DRY
  ① modularity - self contained modules that do specialized work
  ② reusable - same code in multiple contexts
  ③ generalized - use arguments to ↑ capability.
- Func ^ struc. → blocks
  bcq. of statements, grouped in { }

```
{
  ===;
}
```

give it an name to make it reusable.

arguments → generalizations

L2 Func^{n} in C

- declara^{n} - names, inputs, outputs; could be indep. of def^{n}, often moved to header files
- def^{n} - block of code
- Func^{n} properties —
  1. name - similar restrictions as variables
  2. variables - own storage, indep. of others
     ↳ impact on memory usage.
  3. visibility - scoping rules, mechanisms to call/invoke func^{ns} from other sources.
  4. Return values - type of return specified, multiple return need tricks.

L3 What is a Runtime?

- code - executable instructions
- data - instructions operate on data (variables)

→ Where should var. go?
→ What data should a func^{n} see?

- Calling Func^{ns}
  1. Code of func^{n} loaded in memory, lookup table in executable file maps func^{ns} to addresses.
  2. Arguments - load them registers or in memory.
  3. Return values - passed back thru registers or memory.

- Stack - Each func^{n} needs place to store data.
  - ↳ Runtime - func^{ns}, code that handles memory alloca^{n}, func^{n} calling, etc. Allot some memory to each func^{n}.
  (eg. PythonTutor.com)
- Scope - extend block lvl. visibility to func^{ns}.
- Stack Frames - Each func^{n} is allocated a block of memory. Mostly in chunks of fixed size.
  - → Calling Func^{ns}
    1. save existing stack address.
    2. Go to new func^{n} with own stack frame.
    3. Return - restore old stack.

L4 Recursion

- Function Calling
  - → What is a Func^{n}? → Code: CPU jumps to add. of first instruc^{n}, executes; Data: Stack frame to hold variables.
  - → Which Func^{ns} can be called? → Scope: Compiler decides visibility; Name: after compiling a func^{n} is just instructions.
- Recursion → Func^{ns} calls itself. But diff. parameters / arguments.
  - → Base Case - combina^{n} of arguements that has known return value.
  - → Recurrence - express result of func^{n} in terms of same func^{n} with diff. parameters.

→ Induc^n Process - Solve problem by breaking into simpler recursions of same problem.

L5 Scopes in C

- Variables → storage? who can CRUD? In multiple prog., reuse name? How to control visibility & scope of data?
- Where is a var. declared? Can we use a var. w/o declaring it? Declare once, use anywhere? Next func^n? Hide names temporarily for convenience? lifetime?
- C uses simple block level scoping rules.