$\boxed{\text{CSE 13S}}$

$\boxed{9/24}$ - Design PDF first, then start coding
- ssh = secure shell
- scp = secure copy
- sftp = secure FTP
- git tracks files, allows collab (w/ prof & TAs) ⇒ Source code contro
- don't mess w/ .git contents (clone again otherwise)
- vi = standard text editor on Unix
- Vim = Vi IMproved (clone of Vi editor)
  - ↳ everything on Vi is available on Vim
- README.md must be in Markdown
- DESIGN.pdf {answers pre-Lab q's} {describes algorithms & problem, input, output
- WRITEUP.pdf (analysis of running code) (ex: talk about results of code)

[Pics, diagrams, pseudo-code]

- DESIGN due THRS, code assignments due SUN
- git add → commit → push (do w/ all edits)
- git pull for git file to local device (opposite way)

$\boxed{9/27}$ - #include <stdio.h> adds standard I/O package/library
- ~~[illegible]~~ int main(void) main function, void = no arguments
- "cc" for standard compiler ⇒ "cc -o hello hello.c" works instead of clang

- temperatures
    - 0K = absolute zero
    - $°C = K - 273.15$
    - $°F = °C \times 9 \div 5 + 32$
    - print °F to °C table

```
#include <stdio>
int main(void) {
    float fahr, celsius;
    int lower=0, upper=300, step=20;
    fahr = lower;
    while (fahr <= upper) {
        celsius = (5.0/9.0) * (fahr-32);
        printf("%3.0f%6.1f\n", fahr, celsius);
                formatting
    }
    return 0;
}
```

- chars = small ints ⇒ getchar() returns an int
- types = float, int, char

- each pair of { } = scope
- scope of a variable tells where it exists

- EOF = end of file

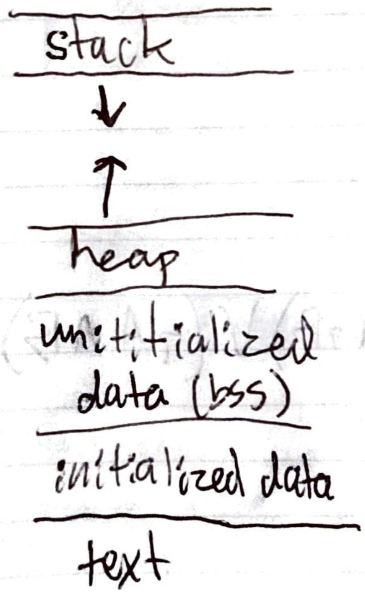
- double quote around string, single quote around char

- #define NAME ~~NUMBER~~ number ⇒ program reads NAME as a NUMBER where the code has NAME

- language translator = program, maps input lang to output lang
- compilation process

$$\begin{tikzcd}
\text{source code (hello.c)} \arrow[r] & \text{pre-processor} \arrow[d, "\text{(hello.i)}"] & \\
& \text{compiler} \arrow[r] & \text{assembly code (hello.S)} \arrow[dl] \\
& \text{assembler} \arrow[r] & \text{object code (hello.o)} \arrow[dl] \\
\text{libraries} \arrow[r] & \text{linker} \arrow[d] & \\
& \text{executable (hello)} &
\end{tikzcd}$$

- memory



- interpreter → directly executes code (doesn't compile)
- compiler → translator, go thru translations & outputs executable
- faster one?

- gcc: GNU C compiler, Linux default
- cc: Unix/Linux variable that points to default compiler
- clang: Mac & FreeBSD default

  ↑
  USE THIS IN ASSIGNMENTS!!

- Boolean
  - ∧ = and = conjunction
  - ∨ = or = disjunction
  - ¬ = not = negation

$A \wedge 1 = A$

$A \wedge 0 = 0$

$A \wedge A = A$

$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$

AND

∧(and)	T	F
T	T	F
F	F	F

OR

∨	T	F
T	T	T
F	T	F

NOT

¬	T	F
T		F
F		T

- Exclusive-or (one or other but not both)

⊕ (XOR)	T	F
T	F	T
F	T	F

↳ $A \oplus B = (A \vee B) \wedge \neg(A \wedge B) = (A \wedge \neg B) \vee (\neg A \wedge B)$

$A \oplus A = 0$

$A \oplus 0 = A$

$A \oplus 1 = \neg A$

$A \oplus (B \oplus C) = (A \oplus B) \oplus C$

- De Morgan's Law: $\neg(A \vee B) = \neg A \wedge \neg B$

  $\neg(A \wedge B) = \neg A \vee \neg B$

- 0 = false (nothing else is false)
- logical expressions have type int
- can have T & F if add #include <stdbool.h>

- if() executes next statement if bool expression is [illegible] true
- always use {} w/ if() to avoid errors
- always use & in front of variable in scanf(%_, variable)

- false && anything = false
- true || anything = true

- switch () ← same requirement/statement
  case 1:
  ....
  break
  case 2:
  ...
  break
  default:
  (not applicable to other cases)

- switch () allows you to select among fixed set of alternatives
- break takes you to the end of the function

- focus on while, for, and do-while in CSE 135
- don't use goto

- while: top-test loop → tests statement in while() first
- executes statement as long as it's true

- for: also top-test loop → parts: initalization
  test
  increment

- any loop can be written as a while loop
- do { } while() = bottom_test loop
  - used when you want to perform statement at least once
  - continues to execute the enclosed statement as long as condition = true
- infinite loops execute forever (escape w/ break)

- = is assignment
- == is equality

- continue jumps back to start of loop
  - use sparingly

- function = block of code that performs a certain task
  - are defined only once
  - must be declared before used
- programs can declare & call functions as many times as wanted
- main() = special function, runs when program starts
  - all other functions are subordinate to it

- functions should
  - define abstractions (consistent & logical)
  - give names to sequences of code
  - hide implementation
- functions can
  - refactor repeated code
  - simplify code for understanding
- functions should never be arbitrary statement sequences

return type function_name(parameters) ← function head
{
  //declarations, assignment statements   function block/body
}

- return_type defines type of return value
  - may be void or any object type other than array
- function_name = function's name
- parameters ~~[illegible]~~ contained in comma-separated declarations list
  - if no parameters, () is empty or has void
- function block/body = declarations (declared variables that are only locally known) & assignment statements (set and/or resets variable values)

- return values can be void (no value)
  - can return any scalar value (char, int, float)
  - can return pointer, ~~[illegible]~~ can return struct (but don't do this)
  - cannot return array

- function naming → same rules as variable naming
  - can't start w/ _ or $ or another func name
  - there are no nested funcs in C
  - can start w/ # or any other punctuation except

- parameters/arguments
  - ~~[illegible]~~ "call by name" is rare → C Preprocessor supports it
  - C uses "call by value" except for arrays & only because they're pointers (pointers make them seem like "call by reference")

- parameters = name of value that's passed to function
  - can be copied to formal parameter
  - reference may be ~~[illegible]~~ bound to formal parameter (the parameter used inside function body)
    - do this by using call by value by passing pointer
- call by value used by all funcs in C
  - arguments passed in func are copied
  - copy of actual parameter places in formal parameter

- swap(int *a, int *b) {

```
  int temp = *a;
  *a = *b;
  *b = temp;
  return;
}
```