CSE 306 Software Quality in Practice

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall
Spent 25 minutes trying to debug a syntax error. Turns out the database connection dropped and I had been typing queries into the bash terminal 😞
LEX08
.h and .c
When stepping through code with debugger, why are declarations skipped?

```c
int foo() {
    int x;
    double y;
    y = f(x) * 3;  // why does debugger skip to here?
    ...
}
```
Declarations are handled by compiler at compile time. They have no run-time analogue.
EXP01

- Team-based: use PRE teams
  - shown in UBLearns gradebook
- Demo
Modeled development process

start

read & understand specifications

write tests

/run tests

fail

implement

pass
The final digit of a Universal Product Code is a check digit computed as follows:

1. Add the digits in the odd-numbered positions (first, third, fifth, etc.) together and multiply by three.
2. Add the digits (up to but not including the check digit) in the even-numbered positions (second, fourth, sixth, etc.) to the result.
3. Take the remainder of the result divided by 10 (modulo operation) and if not 0, subtract this from 10 to derive the check digit.

https://en.wikipedia.org/wiki/Check_digit#UPC
3 \times (0+6+0+2+1+5) = 3 \times 14 = 42

3+0+0+4+4 = 11

42 + 11 = 53
53 \mod 10 = 3
10 - 3 = 7
If c is a char from '0' to '9', how can you convert it to an int from 0 to 9?
If c is a char from '0' to '9', how can you convert it to an int from 0 to 9? Without knowing any library functions:

```c
int convert(char c) { return c - '0'; }
```
If c is a char from '0' to '9', how can you convert it to an int from 0 to 9? Without knowing any library functions:

```c
int convert(char c) { return c - '0'; }
```

```c
int convert(char c) {
    switch (c) {
        case '0': return 0;
        case '1': return 1;
        ...
        case '9': return 9;
        case 'X': return 10; // CAN ALSO HANDLE 'X'
    }
}
```
GNU make

- Who has invoked 'make'? 
- Who has modified a make file? 
- Who has written a make file from scratch?
make
What is it good for?

"You can use [make] to describe any task where some files must be updated automatically from others whenever the others change."

make and makefiles

- makefile contains rules that describe update dependencies
rules

target : prerequisites
recipe
rules

- target : prerequisites
  - recipe

  Must be a tab!
target

- A target is usually the name of a file that needs to be generated/updated during the 'make' process.
- The rule will be used by 'make' when the target is out-of-date, and so should say how to update the target.
A target is usually the name of a file that needs to be generated/updated during the 'make' process.

The rule will be used by 'make' when the target is out-of-date, and so should say how to update the target.

"Bear in mind that make does not know anything about how the recipes work. It is up to you to supply recipes that will update the target file properly. All make does is execute the recipe you have specified when the target file needs to be updated." [p. 5]
target


gcc -c -Wall primOpt.c
A target can be "phony" - an arbitrary label for an action given by the rest of the rule.
target

clean:

```
rm -f primOpt.o main
```

"...the clean target will not work properly if a file named clean is ever created in this directory. Since it has no prerequisites, clean would always be considered up to date and its recipe would not be executed. To avoid this problem you can explicitly declare the target to be phony by making it a prerequisite of the special target .PHONY" [p. 29]
target

.PHONY: clean
clean:
    rm -f primOpt.o main