Tutorial 3

• Review: side effects

• Reading input

• Global variables & mutation

• Loops

```c
while (!understand(loop_knowledge)) {
    ++loop_knowledge;
}
```

CS 136 Winter 2018  Tutorial 3 1

Side effects

(there are 3 types of side effects in functions for now)

• print output

• read input

• mutate a global variable

// MANY SIDE EFFECTS

```c
int y = 2;

int main(void) {
    int x = read_int();
    printf("%d", printf("%d", y));
    if ((x = y - 1)) {
        printf("%d", x = x + 2);
    }
    if (x) {
        y *= y;
        printf("%d", y);
    }
    printf("\n");
}
```

CS 136 Winter 2018  Tutorial 3 2

CS 136 Winter 2018  Tutorial 3 3
Side Effects: printing vs. returning

```c
int pure_functional(int n) {
    return n * n;
}

// effects: displays a message
void just_a_side_effect(int n) {
    printf("n squared is %d\n", n * n);
    return;             // (optional)
}

// effects: displays a message
int has_both_side_effect_and_return_value(int n) {
    printf("n squared is %d\n", n * n);
    return n * n;
}
```

Read Input: Exercise

Modify the code below to read in input and then prints out the average of the input. For example:

- If the input is 1, 2, 3, 2
- It should print out 2.

```c
int main(void) {
    int n = 0;
    while (1) {
        n = read_int();
        if (n == READ_FAIL) {
            break;
        }
    }
}
```

Global variables & mutation

- Global variables are defined outside of functions (at the “top level”).
- A function that mutates a global variable does have a side effect.
- Even if a function does not have a side effect, its behaviour may depend on other mutable global variables.
Loops: for loops & while loops

- Using a loop to solve a problem is called **iteration**.

- **while** is similar to **if** statement but while **repeatedly** "loops back" and executes the statement **until the** expression is **false**.

- General format of a **while** loop:
  ```c
  setup statement
  while (expression) {
    body statement(s)
    update statement
  }
  ```

- **for** loops are a "condensed" version of a **while** loop.

for vs. while

Recall the **for** syntax.

```c
for (setup; expression; update) { body statement(s) }
```

This while example

```c
i = 100; // setup
while (i >= 0) { // expression
  printf("%d\n", i);
  --i; // update
}
```

is equivalent to

```c
for (i = 100; i >= 0; --i) {
  printf("%d\n", i);
}
```

Loop: Exercise

Define the following C function: (use iteration)

```c
// draw_circle(size) draws a circle inside of a square
// with dimensions (size * 2 + 1)
// requires: size >= 1
// effects: produces output
```

- There is a simple example of the output in "simple.expect"