While Statements

- We often need to execute code (statements) many times.
- **if** statements only allow us to execute or not execute code. in other words they allow us to execute code 0 or 1 times
  while loops allow us to execute code 0 or more times
- Like **if**, **while** loops have a condition
  but **while** statements execute their body until
  the condition becomes false

```
while (CONDITION) {
  stmt1;
  stmt2;
  ...
  stmtn;
}
```

While Statements

- C has other looping constructs - but **while** is all you need
- **for** loops can be a little more concise/convenient
  we'll see them later - for now use **while**
- Often use a **loop counter** variable to count loop repetitions
- Can then have a **while** loop execute **n** times.

while Loop - Loop Counter Example

```
// read an integer n
// print n asterisks
int loop_counter, n;

printf("How many asterisks? ");
scanf("%d", &n);

loop_counter = 0;
while (loop_counter < n) {
   printf("*");
   loop_counter = loop_counter + 1;
}
printf("\n");
```

`while` Loop - Loop Counter Pattern

Here is the programming pattern for a while that executes *n* times:

```
int i = 0;
while (i < n) {
    //
    // statements the loop needs to perform
    //

    i = i + 1;
}
```

While Statements - Termination

- Can control termination (stopping) of while loops in many ways.
- Easy to write **while** loop that do not terminate.
- Often a **sentinel** variable is used to stop a while loop when a condition occurs in the body of the loop

while Loop - Sentinel Variable Example

```
// read numbers printing whether even or odd
// stop if zero read
int stop_loop, numbers;

stop_loop = 0;
while (stop_loop != 1) {
    scanf("%d", &number);
    if (number == 0) {
        stop_loop = 1;
    } else if (number % 2 == 1) {
        printf("%d is odd.\n", number);
    } else {
        printf("%d is even.\n", number);
    }
}
```

while Loop - Sentinel Variable Pattern

Here is the programming pattern for a while loop that executes until the sentinel variable is changed.

```
stop_loop = 0;
while (stop_loop != 1) {
    //
    // statements the loop needs to perform
    //
     if (.......) {
         stop_loop = 1;
     }
    //
    // perhaps more statements
    //
}
```

Nested While Loops

- Often need to nest while loops.
- Need a separate loop counter variable for each nested loop.

```
// print a square of 10x10 asterisks

int i = 0;
while (i < 10) {

    int j = 0;
    while (j < 10) {
        printf("* ");
        j = j + 1;
    }

    printf("\n");
    i = i + 1;
}
```

Nested While Loops

- Nested while loops are used when repetition of repetition is required.
- This often happens in problems which have a two-dimensional nature, such as printing a square of asterisks.
- Remember to reset the value of the inner while loop's counter variable each time it runs!