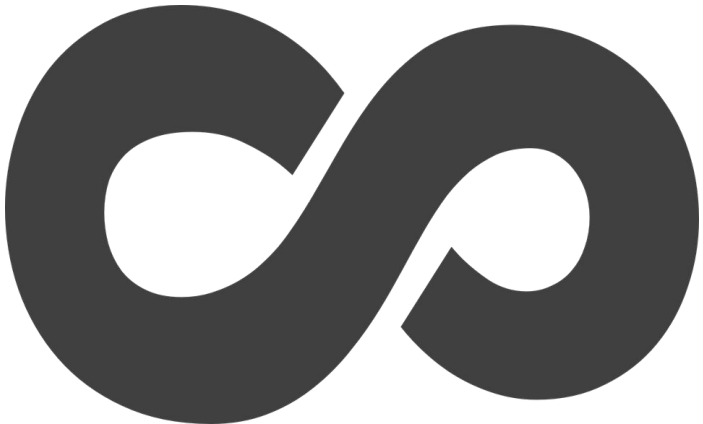
While Loops

Repetitions

- Used to: repeat a certain step in an algorithm
  - Easier to use/code than copying and pasting code
  - Extremely useful for an unspecified number of repetitions

Repetitions

- 3 Ways to do repetitions in C…
  - 1) While loops
  - 2) For loops
  - 3) Do while loops → less important, but will cover them



While Loops

```
While (/* Some condition */) {

    /* Code goes here */

}
```

- Structurally, it is extremely similar to *if* statements

While Loops

```
While (/* Some condition */) {

   /* Code goes here */

}
```

- The loop will repeat, as long as the *condition* is true

While Loops

- Terminating a while loop (2 main ways)

While Loops

- Terminating a while loop (2 main ways)
  - 1) *Terminating Condition*
    - Runs until some condition is reached
  - 2)*Counter*
    - Tells the loop how many times to run

Terminating Conditions

- Reacts to changing variables within the while loop

```
int foo = 10;
int user_input;
While (foo < 100) {
   scanf(“%d”, &user_input);
   foo = foo + user_input;
}
```

Terminating Conditions

- Reacts to changing variables within the while loop

```
int foo = 1;
While (foo != -1) {
   scanf(“%d”, &foo);
   printf(“Foo is %d”, foo);
}
```

ASCII Coding: Part II, Basic Cipher

- Scan in an unspecified amount of characters - store them as *char* variables
- Stop scanning in characters when the user enters “?”

- ADD 10 to the value of the scanned characters
- If the characters are now out of the CAPITAL letter range, print “Not a capital letter” ➔ otherwise, print the *char* and *int* of the variables
    - Use the ASCII table for reference

Terminating Conditions

- This kind of loop can be an *infinite* loop
  - Keeps looping until a special condition is reached

```
int foo = 1;
While (foo != -1) {
   scanf(“%d”, &foo);
   printf(“Foo is %d”, foo);
}
```

Counter

- Tells the loop a specific number of times to run

```
int j= 0;
while (j < 10) {
   printf(“Skip Day Penalty\n”);
   j = j + 1;
}
```

Counter

- This *counter variable* keeps track of how many times the loop runs
- Commonly short letter names (i, j, k)

```
int j= 0;
while (j < 10) {
   printf(“Skip Day Penalty\n”);
   j = j + 1;
}
```

Counter

- Three main parts of a loop
  - 1) **Initial condition**

```
int j= 0;
while (j < 10) {
   printf(“Skip Day Penalty\n”);
   j = j + 1;
}
```

Counter

- Three main parts of a loop
  - 2) **Terminating Condition**

```
int j= 0;
while (j < 10) {
   printf(“Skip Day Penalty\n”);
   j = j + 1;
}
```

Counter

- Three main parts of a loop
  - 3) **Update**

```
int j= 0;
while (j < 10) {
   printf(“Skip Day Penalty\n”);
   j = j + 1;
}
```

Counter

- Can count up or down (standard is up, but down is fine too)

```
int j= 10;
while (j > 0) {
   printf(“Skip Day Penalty\n”);
   j = j - 1;
}
```

Nested While Loops

- Like *if* statements, *while* loops can be nested within each other

```
While (some condition) {
    While (some other condition) {

    }
}
```

While Loop Challenge

Scan in two numbers: a width and a height

Using only while loops, draw a box of asterisks (*) to match the scanned in numbers

Hint: Use nested while loops