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** statements allow us to execute code 0 or more times.
While Statements

• Like `if`, `while` statements have a controlling expression but `while` statements execute their body until the controlling expression is false

```plaintext
while (EXPRESSION) {
    stmt1;
    stmt2;
    ...
    stmtn;
}
```
while Loop - Loop Counter Example

Often use a **loop counter** variable to count loop repetitions. We then have a **while** loop execute **n** times.

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

printf("How many asterisks? ");
scanf("%d", &n);
// initialise our loop counter
loop_counter = 0;
while (loop_counter < n) {
    printf("*");
    loop_counter = loop_counter + 1;
}
```
while Loop - Loop Counter Pattern

Here is the programming pattern for a while that executes \( n \) times:

```javascript
// initialise loop counter
loopCounter = 0;
while (loopCounter < n) {
    // statements the loop needs to perform
    //
    // increment loop counter
    loopCounter = loopCounter + 1;
}
```
While Statements - Termination

It is easy to write `while` loops that do not terminate. A classic example is forgetting to update the loop counter

```c
// i never gets updated so is always 0
// this is an infinite loop!
i = 0;
while (i < n) {
    printf("*");
}
```

Note: You may need to hit `Ctrl C` to stop your program if you accidentally do this.
Sometimes instead of a loop counter, we use a **sentinel** variable. A sentinel variable is a variable that is used to stop a while loop when a condition occurs in the body of the loop.

```c
// read and print numbers
// stop when a negative integer is read
int stopLoop, numbers;
stopLoop = 0;
while (stopLoop != 1) {
    scanf("%d", &number);
    if (number < 0) {
        stopLoop = 1;
    } else {
        printf("%d\n", number);
    }
}
```
**while Loop - Sentinel Variable Pattern**

Here is the programming pattern for a while that executes $n$ times.

```java
stopLoop = 0;
while (stopLoop != 1) {
    //
    // statements the loop needs to perform
    //
    if (..........) {
        stopLoop = 1;
    }
    //
    // perhaps more statements
    //
}
```
Nested While Loops

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

```c
// print a square of 10x10 asterisks
int i, j;
i = 0;
while (i < 10) {
    j = 0;
    while (j < 10) {
        printf("* ");
        j = j + 1;
    }
    printf("\n");
i = i + 1;
}
```
The *for* loop

As well as a while loop there also a *for* Loop:

```java
for (expr1; expr2; expr3) {
    statements;
}
```

- *expr1* is evaluated before the loop starts.
- *expr2* is evaluated at the beginning of each loop; if it is non-zero, the loop is repeated.
- *expr3* is evaluated at the end of each loop.
Example of *for* loop

```c
for (x = 1; x <= 10; x++) {
    printf("%d\n", x * x);
}
```

We can declare our counter variable within the for loop. However we can only use it in the for loop if we do this:

```c
for (int x = 1; x <= 10; x++) {
    printf("%d\n", x * x);
}
```
for loops and while loops

These two are equivalent:

```c
for (expr1; expr2; expr3) {
    statements;
}
expr1;
while (expr2) {
    statements;
    expr3;
}
```
for loops and while loops

These two are equivalent:

```c
for (int i = 0; i < MAX; i++) {
    printf("%d\n",i);
}
```

```c
int i = 0;
while (i < MAX) {
    printf("%d\n",i);
    i++;
}
```
Any of the 3 expressions in the for loop may be omitted ‘;’ must still be present. For example:

```c
printf("Enter starting number for Countdown: ");
scanf("%d", &n); // initial value entered by user
for (; n >= 0; n--) {
    printf("%d\n", n);
}
printf("Blast Off!\n");
```
Although **NOT recommended**, the comma operator ‘, ’ can be used to squeeze multiple statements into `expr1` and `expr3`. For example,

```c
for (int x=0, y=2; x < MAX; x++, y++) {
    ...
}
```