Iteration

Iteration, repetition, or looping, they all mean the same thing. When you want to repeat some portion of the program, you will use some form of iteration. Looping is a very common term to describe this.

The `while` Statement

The most basic looping construct is the **`while`** statement. The form is identical to the **`if`** statement, but uses the keyword **`while`** instead:

```
while ( expression )
  statement
```

Whereas the **`if`** statement caused *statement* to be executed exactly once if *expression* was true, the **`while`** statement causes *statement* to be executed repeatedly as long as *expression* remains true. If *expression* becomes false, then the repetition stops.

Simple example:

```
int count = 5;
int i = 0;

while (i < count)  /* controlling expression  */
  i++;             /* body of the loop        */
```

Example with output: (and a review of expressions)

Version #1

```
int count = 5;
int i = 0;

while (i < count)
{
  i++;
  printf("i is %i\n", i);
}
```

Version #2

```
int count = 5;
int i = 0;

while (i < count)
{
  printf("i is %i\n", ++i);
}
```

Version #3

```
int count = 5;
int i = 0;

while (i++ < count)
{
  printf("i is %i\n", i);
}
```

The output from all 3 examples:

```
i is 1
i is 2
i is 3
i is 4
i is 5
```

Bart Simpson could have used C to make his punishment less painful:

```
int i = 1;
while (i <= 100)
{
  printf("I will include my section in email correspondence with the teacher.\n");
  i++;
}
```

(Partial) output:

```
I will include my section in email correspondence with the teacher.
I will include my section in email correspondence with the teacher.
I will include my section in email correspondence with the teacher.
I will include my section in email correspondence with the teacher.
I will include my section in email correspondence with the teacher.
I will include my section in email correspondence with the teacher.
I will include my section in email correspondence with the teacher.
I will include my section in email correspondence with the teacher.
I will include my section in email correspondence with the teacher.
I will include my section in email correspondence with the teacher.
I will include my section in email correspondence with the teacher.
I will include my section in email correspondence with the teacher.
I will include my section in email correspondence with the teacher.
I will include my section in email correspondence with the teacher.
I will include my section in email correspondence with the teacher.
I will include my section in email correspondence with the teacher.
I will include my section in email correspondence with the teacher.
I will include my section in email correspondence with the teacher.
I will include my section in email correspondence with the teacher.
I will include my section in email correspondence with the teacher.
```

Notes:

- The body of the loop is not guaranteed to execute at all.
- The controlling expression:
  - is executed once before the body of the loop is executed.
  - is executed again after each repetition of the body.
  - usually will eventually evaluate to false, to stop the repetition.
- If the controlling expression *never* evaluates to false, the while loop is called *an infinite loop*, because it never stops.

Infinite loop #1

```
int i = 1;
while (i != 10)
  i += 2;
```

Infinite loop #2

```
int i = 0;
while (i < 10)
  printf("i is %i\n", i);
```

Infinite loop #3

```
while (1)
  printf("This loop never ends...\n");
```

The `do` Statement

Also sometimes referred to as the **`do...while`** statement. The basic format is:

Single statement

```
do
  statement
while ( expression );
```

Multiple statements

```
do
{
  statements
}
while ( expression );
```

The primary difference between the **`while`** statement and the **`do`** statement is that the body of the **`do`** statement is guaranteed to execute at least once. This is simply because the controlling expression is executed *after* the first iteration of the loop body:

Body executes 0 or more times

```
while ( expression )
  statement
```

Body executes 1 or more times

```
do
  statement
while ( expression );
```

Example:

```
int number;
int choice;

do
{
  printf("Enter a number: ");
  scanf("%d", &number);
  printf("You entered %i\n", number);

  printf("Enter another number? (1=yes,0=no) ");
  scanf("%d", &choice);
}
while (choice);
```

Sample run:

```
Enter a number: 23
You entered 23
Enter another number? (1=yes,0=no) 1
Enter a number: 10
You entered 10
Enter another number? (1=yes,0=no) 1
Enter a number: 5
You entered 5
Enter another number? (1=yes,0=no) 3
Enter a number: 12
You entered 12
Enter another number? (1=yes,0=no) 0
```

There really isn't much difference between **`while`** statement and the **`do`** statement. If you need the loop to execute at least once, then the **`do`** statement is the one to use.

The `for` Statement

Now we get to the most complex of the looping mechanisms: the **`for`** statement. The general form is:

```
for ( expression1 ; expression2 ; expression3 )
  statement
```

and the compound version:

```
for ( expression1 ; expression2 ; expression3 )
{
  statements
}
```

The meaning of this is a little involved:

1. First, evaluate *expression*$_1$. This is executed and evaluated exactly once at the beginning of the loop.
2. Evaluate *expression*$_2$.
3. If *expression*$_2$ is true, execute *statement*. (If it's not true, jump out of the loop.)
4. Evaluate *expression*$_3$.
5. Goto step 2.

This process can be written using an equivalent **`while`** statement:

```
expression1;
while ( expression2 )
{
  statements
  expression3;
}
```

This also means that any **`for`** loop can be written as a **`while`** loop and vice-versa.

Simple examples to print the numbers 1 through 10:

for loop #1

```
int i;
for (i = 1; i <= 10; i++)
  printf("%i\n", i);
```

for loop #2

```
int i;
for (i = 0; i < 10; i++)
  printf("%i\n", i + 1);
```

while loop

```
int i = 1;
while (i <= 10)
  printf("%i\n", i++);
```

The **`for`** loops above show the typical ways in which they are used. The variable *i* is sometimes called the *loop control variable* (or simply the *counter*) because it controls when the loop continues or stops. The three expressions generally

1. *expression*$_1$ - initializes the loop variable (or counter)
2. *expression*$_2$ - compares the counter with some value
3. *expression*$_3$ - modifies the counter (usually add/subtract 1)

These are just typical uses of the expressions. You can do practically anything with those expressions.

Count to 20 by 2

```
for (i = 2; i <= 20; i += 2)
  printf("%i\n", i);
```

```
2
4
6
8
10
12
14
16
18
20
```

Count down from 30 by 3

```
for (i = 30; i >= 3; i -= 3)
  printf("%i\n", i);
```

```
30
27
24
21
18
15
12
9
6
3
```

Squares of 1 to 10

```
for (i = 1; i <= 10; i++)
  printf("%i\n", i * i);
```

```
1
4
9
16
25
36
49
64
81
100
```

Of course, all of these could be written as **`while`** loops as well.

Count to 20 by 2

```
i = 2;
while (i <= 20)
{
  printf("%i\n", i);
  i += 2;
}
```

Count down from 30 by 3

```
i = 30;
while (i >= 3)
{
  printf("%i\n", i);
  i -= 3;
}
```

Squares of 1 to 10

```
i = 1;
while (i <= 10)
{
  printf("%i\n", i * i);
  i++;
}
```

Or, if you prefer compact expressions:

Count to 20 by 2

Count down from 30 by 3

Squares of 1 to 10

```
i = 0;
while ( (i += 2)  <= 20)
  printf("%i\n", i);
```

```
i = 33;
while ( (i -= 3) >= 3)
  printf("%i\n", i);
```

```
i = 0;
while (++i <= 10)
  printf("%i\n", i * i);
```

However, I would strongly discourage the use of the last examples above. *Just because C allows you to do something, doesn't mean you should.* If you forget the parentheses in the first two examples, the compiler will say nothing, but you won't get the correct output.

More on Looping

Note that any or all of the expressions in the **`for`** loop can be omitted:

```
i = 1;
for (; i <= 10;)
  printf("%i\n", i++);
```

Of course, this is nothing but a strange-looking while loop now.

```
i = 1;
while (i <= 10)
  printf("%i\n", i++);
```

You can even omit the second expression, but this would lead to an infinite loop, since the default *empty* expression is true!

```
i = 1;
for (;;)
  printf("%i\n", i++);
```

If you want to exit from the loop prematurely, you can use the **`break`** statement:

breaking out of infinite `for`

```
i = 1;
for (;;)
{
  printf("%i\n", i++);
  if (i > 10)
    break;
}
```

breaking out of infinite `while`

```
i = 1;
while (1)
{
  printf("%i\n", i++);
  if (i > 10)
    break;
}
```

The **`break`** statement can be used in any of the looping mechanisms as well as the **`switch`** statement.

You can also have multiple expressions in between the semicolons:

Using a `for` loop

```
for (i = 0, j = 0; i < 16 || j < 3; i +=2, j++)
  printf("%i * %i = %i\n", i, j, i * j);
```

```
0 * 0 = 0
2 * 1 = 2
4 * 2 = 8
6 * 3 = 18
8 * 4 = 32
10 * 5 = 50
12 * 6 = 72
14 * 7 = 98
```

Using a `while` loop

```
i = 0;
j = 0;
while (i < 16 || j < 3)
{
  printf("%i * %i = %i\n", i, j, i * j);
  i += 2;
  j++;
}
```

Note the use of the *comma* operator used in the expressions. This operator has the form:

expression$_1$, *expression*$_2$, *expression*$_3$, etc...

- The expressions are evaluated in order, left to right.
- This order of evaluation is well defined and guaranteed. (Few other operators can make this guarantee).
- The result of the entire expression is the value of the **last** expression.
- What are the values of `i`, `j`, and `c` below? (Look carefully!!)

```
i = 5;
j = 3;
c = ++i, ++j, j + i;
printf("%i, %i, %i\n",i,j,c);
```

```
i = 5;
j = 3;
c = (++i), (++j), (j + i);
printf("%i, %i, %i\n",i,j,c);
```

```
i = 5;
j = 3;
c = (++i, ++j, j + i);
printf("%i, %i, %i\n",i,j,c);
```

The **`continue`** statement is similar to the **`break`** statement in that it causes the loop to deviate from its prescribed course. The difference is subtle, but very important.

`break` statement

```
for (/* expressions */)
{
  /* first statement in loop  */
  /* second statement in loop */
  /* etc...                   */

  break;

  /* last statement in loop   */
}
[break jumps to here]
/* first statement after loop */
```

`continue` statement

```
for (/* expressions */)
{
  /* first statement in loop  */
  /* second statement in loop */
  /* etc...                   */

  continue;

  /* last statement in loop   */
  [continue jumps to here]
}
/* first statement after loop */
```

This prints the even numbers from 2 to 20:

using `for`

```
for (i = 2; i <= 20; i++)
{
  if ( (i % 2) == 1 )
    continue;
  printf("%i\n", i);
}
```

```
2
4
6
8
10
12
14
16
18
20
```

using `while`

```
i = 2;
while (i <= 20)
{
  if ( (i % 2) == 1 )
  {
    i++;
    continue;
  }
  printf("%i\n", i++);
}
```

using `while`

```
i = 2;
while (i <= 20)
{
  if ( (i++ % 2) == 1 )
    continue;
  printf("%i\n", i - 1);
}
```
