C# Do loops and While Loops

As well as using a **for** loop to repeatedly execute some code, you can use a **Do** loop or a **While** loop. We'll start with the Do Loop.

C# Do Loops

Whichever loop you use, the idea is still the same: go round and round and execute the same code until an end condition is met. The difference with the Do and While loops is in the structure. Here's what the Do loop looks like:

```
do
{

} while (true);
```

Notice where the semi-colon is, in the code above. It comes right at the end, after the round brackets. But you start with the word **do**, followed by a pair of curly brackets. After the curly brackets, you type the word **while**. After while, and in between some round brackets, you type your end condition. C# will loop round and round until the end condition between the round brackets is met. Only then will it bail out. Here's an example, using our times table programme:

```
do
{

        answer = multiplyBy * i;
        listBox1.Items.Add(answer.ToString());
        i++;

} while (i <= loopEnd);
```

So this time, we've used a Do Loop instead of a For Loop. The loop will go round and round while the value in the variable called i is less than or equal to the value in the variable called loopEnd. The other thing to notice here is we have to increment (add one to) the value in i ourselves (i++). We do this each time round the loop. If we didn't increment the value in i then it would always be less than loopEnd. We'd have then created an infinite loop, and the programme would crash. But we're really saying this:

"Keep Doing the code in curly brackets while i is less than or equal to loopEnd."

C# While Loops

While loops are very similar in structure to Do loops. Here's what they look like:

```
while (true)
{

}
```

And here's the times table code again:

```
while (i <= loopEnd)
{

        answer = multiplyBy * i;
        listBox1.Items.Add(answer.ToString());
        i++;

}
```

While loops are easier to use than Do loops. If you look at the code above, you can see that the while part is at the start, instead of at the end like a Do Loop. The code you need to execute repeatedly still goes between curly brackets. And you still need a way for the loop to end (i++).

The difference between the two loops is that the code in a Do loop will get executed at least once, because the while part is at the end. With the while part at the beginning, your end condition in round brackets can already be true (i might be more than loopEnd). In which case, C# will bail out immediately, and the code in curly brackets won't get executed at all.

Deciding which loop to use can be quite tricky. Don't worry if you haven't fully understood how to use loops. You'll get lots more practice as you work your way through this book. But loops are difficult to get the hang of, and you shouldn't consider yourself a failure if you haven't yet mastered them. For now, we'll leave this complex subject, and end the section with a problem, and a solution.