while Loop

This lesson essentially explains what a while loop is and how to use it in the code.

WE'LL COVER THE FOLLOWING

while loop

The `while` loop is similar to the `if` statement and essentially works as a repeated `if` statement. Just like `if`, `while` also takes a logical expression and evaluates the block when the logical expression is *true*. The difference is that the `while` statement evaluates the logical expression and executes the statements in the block repeatedly, as long as the logical expression is true, not just once. Repeating a block of code this way is called **looping**.

Here is the syntax of the while statement:

```
while (a_logical_expression) {
// ... expression(s) to execute while true
}
```

For example, the code that displays 'Take cookie' and 'Eat cookie' as long as there is cookie looks like this:

```
import std.stdio;

void main() {
    bool existsCookie = true;
```

```
    while (existsCookie) {
        writeln("Take cookie");

        writeln("Eat cookie");
    }
}
```

That program would continue executing the statements within the scope of the while loop because the value of `existCookie` does not change to false.

`while` is useful when the value of the logical expression changes during the execution of the program. To see this, let's write a program that continues taking a number from the user as long as the number is zero or greater. Remember that the initial value of `int` variables is 0:

```
import std.stdio;

void main() {
    int number;

    while (number >= 0) {
        write("Please enter a number: ");
        readf(" %s", &number);

        writeln("Thank you for ", number);
    }

    writeln("Exited the loop");
}
```

while loop

The program thanks for the provided number and exits the loop only when the number is less than zero.

The continue statement

The **continue statement** starts the next iteration of the loop right away, instead of executing the rest of the expressions of the block.
Let's modify the program above to be a little picky: instead of thanking for any number, let's not accept 13. The following program does not thank for 13 because, in that case, the `continue` statement makes the program go to the beginning of the loop to evaluate the logical expression again:

```
import std.stdio;

void main() {
```

```
    int number;

    while (number >= 0) {
        write("Please enter a number: ");
        readf(" %s", &number);

        if (number == 13) {
            writeln("Sorry, not accepting that one...");
            continue;
        }

        writeln("Thank you for ", number);
    }

    writeln("Exited the loop");
}
```

continue statement

We can define the behavior of the program to take numbers that are greater than or equal to 0 but skip 13.
`continue` works with `do-while`, `for` and `foreach` statements as well. We will see these features later in this chapter.

The break statement

Sometimes it becomes obvious that there is no need to stay in the `while` loop any longer.
In such a case, `break` allows the program to exit the loop right away. The following program exits the loop as soon as it finds a special number:

```
import std.stdio;

void main() {
    int number;

    while (number >= 0) {
        write("Please enter a number: ");
        readf(" %s", &number);

        if (number == 42) {
            writeln("FOUND IT!");
            break;
        }

        writeln("Thank you for ", number);
    }

    writeln("Exited the loop");
}
```

break statement

We can summarize this behavior as taking numbers from the user as long as they are greater than or equal to 0 or until a number is 42.
`break` works with `do-while`, `for`, `foreach` and `switch` statements as well. We will see these features in later chapters.

Unconditional loop

Sometimes the logical expression is intentionally made a constant `true` as shown in the code below. The `break` statement is a common way of exiting such unconditional loops (Infinite loop is an alternative but not completely accurate term that means unconditional loop.)

The following program prints a menu in an unconditional loop; the only way of exiting the loop is a `break` statement:

```
import std.stdio;

void main() {
    /* Unconditional loop, because the logical expression is always
     * true */
    while (true) {
        write("0:Exit, 1:Turkish, 2:English - Your choice? ");

        int choice;
        readf(" %s", &choice);

        if (choice == 0) {
            writeln("See you later...");
            break;   // The only exit of this loop

        } else if (choice == 1) {
            writeln("Merhaba!");

        } else if (choice == 2) {
            writeln("Hello!");

        } else {
            writeln("Sorry, I don't know that language. :/");
        }
    }
}
```

Unconditional loop

> **Note:** Exceptions can terminate an unconditional loop as well. We will see exceptions in a later chapter.

Now that we have learned about conditional statements and loops, our next lesson is about name scopes.