04-Loops

In computer science, a *loop* -- put simply -- is a segment of code that is executed repeatedly. A simple example of a loop using `goto` may look like:

```
#include <iostream>

int main() {
Loop:
	std::cout << "Hello, world!\n";
	goto Loop;

	__builtin_unreachable(); // this will never be reached
}
```

The output of this program may look something like:

```
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
... # Stopping here, you'll thank me if this is ever printed.
```

On their own, unterminated loops -- for the most part -- are not very useful. A program must exit at some point, even in the event of some failure. Loops should often have a *loop condition*, or have some way to ***break***. Two such types of loops in C++ exist: `for` loops and `while` loops.

`while` Loops

In C++, a `while` loop is a loop which repeats a segment of code as long as it's *loop condition* is true. Its structure is similar to that of an `if` statement:

```
while (condition) {
	// ... runs as long as condition is true
}
```

Contrary to an `if` statement, however, `while` conditions are evaluated once, and then every time the end of the `while` loop is reached. For example, if we want to declare a loop which "counts" to `5`, we may do:

```
int i = 1;
while (i <= 5) {
        std::cout << "i = " << i << '\n';
        ++i; // increment i by 1
}
```

Output:

```
i = 1
i = 2
i = 3
i = 4
i = 5
```

`for` Loops

In C++, a `for` loop is a loop which, similar to a `while` loop, contains a loop condition which determines whether its statement will repeat itself. Unlike `while` loops, however, `for` loops contain two extra components: an *init statement* and an *expression* that runs at the end of every *iteration* (loop repetition):

```
for (init statement; condition; expression) {
        // ...
}
```

> Notice that each statement within the `for` loop header is terminated by a semicolon.

`for` loops, unlike `while` loops, allow you to specify a *method of execution* for a segment of code, meaning that you can outline *what* is being checked, *how* it is checked, and *how* it is changed every loop. For instance, in our previous example, we count from `1` to `5` with a variable `i`. We can express this as a `for` loop via:

```
for (int i = 1; i <= 5; ++i) {
        std::cout << "i = " << i << '\n';
}
// `int i` is no longer in scope after the for loop.
```

What's Next?

In the next section, we will learn how to utilize loops and conditional logic to write and access lists of variables, using *arrays*.

Courtesy of *DeCentralize the Web* (2024)