8.1 — Control flow introduction

learncpp.com/cpp-tutorial/control-flow-introduction/

When a program is run, the CPU begins execution at the top of `main()`, executes some number of statements (in sequential order by default), and then the program terminates at the end of `main()`. The specific sequence of statements that the CPU executes is called the program's **execution path** (or **path**, for short).

Consider the following program:

```
#include <iostream>

int main()
{
    std::cout << "Enter an integer: ";

    int x{};
    std::cin >> x;

    std::cout << "You entered " << x << '\n';

    return 0;
}
```

The execution path of this program includes lines 5, 7, 8, 10, and 12, in that order. This is an example of a **straight-line program**. Straight-line programs take the same path (execute the same statements in the same order) every time they are run.

However, often this is not what we desire. For example, if we ask the user for input, and the user enters something invalid, ideally we'd like to ask the user to make another choice. This is not possible in a straight-line program. In fact, the user may repeatedly enter invalid input, so the number of times we might need to ask them to make another selection isn't knowable until runtime.

Fortunately, C++ provides a number of different **control flow statements** (also called **flow control statements**), which are statements that allow the programmer to change the normal path of execution through the program. You've already seen an example of this with if-statements (introduced in lesson 4.10 -- Introduction to if statements) that let us execute a statement only if a conditional expression is true.

When a control flow statement causes point of execution to change to a non-sequential statement, this is called **branching**.

Categories of flow control statements

Category	Meaning	Implemented in C++ by
Conditional statements	Causes a sequence of code to execute only if some condition is met.	if, else, switch
Jumps	Tells the CPU to start executing the statements at some other location.	goto, break, continue
Function calls	Jump to some other location and back.	function calls, return
Loops	Repeatedly execute some sequence of code zero or more times, until some condition is met.	while, do-while, for, ranged-for
Halts	Terminate the program.	std::exit(), std::abort()
Exceptions	A special kind of flow control structure designed for error handling.	try, throw, catch

We'll cover all of these categories in detail throughout this chapter, with the exception of exceptions (ha) which we'll devote an entire future chapter to (chapter 27).

Prior to this chapter, the number of things you could have a program do was fairly limited. Being able to control the flow of your program (particularly using loops) makes any number of interesting things possible! No longer will you be restricted to toy programs -- you will be able to write programs that have real utility.

This is where the real fun begins. So let's get to it!