6.5 — The comma operator

Operator	Symbol	Form	Operation
Comma	,	x, y	Evaluate x then y, returns value of y

The **comma operator (,)** allows you to evaluate multiple expressions wherever a single expression is allowed. The comma operator evaluates the left operand, then the right operand, and then returns the result of the right operand.

For example:

```
#include <iostream>

int main()
{
    int x{ 1 };
    int y{ 2 };

    std::cout << (++x, ++y) << '\n'; // increment x and y, evaluates to the right operand

    return 0;
}
```

First the left operand of the comma operator is evaluated, which increments *x* from *1* to *2*. Next, the right operand is evaluated, which increments *y* from *2* to *3*. The comma operator returns the result of the right operand (*3*), which is subsequently printed to the console.

Note that comma has the lowest precedence of all the operators, even lower than assignment. Because of this, the following two lines of code do different things:

```
z = (a, b); // evaluate (a, b) first to get result of b, then assign that value to variable z.
z = a, b; // evaluates as "(z = a), b", so z gets assigned the value of a, and b is evaluated and discarded.
```

This makes the comma operator somewhat dangerous to use.

In almost every case, a statement written using the comma operator would be better written as separate statements. For example, the above code could be written as:

```
#include <iostream>

int main()
{
    int x{ 1 };
    int y{ 2 };

    ++x;
    std::cout << ++y << '\n';

    return 0;
}
```

Most programmers do not use the comma operator at all, with the single exception of inside *for loops*, where its use is fairly common. We discuss for loops in future lesson 8.10 -- For statements.

Best practice

Avoid using the comma operator, except within *for loops*.

Comma as a separator

In C++, the comma symbol is often used as a separator, and these uses do not invoke the comma operator. Some examples of separator commas:

```
void foo(int x, int y) // Separator comma used to separate parameters in function definition
{
    add(x, y); // Separator comma used to separate arguments in function call
    constexpr int z{ 3 }, w{ 5 }; // Separator comma used to separate multiple variables being defined on the same line (don't do this)
}
```

There is no need to avoid separator commas (except when declaring multiple variables, which you should not do).