C++ basics

There are many different kinds of statements in C++:

- Declaration statements
- Jump statements
- Expression statements
- Compound statements
- Selection statements (conditionals)
- Iteration statements (loops)
- Try blocks

Variable assignment and initialization

```
int a;          // no initializer (default initialization)
int b = 5;      // initial value after equals sign (copy initialization)
int c( 6 );     // initial value in parenthesis (direct initialization)

// List initialization methods (C++11) (preferred)
int d { 7 };    // initial value in braces (direct list initialization)
int e = { 8 }; // initial value in braces after equals sign (copy list
initialization)
int f {};       // initializer is empty braces (value initialization)
```

Best practice

Prefer direct list initialization (or value initialization) for initializing your variables. Prevents warnings for unused variables.

```
[[maybe_unused]] double pi { 3.14159 };
[[maybe_unused]] double gravity { 9.8 };
[[maybe_unused]] double phi { 1.61803 };
```

PREFER LIST INITIALIZATION!! example: int x {5}

IOSTREAM

Using std::endl is often inefficient, as it actually does two jobs: it **outputs a newline** (moving the cursor to the next line of the console), and it **flushes the buffer** (which is slow).

Uninitialized variables and undefined behavior

- Initialized = The object is given a known value at the point of definition.
- Assignment = The object is given a known value beyond the point of definition.
- Uninitialized = The object has not been given a known value yet.

Undefined behaviour

- Your program produces different results every time it is run.
- Your program consistently produces the same incorrect result.
- Your program behaves inconsistently (sometimes produces the correct result, sometimes not).
- Your program seems like it's working but produces incorrect results later in the program.
- Your program crashes, either immediately or later.
- Your program works on some compilers but not others.
- Your program works until you change some other seemingly unrelated code.

Operators

arity

- **Unary** operators act on one operand. An example of a unary operator is the - operator. For example, given -5, operator- takes literal operand 5 and flips its sign to produce new output value -5.
- **Binary** operators act on two operands (often called left and right, as the left operand appears on the left side of the operator, and the right operand appears on the right side of the operator). An example of a binary operator is the + operator. For example, given 3 + 4, operator+ takes the left operand 3 and the right operand 4 and applies mathematical addition to produce new output value 7. The insertion (<<) and extraction (>>) operators are binary operators, taking std::cout or std::cin on the left side, and the value to output or variable to input to on the right side.
- **Ternary** operators act on three operands. There is only one of these in C++ (the conditional operator), which we'll cover later.
- **Nullary** operators act on zero operands. There is also only one of these in C++ (the throw operator), which we'll also cover later.

Expressions

- An **expression** is a non-empty sequence of literals, variables, operators, and function calls that calculates a single value. The process of executing an expression is called **evaluation**, and the single value produced is called the **result** of the expression.
- Statements are used when we want the program to perform an action. Expressions are used when we want the program to calculate a value.