Swapping two variables

This will discuss swapping local variables, and for now we will assume that all local variables being swapped have the same type.

The standard means by which two variables can be swapped is to initialize a temporary local variable with the value of the first variable, to then assign that first variable the value of the second variable, and to finish with assigning the second variable the value stored in the temporary local variable:

```
// Swapping local variables 'a' and 'b'
typename tmp{a};
a = b;
b = tmp;
```

The temporary local variable should have the same type as that of `a` and `b`.

Now, some simplifications are to first allow the compiler to determine the type:

```
// Swapping local variables 'a' and 'b'
auto tmp{a};
a = b;
b = tmp;
```

The `auto` keyword says to the compiler: if you can unambiguously determine the type that this local variable should have, then use that type.

Swapping without local variables

It is possible to swap without local variables using arithmetic and bitwise operations. Consider the following:

```
// Swapping local variables 'a' and 'b'
a += b;
b = b - a;
a -= b;
```

Suppose, that `a` is assigned `13` and `b` is assigned `29`.

```
a += b;     // 'a' is now assigned 42
b = a - b;  // 'b' is now assigned 42 - 29 which equals 13
a -= b;     // 'a' is now assigned 42 - 13 which equals 29
```

Your first thought may be, “great, fewer operations!” But that is not really the case, as the second statement first requires the calculation of `a - b`, and only then can the result be moved to `b`. Also, these are arithmetic operations, which require more energy.

Try to convince yourself that this still works with overflow and underflow for integer data types.

Explain under which circumstance this may not work if you were to use this with double-precision floating-point numbers. Specifically, consider this example:

```
// Pre-processor include directives
#include <iostream>

// Function declarations
int main();

// Function definitions
int main() {
        double x{1.2e308};
        double y{1.5e308};

        std::cout << "Swapping 1.2 x 10^308 and 1.5 x 10^308"
                  << std::endl;
        std::cout << x << ", " << y << std::endl;

        x += y;
        y = x - y;
        x -= y;

        std::cout << x << ", " << y << std::endl << std::endl;

        double u{6.8e86};
        double v{42.0};

        std::cout << "Swapping 6.8 x 10^86 and 42" << std::endl;
        std::cout << u << ", " << v << std::endl;

        u += v;
        v = u - v;
        u -= v;

        std::cout << u << ", " << v << std::endl;

        return 0;
}
```

The output is:

```
Swapping 1.2 x 10^308 and 1.5 x 10^308
1.2e+308, 1.5e+308
-nan, inf

Swapping 6.8 x 10^86 and 42
6.8e+86, 42
0, 6.8e+86
```

The other approach is to use the exclusive-OR operator:

```
// Swapping local variables 'a' and 'b'
a ^= b;
b ^= a;
a ^= b;
```

This is actually cheaper as a bit-wise exclusive or operation requires no carries (unlike addition and subtraction).

Suppose, that `a` is assigned `00101101` and `b` is assigned `10010110`.

```
a ^= b;     // 'a' is now assigned 10111011
b ^= a;     // 'b' is now assigned 00101101
a -= b;     // 'b' is now assigned 10010110
```

and you will see that the two variables are swapped.

Swapping objects

If a class has a move constructor and move assignment operator defined, it is possibly to explicitly call that operator using std::move(...):

```
// Swapping local variables 'a' and 'b'
auto tmp{std::move( a )};
a = std::move( b );
b = std::move( tmp );
```

In general

In general, just use `std::swap` to swap two local variables. If you are defining a very complex class, `std::swap` will still use move constructors and move assignment operators if they are defined, and only under very specific circumstances may it be even better to author your own swap function for a very complex class.