Classes in C++

Classes (and structs) are data structures that contain data (member variables), and functions (sometimes called methods)
Some are provided by c++ libraries; we can also write our own.
This can be an extremely useful tool for encapsulation.

See also the examples at: https://github.com/benroberts999/cpp-cheatsheet

---

Classes and Structs

We may define classes and structs in the following way:

```
class MyClass{};
struct MyStruct{};
```

- Notice the semi-colon
- The only difference between class and struct: members in class are *private* by default; those of struct are *public* by default (we'll see what this means below)
- Public mean accessible outside the class; private means not
- By convention, structs are used for very small objects (one or two members, no functions), and classes are used for large ones with functions

Classes are not useful unless that contain something (data/functions).
Consider this example:

```
// define a class:
class MyClass{
private:
  int a = 1;
public:
  int b = 2;
};

// Construct object of type 'MyClass', call it mc:
MyClass mc;

// Access the data members of MyClass
std::cout << mc.b << '\n';
// std::cout << mc.a << '\n'; - does not work, "a is private in this scope"

mc.b = 42;
std::cout << mc.b << '\n';
```

Constructor, member initializer list

- Constructor is a function that is called automatically when you make a new variable of this type

- *member initializer list* initialises member variables
  - funny syntax: a colon ':' after (), but before {}

```
class MyClass{
public:

  // data stored in the class
  int a;

  // Constructor, using member initializer list:
  MyClass(int in_a) : a(in_a) {}

  // We could also write it as:
  // MyClass(int in_a){
  //   a = in_a;
  // }
  // However, in this case, the int a in constructed, then set to in_a
  // With the member initializer list, these happen at the same time
};
```

Member functions

- Member functions work just like other functions, but have access to class data
- called using the '.': object.function()

Basic example: https://godbolt.org/z/xWYddacjW

```
class MyClass {
 private:
  int b = 7;
 public:
  int a;
  MyClass(int in_a) : a(in_a) {}

  // Member function:
  double square_a() { return a * a; }
  // Can use member functions to interface with private variables
  void print_b() { std::cout << b << '\n'; }
};
```

```
// Use it:
MyClass mc{4};
std::cout << mc.a << '\n';
// std::cout << mc.b << '\n'; //won't compile: b is private
std::cout << mc.square_a() << '\n';
mc.print_b();
```

Operator overloading

- In c++, we may define operators that act on our own user-defined classes
- We may overload: +,-,/,*,==,>,<, <<, >> etc. etc.
- This is very useful: and leads to nice-looking code. e.g.,
  - Matrix3 = Matrix1 + Matrix2

- instead of: Matrix3 = matrix_addition(Matrix1,Matrix2);

Super basic example: https://godbolt.org/z/MnEM8s3sq

```
class MyComplex {
 public:
  double re, im;
  MyComplex(double in_r, double in_i) : re(in_r), im(in_i) {}
};

MyComplex operator*(MyComplex lhs, MyComplex rhs) {
  double res_re = lhs.re * rhs.re - lhs.im * rhs.im;
  double res_im = lhs.re * rhs.im + lhs.im * rhs.re;
  MyComplex result{res_re, res_im};
  return result;
}
```

```
  MyComplex mc1{4.0, 3.0};
  MyComplex mc2{6.0, -1.0};
  MyComplex mc3 = mc1 * mc2;
  std::cout << mc1.re << " + " << mc1.im << "i\n";
  std::cout << mc2.re << " + " << mc2.im << "i\n";
  std::cout << mc3.re << " + " << mc3.im << "i\n";
```