Classes in C++

Picked from Lippman 5th edition

Basic Syntax and Concepts

- Member functions are defined similar to regular functions, but the declarations must occur inside the class definition.
- The functions may be defined outside, and those defined inside the class body are implicitly `inline`.
- Member functions called inside other member functions have this access through the `this` pointer.
- It is illegal to define a parameter or a vriable named `this`.
- The type of `this` is a const pointer to the nonconst version of the class type.
- This means that in general, we cannot call an ordinary member function on a const object.
- The way to indicate that the `this` pointer needs to be const is adding `const` in between the parameter list and the function body. For eg:

```
double Sales_data::avg_price() const {
    ...
}
```

- For defining a class member function outsude the class, you need to validate it using the scope operator, `class_name::func()`.
- To return the object itself, you should `return *this`.
- One can also return an lvalue reference to the object.
- Non-member functions that are part of the interface are usually declared in the same header.
- Class object initialization is controlled through constructors. Constructors have no return type, and have the same name as the class.
- Constructors may not be declared const, as they invariably need to change the object.
- A default constructor is one that takes no arguments.
- The compiler supplies a special default constructor, the synthesized constructor, which is defined implictly if no constructor is defined explicitly by us. It works as:
  1. If there is an in-class initializer, use it.
  2. Otherwise, deault-initialize the member.
- If any constructor is defined explicitly, the synthesized constructor flies out.
- If a class has a member which is another class, which does not have any default constructor, the compiler cannot synthesize a constructor implicitly.
- If we want to define a default constructor which is the same as the synthe-

sized one, and more constructors, we can write `Sales_data() = default;`. This can appear with the declarations inside or as a definition outside.
- The constructor can explicitly initialize the members instead of just assigning them, though the constructor initializer list.
- The constructor initializer list is a list of member names each followed by its initialized value in parantheses, separated by commas. It appears between the parameter list and the body. For eg:

```
Sales_data(const std:string &s): bookNo(s) {}
```

- The constructor's initializer list wins over te in-class initializer if both are present.
- It is usually better to use in-class initializer instead.
- Classes also define what happens on copy, assignment and destruction.
- Classes that deal with pointers, or dynamic memory, would need to redefine all of the above, and probably cannot use the synthesized versions.
- C++ uses access control specifiers to enforce encapsulation.
- struct deafults to public where as class defaults to private.
- A class can allow another class or nonmember function to access its nonpublic members by declaring them friends, basically by writing out the prototype preceded by the keyword `friend` in the class definition. For eg:

```
friend Sales_data add(const Sales_data &, const Sales_data &);
```

- The friendship is not affected by the access modifier of where it is declared.
- We also need to declare the function outside separately, a friend declaration is not a declaration.
- A class can also define local names for types, using `typedef`. They should appear before they are used.
- For a function defined outside, we can declare it inline either during the declaration or during the definition outside.
- It is better to define inline functions in the corresponding class header.
- Member functions can also be overloaded.
- Some datamembers must be allowed to change, even inside a const object. Such datamembers need to be declared `mutable`.
- In-class initializers could also be brace initialized in nature.
- Functions that return a reference to `*this` allow for chaining of functions on them one after the other.
- We can overload a function based on just the constness of the `this` pointer. This will resolve by const objects calling one version and nonconst ones the other.
- Two classes with the same contents still define two different unrelated types.
- We can declare a class without defining it, by just writing `class Screen;`.
- This makes `Screen` an incomplete type. We can only define pointers or references to such types, and declare functions that use these.
- Data members of a class type need to have that class defined before they themselves are specified.

- We can declare an entire class as a friend, allowing all member functions of that class access to all of our members.
- We can also make a member function of another class a friend of ours, by scoping the function during the friend declaration.
- This requires a careful ordering:
  1. Define the other class and declare the function.
  2. Put a friend declaration for that member function in this class.
  3. Define the member function outside.
- Each class defines a scope, called the class scope.
- During a function definition outside the class, the return type comes before the function name, so if the return type is of a type defined by the class, you need to scope the return type also. This need not be done for the parameters.
- Compilers look at classes in two phases. First all declarations are compiled, then all definitions (function bodies).
- Names used in declarations still need to occur before they are used.
- Name lookup thus proceeds as:
  1. the part of the function code seen yet.
  2. The whole code of the class definition.
  3. Surrounding scopes.
- We can override 1 and get to 2 by using scoping.
- The constructor initializer is needed for types which need to be initialized distinct from assignment. This is true for const types and reference types.
- The order of member initialization is not the same as that specified in the constructor initializer list, but rather follows the order in which they appear in the class definition.
- This becomes important when one datamember is initialized in terms of others.
- A constructor with default arguments for all parameters also ends up defining the default constructor.