Initialization

Starting with C++11 you can provide in-definition initializers for each of your data members, just like Java. You should definitely take advantage of this as it will eliminate uninitialized data members.

```
struct Date
{
    std::string month;    // no initializer required
    int day = 0;          // legacy initialization
    int year{0};          // uniform initialization
};
```

Use legacy ("assignment") initialization (`day`), or uniform initialization (`year`). You **may not use direct initialization** with parentheses instead of braces. Note that `month` does not need an initializer, since it is a library type, and it will automatically be initialized by its constructor. However, you **may** explicitly initialize it as well, if you like.

Aggregate Initialization

You may **explicitly initialize** a structure variable by supplying **a list of values**, one for every data member, inside curly braces, separated by commas and ending with a semicolon. This is called **aggregate initialization**.

```
Date birthday = {"February", 2, 1950};
Date empty = {};
```

If you supply no initializers, as with the `Date empty`, then all members are **default initialized**. In this case, that means that `day` and `year` are set to `0` instead of a random number. If the members already have default initializers (from the structure definition), then those default initializers are used instead.