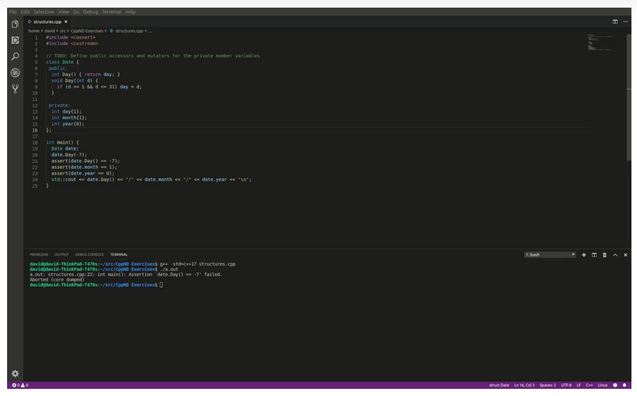

https://youtu.be/FTzrwV2LP5g

Classes

Classes, like structures, provide a way for C++ programmers to aggregate data together in a way that makes sense in the context of a specific program. By convention, programmers use structures when member variables are independent of each other, and **use classes when member variables are related by an "invariant"**.

Invariants

An "invariant" is a rule that limits the values of member variables.

For example, in a `Date` class, an invariant would specify that the member variable `day` cannot be less than 0. Another invariant would specify that the value of `day` cannot exceed 28, 29, 30, or 31, depending on the month and year. Yet another invariant would limit the value of `month` to the range of 1 to 12.

`Date` Class

Let's define a `Date` class:

```
// Use the keyword “class” to define a Date class:
class Date {
  int day{1};
  int month{1};
  int year{0};
};
```

So far, this class definition provides no invariants. The data members can vary independently of each other.

There is one subtle but important change that takes place when we change `struct Date` to `class Date`. By default, all members of a `struct` default to public, whereas all members of a `class` default to private. Since we have not specified access for the members of `class Date`, all of the members are private. In fact, we are not able to assign value to them at all!

`Date` Accessors And Mutators

As the first step to adding the appropriate invariants, let's specify that the member variable `day` is private. In order to access this member, we'll provide accessor and mutatot functions. Then we can add the appropriate invariants to the mutators.

```
class Date {
 public:
  int Day() { return day_; }
  void Day(int d) { day_ = d; }

 private:
  int day_{1};
  int month_{1};
  int year_{0};
};
```

`Date` Invariants

Now we can add the invariants within the mutators.

```
class Date {
 public:
  int Day() { return day; }
  void Day(int d) {
    if (d >= 1 && d <= 31) day_ = d;
  }

 private:
  int day_{1};
  int month_{1};
  int year_{0};
};
```

Now we have a set of invariants for the the class members!

As a general rule, member data subject to an invariant should be specified `private`, in order to enforce the invariant before updating the member's value.