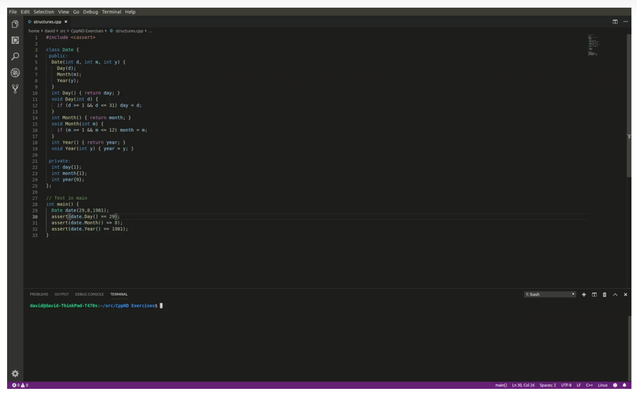

https://youtu.be/U2ItvJEwuHQ

Scope Resolution

C++ allows different **identifiers** (variable and function names) to have the same name, as long as they have different scope. For example, two different functions can each declare the variable `int i`, because each variable only exists within the scope of its parent function.

In some cases, scopes can overlap, in which case the compiler may need assistance in determining which identifier the programmer means to use. The process of determining which identifier to use is called **"scope resolution"**.

Scope Resultion Operator

`::` is the **scope resolution operator**. We can use this operator to specify which namespace or class to search in order to resolve an identifier.

```
Person::move(); \\ Call the move the function that is a member of the Person class.
std::map m; \\ Initialize the map container from the C++ Standard Library.
```

Class

Each class provides its own scope. We can use the scope resolution operator to specify identifiers from a class.

This becomes particularly useful if we want to separate class *declaration* from class *definition*.

```
class Date {
 public:
  int Day() const { return day; }
  void Day(int day);  // Declare member function Date::Day().
  int Month() const { return month; }
  void Month(int month) {
    if (month >= 1 && month <= 12) Date::month = month;
  }
  int Year() const { return year; }
  void Year(int year) { Date::year = year; }

 private:
  int day{1};
  int month{1};
  int year{0};
};

// Define member function Date::Day().
void Date::Day(int day) {
  if (day >= 1 && day <= 31) Date::day = day;
}
```

Namespaces

Namespaces allow programmers to group logically related variables and functions together. Namespaces also help to avoid conflicts between to variables that have the same name in different parts of a program.

```
namespace English {
void Hello() { std::cout << "Hello, World!\n"; }
}  // namespace English

namespace Spanish {
void Hello() { std::cout << "Hola, Mundo!\n"; }
}  // namespace Spanish

int main() {
  English::Hello();
  Spanish::Hello();
}
```

In this example, we have two different `void Hello()` functions. If we put both of these functions in the same namespace, they would conflict and the program would not compile. However, by declaring each of these functions in a separate namespace, they are able to co-exist. Furthermore, we can specify which function to call by prefixing `Hello()` with the appropriate namespace, followed by the `::` operator.

`std` Namespace

You are already familiar with the `std` namespace, even if you didn't realize quite what it was. `std` is the namespace used by the **C++ Standard Library**.

Classes like `std::vector` and functions like `std::sort` are defined within the `std` namespace.