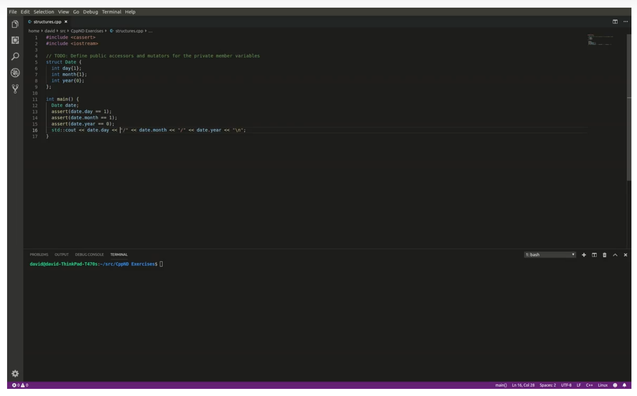

https://youtu.be/ZEt2LTRc2D0

Access Specifiers

Members of a structure can be specified as `public` or `private`.

By default, all members of a structure are public, unless they are specifically marked `private`.

Public members can be changed directly, by any user of the object, whereas private members can only be changed by the object itself.

Private Members

This is an implementation of the `Date` structure, with all members marked as private.

```
struct Date {
 private:
  int day{1};
  int month{1};
  int year{0};
};
```

Private members of a class are accessible only from within other member functions of the same class (or from their "friends", which we'll talk about later).

There is a third access modifier called `protected`, which implies that members are accessible from other member functions of the same class (or from their "friends"), and also from members of their derived classes. We'll also discuss about derived classes later, when we learn about inheritance.

Accessors And Mutators

To access private members, we typically define public "accessor" and "mutator" member functions (sometimes called "getter" and "setter" functions).

```
struct Date {
 public:
  int Day() { return day; }
  void Day(int day) { this.day = day; }
  int Month() { return month; }
  void Month(int month) { this.month = month; }
  int Year() { return year; }
  void Year(int year) { this.year = year; }

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

In the last example, you saw how to create a setter function for class member attributes. Check out the code in the Notebook below to play around a bit with access modifiers as well as setter and getter functions!

Avoid Trivial Getters And Setters

Sometimes accessors are not necessary, or even advisable. The **C++ Core Guidelines** recommend, "A trivial getter or setter adds no semantic value; the data item could just as well be public."

Here is the example from the Core Guidelines:

```
class Point {
    int x;
    int y;
public:
    Point(int xx, int yy) : x{xx}, y{yy} { }
    int get_x() const { return x; }  // const here promises not to modify the object
    void set_x(int xx) { x = xx; }
    int get_y() const { return y; }  // const here promises not to modify the object
    void set_y(int yy) { y = yy; }
    // no behavioral member functions
};
```

This `class` could be made into a `struct`, with no logic or "invariants", just passive data. The member variables could both be public, with no accessor functions:

```
struct Point {   // Good: concise
    int x {0};   // public member variable with a default initializer of 0
    int y {0};   // public member variable with a default initializer of 0
};
```