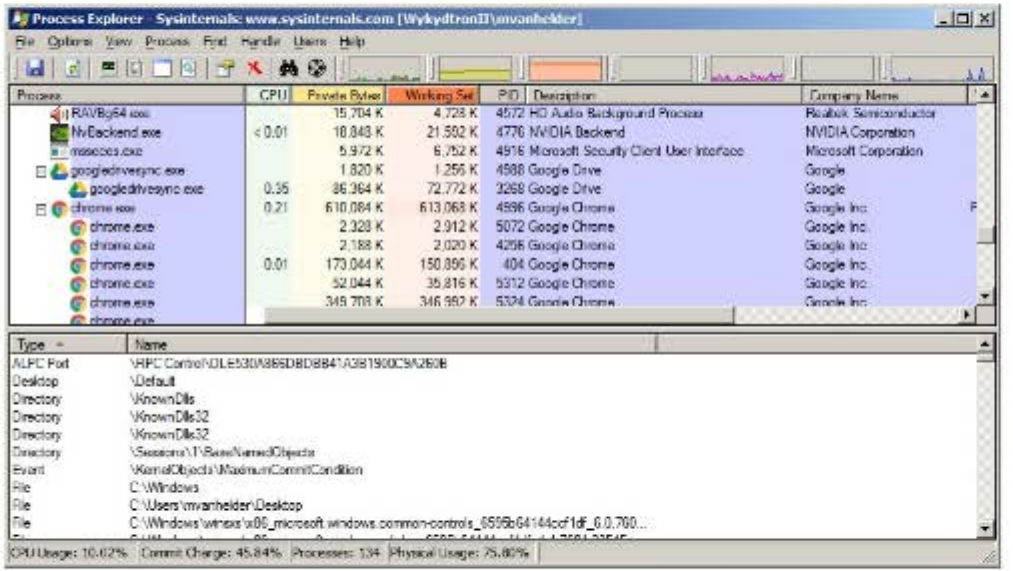
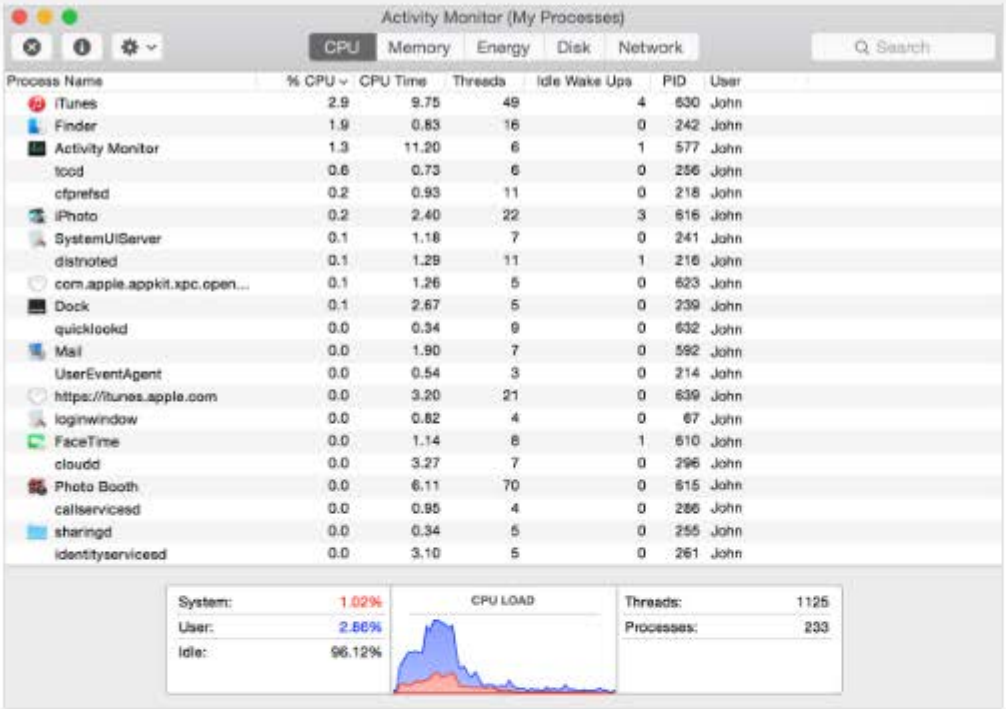
If you use Windows, then you might be familiar with the Process Explorer or Process Monitor applications:



Source: PC World

On the other hand, if you are a Mac user, you've probably seen the Activity Monitor before:



And finally, if you are a Linux user, you may have used top or htop to view active processes on your computer.

At the end of this course, you will use C++ to build a process monitor, similar to htop . This process monitor will run on Linux.

The process monitor will allow you to see all the active processes on the system, with their corresponding process ids (PIDs), CPU usage, and memory usage:

root@77e30fca8a01: /home/workspace/CppND-Object-Oriented

root@77e30fca8a01: /home/workspace/CppND-Object-Oriented 108x20

```
  1  [|                                        0.7%]   Tasks: 38, 189 thr; 2 running
  2  [                                         0.0%]   Load average: 0.03 0.01 0.00
  3  [||                                       2.7%]   Uptime: 00:52:30
  4  [|                                        0.7%]
  Mem[||||||||||                         984M/15.7G]
  Swp[                                     0K/20.0G]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 1672 root       20   0 49536  6552  5076 R  2.0  0.0  0:09.07 ./a.out
   30 root       20   0 72968 35364 14992 S  0.7  0.2  0:07.99 /opt/TurboVNC/bin/Xvnc :1 -desktop TurboVNC:
 1358 root       20   0 1482M  101M 80044 S  0.7  0.6  0:01.55 /usr/lib/firefox/firefox -contentproc -childI
 1605 root       20   0 57944  6572  5132 R  0.0  0.0  0:02.24 htop
 1206 root       20   0 2002M  255M  133M S  0.0  1.6  0:00.14 /usr/lib/firefox/firefox
   69 root       20   0 87472 23244  9540 S  0.0  0.1  0:00.42 python /opt/noVNC/utils/websockify/run --web
 1042 root       20   0 96284 20344  6520 S  0.0  0.1  0:03.69 python /opt/noVNC/utils/websockify/run --web
 1611 root       20   0  529M 45144 26792 S  0.0  0.3  0:00.86 /usr/bin/python /usr/bin/terminator -e vglrun
 1068 root       20   0  529M 45584 27056 S  0.0  0.3  0:02.29 /usr/bin/python /usr/bin/terminator -e vglrun
 1369 root       20   0 1482M  101M 80044 S  0.0  0.6  0:00.28 /usr/lib/firefox/firefox -contentproc -childI
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

```
PID:   User:  CPU[%]:   RAM[MB]: Uptime:  CMD:
1192   root   342.5     0.328     10.09    /usr/lib/firefox/firefox
1358   root   51.23     0.067     1.070    /usr/lib/firefox/firefox
1379   root   103.4     0.086     0.000    /usr/lib/firefox/firefox
1439   root   68.98     0.030     0.000    /usr/lib/firefox/firefox
1515   root   36.17     0.005     0.000    /usr/lib/firefox/firefox
1605   root   1.476     0.286     0.590    htop
1611   root   50.35     0.028     0.720    /usr/bin/python
1620   root   0.230     0.046     0.120    gnome-pty-helper
1625   root   1.726     0.034     0.000    bash
1672   root   1.578     0.812     5.060    ./a.out
```

Jupyter with C++

In this lesson, you'll be writing and testing lots of C++ code. C++ is a *compiled* language, which is to say there is a separate program - the compiler - that converts your code to an executable program that the computer can run. This means that, after you save a new C++ program to file, running it is normally a two step process:

1. Compile your code with a compiler.
2. Run the executable file that the compiler outputs.

For example, in the notebook exercises that follow, you'll be saving your code in a file, let's say `filename.cpp` in a folder called `/code`. To compile it using the C++17 standard, you can run the following command:

```
g++ -std=c++17 ./code/filename.cpp
```

And then to run the resulting executable file, you can run:

```
./a.out
```

Jupyter Notebooks in Udacity Classroom

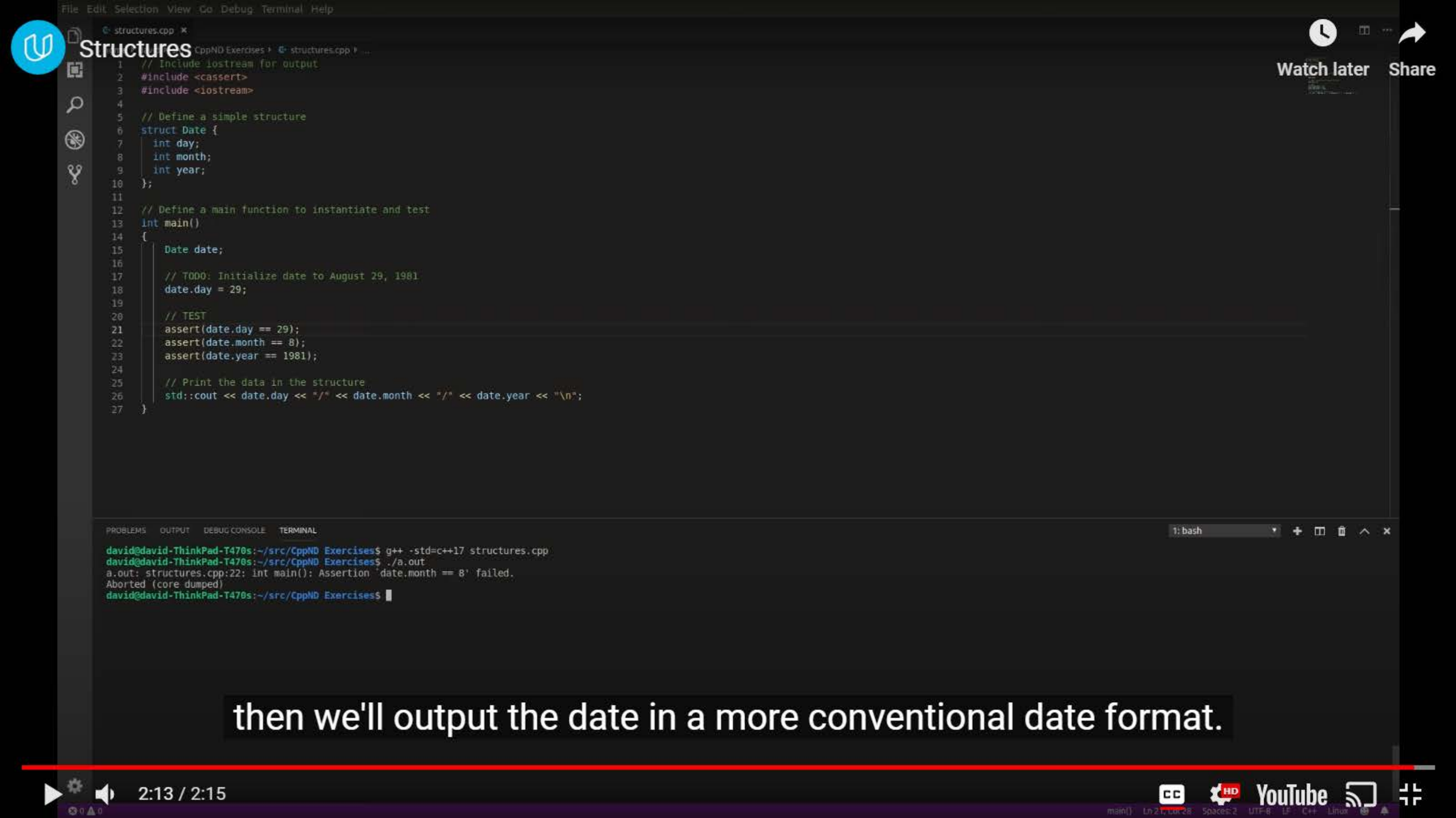
In this lesson, you will save, compile, and run executables over and over. To make your life simpler, we've set things up so you can save, compile and run with the click of a single `Compile & Run` button in the Jupyter Notebooks.

Later, when you build the project, you'll move out of the notebooks and into a Linux virtual machine. At that point, you'll need to remember to compile and run the programs yourself!

If you haven't seen Jupyter Notebooks before, you can test one out below. A Notebook is a web application that allows for code, text, and visualizations to be combined and shared.

Check out the Notebook below for an example of how these will be used in the course.

When you use a Notebook Workspace, we encourage you to expand to a full screen view. Click on the `EXPAND` button in the lower left corner.


Structures
Watch later
Share
structures.cpp
CppND Exercises › structures.cpp
// Include iostream for output
#include <cassert>
#include <iostream>
// Define a simple structure
struct Date {
int day;
int month;
int year;
};
// Define a main function to instantiate and test
int main()
{
Date date;
// TODO: Initialize date to August 29, 1981
date.day = 29;
// TEST
assert(date.day == 29);
assert(date.month == 8);
assert(date.year == 1981);
// Print the data in the structure
std::cout << date.day << "/" << date.month << "/" << date.year << "\n";
}
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
david@david-ThinkPad-T470s:~/src/CppND Exercises$ g++ -std=c++17 structures.cpp
david@david-ThinkPad-T470s:~/src/CppND Exercises$ ./a.out
a.out: structures.cpp:22: int main(): Assertion 'date.month == 8' failed.
Aborted (core dumped)
david@david-ThinkPad-T470s:~/src/CppND Exercises$
then we'll output the date in a more conventional date format.
2:13 / 2:15
YouTube

Structures

Structures allow developers to create their own types ("user-defined" types) to aggregate data relevant to their needs.

For example, a user might define a `Rectangle` structure to hold data about rectangles used in a program.

```
struct Rectangle {
  float length;
  float width;
};
```

Types

Every C++ variable is defined with a type.

```
int value;
Rectangle rectangle;
Sphere earth;
```

In this example, the "type" of `value` is `int`. Furthermore, `rectangle` is "of type" `Rectangle`, and `earth` has type `Sphere`.

Fundamental Types

C++ includes fundamental types, such as `int` and `float`. These fundamental types are sometimes called "primitives".

Fundamental Types

C++ includes fundamental types, such as `int` and `float`. These fundamental types are sometimes called "primitives".

The Standard Library [includes additional types](, such as `std::size_t` and `std::string`.

User-Defined Types

Structures are "user-defined" types. Structures are a way for programmers to create types that aggregate and store data in way that makes sense in the context of a program.

For example, C++ does not have a fundamental type for storing a date. (The Standard Library does include types related to time, which can be converted to dates.)

A programmer might desire to create a type to store a date.

Consider the following example:

```
struct Date {
  int day;
  int month;
  int year;
};
```

The code above creates a structure containing three "member variables" of type `int`: `day`, `month` and `year`.

If you then create an "instance" of this structure, you can initialize these member variables:

```
// Create an instance of the Date structure
Date date;
// Initialize the attributes of Date
date.day = 1;
date.month = 10;
date.year = 2019;
```

Member Initialization

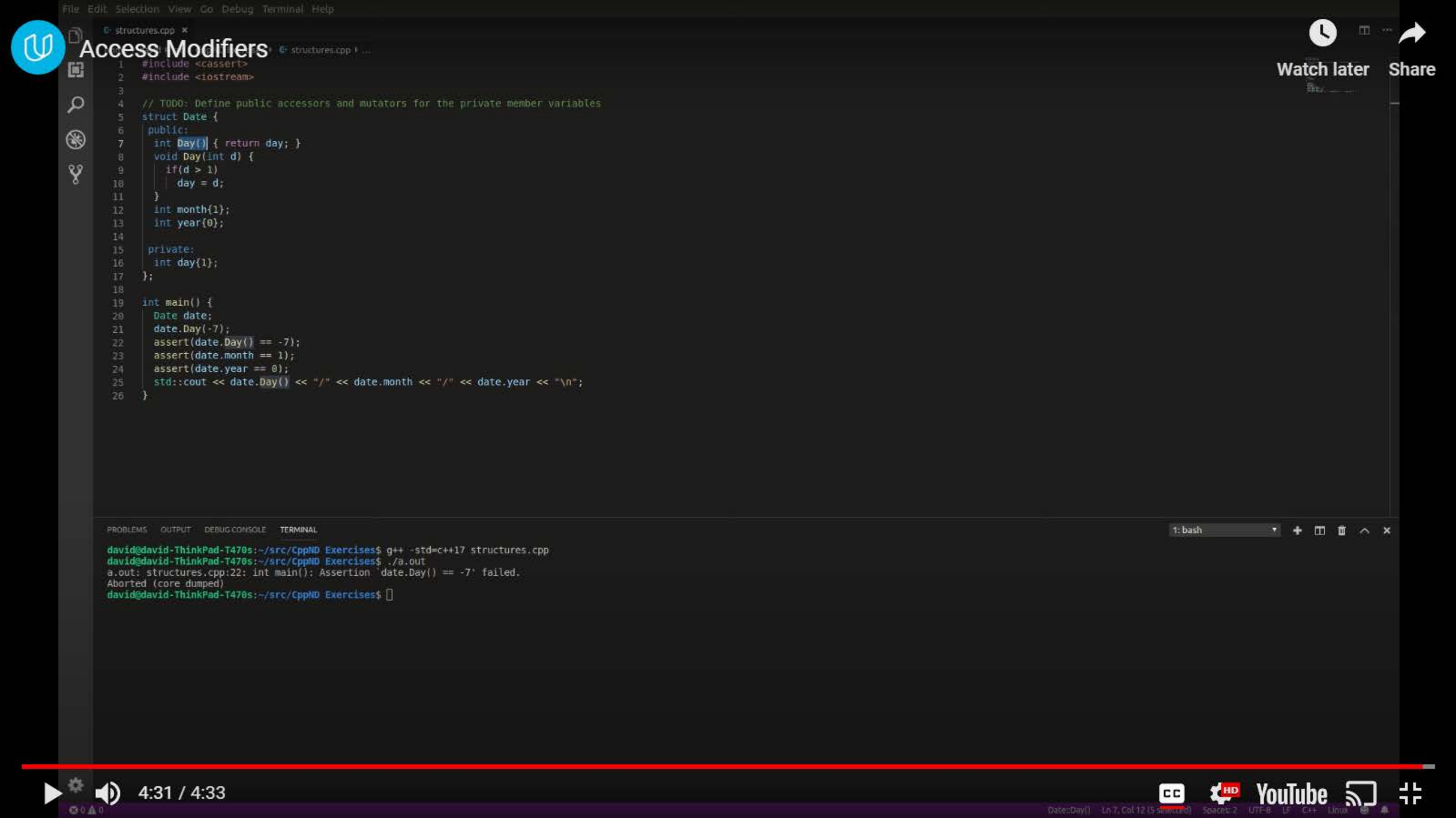
Generally, we want to avoid instantiating an object with undefined members. Ideally, we would like all members of an object to be in a valid state once the object is instantiated. We can change the values of the members later, but we want to avoid any situation in which the members are ever in an invalid state or undefined.

In order to ensure that objects of our `Date` structure always start in a valid state, we can initialize the members from within the structure definition.

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

There are also several other approaches to either initialize or assign member variables when the object is instantiated. For now, however, this approach ensures that every object of `Date` begins its life in a defined and valid state.

Access Modifiers



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

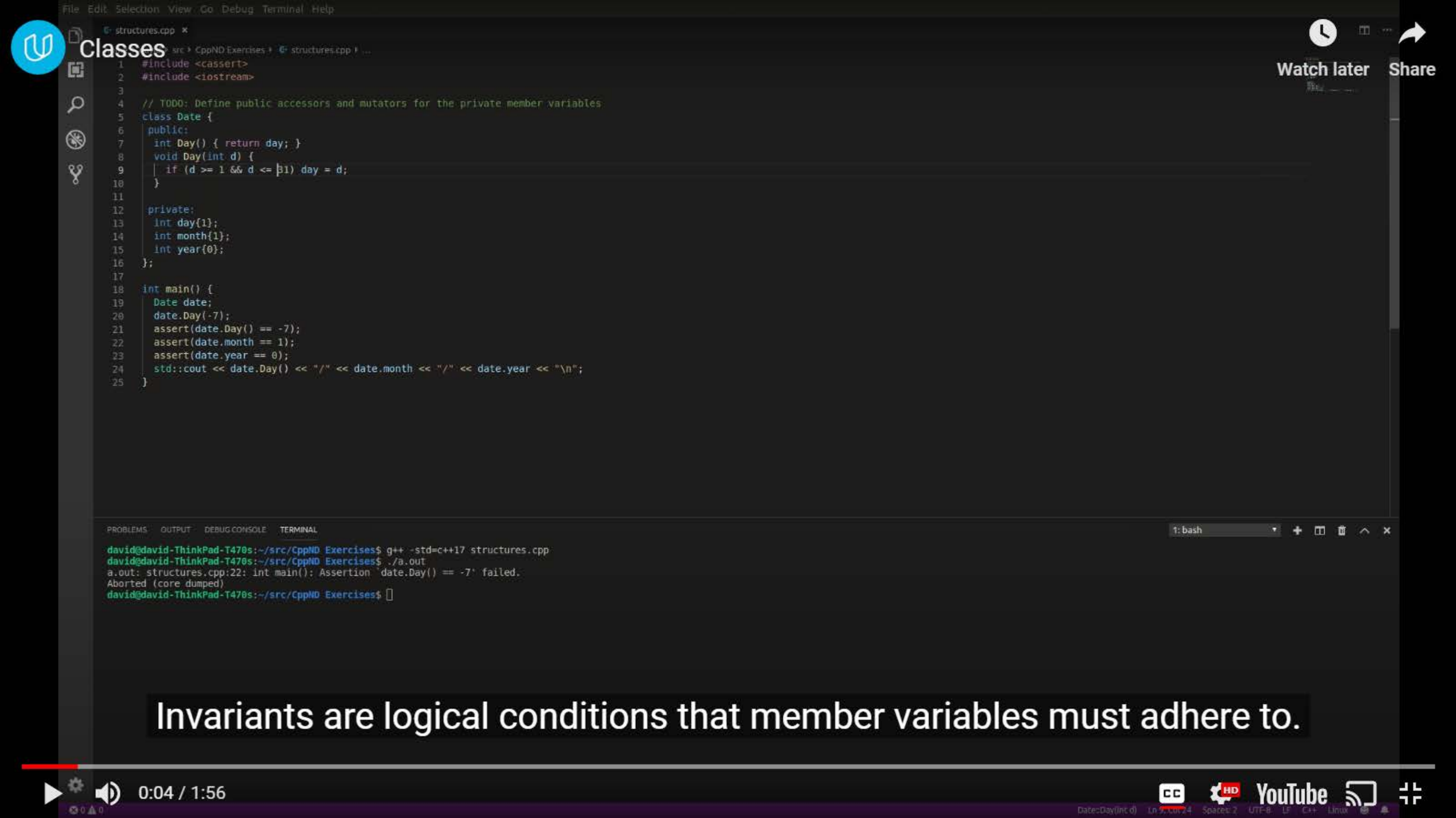
Sometimes accessors are not necessary, or even advisable. The C++ Core Guidelines recommend, "A trivial getter or setter adds no semantic value; the data item could just as well be public."

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
};
```


Classes
Watch later
Share
structures.cpp
src › CppND Exercises › structures.cpp
#include <cassert>
#include <iostream>
// TODO: Define public accessors and mutators for the private member variables
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};
};
int main() {
Date date;
date.Day(-7);
assert(date.Day() == -7);
assert(date.month == 1);
assert(date.year == 0);
std::cout << date.Day() << "/" << date.month << "/" << date.year << "\n";
}
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
david@david-ThinkPad-T470s:~/src/CppND Exercises$ g++ -std=c++17 structures.cpp
david@david-ThinkPad-T470s:~/src/CppND Exercises$ ./a.out
a.out: structures.cpp:22: int main(): Assertion `date.Day() == -7' failed.
Aborted (core dumped)
david@david-ThinkPad-T470s:~/src/CppND Exercises$
Invariants are logical conditions that member variables must adhere to.
0:04 / 1:56
YouTube

File Edit Selection View Go Debug Terminal Help

structures.cpp

src › CppND Exercises › structures.cpp › ...

```cpp
#include <cassert>
#include <iostream>

// TODO: Define public accessors and mutators for the private member variables
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};
};

int main() {
  Date date;
  date.Day(-7);
  assert(date.Day() == -7);
  assert(date.month == 1);
  assert(date.year == 0);
  std::cout << date.Day() << "/" << date.month << "/" << date.year << "\n";
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: bash

```
david@david-ThinkPad-T470s:~/src/CppND Exercises$ g++ -std=c++17 structures.cpp
david@david-ThinkPad-T470s:~/src/CppND Exercises$ ./a.out
a.out: structures.cpp:22: int main(): Assertion `date.Day() == -7' failed.
Aborted (core dumped)
david@david-ThinkPad-T470s:~/src/CppND Exercises$
```

then the convention is typically to convert from a struct into a class

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.

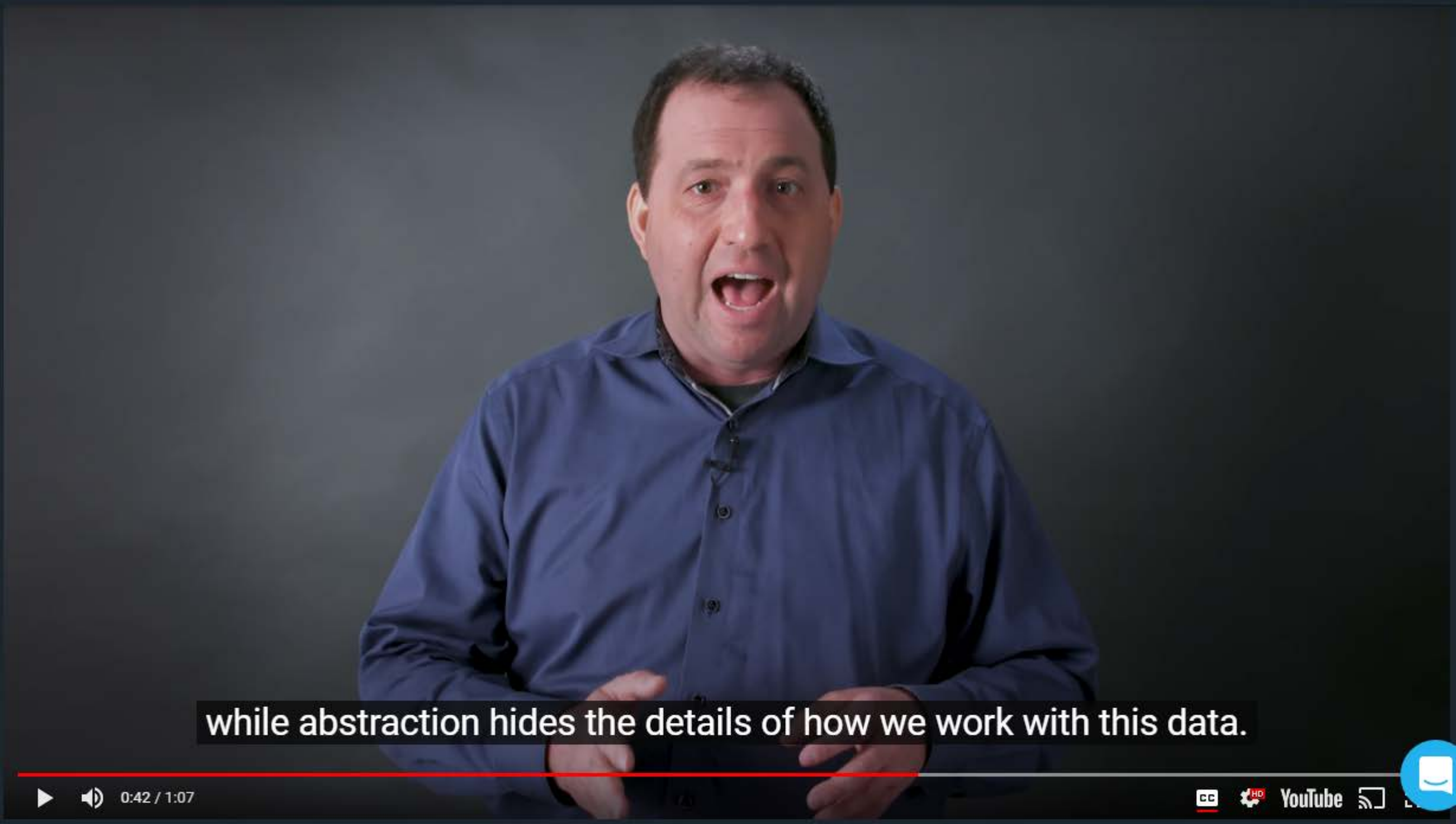
```
class Date {
 public:
  int Day() { return day; }
  void Day(int d) {
    if (d >= 1 && d <= 31) day_ = d;
  }

 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.


Encapsulation and abstraction are related because
0:34 / 1:07
YouTube


encapsulation groups relevant data together in a class,
0:38 / 1:07
YouTube


while abstraction hides the details of how we work with this data.
0:42 / 1:07
YouTube


I provide what's called a public interface and private part the implementation details.
1:00 / 1:12
YouTube

Constructors

Constructors are member functions of a class or struct that initialize an object. The Core Guidelines define a constructor) as:

> *constructor*: an operation that initializes ("constructs") an object. Typically a constructor establishes an invariant and often acquires resources needed for an object to be used (which are then typically released by a destructor).

A constructor can take arguments, which can be used to assign values to member variables.

```
class Date {
 public:
  Date(int d, int m, int y) {  // This is a constructor.
    Day(d);
  }
  int Day() { return day; }
  void Day(int d) {
    if (d >= 1 && d <= 31) day = d;
  }
  int Month() { return month; }
  void Month(int m) {
    if (m >= 1 && m <= 12) month = m;
  }
  int Year() { return year_; }
  void Year(int y) { year = y; }

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

As you can see, a constructor is also able to call other member functions of the object it is constructing.
In the example above, `Date(int d, int m, int y)` assigns a member variable by calling `Day(int d)`.

Default Constructor

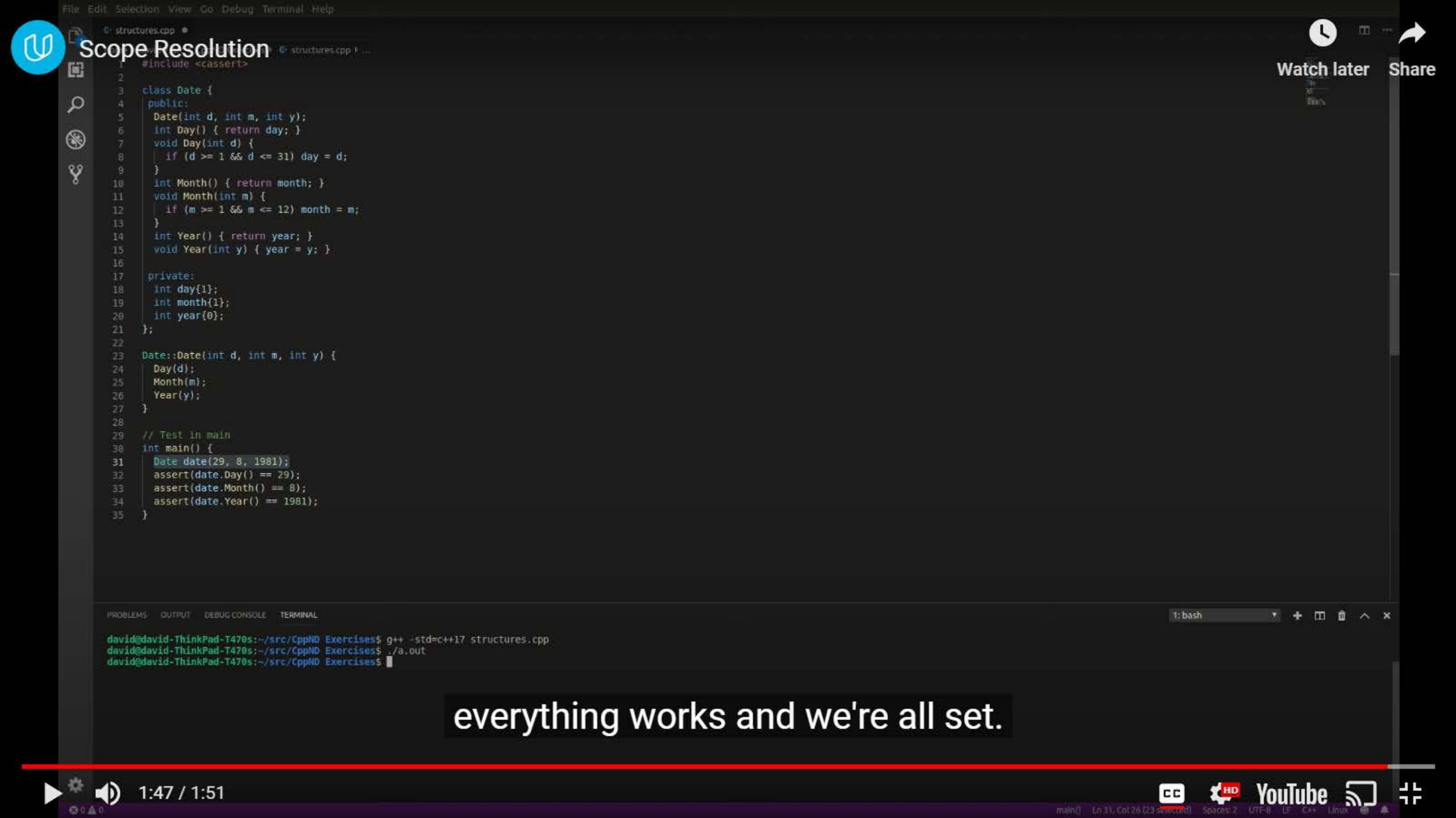
A class object is always initialized by calling a constructor. That might lead you to wonder how it is possible to initialize a class or structure that does not define any constructor at all.

For example:

```
class Date {
  int day{1};
  int month{1};
  int year{0};
};
```

We can initialize an object of this class, even though this class does not explicitly define a constructor.

This is possible because of the default constructor. The compiler will define a default constructor, which accepts no arguments, for any class or structure that does not contain an explicitly-defined constructor.


Scope Resolution
Watch later
Share
structures.cpp
#include <cassert>
class Date {
public:
Date(int d, int m, int y);
int Day() { return day; }
void Day(int d) {
if (d >= 1 && d <= 31) day = d;
}
int Month() { return month; }
void Month(int m) {
if (m >= 1 && m <= 12) month = m;
}
int Year() { return year; }
void Year(int y) { year = y; }
private:
int day{1};
int month{1};
int year{0};
};
Date::Date(int d, int m, int y) {
Day(d);
Month(m);
Year(y);
}
// Test in main
int main() {
Date date(29, 8, 1981);
assert(date.Day() == 29);
assert(date.Month() == 8);
assert(date.Year() == 1981);
}
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
david@david-ThinkPad-T470s:~/src/CppND Exercises$ g++ -std=c++17 structures.cpp
david@david-ThinkPad-T470s:~/src/CppND Exercises$ ./a.out
david@david-ThinkPad-T470s:~/src/CppND Exercises$
everything works and we're all set.
1:47 / 1:51
YouTube

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.

```cpp
class Date {
 public:
  Date(int day, int month, int year);
  int Day() { return day; }
  void Day(int day);
  int Month() { return month; }
  void Month(int month);
  int Year() { return year; }
  void Year(int year) { Date::year = year; }

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

Date::Date(int d, int m, int y) : year(y) {
  Day(d);
  Month(m);
}

void Date::Day(int day) {
  if (day >= 1 && day <= 31) Date::day = day;
}

void Date::Month(int month) {
  if (month >= 1 && month <= 12) Date::month = month;
}

// Test in main
int main() {
  Date date(-2, 1000, 1981);
  assert(date.Day() == -2);
  assert(date.Month() == 1000);
  assert(date.Year() == 1981);
  std::cout << date.Day() << "/" << date.Month() << "/" << date.Year() << "\n";
}
```

```
david@david-ThinkPad-T470s:~/src/CppND Exercises$ g++ -std=c++17 structures.cpp
david@david-ThinkPad-T470s:~/src/CppND Exercises$ ./a.out
29/8/1981
david@david-ThinkPad-T470s:~/src/CppND Exercises$ g++ -std=c++17 structures.cpp
david@david-ThinkPad-T470s:~/src/CppND Exercises$ ./a.out
29/8/1981
david@david-ThinkPad-T470s:~/src/
david@david-ThinkPad-T470s:~/src/
-2/1000/1981
david@david-ThinkPad-T470s:~/src/CppND Exercises$
```

the attached invariants when we construct the object.

Initializer Lists

Initializer lists initialize member variables to specific values, just before the class constructor runs. This initialization ensures that class members are automatically initialized when an instance of the class is created.

```
Date::Date(int day, int month, int year) : year_(y) {
  Day(day);
  Month(month);
}
```

In this example, the member value `year` is initialized through the initializer list, while `day` and `month` are assigned from within the constructor. Assigning `day` and `month` allows us to apply the invariants set in the mutator.

In general, prefer initialization to assignment. Initialization sets the value as soon as the object exists, whereas assignment sets the value only after the object comes into being. This means that assignment creates and opportunity to accidentally use a variable before its value is set.

In fact, initialization lists ensure that member variables are initialized *before* the object is created. This is why class member variables can be declared `const`, but only if the member variable is initialized through an initialization list. Trying to initialize a `const` class member within the body of the constructor will not work.

Instructions

1. Declare `class Person`.
2. Add `std::string name` to `class Person`.
3. Create a constructor for `class Person`.
4. Add an initializer list to the constructor.
5. Create class object.

Initializing Constant Members

Watch later Share

```cpp
#include <cassert>

class Birthday {
 public:
  Birthday(int d, int m, int y);
  int Day() { return day; }
  int Month() { return month; }
  int Year() { return year; }

 private:
  int const day;
  int const month;
  int const year;
};

const int Birthday::day
Birthday::Birthday(int d, int m, int y) : day(d), month(m), year(y) {}

// Test in main
int main() {
  Birthday date(-29, 8, 1981);
  assert(date.Day() == -29);
  assert(date.Month() == 8);
  assert(date.Year() == 1981);
}
```

```
david@david-ThinkPad-T470s:~/src/CppND Exercises$ g++ -std=c++17 structures.cpp
david@david-ThinkPad-T470s:~/src/CppND Exercises$ ./a.out
david@david-ThinkPad-T470s:~/src/CppND Exercises$ g++ -std=c++17 structures.cpp
david@david-ThinkPad-T470s:~/src/CppND Exercises$ ./a.out
david@david-ThinkPad-T470s:~/src/CppND Exercises$
```

what we need to do in order to initialize them is to use these initializer lists.

3:29 / 3:33

Exercise: Constructor Syntax

Initializer lists exist for a number of reasons. First, the compiler can optimize initialization faster from an initialization list than from within the constructor.

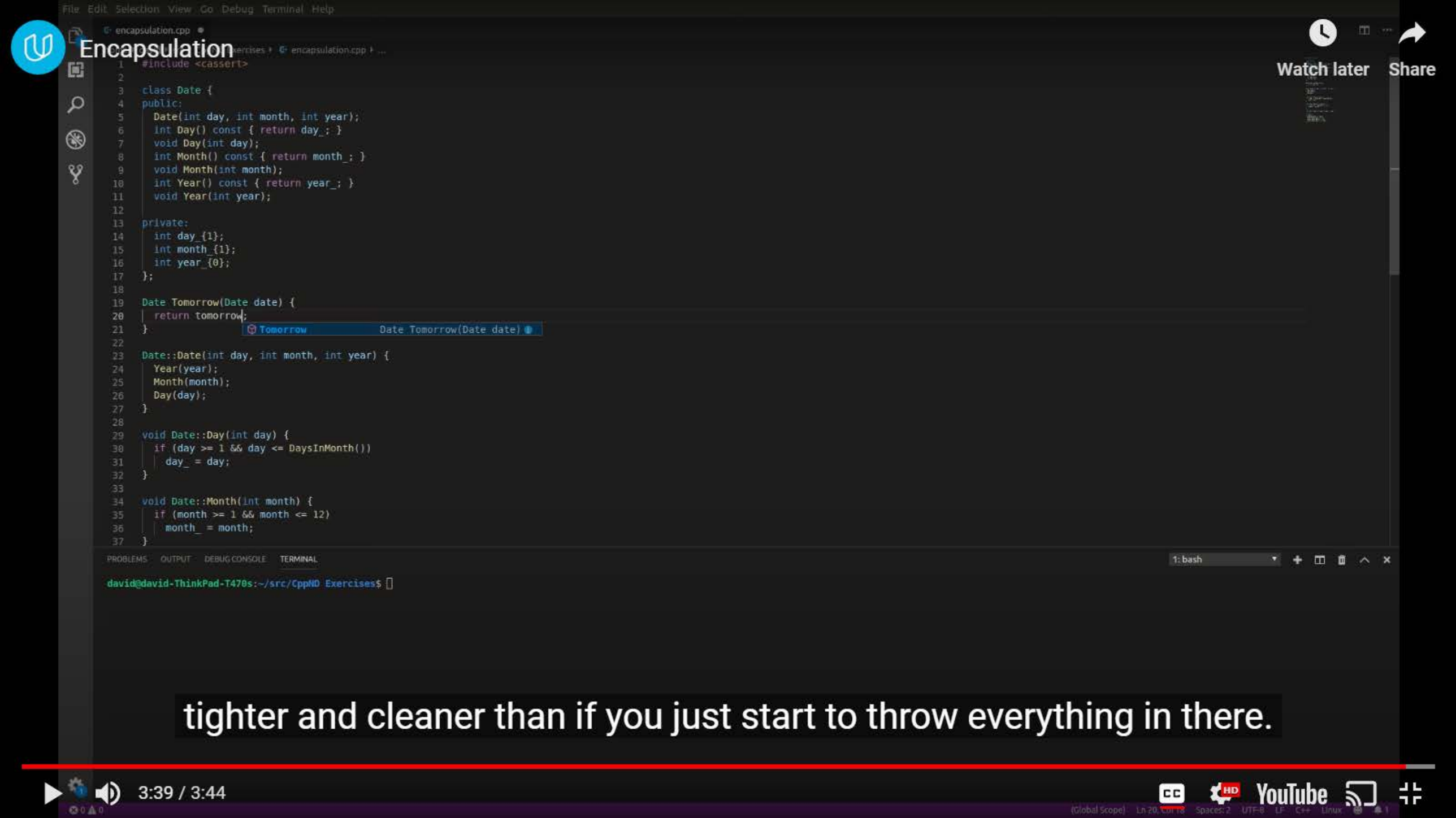
A second reason is a bit of a technical paradox. If you have a `const` class attribute, you can only initialize it using an initialization list. Otherwise, you would violate the `const` keyword simply by initializing the member in the constructor!

The third reason is that attributes defined as references must use initialization lists.

This exercise showcases several advantages of initializer lists.

Instructions

1. Modify the exist code to use an initialization list.
2. Verify that the test passes.


Encapsulation
Watch later
Share
encapsulation.cpp
#include <cassert>
class Date {
public:
Date(int day, int month, int year);
int Day() const { return day_; }
void Day(int day);
int Month() const { return month_; }
void Month(int month);
int Year() const { return year_; }
void Year(int year);
private:
int day_{1};
int month_{1};
int year_{0};
};
Date Tomorrow(Date date) {
return tomorrow;
}
Tomorrow Date Tomorrow(Date date)
Date::Date(int day, int month, int year) {
Year(year);
Month(month);
Day(day);
}
void Date::Day(int day) {
if (day >= 1 && day <= DaysInMonth())
day_ = day;
}
void Date::Month(int month) {
if (month >= 1 && month <= 12)
month_ = month;
}
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
david@david-ThinkPad-T470s:~/src/CppND Exercises$
tighter and cleaner than if you just start to throw everything in there.
3:39 / 3:44
YouTube

Encapsulation

Encapsulation is the grouping together of data and logic into a single unit. In object-oriented programming, classes encapsulate data and functions that operate on that data.

This can be a delicate balance, because on the one hand we want to group together relevant data and functions, but on the hand we want to **limit member functions to only those functions that need direct access to the representation of a class**.

In the context of a `Date` class, a function `Date Tomorrow(Date const & date)` probably does not need to be encapsulated as a class member. It can exist outside the `Date` class.

However, a function that calculates the number of days in a month probably should be encapsulated with the class, because the class needs this function in order to operate correctly.

encapsulation.cpp

```cpp
#include <cassert>

class Date {
public:
  Date(int day, int month, int year);
  int Day() const { return day_; }
  void Day(int day);
  int Month() const { return month_; }
  void Month(int month);
  int Year() const { return year_; }
  void Year(int year);

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

Date Tomorrow(Date date) {
  return tomorrow;
}

Date::Date(int day, int month, int year) {
  Year(year);
  Month(month);
  Day(day);
}

void Date::Day(int day) {
  if (day >= 1 && day <= DaysInMonth())
    day_ = day;
}

void Date::Month(int month) {
  if (month >= 1 && month <= 12)
    month_ = month;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: bash

```
david@david-ThinkPad-T470s:~/src/CppND Exercises$
```

They are typically marked as constant because they are

Accessor Functions

Accessor functions are public member functions that allow users to access an object's data, albeit indirectly.

`const`

Accessors should only retrieve data. They should not change the data stored in the object.

The main role of the `const` specifier in accessor methods is to protect member data. When you specify a member function as `const`, the compiler will prohibit that function from changing any of the object's member data.

Exercise: Bank Account Class

Your task is to design and implement class called `BankAccount`. This will be a generic account defined by its account number, the name of the owner and the funds available.

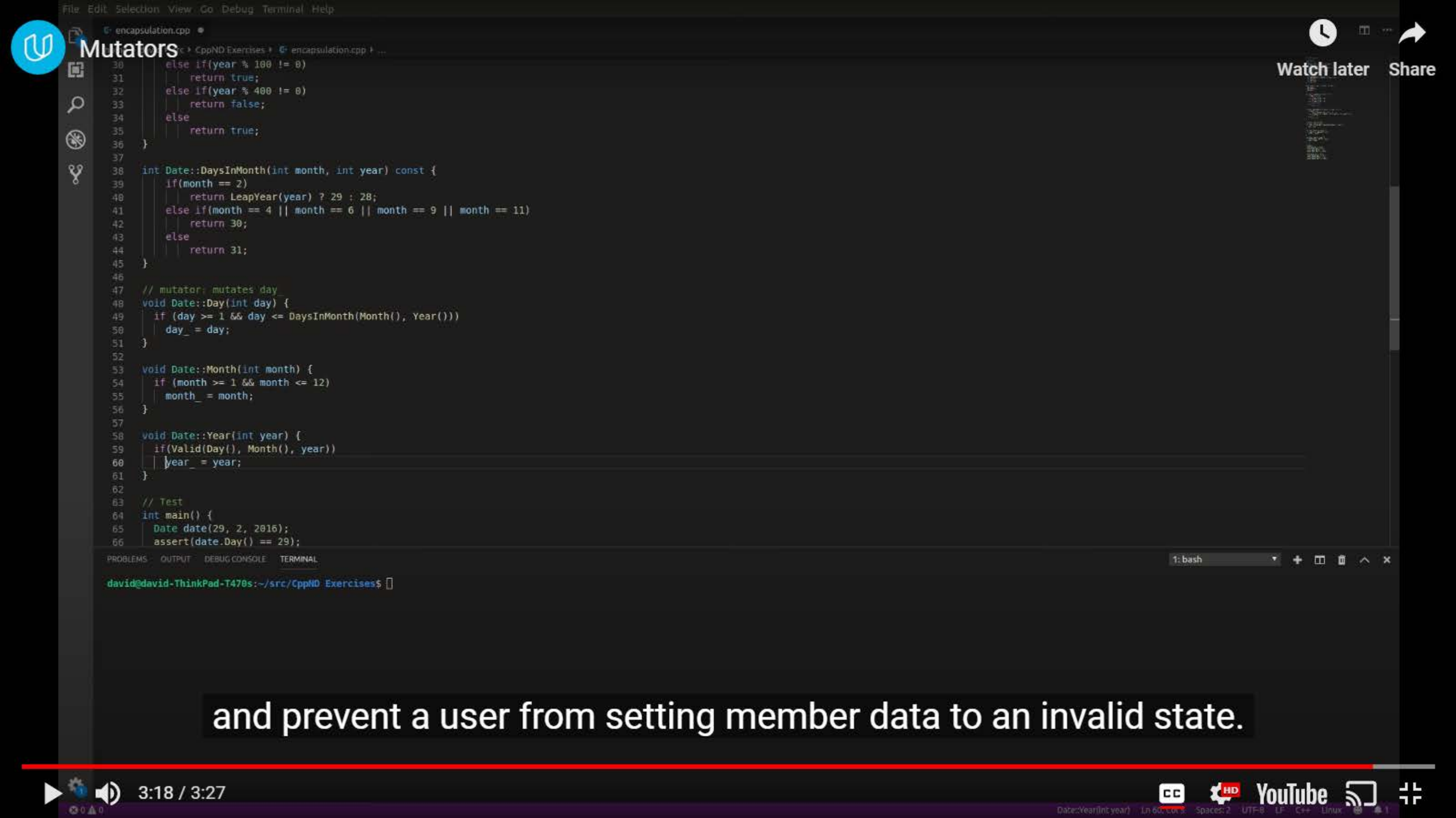
Complete the following steps:

1. Create class called `BankAccount`
2. Use typical info about bank accounts to design attributes, such as the account number, the owner name, and the available funds.
3. Specify access so that member data are protected from other parts of the program.
4. Create accessor and mutator functions for member data.

```
30      else if(year % 100 != 0)
31          return true;
32      else if(year % 400 != 0)
33          return false;
34      else
35          return true;
36  }
37
38  int Date::DaysInMonth(int month, int year) const {
39      if(month == 2)
40          return LeapYear(year) ? 29 : 28;
41      else if(month == 4 || month == 6 || month == 9 || month == 11)
42          return 30;
43      else
44          return 31;
45  }
46
47  // mutator: mutates day_
48  void Date::Day(int day) {
49    if (day >= 1 && day <= DaysInMonth(Month(), Year()))
50      day_ = day;
51  }
52
53  void Date::Month(int month) {
54    if (month >= 1 && month <= 12)
55      month_ = month;
56  }
57
58  void Date::Year(int year) {
59    if(Valid(Day(), Month(), year))
60      year_ = year;
61  }
62
63  // Test
64  int main() {
65    Date date(29, 2, 2016);
66    assert(date.Day() == 29);
```

```
david@david-ThinkPad-T470s:~/src/CppND Exercises$
```

But that's the whole point of mutators to allow us to set invariant logic for the class,


Mutators
Watch later
Share
encapsulation.cpp
CppND Exercises
encapsulation.cpp
else if(year % 100 != 0)
return true;
else if(year % 400 != 0)
return false;
else
return true;
}
int Date::DaysInMonth(int month, int year) const {
if(month == 2)
return LeapYear(year) ? 29 : 28;
else if(month == 4 || month == 6 || month == 9 || month == 11)
return 30;
else
return 31;
}
// mutator: mutates day_
void Date::Day(int day) {
if (day >= 1 && day <= DaysInMonth(Month(), Year()))
day_ = day;
}
void Date::Month(int month) {
if (month >= 1 && month <= 12)
month_ = month;
}
void Date::Year(int year) {
if(Valid(Day(), Month(), year))
year_ = year;
}
// Test
int main() {
Date date(29, 2, 2016);
assert(date.Day() == 29);
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
david@david-ThinkPad-T470s:~/src/CppND Exercises$
and prevent a user from setting member data to an invalid state.
3:18 / 3:27
YouTube

Mutator Functions

A mutator ("setter") function can apply logic ("invariants") when updating member data.

Exercise: Car Class

In this lab you will create a setter method that receives data as an argument an converts it to a different type. Specifically, you will receive a string as input and convert it to a character array.

1. Create a class called `Car`.
2. Create 3 member variables: `horsepower`, `weight` and `brand`. The `brand` attribute must be a character array.
3. Create accessor and mutator functions for all member data. The mutator function for `brand` must accept a C++ string as a parameter and convert that string into a C-style string (a character array ending in null character) to set the value of `brand`.
4. The accessor function for the `brand` must return a string, so in this function you first will need to convert `brand` to `std::string`, and then return it.

QUESTION 1 OF 3

The constructor function of a class is a special member function that defines any input parameters or logic that must be included upon instantiation of a class. From what you've seen so far is it required to define a constructor in C++ classes?

- ✓ No, if undefined C++ will define a default constructor
- Yes, without a constructor defined you cannot instantiate a class.

SUBMIT

QUESTION 2 OF 3

What are the three options for access modifiers in C++?

- Public (access to anyone), Private (access only within the class) and Permitted (access in friend classes)
- Public (access to anyone), Protected (access in friend classes) and Permitted (access only within the class)
- ✓ Public (access to anyone), Private (access only within the class) and Protected (access in friend classes)
- Public (access in friend classes), Private (access only within the class) and Protected (access to anyone)

SUBMIT

QUESTION 3 OF 3

Why does it make sense to specify private member variables with accessor and mutator functions, instead of public member variables?

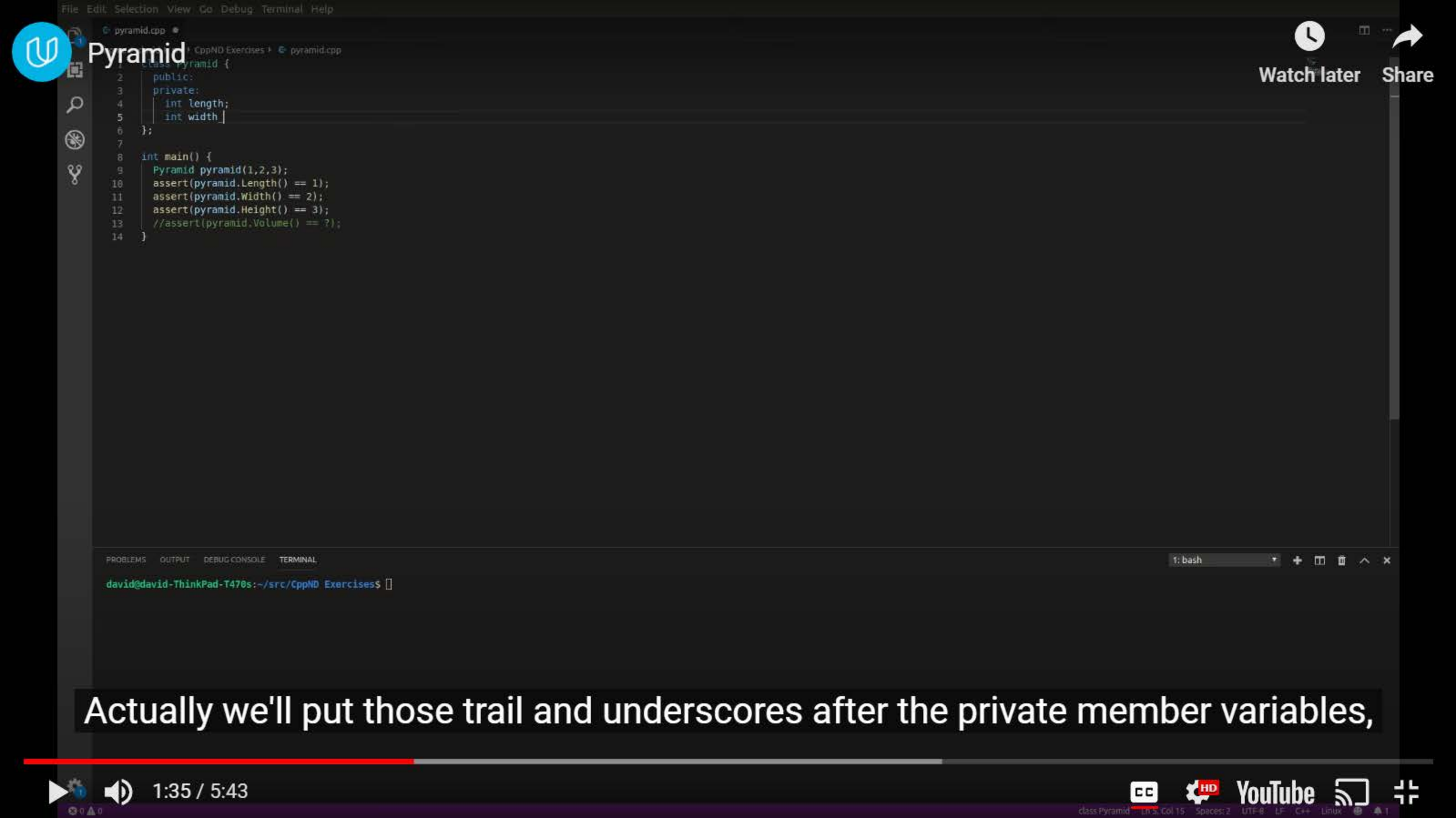
- It doesn't matter actually, you could just as well make them public.
- Using getter and setter functions is the only way to modify class member variables in C++.
- ✓ Often times you want to limit the user's access to class member variables, possibly because of an invariant.

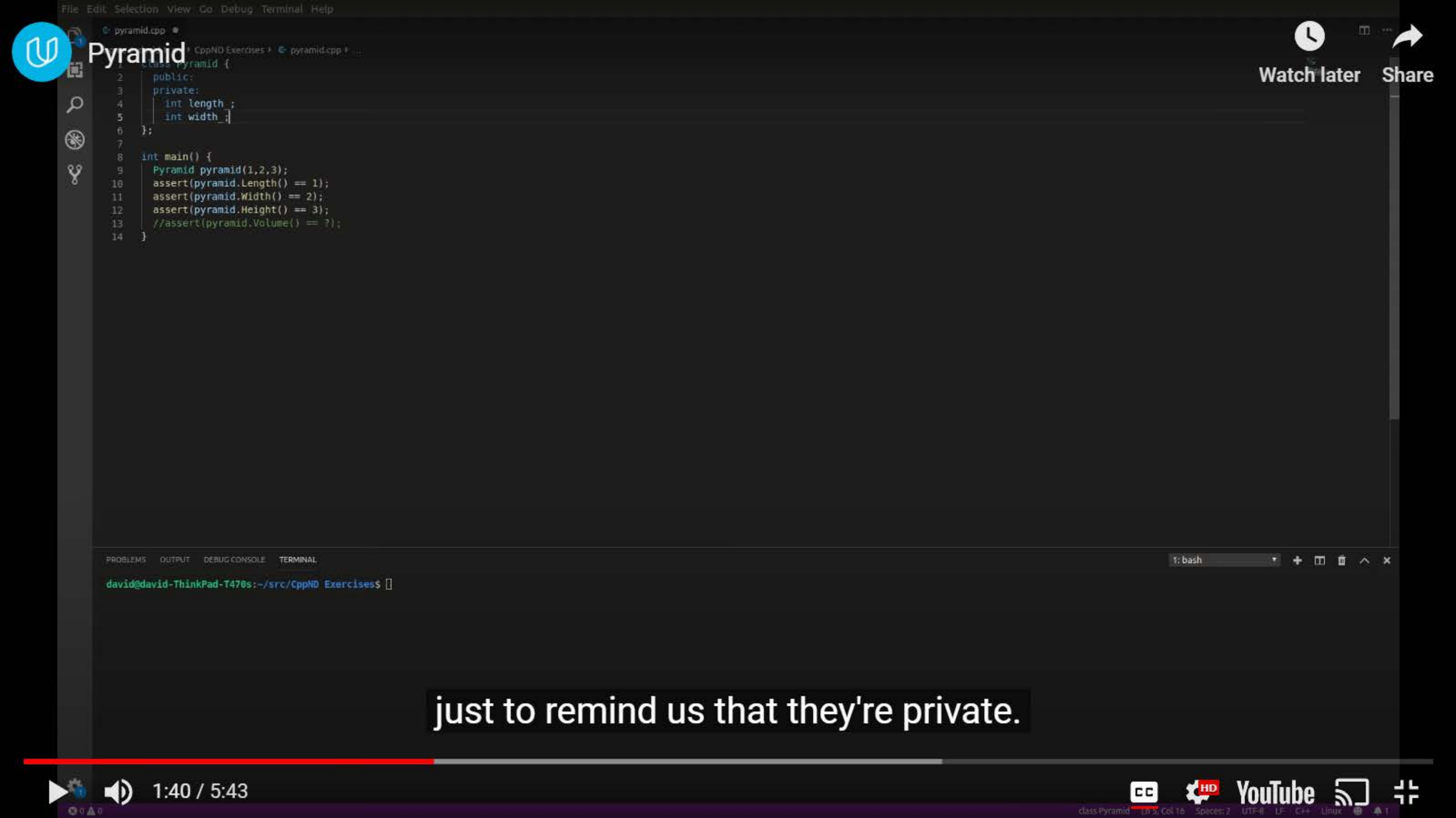
SUBMIT

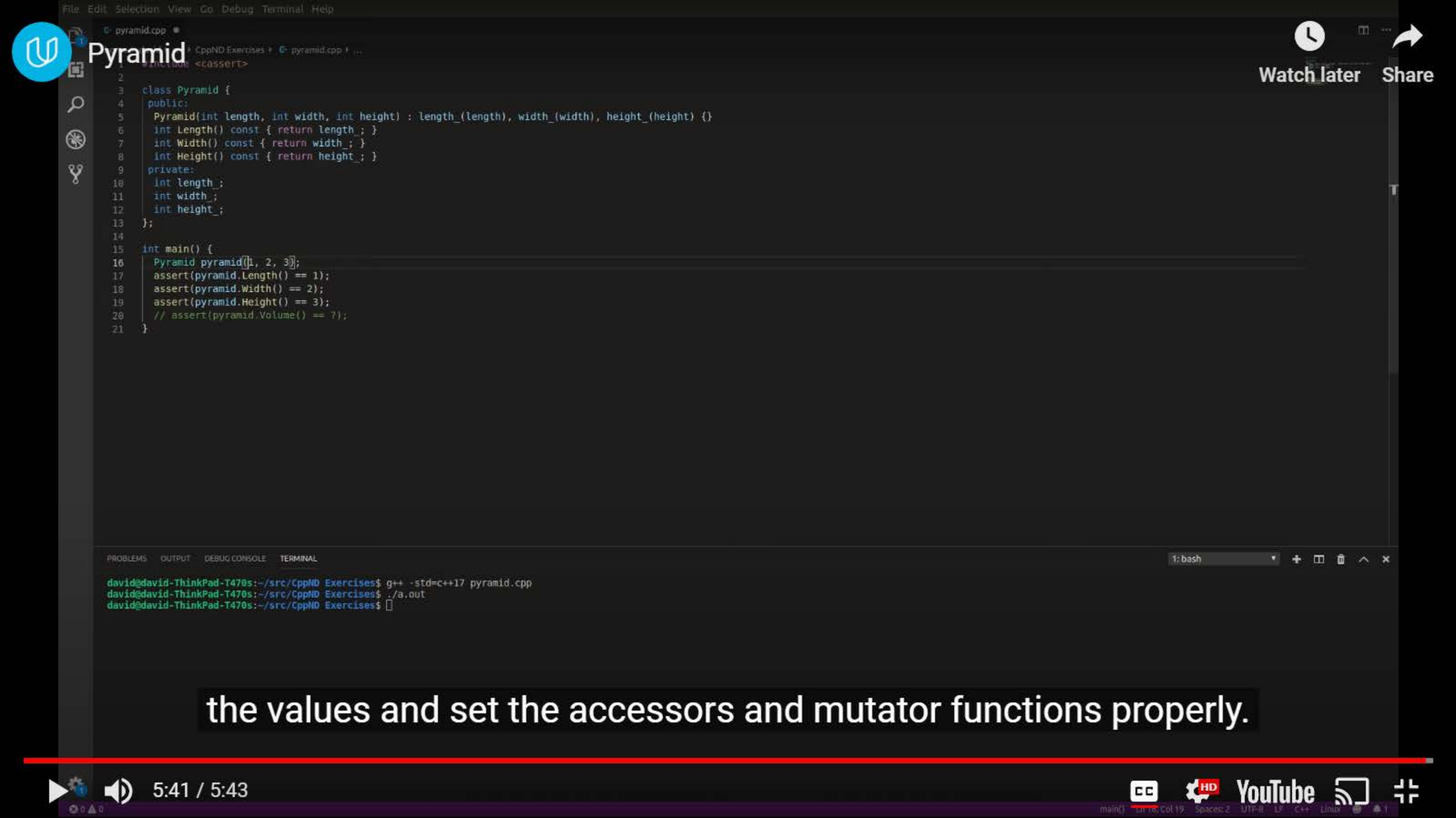
Thanks for completing that!

Indeed, your getter and setter functions can serve as a firewall between users and class member variables to limit how they can access or modify them.

CONTINUE


Pyramid
Watch later
Share
pyramid.cpp
CppND Exercises
pyramid.cpp
class Pyramid {
public:
private:
int length;
int width_
};
int main() {
Pyramid pyramid(1,2,3);
assert(pyramid.Length() == 1);
assert(pyramid.Width() == 2);
assert(pyramid.Height() == 3);
//assert(pyramid.Volume() == ?);
}
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
david@david-ThinkPad-T470s:~/src/CppND Exercises$
Actually we'll put those trail and underscores after the private member variables,
1:35 / 5:43


Pyramid
Watch later
Share
pyramid.cpp
CppND Exercises
pyramid.cpp
class Pyramid {
public:
private:
int length_;
int width_;
};
int main() {
Pyramid pyramid(1,2,3);
assert(pyramid.Length() == 1);
assert(pyramid.Width() == 2);
assert(pyramid.Height() == 3);
//assert(pyramid.Volume() == ?);
}
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
david@david-ThinkPad-T470s:~/src/CppND Exercises$
just to remind us that they're private.
1:40 / 5:43


Pyramid
Watch later
Share
pyramid.cpp
CppND Exercises
pyramid.cpp
#include <cassert>
class Pyramid {
public:
Pyramid(int length, int width, int height) : length_(length), width_(width), height_(height) {}
int Length() const { return length_; }
int Width() const { return width_; }
int Height() const { return height_; }
private:
int length_;
int width_;
int height_;
};
int main() {
Pyramid pyramid{1, 2, 3};
assert(pyramid.Length() == 1);
assert(pyramid.Width() == 2);
assert(pyramid.Height() == 3);
// assert(pyramid.Volume() == 7);
}
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
david@david-ThinkPad-T470s:~/src/CppND Exercises$ g++ -std=c++17 pyramid.cpp
david@david-ThinkPad-T470s:~/src/CppND Exercises$ ./a.out
david@david-ThinkPad-T470s:~/src/CppND Exercises$
the values and set the accessors and mutator functions properly.
5:41 / 5:43

```
3
4   // TODO: Define "Student" class
5   class Student {
6    public:
7     // constructor
8     // accessors
9     // mutators
10
11   private:
12    // name
13    // grade
14    // GPA
15  };
16
17  // TODO: Test
18  int main() {
19    Student student("fake name", 2, 3.2);
20    assert(student.Name() == "fake name");
21
22    bool myexception{false};
23    try {
24      student.Grade(-100);
25    }
26    catch(...) {
27      myexception = true;
28    }
29    assert(myexception);
30  }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: bash

```
david@david-ThinkPad-T470s:~/src/CppND Exercises$ ./a.out
david@david-ThinkPad-T470s:~/src/CppND Exercises$
```

then you should be able to pass this exercise.

QUESTION 1 OF 3

In the context of object oriented programming, encapsulation refers to:

- A requirement that data and logic be packaged separately in distinct objects
- ✓ The notion that data and logic can be packaged together and passed around within a program as a single object.
- The restriction that logic within a particular object can only operate on data stored within that same object.

SUBMIT

QUESTION 2 OF 3

Invoking the `const` keyword in an accessor function allows you to:

- Require that the data type of the output will be the same as that of the input.
- ✓ Ensure the user cannot do anything to change the private attributes of the object.
- Pass in constant attribute values to the accessor function.

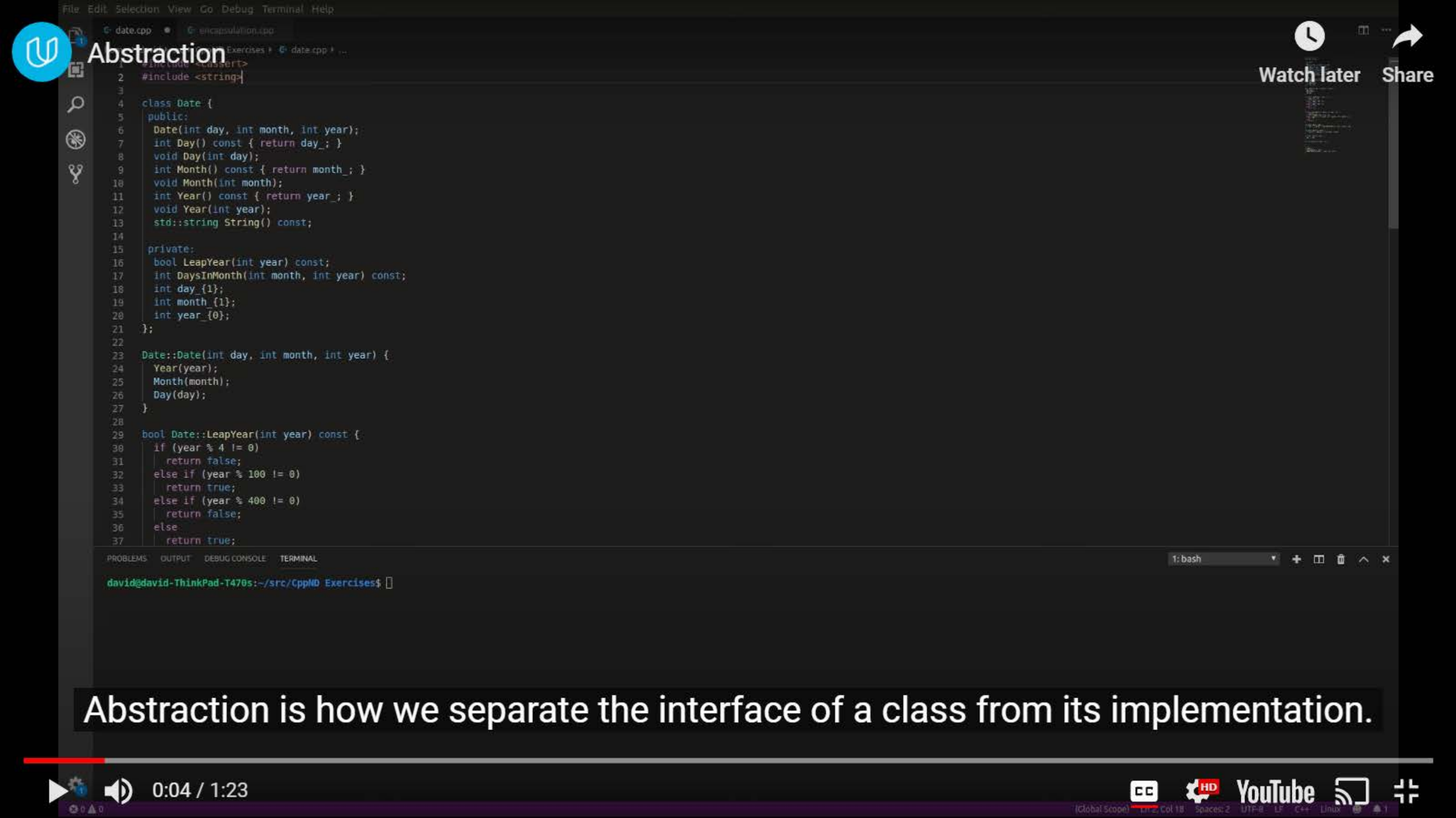
SUBMIT

QUESTION 3 OF 3

Making class attributes private and assigning them with a mutator function allows you to:

- Ensure that only class member functions have access to private class attributes.
- ✓ Invoke logic that checks whether the input data are valid before setting attributes.
- Prevent users from changing non-public class attributes.

SUBMIT


Abstraction
Watch later
Share
File Edit Selection View Go Debug Terminal Help
date.cpp
encapsulation.cpp
Exercises › date.cpp › ...
#include <cassert>
#include <string>
class Date {
public:
Date(int day, int month, int year);
int Day() const { return day_; }
void Day(int day);
int Month() const { return month_; }
void Month(int month);
int Year() const { return year_; }
void Year(int year);
std::string String() const;
private:
bool LeapYear(int year) const;
int DaysInMonth(int month, int year) const;
int day_{1};
int month_{1};
int year_{0};
};
Date::Date(int day, int month, int year) {
Year(year);
Month(month);
Day(day);
}
bool Date::LeapYear(int year) const {
if (year % 4 != 0)
return false;
else if (year % 100 != 0)
return true;
else if (year % 400 != 0)
return false;
else
return true;
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
david@david-ThinkPad-T470s:~/src/CppND Exercises$
Abstraction is how we separate the interface of a class from its implementation.
0:04 / 1:23
YouTube

Abstraction

Abstraction refers to the separation of a class's interface from the details of its implementation. The interface provides a way to interact with an object, while hiding the details and implementation of how the class works.

Example

The `String()` function within this `Date` class is an example of abstraction.

```
class Date {
 public:
  ...
  std::string String() const;
  ...
};
```

The user is able to interact with the `Date` class through the `String()` function, but the user does not need to know about the implementation of either `Date` or `String()`.

For example, the user does not know, or need to know, that this object internally contains three `int` member variables. The user can just call the `String()` method to get data.

If the designer of this class ever decides to change how the data is stored internally -- using a vector of `int`s instead of three separate `int`s, for example -- the user of the `Date` class will not need to know.

Exercise: Sphere Class

In this exercise you will practice abstraction by creating a class which represents a sphere.

Declare:

1. A constructor function that takes the radius as an argument
2. A member function that returns the volume

Directions

1. Define a class called `Sphere`.
2. Add one private member variable: `radius`.
3. Define a constructor to initialize the radius.
4. Define an accessor method that returns the radius.
5. Define a member function to return the volume of the sphere.
6. Write a `main()` function to initialize an object of type `Sphere`.

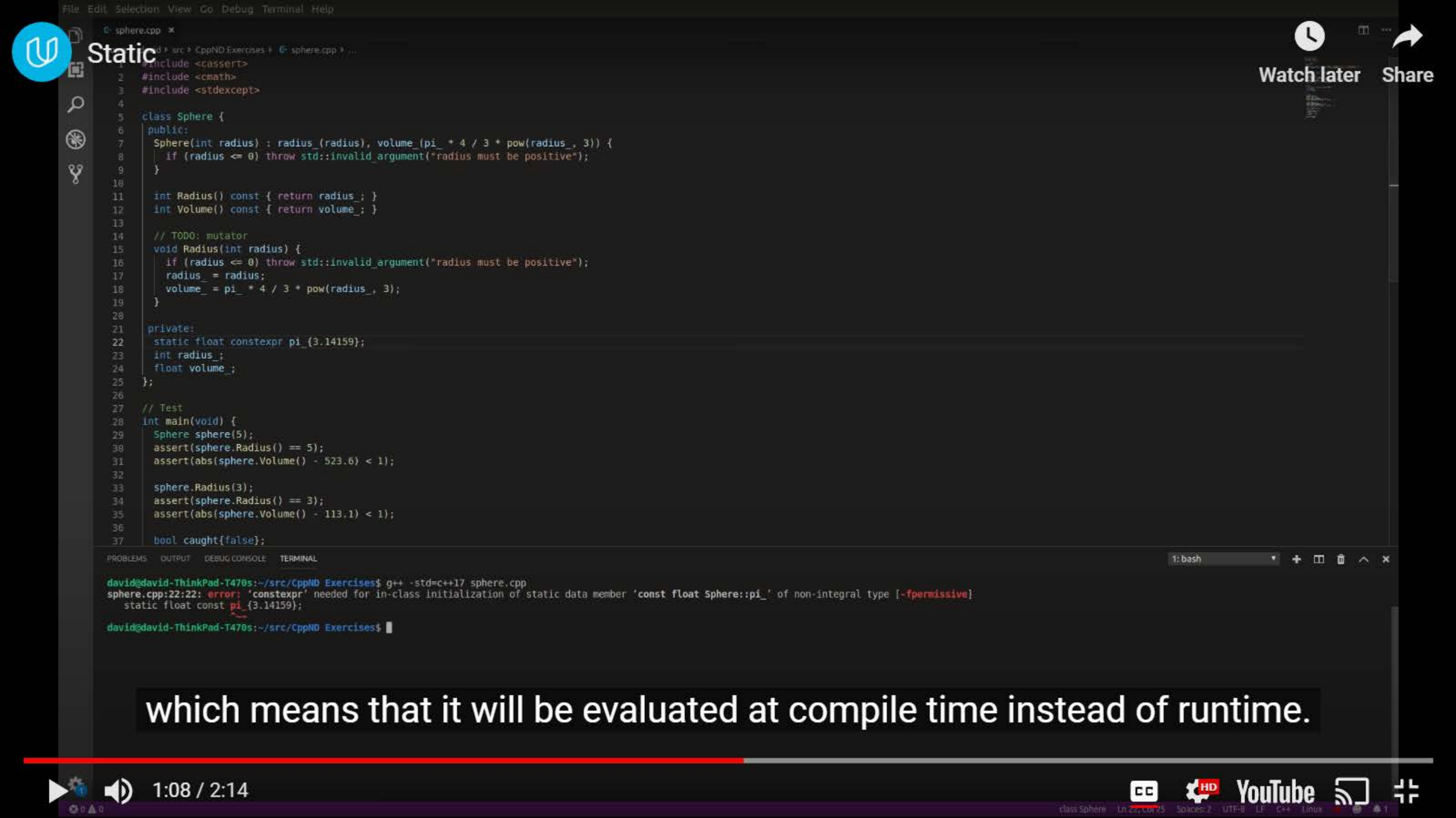
Exercise: Private Method

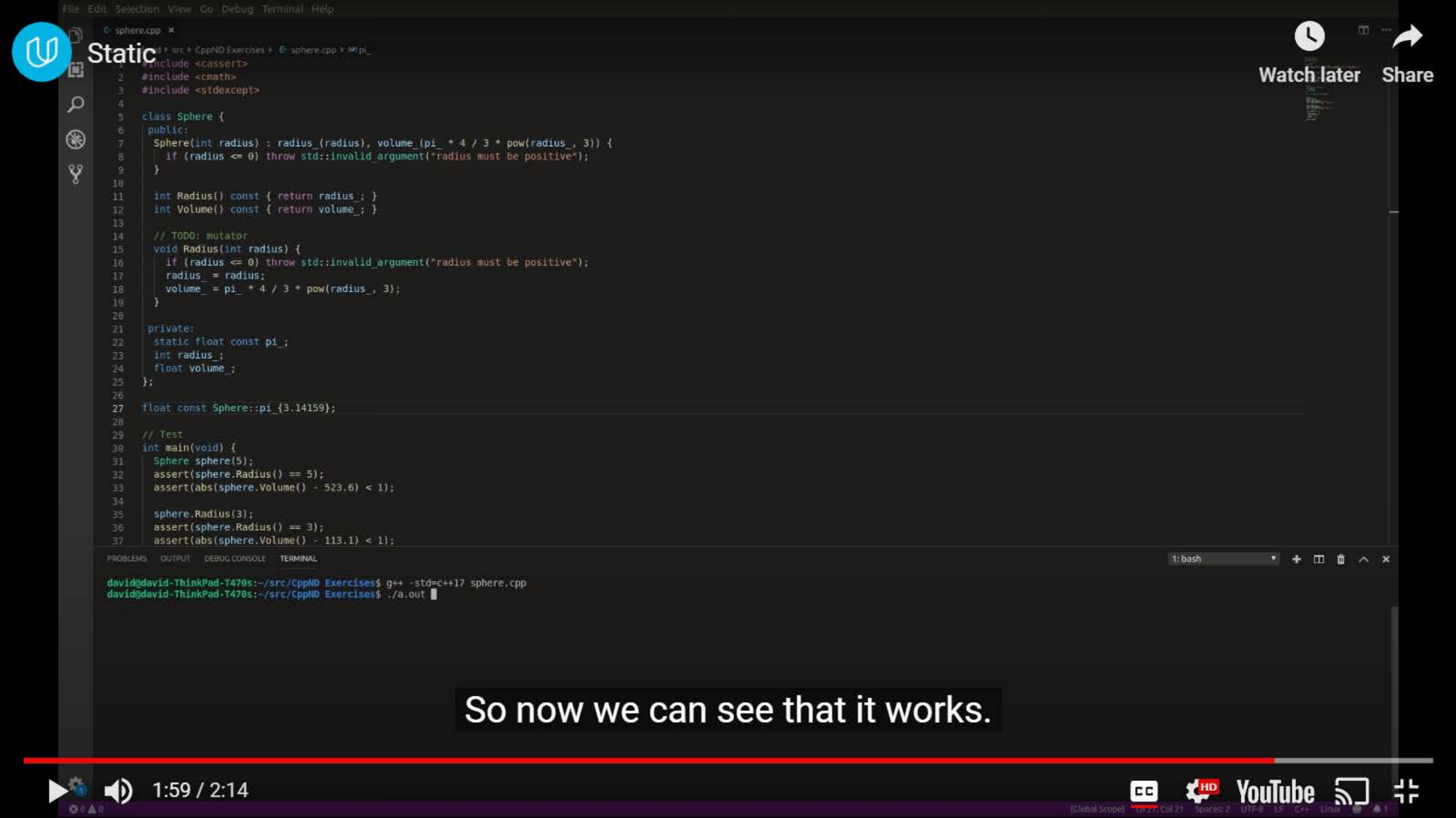
Abstraction is used to expose only relevant information to the user. By hiding implementation details, we give ourselves flexibility to modify how the program works. In this example, you'll practice abstracting implementation details from the user.

Directions

In this exercise, you'll update the `class Sphere` so that it becomes possible to change the radius of a sphere after it has been initialized. In order to do this, you'll move the two class invariants into private member functions.

1. Move the range-check on `radius_` into a private member function.
2. Move the `volume_` calculation, which depends on the value of `radius_` into the same private member function.
3. Verify that the class still functions correctly.
4. Add a mutator method to change the radius of an existing `Sphere`.
5. Verify that the mutator method successfully updates both the radius and the volume.


Static
Watch later
Share
sphere.cpp
#include <cassert>
#include <cmath>
#include <stdexcept>
class Sphere {
public:
Sphere(int radius) : radius_(radius), volume_(pi_ * 4 / 3 * pow(radius_, 3)) {
if (radius <= 0) throw std::invalid_argument("radius must be positive");
}
int Radius() const { return radius_; }
int Volume() const { return volume_; }
// TODO: mutator
void Radius(int radius) {
if (radius <= 0) throw std::invalid_argument("radius must be positive");
radius_ = radius;
volume_ = pi_ * 4 / 3 * pow(radius_, 3);
}
private:
static float constexpr pi_{3.14159};
int radius_;
float volume_;
};
// Test
int main(void) {
Sphere sphere(5);
assert(sphere.Radius() == 5);
assert(abs(sphere.Volume() - 523.6) < 1);
sphere.Radius(3);
assert(sphere.Radius() == 3);
assert(abs(sphere.Volume() - 113.1) < 1);
bool caught{false};
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
david@david-ThinkPad-T470s:~/src/CppND Exercises$ g++ -std=c++17 sphere.cpp
sphere.cpp:22:22: error: 'constexpr' needed for in-class initialization of static data member 'const float Sphere::pi_' of non-integral type [-fpermissive]
static float const pi_{3.14159};
david@david-ThinkPad-T470s:~/src/CppND Exercises$
which means that it will be evaluated at compile time instead of runtime.
1:08 / 2:14
YouTube


Static
Watch later
Share
sphere.cpp
#include <cassert>
#include <cmath>
#include <stdexcept>
class Sphere {
public:
Sphere(int radius) : radius_(radius), volume_(pi_ * 4 / 3 * pow(radius_, 3)) {
if (radius <= 0) throw std::invalid_argument("radius must be positive");
}
int Radius() const { return radius_; }
int Volume() const { return volume_; }
// TODO: mutator
void Radius(int radius) {
if (radius <= 0) throw std::invalid_argument("radius must be positive");
radius_ = radius;
volume_ = pi_ * 4 / 3 * pow(radius_, 3);
}
private:
static float const pi_;
int radius_;
float volume_;
};
float const Sphere::pi_{3.14159};
// Test
int main(void) {
Sphere sphere(5);
assert(sphere.Radius() == 5);
assert(abs(sphere.Volume() - 523.6) < 1);
sphere.Radius(3);
assert(sphere.Radius() == 3);
assert(abs(sphere.Volume() - 113.1) < 1);
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
david@david-ThinkPad-T470s:~/src/CppND Exercises$ g++ -std=c++17 sphere.cpp
david@david-ThinkPad-T470s:~/src/CppND Exercises$ ./a.out
So now we can see that it works.
1:59 / 2:14
HD
YouTube

Static Members

Class members can be declared `static`, which means that the member belongs to the entire class, instead of to a specific instance of the class. More specifically, a `static` member is created only once and then shared by all instances (i.e. objects) of the class. That means that if the `static` member gets changed, either by a user of the class or within a member function of the class itself, then all members of the class will see that change the next time they access the `static` member.

QUIZ QUESTION

Imagine you have a `class Sphere` with a `static int counter` member. `Sphere` increments `counter` in the constructor and uses this to track how many `Sphere`s have been created. What would happen if you instantiated a new classes (`Cube`, for instance) that also had a `static int counter`? Would the two `counter`s conflict?

- Yes, instantiating a class of a different name that has a static attribute `counter` will increment the same `counter` as before.
- ✓ No, because the new static attribute `counter` is defined within the `Cube` class, it has nothing to do with `Sphere::counter`.

Only if both classes are instantiated within the same scope do the two `counter` attributes conflict.

SUBMIT

Implementation

`static` members are **declared** within their `class` (often in a header file) but in most cases they must be **defined** within the global scope. That's because memory is allocated for `static` variables immediately when the program begins, at the same time any global variables are initialized.

Here is an example:

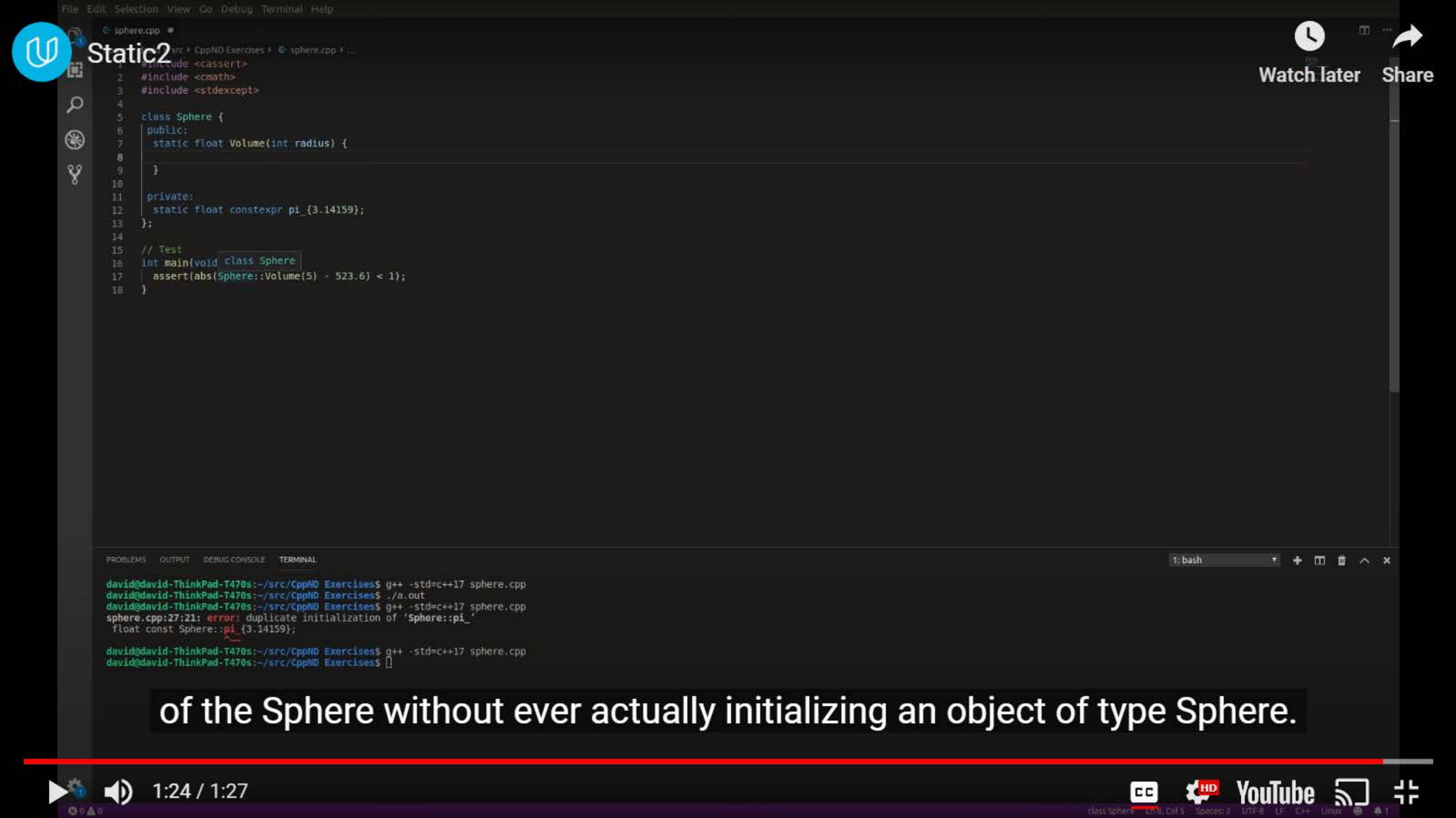
```
#include <cassert>

class Foo {
 public:
  static int count;
  Foo() { Foo::count += 1; }
};

int Foo::count{0};

int main() {
  Foo f{};
  assert(Foo::count == 1);
}
```

An exception to the global definition of `static` members is if such members can be marked as **`constexpr`**. In that case, the `static` member variable can be both declared and defined within the `class` definition:

```
struct Kilometer {
  static constexpr int meters{1000};
};
```

Exercise: Pi

`class Sphere` has a member `const double pi`. Experiment with specifying `pi` to be `const`, `constexpr`, and `static`. Which specifications work and which break? Do you understand why?


Static2
Watch later
Share
sphere.cpp
#include <cassert>
#include <cmath>
#include <stdexcept>
class Sphere {
public:
static float Volume(int radius) {
}
private:
static float constexpr pi_{3.14159};
};
// Test
int main(void
class Sphere
assert(abs(Sphere::Volume(5) - 523.6) < 1);
}
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
david@david-ThinkPad-T470s:~/src/CppND Exercises$ g++ -std=c++17 sphere.cpp
david@david-ThinkPad-T470s:~/src/CppND Exercises$ ./a.out
david@david-ThinkPad-T470s:~/src/CppND Exercises$ g++ -std=c++17 sphere.cpp
sphere.cpp:27:21: error: duplicate initialization of 'Sphere::pi_'
float const Sphere::pi_{3.14159};
david@david-ThinkPad-T470s:~/src/CppND Exercises$ g++ -std=c++17 sphere.cpp
david@david-ThinkPad-T470s:~/src/CppND Exercises$
of the Sphere without ever actually initializing an object of type Sphere.
1:24 / 1:27
YouTube

Exercise: Static Method

In addition to `static` member variables, C++ supports `static` member functions (or "methods"). Just like `static` member variables, `static` member functions are instance-independent: they belong to the class, not to any particular instance of the class.

One corollary to this is that we can method invoke a `static` member function *without ever creating an instance of the class*.

You will try just that in this exercise.

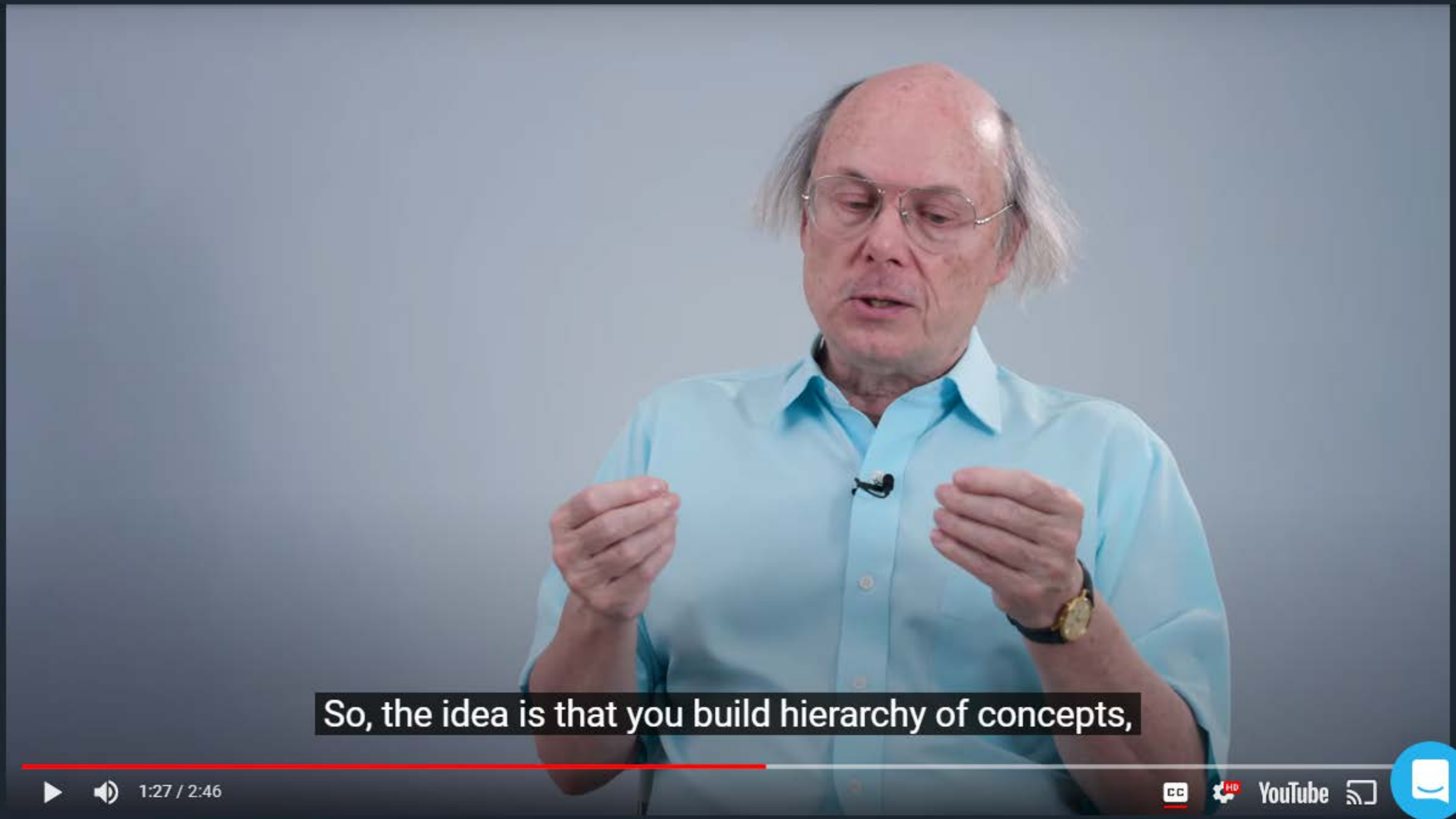
Instructions

1. Refactor `class Sphere` to move the volume calculation into a `static` function.
2. Verify that the class still functions as intended.
3. Make that `static` function public.
4. Call that static function directly from `main()` to calculate the hypothetical volume of a sphere you have not yet instantiated.

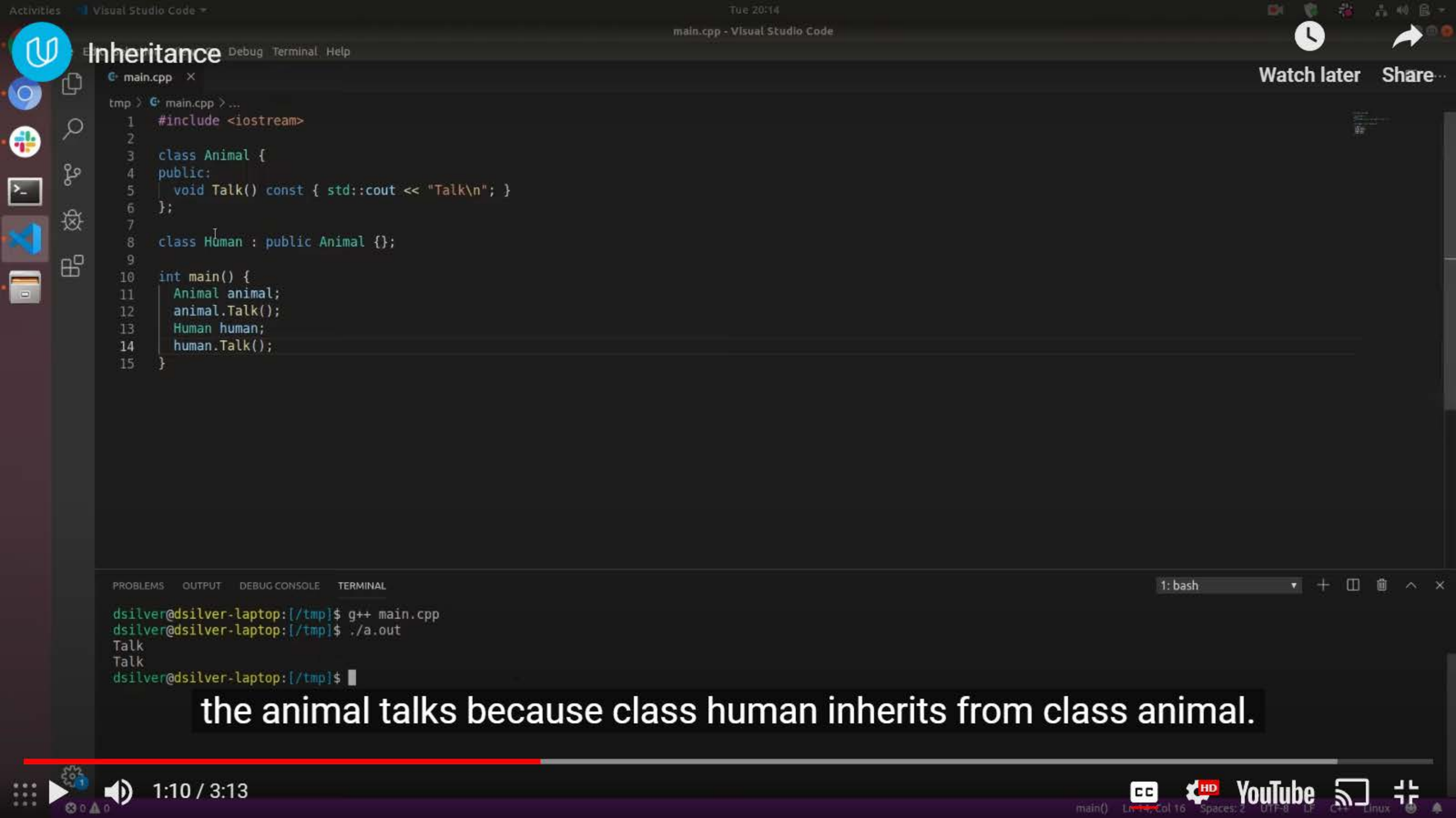

Inheritance is one way that classes can relate to each other.
0:08 / 2:02
CC
HD
YouTube

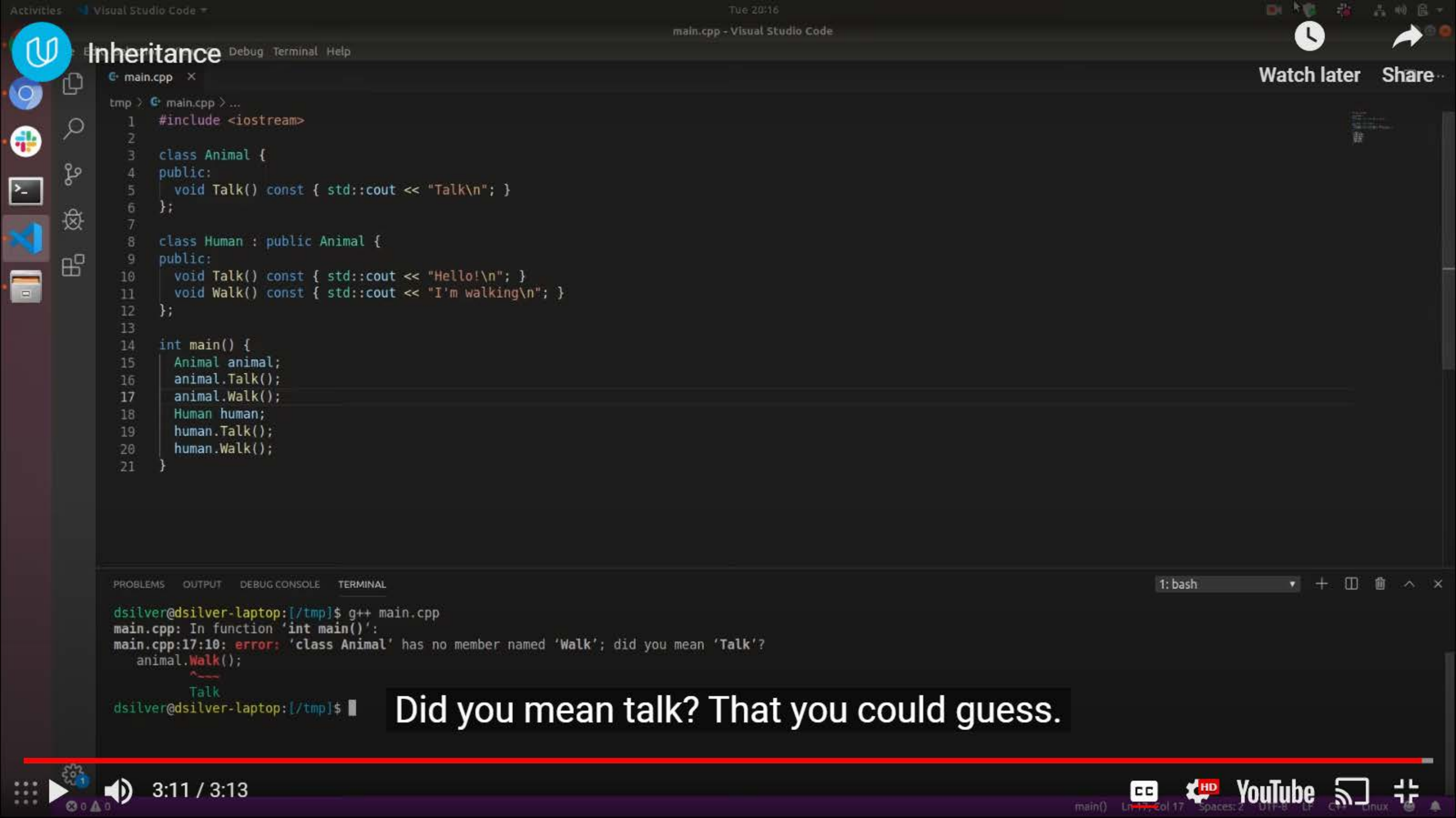

Polymorphism is a related concept that
1:09 / 2:02
CC
HD
YouTube


allows an interface to work with several different types.
1:12 / 2:02
CC
HD
YouTube


So, the idea is that you build hierarchy of concepts,
1:27 / 2:46
YouTube


starting from the most general and then more and more specialized.
1:31 / 2:46
YouTube

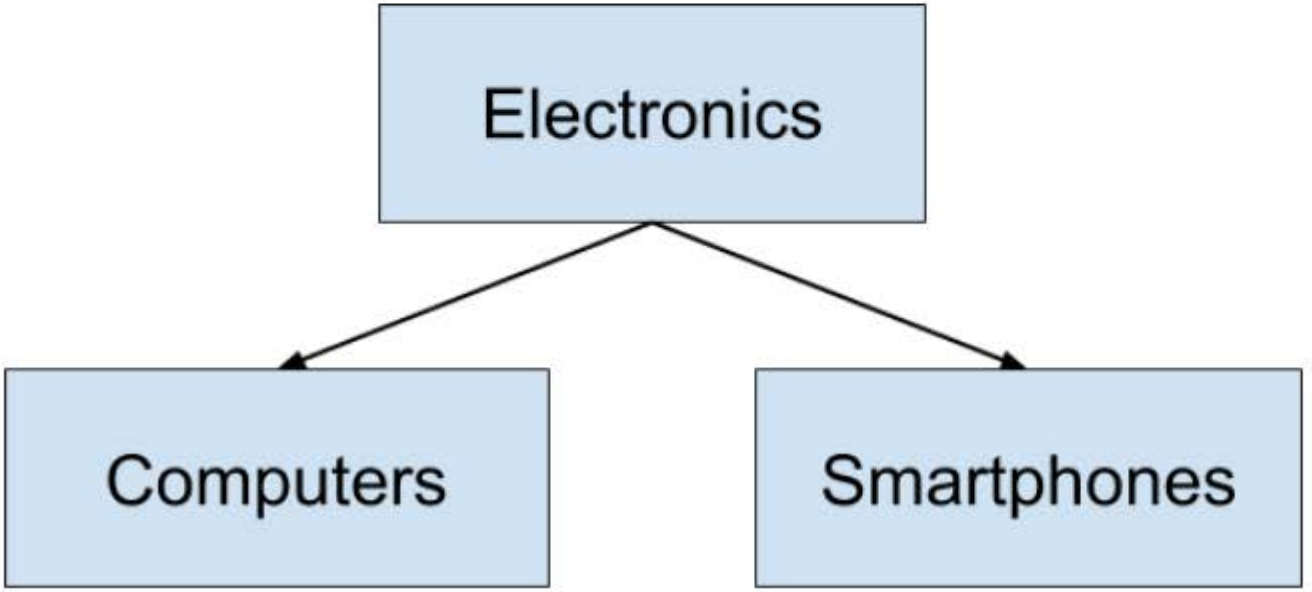

Activities
Visual Studio Code
Tue 20:14
main.cpp - Visual Studio Code
Inheritance
Watch later
Share
File Edit Selection View Go Debug Terminal Help
main.cpp
tmp > main.cpp > ...
#include <iostream>
class Animal {
public:
void Talk() const { std::cout << "Talk\n"; }
};
class Human : public Animal {};
int main() {
Animal animal;
animal.Talk();
Human human;
human.Talk();
}
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
dsilver@dsilver-laptop:[/tmp]$ g++ main.cpp
dsilver@dsilver-laptop:[/tmp]$ ./a.out
Talk
Talk
dsilver@dsilver-laptop:[/tmp]$
the animal talks because class human inherits from class animal.
1:10 / 3:13
YouTube


Activities
Visual Studio Code
Tue 20:16
main.cpp - Visual Studio Code
Inheritance
Debug Terminal Help
Watch later
Share
main.cpp
tmp > main.cpp > ...
#include <iostream>
class Animal {
public:
void Talk() const { std::cout << "Talk\n"; }
};
class Human : public Animal {
public:
void Talk() const { std::cout << "Hello!\n"; }
void Walk() const { std::cout << "I'm walking\n"; }
};
int main() {
Animal animal;
animal.Talk();
animal.Walk();
Human human;
human.Talk();
human.Walk();
}
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
dsilver@dsilver-laptop:[/tmp]$ g++ main.cpp
main.cpp: In function 'int main()':
main.cpp:17:10: error: 'class Animal' has no member named 'Walk'; did you mean 'Talk'?
animal.Walk();
^~~~
Talk
dsilver@dsilver-laptop:[/tmp]$
Did you mean talk? That you could guess.
3:11 / 3:13
YouTube

Inheritence

In our everyday life, we tend to divide things into groups, based on their shared characteristics. Here are some groups that you have probably used yourself: electronics, tools, vehicles, or plants.

Sometimes these groups have hierarchies. For example, computers and smartphones are both types of electronics, but computers and smartphones are also groups in and of themselves. You can imagine a tree with "electronics" at the top, and "computers" and "smartphones" each as children of the "electronics" node.



Object-oriented programming uses the same principles! For instance, imagine a `Vehicle` class:

```
class Vehicle {
public:
  int wheels = 0;
  string color = "blue";

  void Print() const
  {
    std::cout << "This " << color << " vehicle has " << wheels << " wheels!\n";
  }
};
```

We can derive other classes from `Vehicle`, such as `Car` or `Bicycle`. One advantage is that this saves us from having to re-define all of the common member variables - in this case, `wheels` and `color` - in each derived class.

Another benefit is that derived classes, for example `Car` and `Bicycle`, can have distinct member variables, such as `sunroof` or `kickstand`. Different derived classes will have different member variables:

```
class Car : public Vehicle {
public:
  bool sunroof = false;
};

class Bicycle : public Vehicle {
public:
  bool kickstand = true;
};
```

Instructions

1. Add a new member variable to `class Vehicle`.
2. Output that new member in `main()`.
3. Derive a new class from `Vehicle`, alongside `Car` and `Bicycle`.
4. Instantiate an object of that new class.
5. Print the object.

Accessmodifiers

Watch later Share

```cpp
#include <iostream>
#include <string>

class Animal {
public:
  void Talk() const { std::cout << "Talk\n"; }
};

class Human : public Animal {
public:
  void Talk(std::string content) const { std::cout << content << "\n"; }
};

class Baby : private Human {
public:
  void Cry() { Talk("Whaa!"); }
};

int main() {
  Human human;
  human.Talk("Hello, World!");
  Baby baby;
  baby.Cry();
  baby.Talk("The square root of 9 is 3.");
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: bash

```
dsilver@dsilver-laptop:[/tmp]$ g++ main.cpp
main.cpp: In function 'int main()':
main.cpp:24:41: error: 'void Human::Talk(std::__cxx11::string) const' is inaccessible within this context
   baby.Talk("The square root of 9 is 3.");
                                         ^

main.cpp:11:8: note: declared here
   void Talk(std::string co
        ^~~~
main.cpp:24:41: error: 'Human' is not an accessible base of 'Baby'
   baby.Talk("The square root of 9 is 3.");
                                         ^
dsilver@dsilver-laptop:[/tmp]$
```

So the user class Baby can't reach up and call talk itself.

2:39 / 3:12

Accessmodifiers

Watch later Share

```cpp
#include <iostream>
#include <string>

class Animal {
public:
  void Talk() const { std::cout << "Talk\n"; }
};

class Human : public Animal {
public:
  void Talk(std::string content) const { std::cout << content << "\n"; }
};

class Baby : private Human {
public:
  void Cry() { Talk("Whaa!"); }
};

int main() {
  Human human;
  human.Talk("Hello, World!");
  Baby baby;
  baby.Cry();
  baby.Talk("The square root of 9 is 3.");
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: bash

```
dsilver@dsilver-laptop:[/tmp]$ g++ main.cpp
main.cpp: In function 'int main()':
main.cpp:24:41: error: 'void Human::Talk(std::__cxx11::string) const' is inaccessible within this context
   baby.Talk("The square root of 9 is 3.");
                                         ^
main.cpp:11:8: note: declared here
   void Talk(std::string content) const { st
        ^~~~
main.cpp:24:41: error: 'Human' is not an accessible base of 'Baby'
   baby.Talk("The square root of 9 is 3.");
dsilver@dsilver-laptop:[/tmp]$
```

However, Baby itself is able to call Talk,

2:43 / 3:12

Inherited Access Specifiers

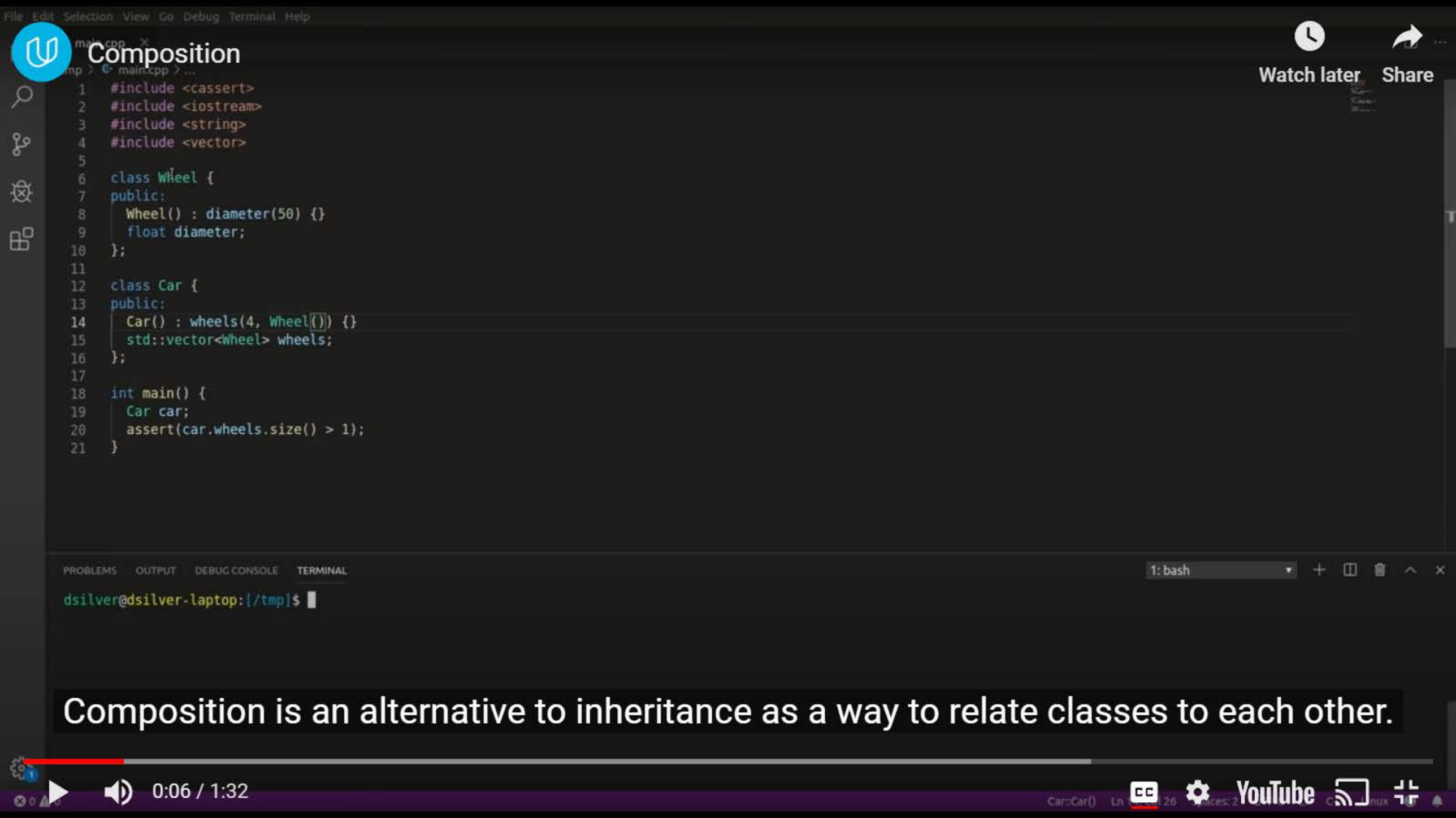
Just as access specifiers (i.e. `public`, `protected`, and `private`) define which class members *users* can access, the same access modifiers also define which class members *users of a derived classes* can access.

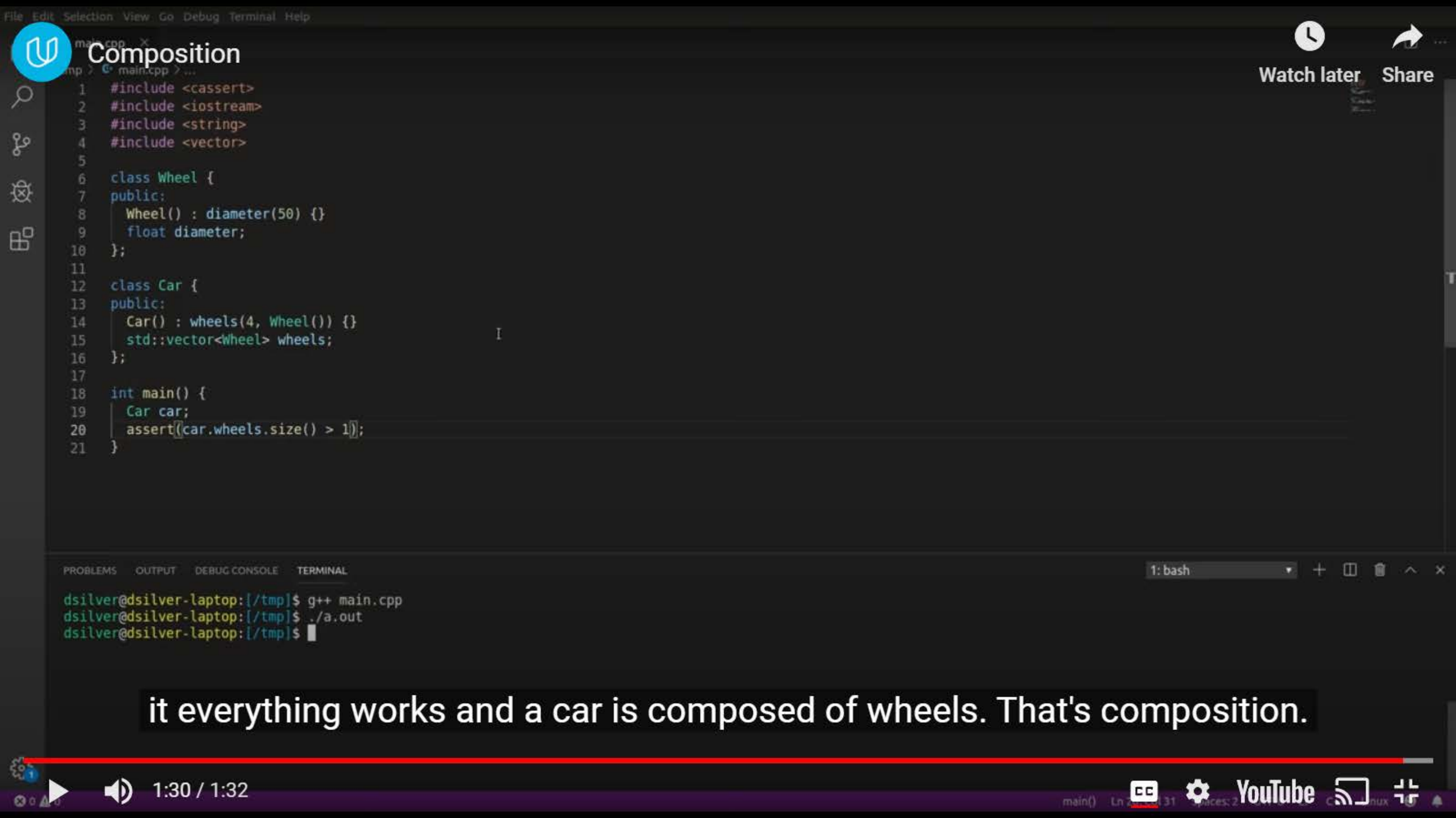
> **Public inheritance:** the public and protected members of the base class listed after the specifier keep their member access in the derived class
>
> **Protected inheritance:** the public and protected members of the base class listed after the specifier are protected members of the derived class
>
> **Private inheritance:** the public and protected members of the base class listed after the specifier are private members of the derived class
>
> Source: C++ reference

In the exercise below, you'll experiment with access modifiers.

Instructions

1. Update the derived classes so that one has `protected` inheritance and one has `private` inheritance.
2. Try to access a `protected` member from `main()`. Is it possible?
3. Try to access a `private` member from `main()`. Is it possible?
4. Try to access a member of the base class from within the derived class that has `protected` inheritance. Is it possible?
5. Try to access a member of the base class from within the derived class that has `private` inheritance. Is it possible?


Composition
Watch later
Share
#include <cassert>
#include <iostream>
#include <string>
#include <vector>
class Wheel {
public:
Wheel() : diameter(50) {}
float diameter;
};
class Car {
public:
Car() : wheels(4, Wheel()) {}
std::vector<Wheel> wheels;
};
int main() {
Car car;
assert(car.wheels.size() > 1);
}
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
dsilver@dsilver-laptop:[/tmp]$
Composition is an alternative to inheritance as a way to relate classes to each other.
0:06 / 1:32
YouTube


Composition
Watch later
Share
#include <cassert>
#include <iostream>
#include <string>
#include <vector>
class Wheel {
public:
Wheel() : diameter(50) {}
float diameter;
};
class Car {
public:
Car() : wheels(4, Wheel()) {}
std::vector<Wheel> wheels;
};
int main() {
Car car;
assert(car.wheels.size() > 1);
}
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
dsilver@dsilver-laptop:[/tmp]$ g++ main.cpp
dsilver@dsilver-laptop:[/tmp]$ ./a.out
dsilver@dsilver-laptop:[/tmp]$
it everything works and a car is composed of wheels. That's composition.
1:30 / 1:32
YouTube

Composition

Composition is a closely related alternative to inheritance. Composition involves constructing ("composing") classes from other classes, instead of inheriting traits from a parent class.

A common way to distinguish "composition" from "inheritance" is to think about what an object can do, rather than what it is. This is often expressed as **"has a"** versus **"is a"**.

From the standpoint of composition, a cat "has a" head and "has a" set of paws and "has a" tail.

From the standpoint of inheritance, a cat "is a" mammal.

There is no hard and fast rule about when to prefer composition over inheritance. In general, if a class needs only extend a small amount of functionality beyond what is already offered by another class, it makes sense to **inherit** from that other class. However, if a class needs to contain functionality from a variety of otherwise unrelated classes, it makes sense to **compose** the class from those other classes.

Exercise: Class Hierarchy

Multi-level inheritance is term used for chained classes in an inheritance tree. Have a look at the example in the notebook below to get a feel for multi-level inheritance.

```cpp
#include <cassert>

class Heart {
private:
  int rate{80};
  friend class Human;
};

class Human {
public:
  Heart heart;
  void Exercise() { heart.rate = 150; }
  int HeartRate() { return heart.rate; }
};

int main() {
  Human human;
  assert(human.HeartRate() == 80);
  human.Exercise();
  assert(human.HeartRate() == 150);
}
```

```
dsilver@dsilver-laptop:[/tmp]$ g++ main.cpp
dsilver@dsilver-laptop:[/tmp]$ ./a.out
dsilver@dsilver-laptop:[/tmp]$
```

we see that it works

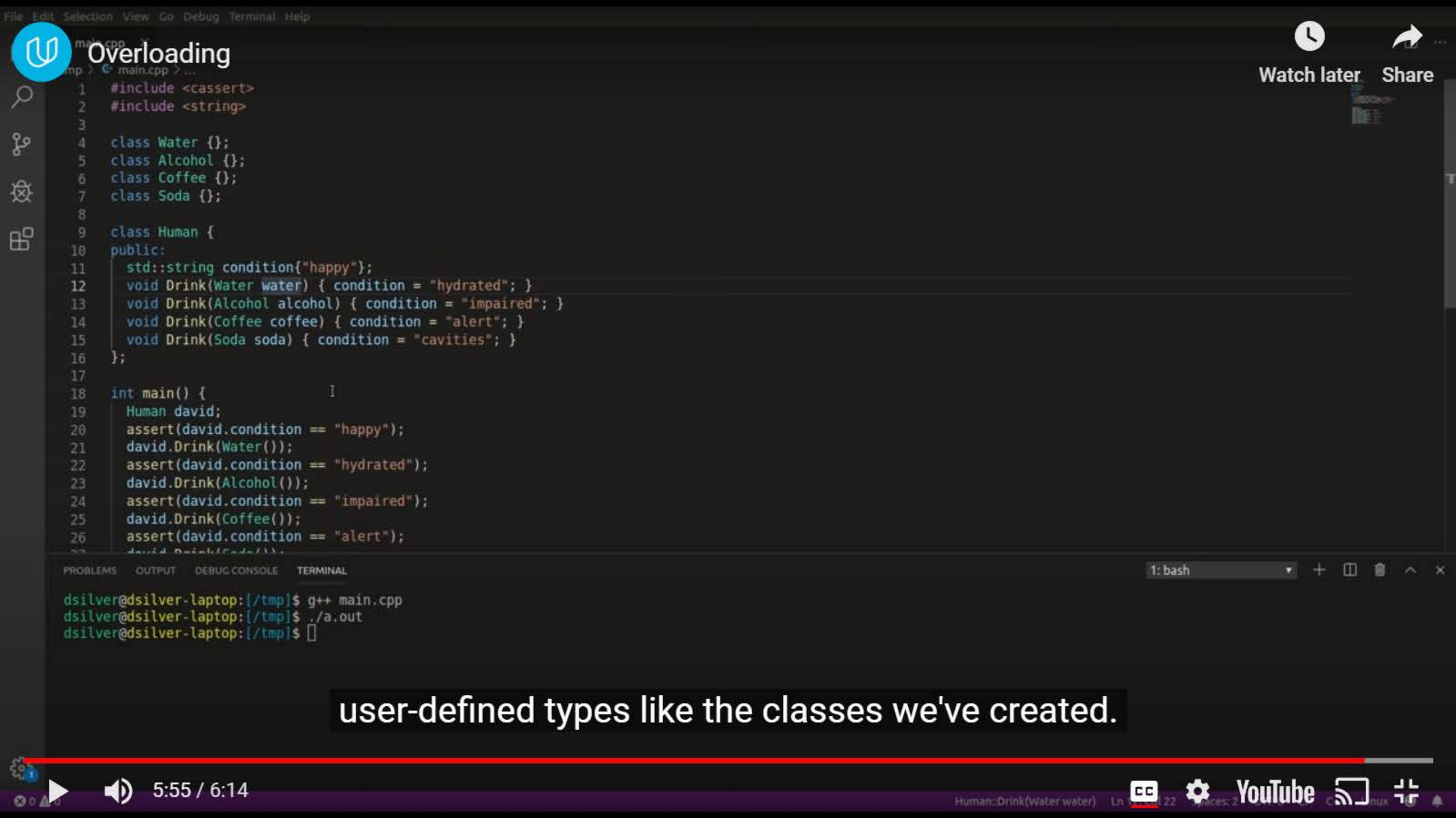
Friends

In C++, `friend` classes provide an alternative inheritance mechanism to derived classes. The main difference between classical inheritance and friend inheritance is that a `friend` class can access private members of the base class, which isn't the case for classical inheritance. In classical inheritance, a derived class can only access public and protected members of the base class.

Instructions

In this exercise you will experiment with friend classes. In the notebook below, implement the following steps:

1. Declare a class `Rectangle`.
2. Define a class `Square`.
3. Add class `Rectangle` as a friend of the class `Square`.
4. Add a private attribute `side` to class `Square`.
5. Create a public constructor in class `Square` that initializes the `side` attribute.
6. Add private members `width` and `height` to class `Rectangle`.
7. Add a `Rectangle()` constructor that takes a `Square` as an argument.
8. Add an `Area()` function to `class Rectangle`.


Overloading
Watch later
Share
#include <cassert>
#include <string>
class Water {};
class Alcohol {};
class Coffee {};
class Soda {};
class Human {
public:
std::string condition{"happy"};
void Drink(Water water) { condition = "hydrated"; }
void Drink(Alcohol alcohol) { condition = "impaired"; }
void Drink(Coffee coffee) { condition = "alert"; }
void Drink(Soda soda) { condition = "cavities"; }
};
int main() {
Human david;
assert(david.condition == "happy");
david.Drink(Water());
assert(david.condition == "hydrated");
david.Drink(Alcohol());
assert(david.condition == "impaired");
david.Drink(Coffee());
assert(david.condition == "alert");
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
dsilver@dsilver-laptop:[/tmp]$ g++ main.cpp
dsilver@dsilver-laptop:[/tmp]$ ./a.out
dsilver@dsilver-laptop:[/tmp]$
user-defined types like the classes we've created.
5:55 / 6:14
YouTube

```
 9  class Human {
10  public:
11    std::string condition{"happy"};
12    void Drink(Water water) { condition = "hydrated"; }
13    void Drink(Alcohol alcohol) { condition = "impaired"; }
14    void Drink(Coffee coffee) { condition = "alert"; }
15    void Drink(Soda soda) { condition = "cavities"; }
16  };
17
18  int main() {
19    Human david;
20    assert(david.condition == "happy");
21    david.Drink(Water());
22    assert(david.condition == "hydrated");
23    david.Drink(Alcohol());
24    assert(david.condition == "impaired");
25    david.Drink(Coffee());
26    assert(david.condition == "alert");
27    david.Drink(Soda());
28    assert(david.condition == "cavities");
29  }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: bash

```
dsilver@dsilver-laptop:[/tmp]$ g++ main.cpp
dsilver@dsilver-laptop:[/tmp]$ ./a.out
dsilver@dsilver-laptop:[/tmp]$ g++ main.cpp
dsilver@dsilver-laptop:[/tmp]$ ./a.out
dsilver@dsilver-laptop:[/tmp]$
```

what arguments you pass in.

Polymorphism

Polymorphism is means "assuming many forms".

In the context of object-oriented programming, polymorphism) describes a paradigm in which a function may behave differently depending on how it is called. In particular, the function will perform differently based on its inputs.

Polymorphism can be achieved in two ways in C++: overloading and overriding. In this exercise we will focus on overloading.

Overloading

In C++, you can write two (or more) versions of a function with the same name. This is called "overloading". Overloading requires that we leave the function name the same, but we modify the function signature. For example, we might define the same function name with multiple different configurations of input arguments.

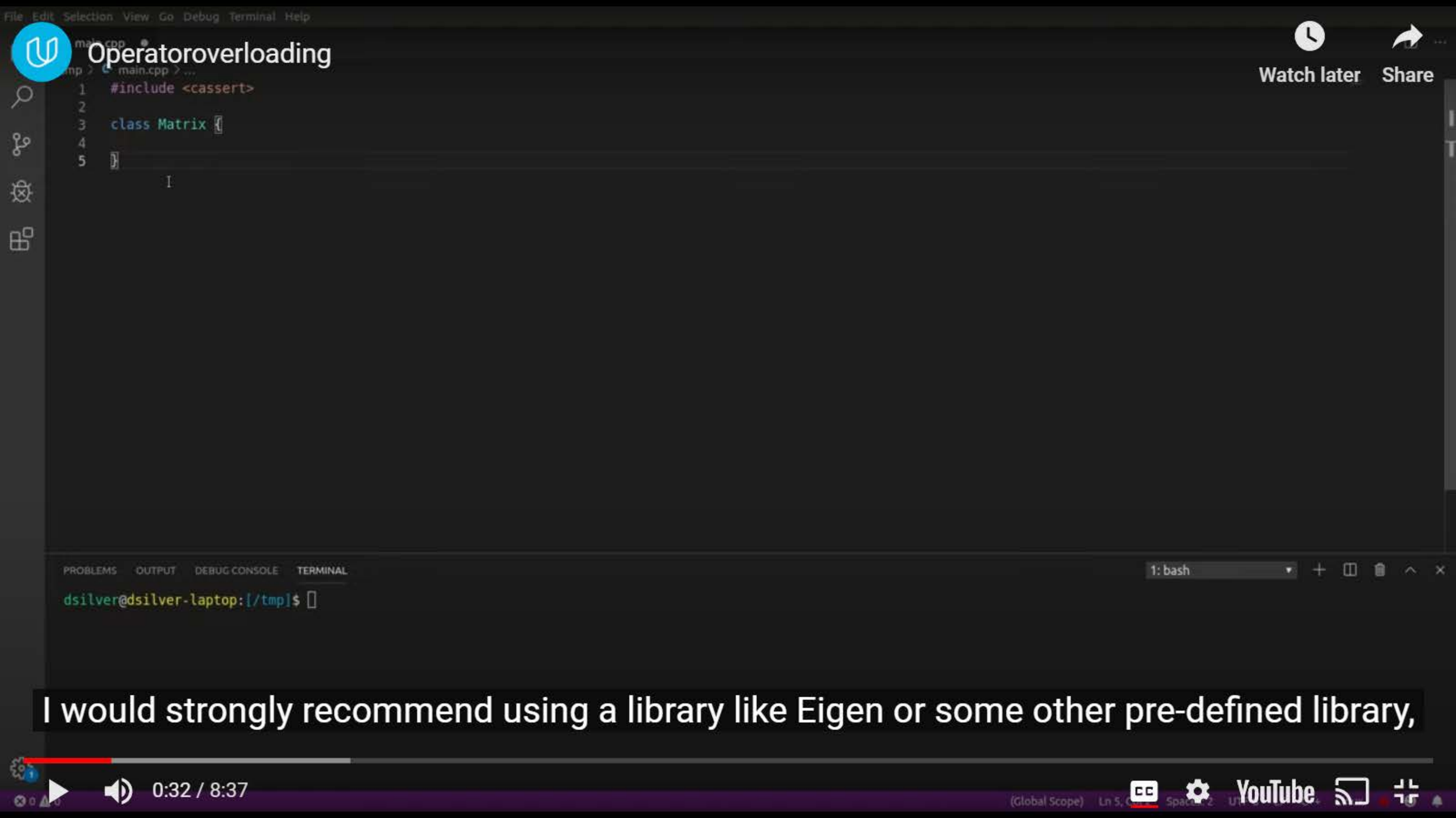
This example of `class Date` overloads:

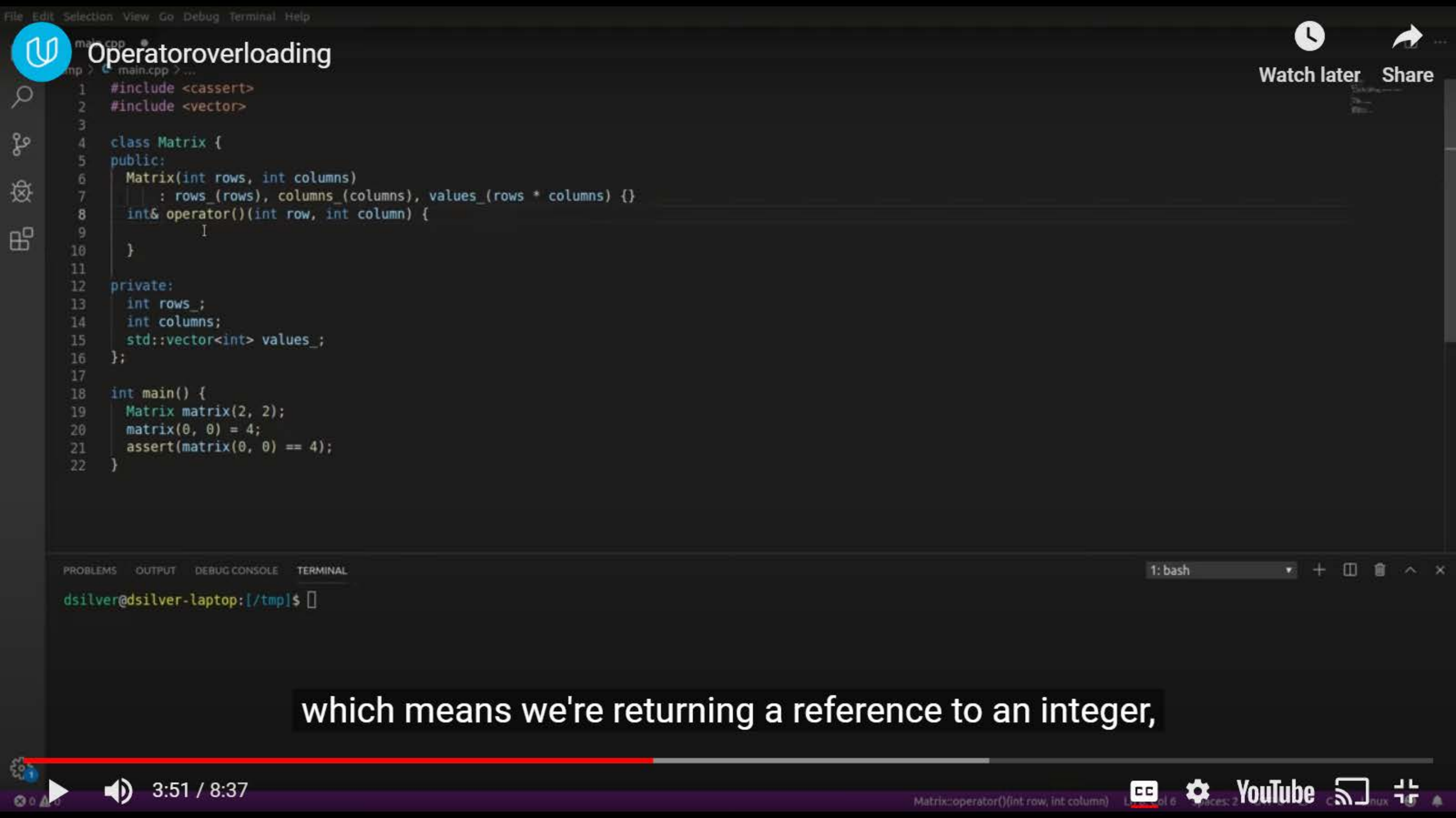
```
#include <ctime>

class Date {
public:
    Date(int day, int month, int year) : day_(day), month_(month), year_(year) {}
    Date(int day, int month) : day_(day), month_(month)  // automatically sets the Date to the current ye
    {
        time_t t = time(NULL);
        tm* timePtr = localtime(&t);
        year_ = timePtr->tm_year;
    }

private:
    int day_;
    int month_;
    int year_;
};
```

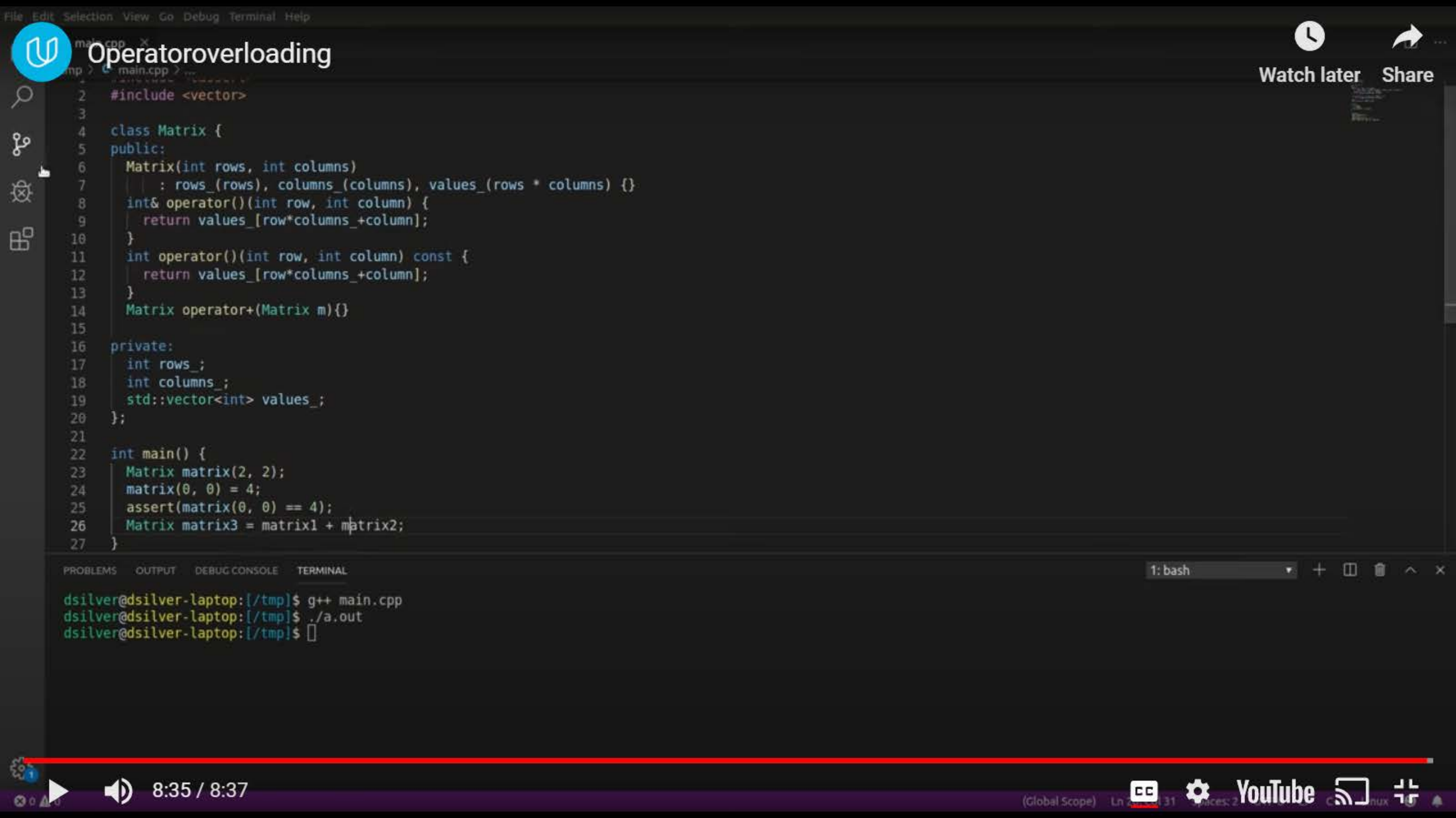
Instructions

Overloading can happen outside of an object-oriented context, too. In this exercise, you will practice overloading a normal function that is not a class member.

1. Create a function `hello()` that outputs, "Hello, World!"
2. Create a `class Human`.
3. Overload `hello()` by creating a function `hello(Human human)`. This function should output, "Hello, Human!"
4. Create 2 more classes and use those classes to further overload the `hello()` function.


Operatoroverloading
Watch later
Share
File Edit Selection View Go Debug Terminal Help
main.cpp
tmp > main.cpp > ...
1 #include <cassert>
2
3 class Matrix {
4
5 }
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
dsilver@dsilver-laptop:[/tmp]$
I would strongly recommend using a library like Eigen or some other pre-defined library,
0:32 / 8:37
YouTube


Operatoroverloading
Watch later
Share
#include <cassert>
#include <vector>
class Matrix {
public:
Matrix(int rows, int columns)
: rows_(rows), columns_(columns), values_(rows * columns) {}
int& operator()(int row, int column) {
}
private:
int rows_;
int columns;
std::vector<int> values_;
};
int main() {
Matrix matrix(2, 2);
matrix(0, 0) = 4;
assert(matrix(0, 0) == 4);
}
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
dsilver@dsilver-laptop:[/tmp]$
which means we're returning a reference to an integer,
3:51 / 8:37
YouTube


Operatoroverloading
Watch later
Share
#include <vector>
class Matrix {
public:
Matrix(int rows, int columns)
: rows_(rows), columns_(columns), values_(rows * columns) {}
int& operator()(int row, int column) {
return values_[row*columns_+column];
}
int operator()(int row, int column) const {
return values_[row*columns_+column];
}
Matrix operator+(Matrix m){}
private:
int rows_;
int columns_;
std::vector<int> values_;
};
int main() {
Matrix matrix(2, 2);
matrix(0, 0) = 4;
assert(matrix(0, 0) == 4);
Matrix matrix3 = matrix1 + matrix2;
}
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
dsilver@dsilver-laptop:[/tmp]$ g++ main.cpp
dsilver@dsilver-laptop:[/tmp]$ ./a.out
dsilver@dsilver-laptop:[/tmp]$
8:35 / 8:37
YouTube

Operator Overloading

. In this exercise you'll see how to achieve polymorphism with operator overloading. You can choose any operator from the ASCII table and give it your own set of rules!

Operator overloading can be useful for many things. Consider the `+` operator. We can use it to add `int`s, `double`s, `float`s, or even `std::string`s.

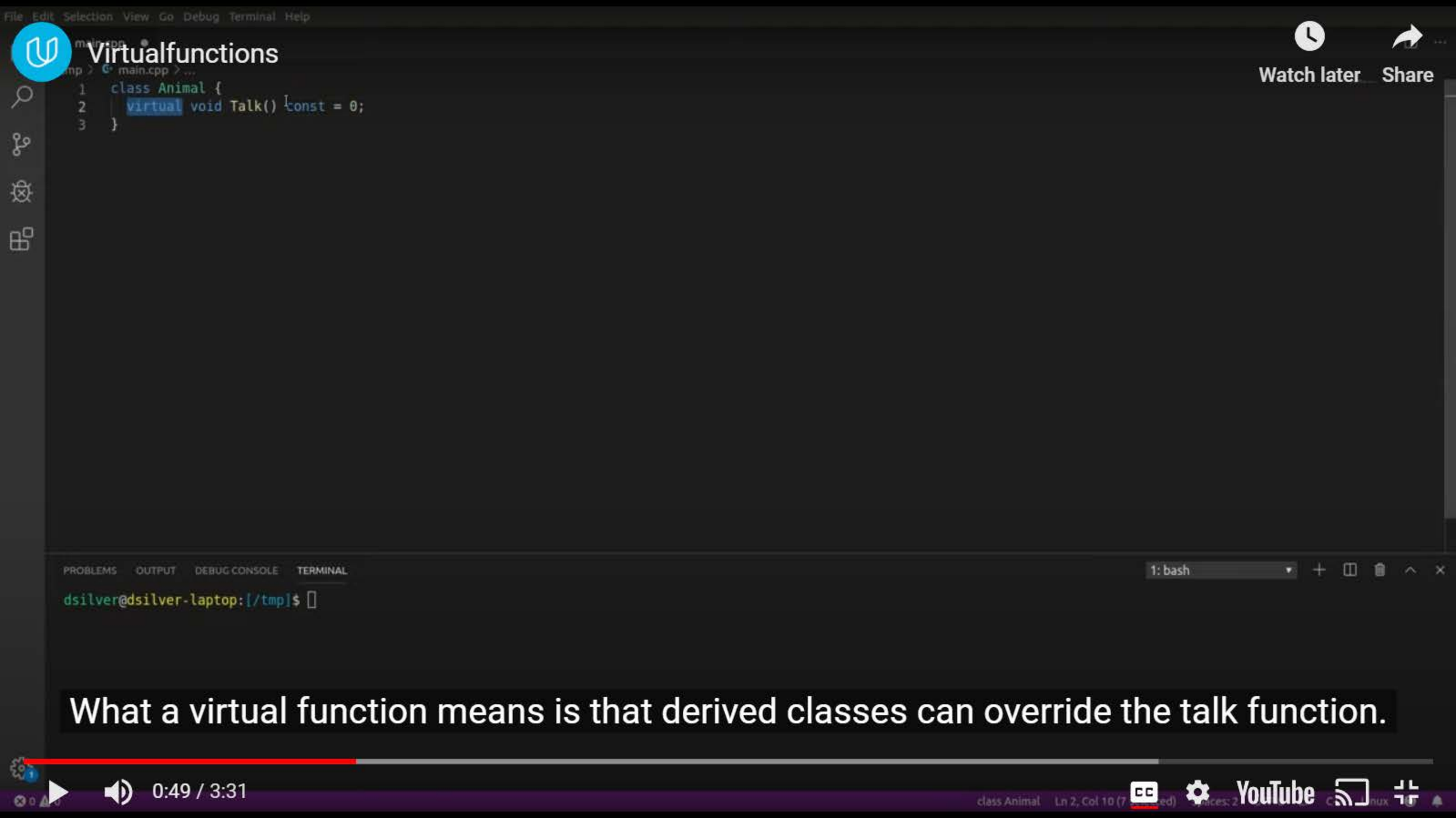
In order to overload an operator, use the `operator` keyword in the function signature:

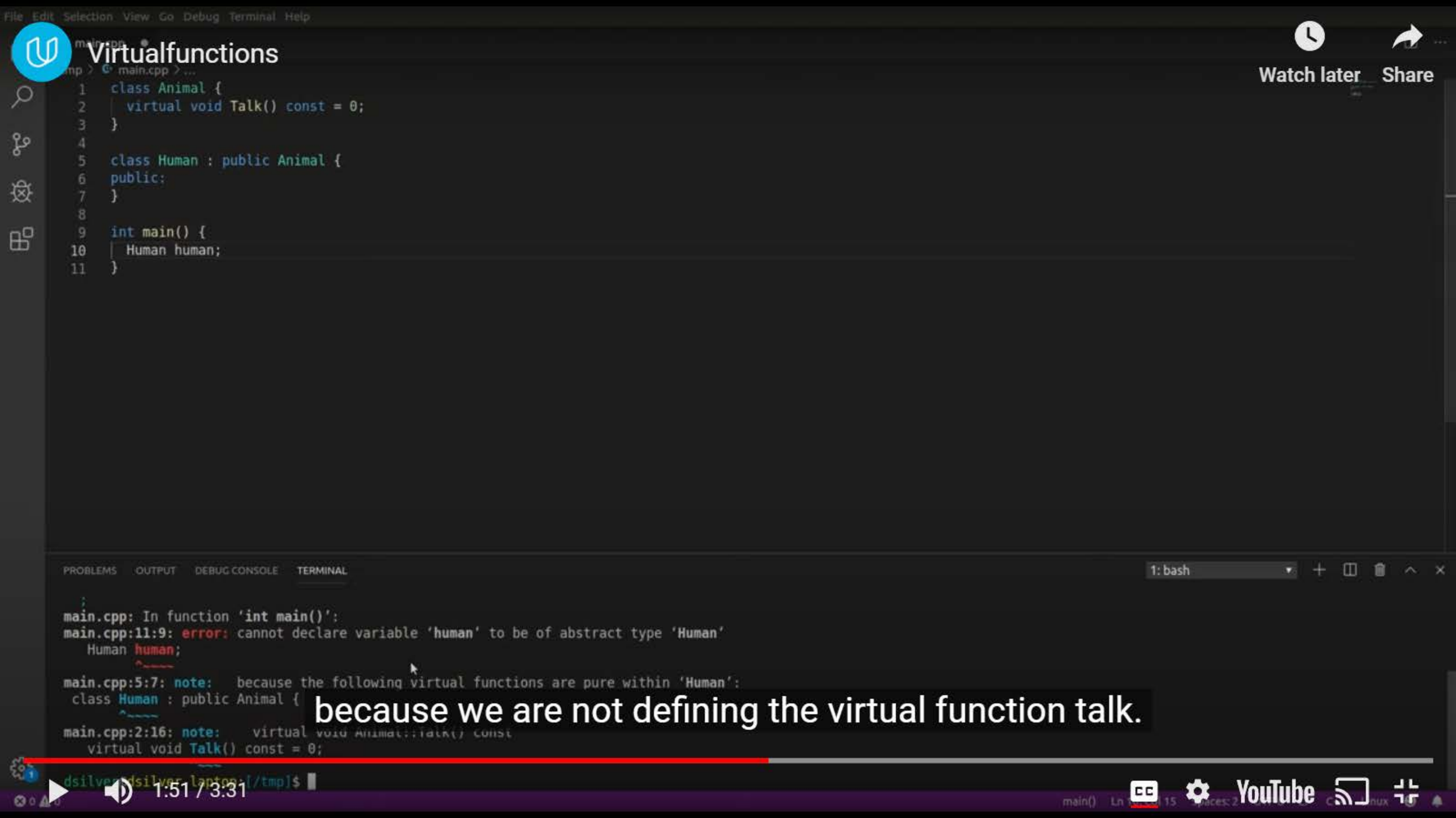
```
Complex operator+(const Complex& addend) {
  //...logic to add complex numbers
}
```

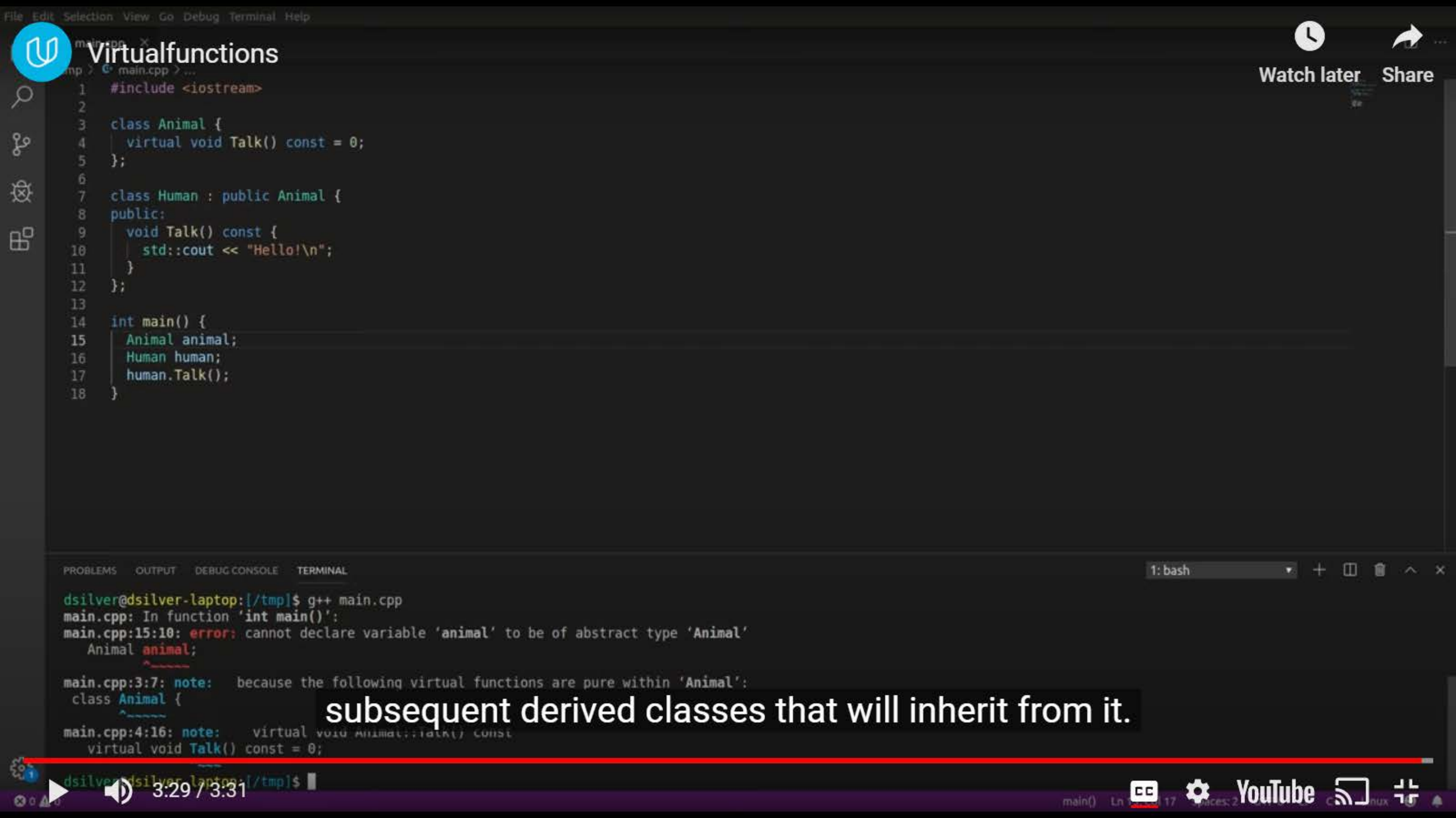
Imagine vector addition. You might want to perform vector addition on a pair of points to add their x and y components. The compiler won't recognize this type of operation on its own, because this data is user defined. However, you can overload the `+` operator so it performs the action that you want to implement.

Instructions

1. Define class `Point`.
2. Declare a prototype of overload method for `+` operator.
3. Confirm the tests pass.


Virtualfunctions
Watch later
Share
main.cpp
1 class Animal {
2 virtual void Talk() const = 0;
3 }
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
dsilver@dsilver-laptop:[/tmp]$
What a virtual function means is that derived classes can override the talk function.
0:49 / 3:31
YouTube


Virtualfunctions
Watch later
Share
main.cpp
1 class Animal {
2 virtual void Talk() const = 0;
3 }
4
5 class Human : public Animal {
6 public:
7 }
8
9 int main() {
10 Human human;
11 }
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
main.cpp: In function 'int main()':
main.cpp:11:9: error: cannot declare variable 'human' to be of abstract type 'Human'
Human human;
main.cpp:5:7: note: because the following virtual functions are pure within 'Human':
class Human : public Animal {
main.cpp:2:16: note: virtual void Talk() const = 0;
because we are not defining the virtual function talk.
1:51 / 3:31
YouTube


Virtualfunctions
Watch later
Share
#include <iostream>
class Animal {
virtual void Talk() const = 0;
};
class Human : public Animal {
public:
void Talk() const {
std::cout << "Hello!\n";
}
};
int main() {
Animal animal;
Human human;
human.Talk();
}
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
dsilver@dsilver-laptop:[/tmp]$ g++ main.cpp
main.cpp: In function 'int main()':
main.cpp:15:10: error: cannot declare variable 'animal' to be of abstract type 'Animal'
Animal animal;
main.cpp:3:7: note: because the following virtual functions are pure within 'Animal':
class Animal {
main.cpp:4:16: note: virtual void Talk() const = 0;
dsilver@dsilver-laptop:[/tmp]$
subsequent derived classes that will inherit from it.
3:29 / 3:31
YouTube

Virtual Functions

Virtual functions are a polymorphic feature. These functions are declared (and possibly defined) in a base class, and can be overridden by derived classes.

This approach declares an interface at the base level, but delegates the implementation of the interface to the derived classes.

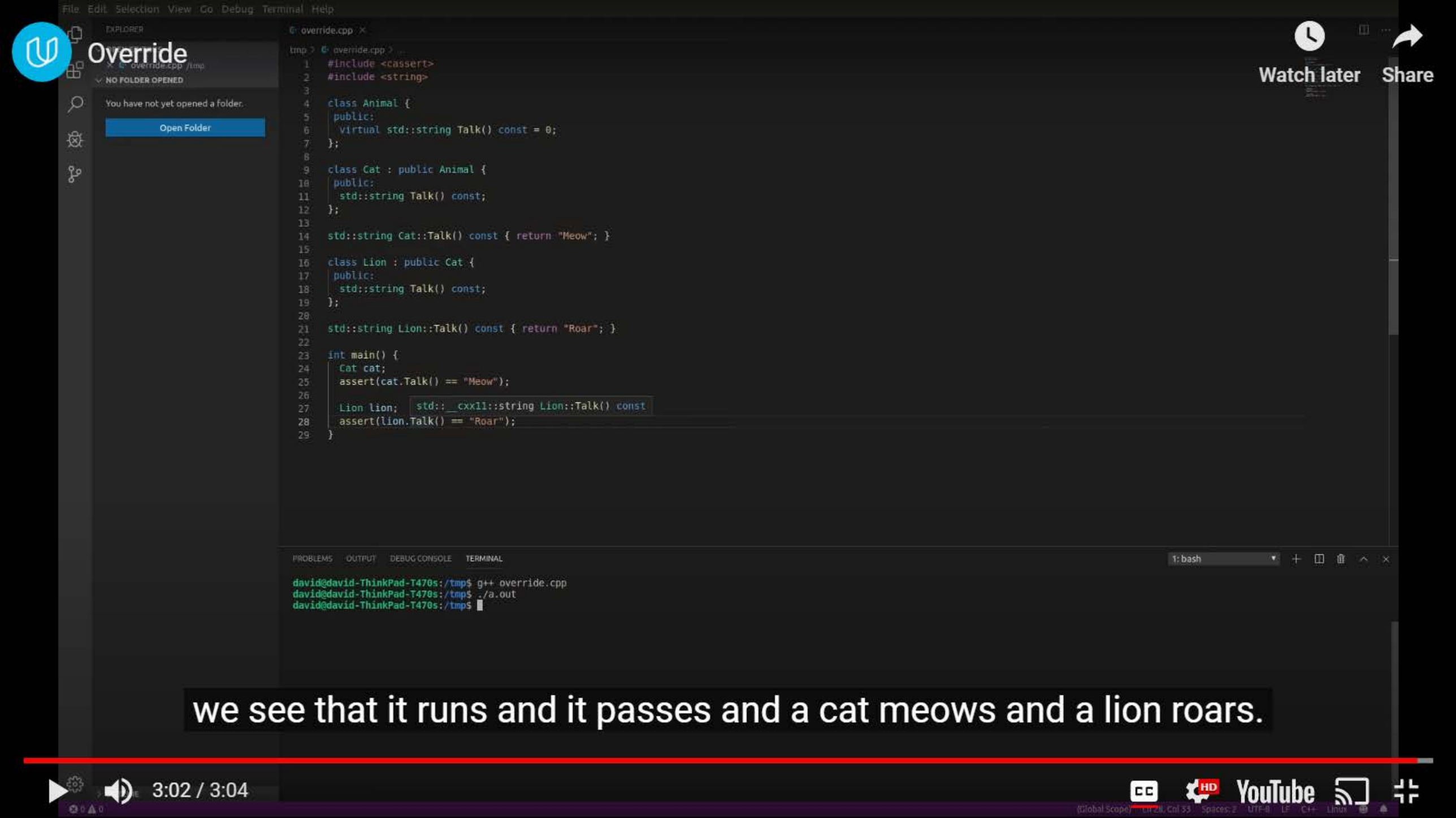
In this exercise, `class Shape` is the base class. Geometrical shapes possess both an area and a perimeter. `Area()` and `Perimeter()` should be virtual functions of the base class interface. Append `= 0` to each of these functions in order to declare them to be "pure" virtual functions.

A pure virtual function is a virtual function that the base class declares but does not define.

A pure virtual function has the side effect of making its class abstract. This means that the class cannot be instantiated. Instead, only classes that derive from the abstract class and override the pure virtual function can be instantiated.

```
class Shape {
    public:
        Shape() {}
        virtual double Area() const = 0;
        virtual double Perimeter() const = 0;
};
```

Virtual functions can be defined by derived classes, but this is not required. However, if we mark the virtual function with `= 0` in the base class, then we are declaring the function to be a pure virtual function. This means that the base class does not define this function. A derived class must define this function, or else the derived class will be abstract.


Override
Watch later
Share
override.cpp
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
#include <cassert>
#include <string>
class Animal {
public:
virtual std::string Talk() const = 0;
};
class Cat : public Animal {
public:
std::string Talk() const;
};
std::string Cat::Talk() const { return "Meow"; }
class Lion : public Cat {
public:
std::string Talk() const;
};
std::string Lion::Talk() const { return "Roar"; }
int main() {
Cat cat;
assert(cat.Talk() == "Meow");
Lion lion;
assert(lion.Talk() == "Roar");
}
std::__cxx11::string Lion::Talk() const
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
david@david-ThinkPad-T470s:/tmp$ g++ override.cpp
david@david-ThinkPad-T470s:/tmp$ ./a.out
david@david-ThinkPad-T470s:/tmp$
we see that it runs and it passes and a cat meows and a lion roars.
3:02 / 3:04
HD
YouTube

Polymorphism: Overriding

"Overriding" a function occurs when:

1. A base class declares a `virtual` function.
2. A derived class *overrides* that virtual function by defining its own implementation with an identical function signature (i.e. the same function name and argument types).

```
class Animal {
public:
  virtual std::string Talk() const = 0;
};

class Cat {
public:
  std::string Talk() const { return std::string("Meow"); }
};
```

In this example, `Animal` exposes a `virtual` function: `Talk()`, but does not define it. Because `Animal::Talk()` is undefined, it is called a *pure virtual function*, as opposed to an ordinary (impure? 😐) *virtual function*.

Furthermore, because `Animal` contains a pure virtual function, the user cannot instantiate an object of type `Animal`. This makes `Animal` an ***abstract class***.

`Cat`, however, inherits from `Animal` and overrides `Animal::Talk()` with `Cat::Talk()`, which is defined. Therefore, it is possible to instantiate an object of type `Cat`.

Instructions

1. Create a class `Dog` to inherit from `Animal`.
2. Define `Dog::Talk()` to override the virtual function `Animal::Talk()`.
3. Confirm that the tests pass.

Function Hiding

Function hiding is closely related, but distinct from, overriding.

A derived class hides a base class function, as opposed to overriding it, if the base class function is not specified to be `virtual`.

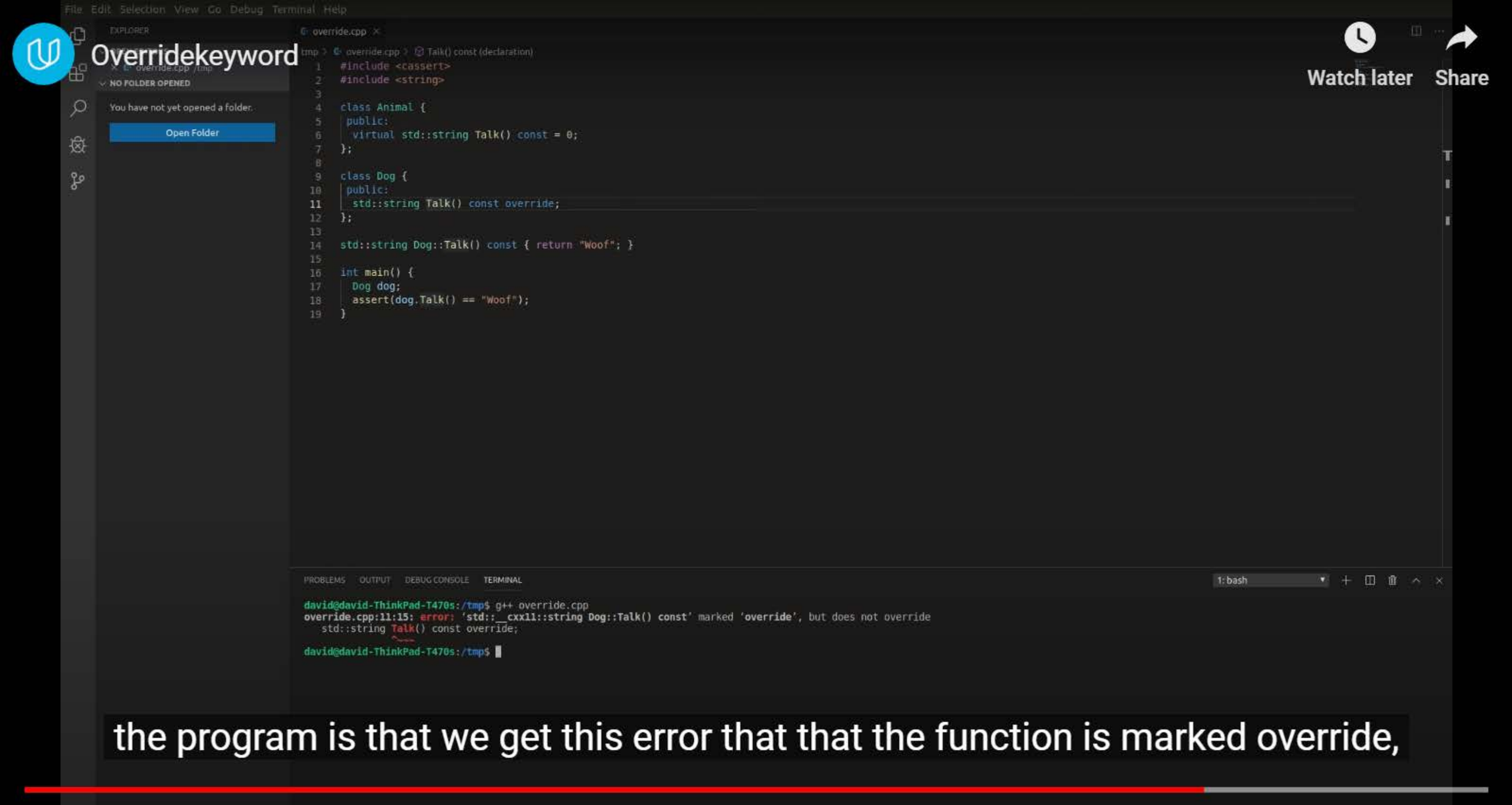
```
class Cat { // Here, Cat does not derive from a base class
public:
  std::string Talk() const { return std::string("Meow"); }
};

class Lion : public Cat {
public:
  std::string Talk() const { return std::string("Roar"); }
};
```

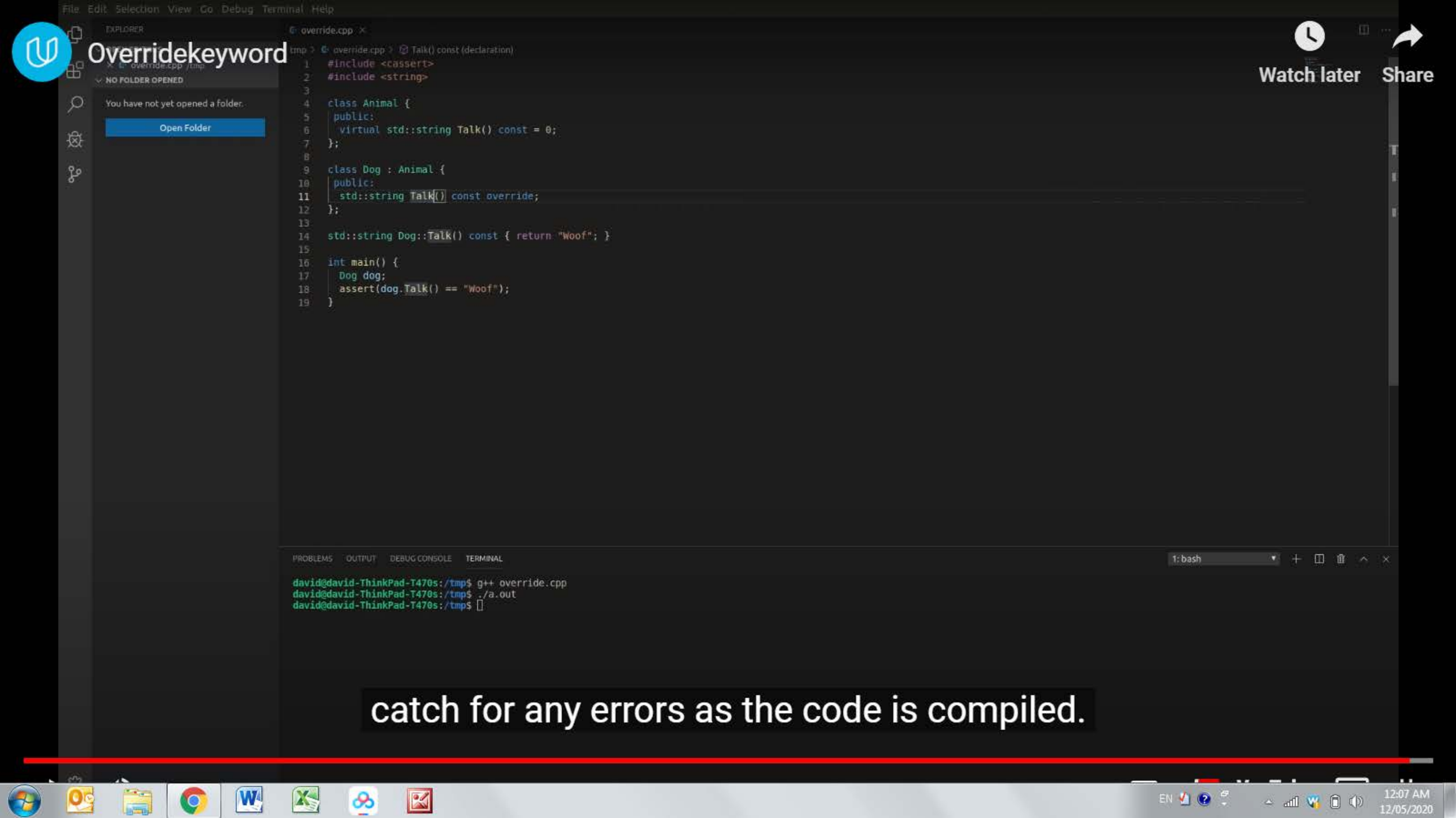
In this example, `Cat` is the base class and `Lion` is the derived class. Both `Cat` and `Lion` have `Talk()` member functions.

When an object of type `Lion` calls `Talk()`, the object will run `Lion::Talk()`, not `Cat::Talk()`.

In this situation, `Lion::Talk()` is *hiding* `Cat::Talk()`. If `Cat::Talk()` were `virtual`, then `Lion::Talk()` would *override* `Cat::Talk()`, instead of *hiding* it. *Overriding* requires a `virtual` function in the base class.

The distinction between *overriding* and *hiding* is subtle and not terribly significant, but in certain situations *hiding* can lead to bizarre errors, particularly when the two functions have slightly different function signatures.


Overridekeyword
Watch later
Share
override.cpp
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
#include <cassert>
#include <string>
class Animal {
public:
virtual std::string Talk() const = 0;
};
class Dog {
public:
std::string Talk() const override;
};
std::string Dog::Talk() const { return "Woof"; }
int main() {
Dog dog;
assert(dog.Talk() == "Woof");
}
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
david@david-ThinkPad-T470s:/tmp$ g++ override.cpp
override.cpp:11:15: error: 'std::__cxx11::string Dog::Talk() const' marked 'override', but does not override
std::string Talk() const override;
david@david-ThinkPad-T470s:/tmp$
the program is that we get this error that that the function is marked override,


Overridekeyword
Watch later
Share
override.cpp
#include <cassert>
#include <string>
class Animal {
public:
virtual std::string Talk() const = 0;
};
class Dog : Animal {
public:
std::string Talk() const override;
};
std::string Dog::Talk() const { return "Woof"; }
int main() {
Dog dog;
assert(dog.Talk() == "Woof");
}
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
david@david-ThinkPad-T470s:/tmp$ g++ override.cpp
david@david-ThinkPad-T470s:/tmp$ ./a.out
david@david-ThinkPad-T470s:/tmp$
catch for any errors as the code is compiled.

Override

"Overriding" a function occurs when a derived class defines the implementation of a `virtual` function that it inherits from a base class.

It is possible, but not required, to specify a function declaration as `override`.

```
class Shape {
public:
  virtual double Area() const = 0;
  virtual double Perimeter() const = 0;
};

class Circle : public Shape {
public:
  Circle(double radius) : radius_(radius) {}
  double Area() const override { return pow(radius_, 2) * PI; } // specified as an override function
  double Perimeter() const override { return 2 * radius_ * PI; } // specified as an override function

private:
  double radius_;
};
```

This specification tells both the compiler and the human programmer that the purpose of this function is to override a virtual function. The compiler will verify that a function specified as `override` does indeed override some other virtual function, or otherwise the compiler will generate an error.

Specifying a function as `override` is good practice, as it empowers the compiler to verify the code, and communicates the intention of the code to future users.

Exercise

In this exercise, you will build two vehicle motion models, and override the `Move()` member function.

The first motion model will be `class ParticleModel`. In this model, the state is `x`, `y`, and `theta` (heading). The `Move(double v, double theta)` function for this model includes instantaneous steering:

```
theta += phi
```

```
x += v * cos(theta)
```

```
y += v * cos(theta)
```

The second motion model will be `class BicycleModel`. In this model, the state is `x`, `y`, `theta` (heading), and `L` (the length of the vehicle). The `Move(double v, double theta)` function for this model is affected by the length of the vehicle:

```
theta += v / L * tan(phi)
```

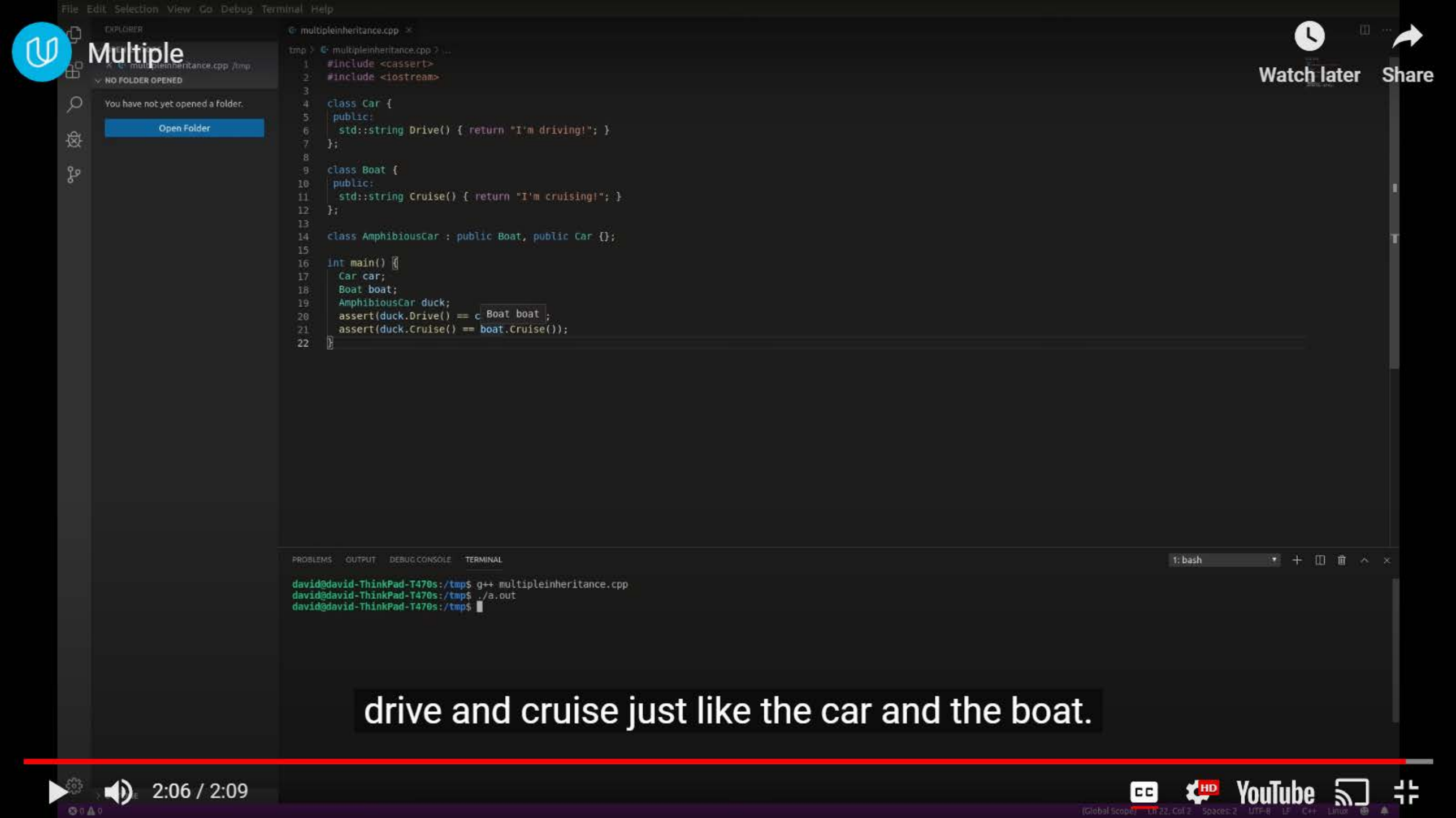
```
x += v * cos(theta)
```

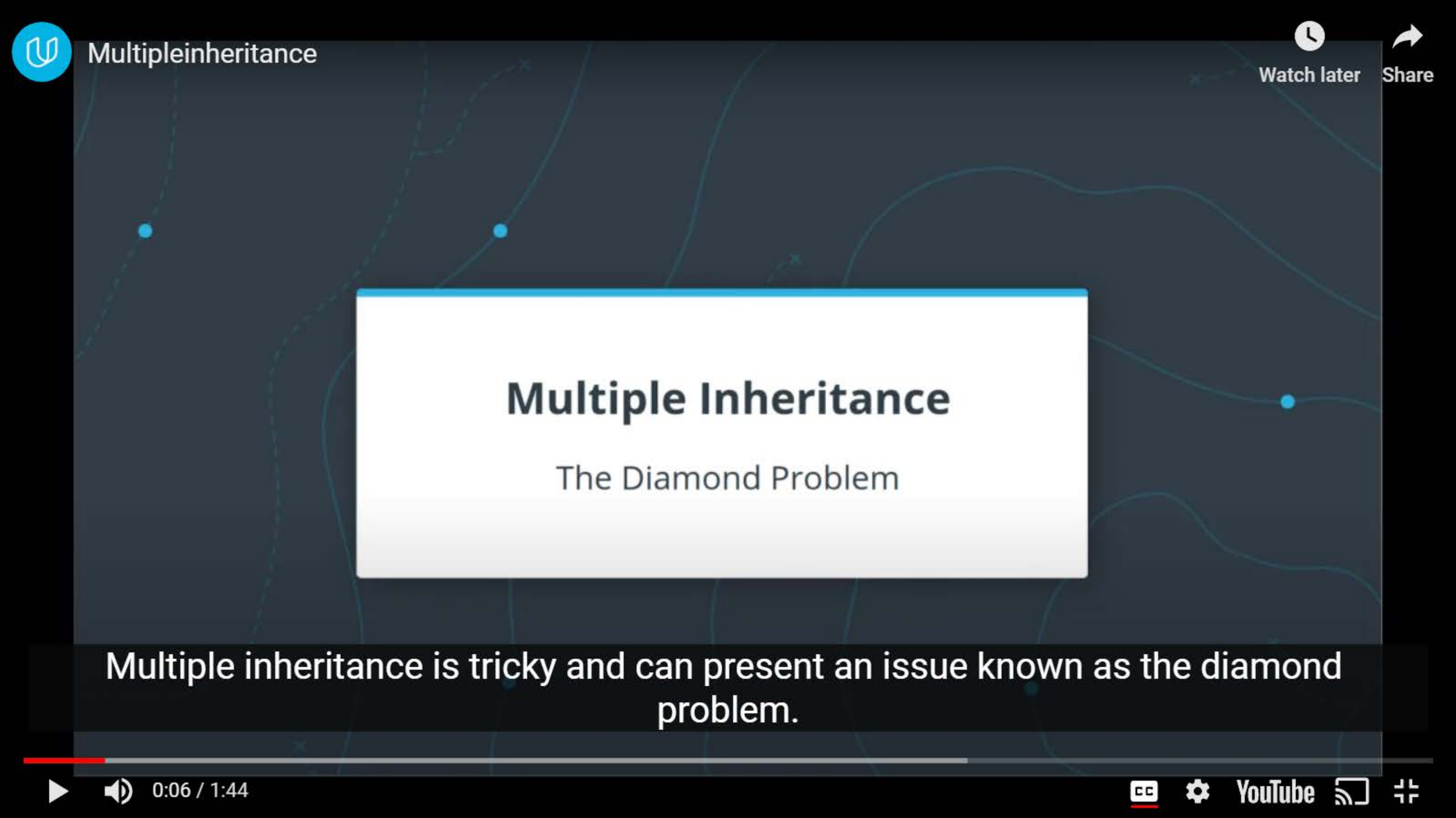
```
y += v * cos(theta)
```

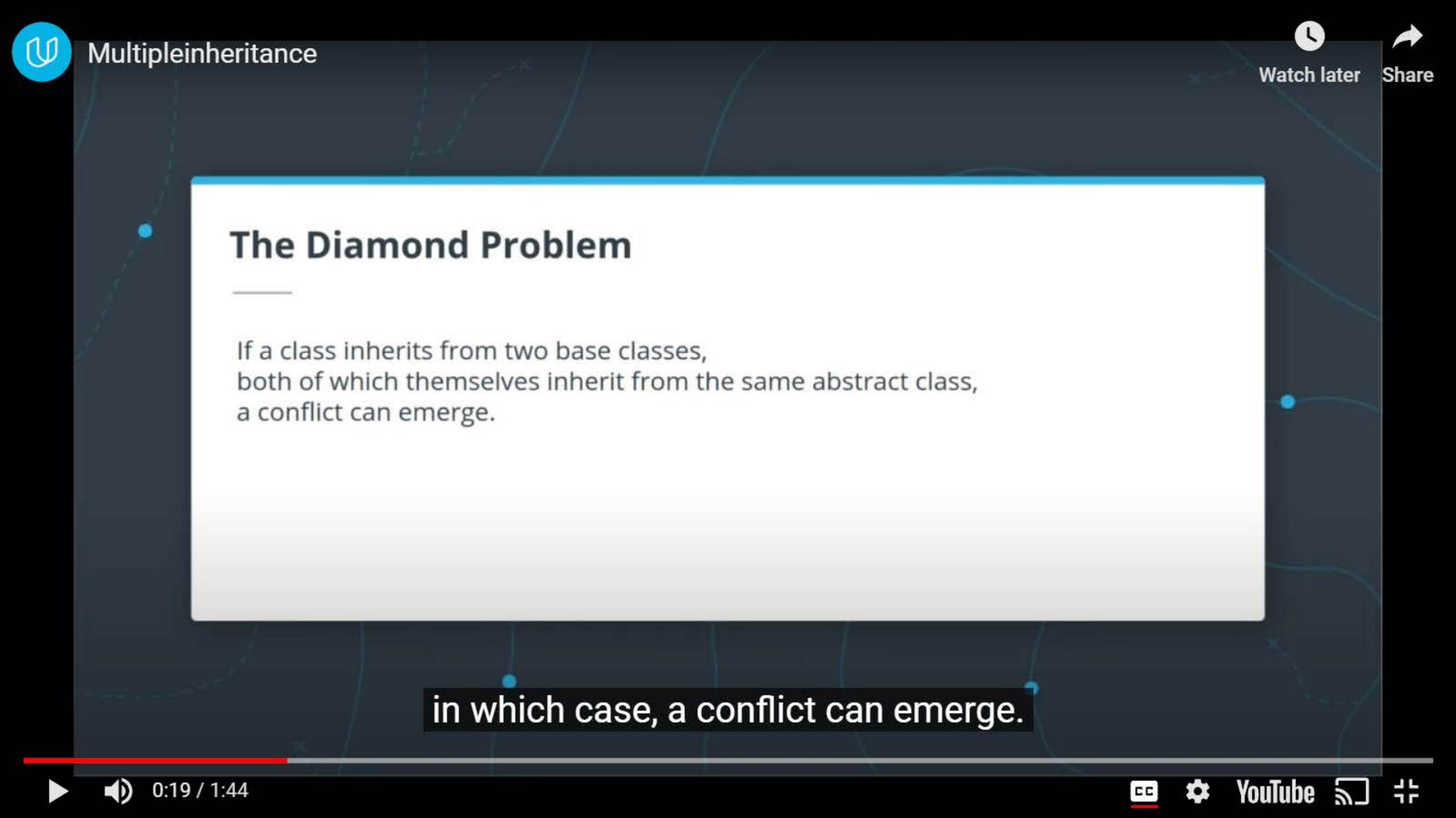
You are encouraged to read more about vehicle motion, but for the purposes of practicing function overriding, the precise motion models are not so important. What is important is that the two models, and thus to the two `Move()` functions, are *different*.

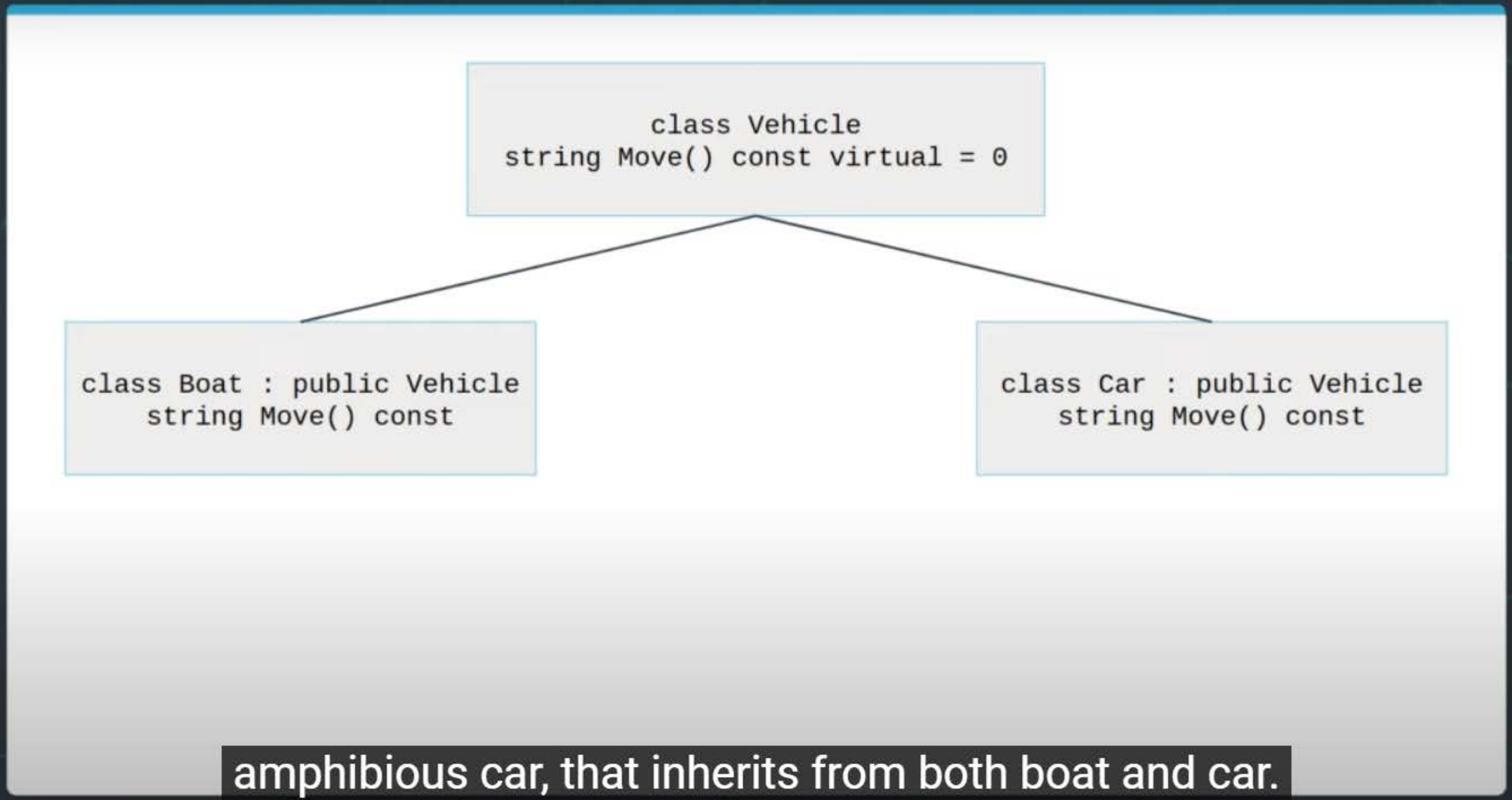
Instructions

1. Define `class ParticleModel`, including its state and `Move()` function.
2. Extend `class BicycleModel` from `class ParticleModel`.
3. Override the `Move()` function within `class BicycleModel`.
4. Specify `BicycleModel::Move()` as **`override`**.
5. Pass the tests in `main()` by verifying that the two `Move()` functions override each other in different scenarios.

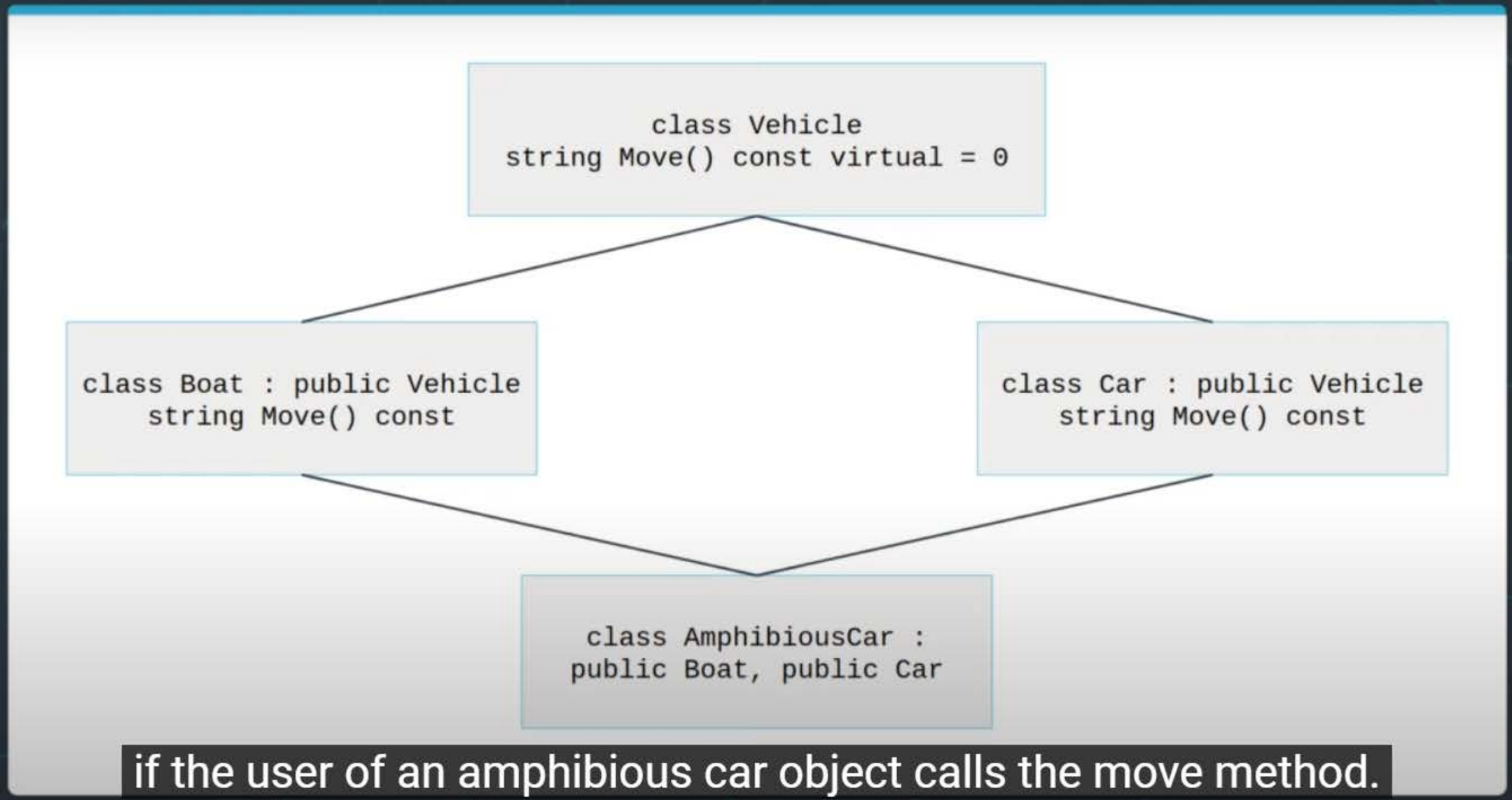

Multiple
Watch later
Share
multipleinheritance.cpp
#include <cassert>
#include <iostream>
class Car {
public:
std::string Drive() { return "I'm driving!"; }
};
class Boat {
public:
std::string Cruise() { return "I'm cruising!"; }
};
class AmphibiousCar : public Boat, public Car {};
int main() {
Car car;
Boat boat;
AmphibiousCar duck;
assert(duck.Drive() == c
Boat boat
;
assert(duck.Cruise() == boat.Cruise());
}
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
david@david-ThinkPad-T470s:/tmp$ g++ multipleinheritance.cpp
david@david-ThinkPad-T470s:/tmp$ ./a.out
david@david-ThinkPad-T470s:/tmp$
drive and cruise just like the car and the boat.
2:06 / 2:09


Multipleinheritance
Watch later
Share
Multiple Inheritance
The Diamond Problem
Multiple inheritance is tricky and can present an issue known as the diamond problem.
0:06 / 1:44
YouTube


Multipleinheritance
Watch later
Share
The Diamond Problem
If a class inherits from two base classes,
both of which themselves inherit from the same abstract class,
a conflict can emerge.
in which case, a conflict can emerge.
0:19 / 1:44
YouTube



amphibious car, that inherits from both boat and car.



if the user of an amphibious car object calls the move method.

Multiple Inheritance

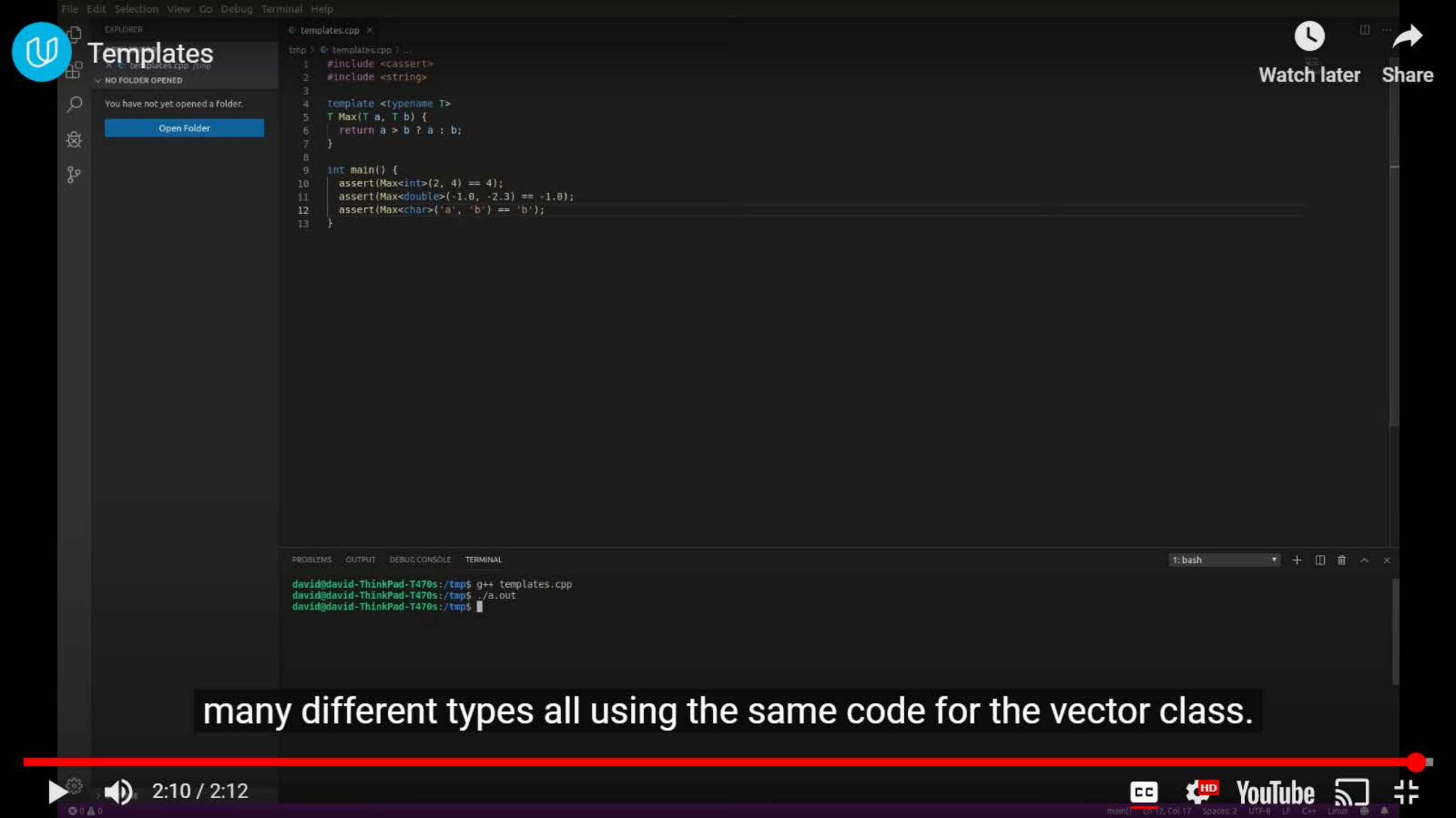
In this exercise, you'll get some practical experience with multiple inheritance. If you have class `Animal` and another class `Pet`, then you can construct a class `Dog`, which inherits from both of these base classes. In doing this, you are able to incorporate attributes of multiple base classes.

The Core Guidelines have some worthwhile recommendations about how and when to use multiple inheritance:

- "Use multiple inheritance to represent multiple distinct interfaces"
- "Use multiple inheritance to represent the union of implementation attributes"

Instructions

1. Review `class Dog`, which inherits from both `Animal` and `Pet`.
2. Declare a `class Cat`, with a member attribute `color`, that also inherits from both `Animal` and `Pet`.
3. Instantiate an object of `class Cat`.
4. Configure that object to pass the tests in `main()`.


Templates
Watch later
Share
templates.cpp
#include <cassert>
#include <string>
template <typename T>
T Max(T a, T b) {
return a > b ? a : b;
}
int main() {
assert(Max<int>(2, 4) == 4);
assert(Max<double>(-1.0, -2.3) == -1.0);
assert(Max<char>('a', 'b') == 'b');
}
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
david@david-ThinkPad-T470s:/tmp$ g++ templates.cpp
david@david-ThinkPad-T470s:/tmp$ ./a.out
david@david-ThinkPad-T470s:/tmp$
many different types all using the same code for the vector class.
2:10 / 2:12

Templates

Templates enable generic programming by generalizing a function to apply to any class. Specifically, templates use *types* as parameters so that the same implementation can operate on different data types.

For example, you might need a function to accept many different data types. The function acts on those arguments, perhaps dividing them or sorting them or something else. Rather than writing and maintaining the multiple function declarations, each accepting slightly different arguments, you can write one function and pass the argument types as parameters. At compile time, the compiler then expands the code using the types that are passed as parameters.

```
template <typename Type> Type Sum(Type a, Type b) { return a + b; }

int main() { std::cout << Sum<double>(20.0, 13.7) << "\n"; }
```

Because `Sum()` is defined with a template, when the program calls `Sum()` with `double`s as parameters, the function expands to become:

```
double Sum(double a, double b) {
    return a+b;
}
```

Or in this case:

```
std::cout << Sum<char>('Z', 'j') << "\n";
```

The program expands to become:

```
char Sum(char a, char b) {
    return a+b;
}
```

We use the keyword `template` to specify which function is generic. Generic code is the term for code that is independent of types. It is mandatory to put the `template<>` tag before the function signature, to specify and mark that the declaration is generic.

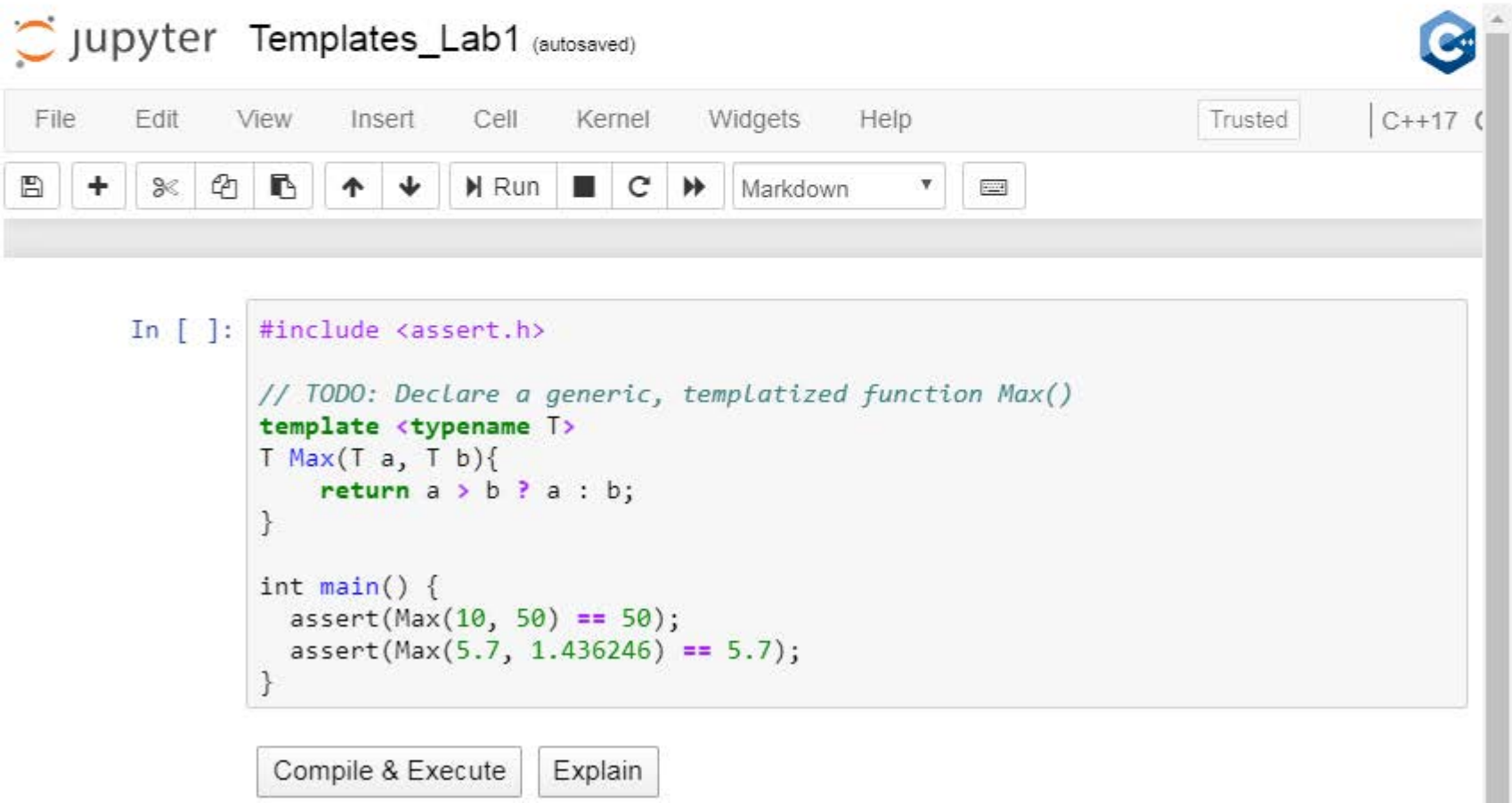
Besides `template`, the keyword `typename` (or, alternatively, `class`) specifies the generic type in the function prototype. The parameters that follow `typename` (or `class`) represent generic types in the function declaration.

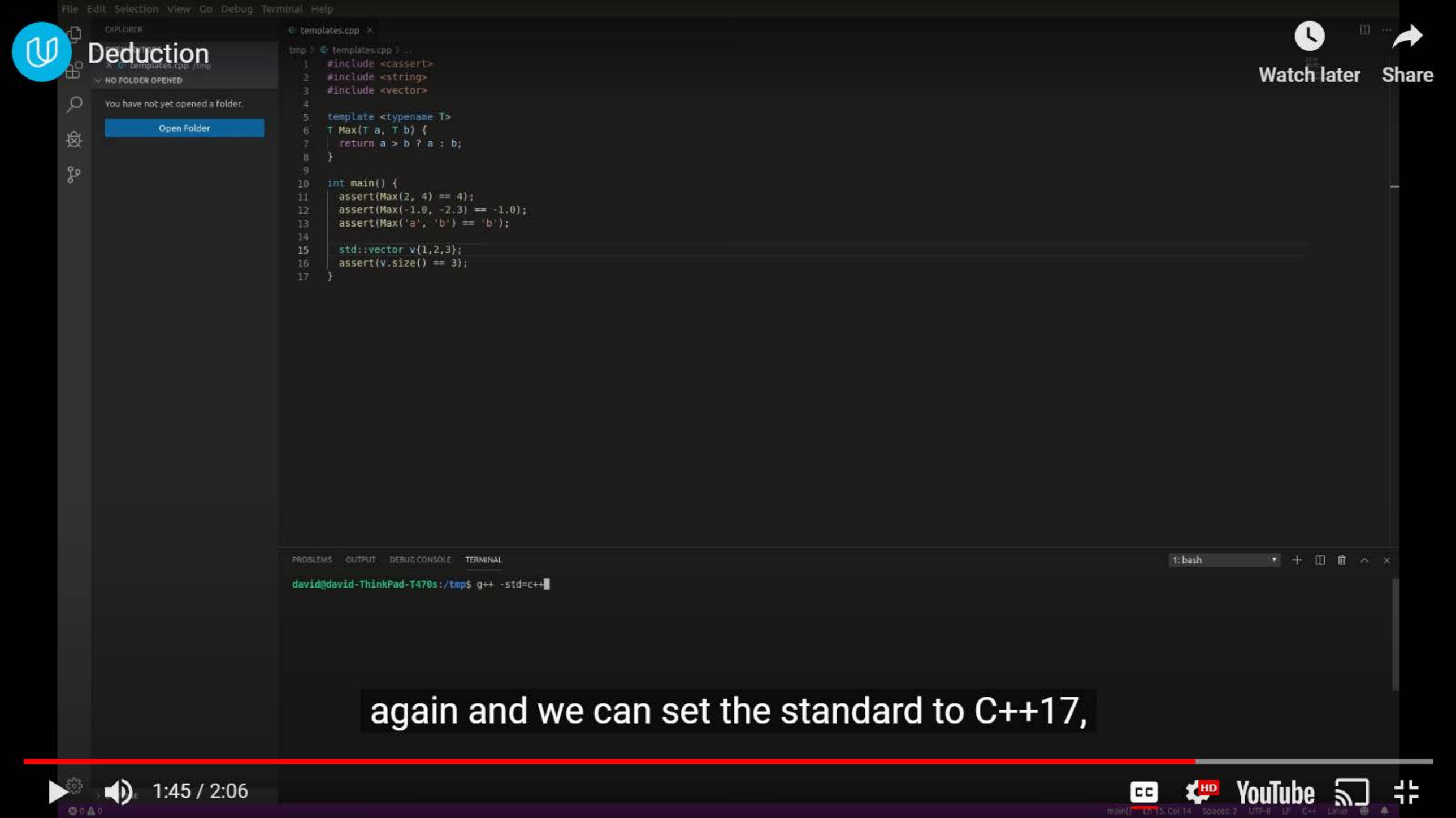
In order to instantiate a templatized class, use a templatized constructor, for example: `Sum<double>(20.0, 13.7)`. You might recognize this form as the same form used to construct a `vector`. That's because `vector`s are indeed a generic class!

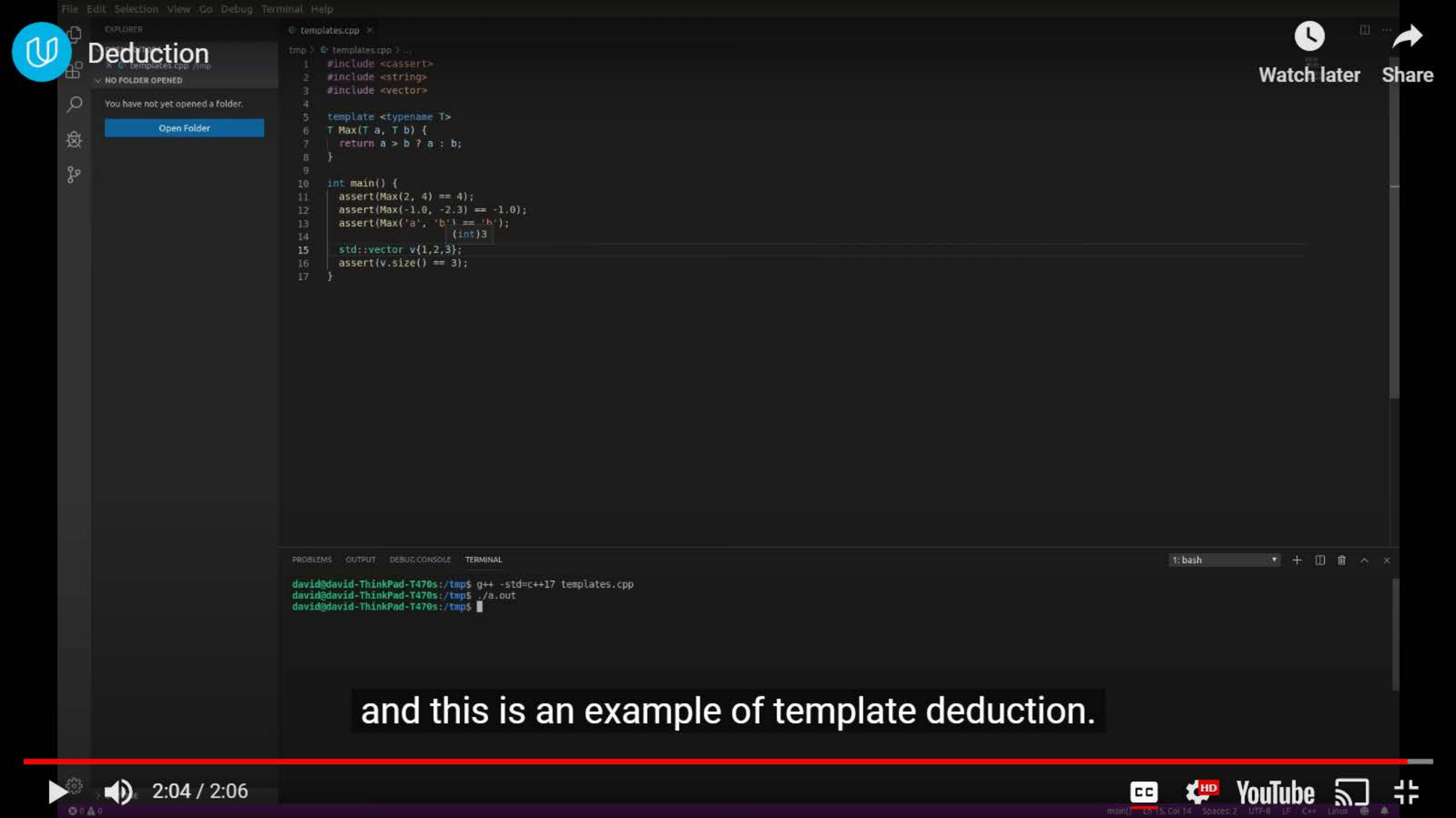
Exercise: Comparison Operator

This exercise demonstrates how a simple comparison between two variables of unknown type can work using templates. In this case, by defining a template that performs a comparison using the `>` operator, you can compare two variables of any type (both variables must be of the same type, though) as long as the operator `>` is defined for that type.

Check out the notebook below to see how that works.




Deduction
Watch later
Share
templates.cpp
#include <cassert>
#include <string>
#include <vector>
template <typename T>
T Max(T a, T b) {
return a > b ? a : b;
}
int main() {
assert(Max(2, 4) == 4);
assert(Max(-1.0, -2.3) == -1.0);
assert(Max('a', 'b') == 'b');
std::vector v{1,2,3};
assert(v.size() == 3);
}
david@david-ThinkPad-T470s:/tmp$ g++ -std=c++
again and we can set the standard to C++17,
1:45 / 2:06


Deduction
Watch later
Share
templates.cpp
#include <cassert>
#include <string>
#include <vector>
template <typename T>
T Max(T a, T b) {
return a > b ? a : b;
}
int main() {
assert(Max(2, 4) == 4);
assert(Max(-1.0, -2.3) == -1.0);
assert(Max('a', 'b') == 'b');
(int)3
std::vector v{1,2,3};
assert(v.size() == 3);
}
david@david-ThinkPad-T470s:/tmp$ g++ -std=c++17 templates.cpp
david@david-ThinkPad-T470s:/tmp$ ./a.out
and this is an example of template deduction.
2:04 / 2:06

Deduction

In this example, you will see the difference between total and partial deduction.

Deduction occurs when you instantiate an object without explicitly identifying the types. Instead, the compiler "deduces" the types. This can be helpful for writing code that is generic and can handle a variety of inputs.

In this exercise, we will use templates to overload the '#' operator to average two numbers.

Instructions

1. Use a template to overload the # operator.
2. Confirm that the tests pass.

Exercise: Class Template

Classes are the building blocks of object oriented programming in C++. Templates support the creation of generic classes!

Class templates can declare and implement generic attributes for use by generic methods. These templates can be very useful when building classes that will serve multiple purposes.

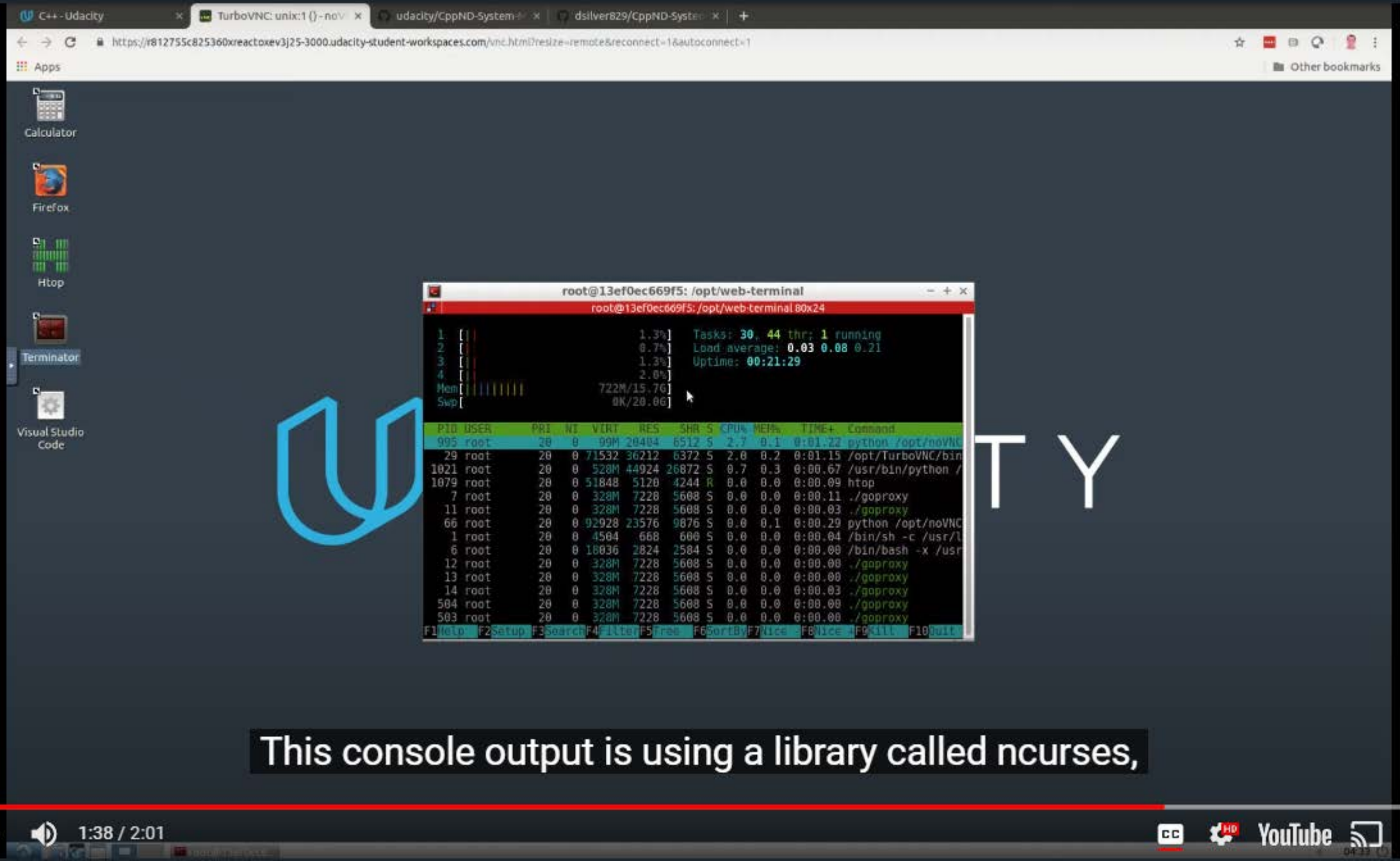
In this exercise you will create a `class Mapping` that maps a generic key to a generic value.

All of the code has been written for you, except the initial template specification.

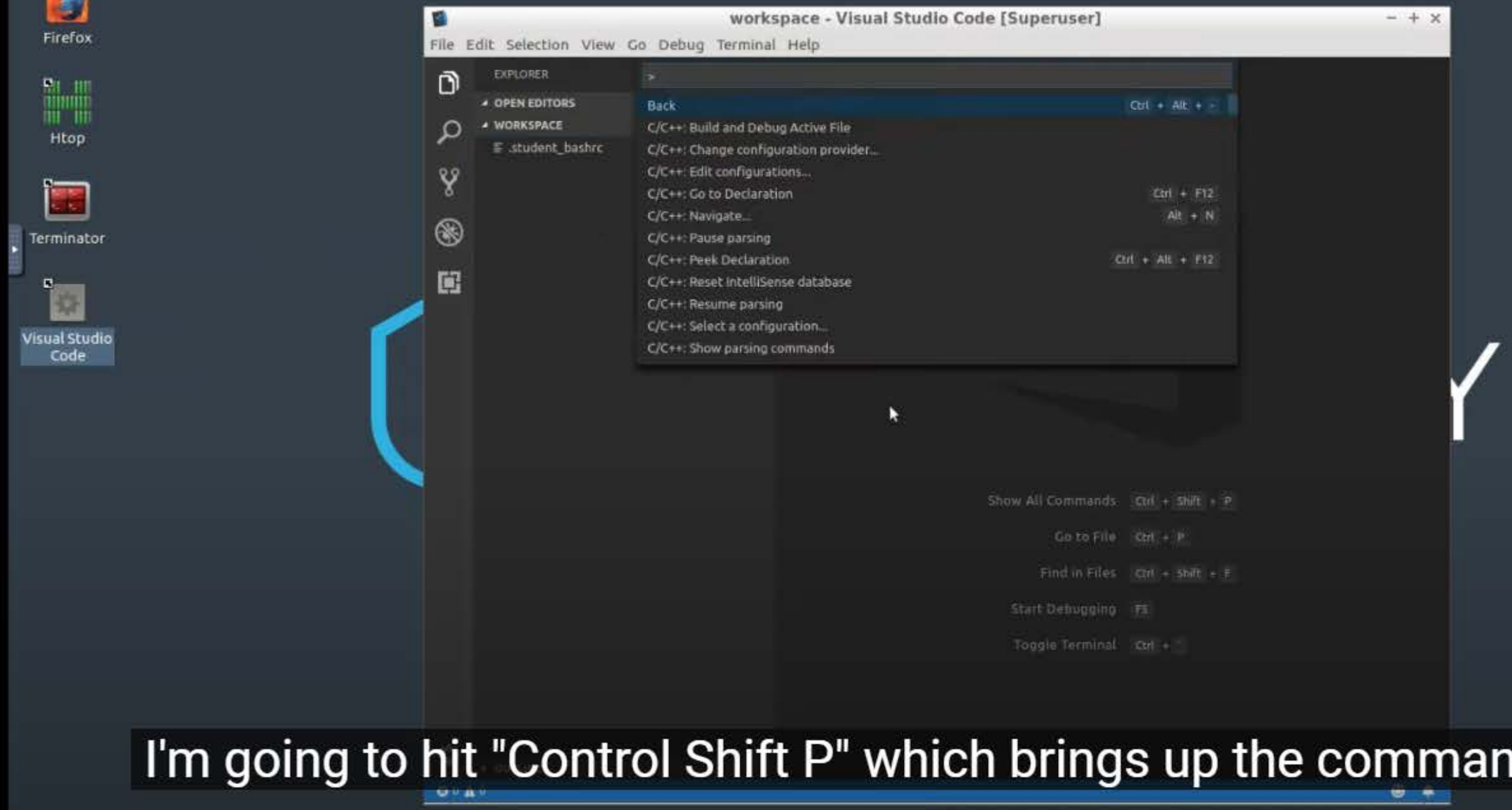
In order for this template specification to work, you will need to include two generic types: `KeyName` and `ValueName`. Can you imagine how to do that?

Instructions

1. Write the template specification.
2. Verify that the test passes.


C++ - Udacity
TurboVNC: unix:1 () - noV
udacity/CppND-System-
dsilver829/CppND-Syste
https://r81275Sc825360xreactoxev3j25-3000.udacity-student-workspaces.com/vnc.html?resize=remote&reconnect=1&autoconnect=1
Apps
Other bookmarks
Calculator
Firefox
Htop
Terminator
Visual Studio Code
root@13ef0ec669f5: /opt/web-terminal
root@13ef0ec669f5: /opt/web-terminal 80x24
1 [|| 1.3%]
2 [| 0.7%]
3 [|| 1.3%]
4 [|| 2.0%]
Mem[|||||||| 722M/15.7G]
Swp[0K/20.0G]
Tasks: 30, 44 thr; 1 running
Load average: 0.03 0.08 0.21
Uptime: 00:21:29
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
995 root 20 0 99M 20404 6512 S 2.7 0.1 0:01.22 python /opt/noVNC
29 root 20 0 71532 36212 6372 S 2.0 0.2 0:01.15 /opt/TurboVNC/bin
1021 root 20 0 520M 44924 26872 S 0.7 0.3 0:00.67 /usr/bin/python /
1079 root 20 0 51848 5120 4244 R 0.0 0.0 0:00.09 htop
7 root 20 0 328M 7228 5608 S 0.0 0.0 0:00.11 ./goproxy
11 root 20 0 328M 7228 5608 S 0.0 0.0 0:00.03 ./goproxy
66 root 20 0 92928 23576 9876 S 0.0 0.1 0:00.29 python /opt/noVNC
1 root 20 0 4504 668 600 S 0.0 0.0 0:00.04 /bin/sh -c /usr/l
6 root 20 0 18036 2824 2584 S 0.0 0.0 0:00.00 /bin/bash -x /usr
12 root 20 0 328M 7228 5608 S 0.0 0.0 0:00.00 ./goproxy
13 root 20 0 328M 7228 5608 S 0.0 0.0 0:00.00 ./goproxy
14 root 20 0 328M 7228 5608 S 0.0 0.0 0:00.03 ./goproxy
504 root 20 0 328M 7228 5608 S 0.0 0.0 0:00.00 ./goproxy
503 root 20 0 328M 7228 5608 S 0.0 0.0 0:00.00 ./goproxy
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice - F8Nice + F9Kill F10Quit
T Y
This console output is using a library called ncurses,
1:38 / 2:01
CC
HD
YouTube

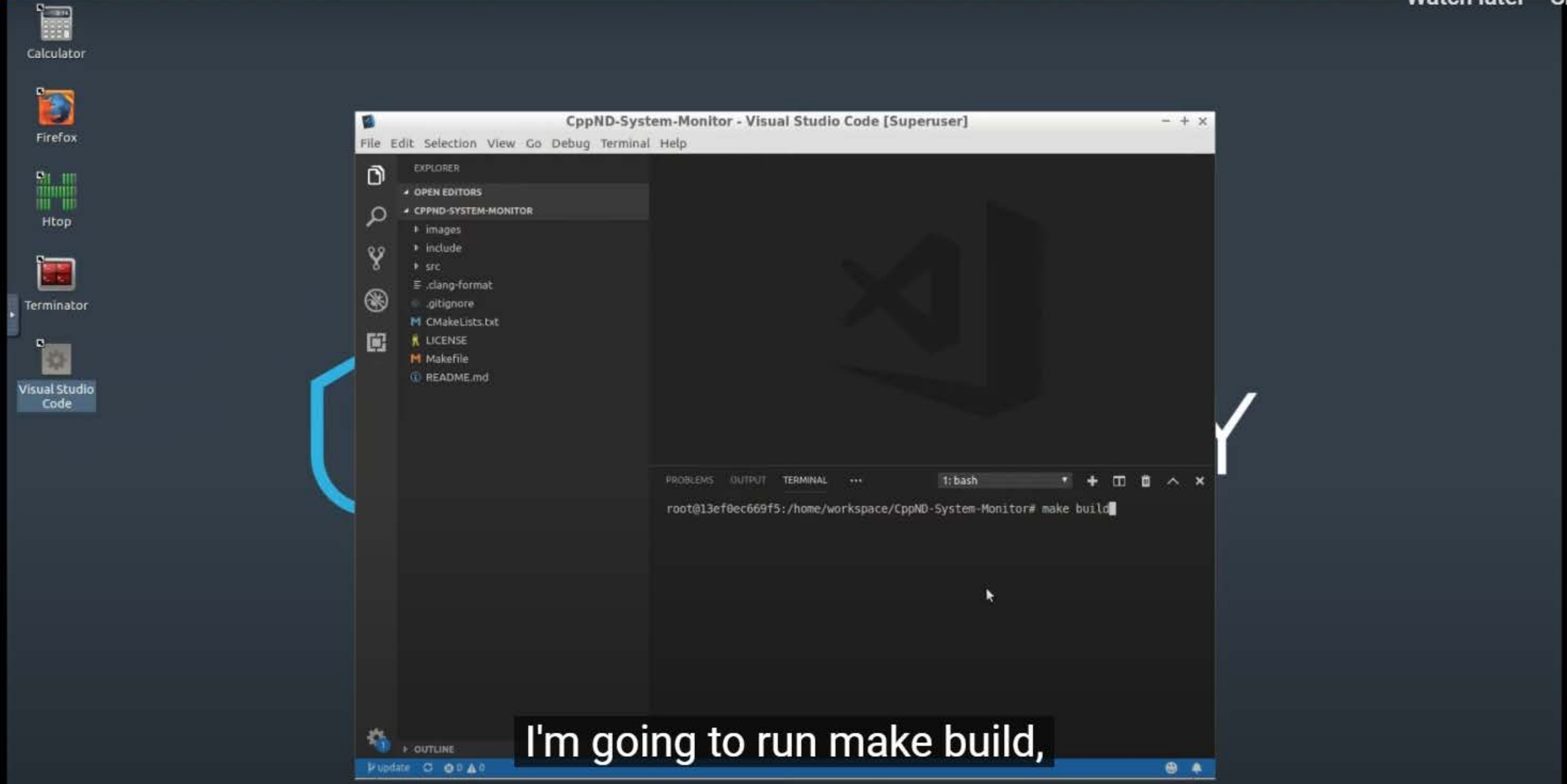
CPPND System Monitor Project Starter Code



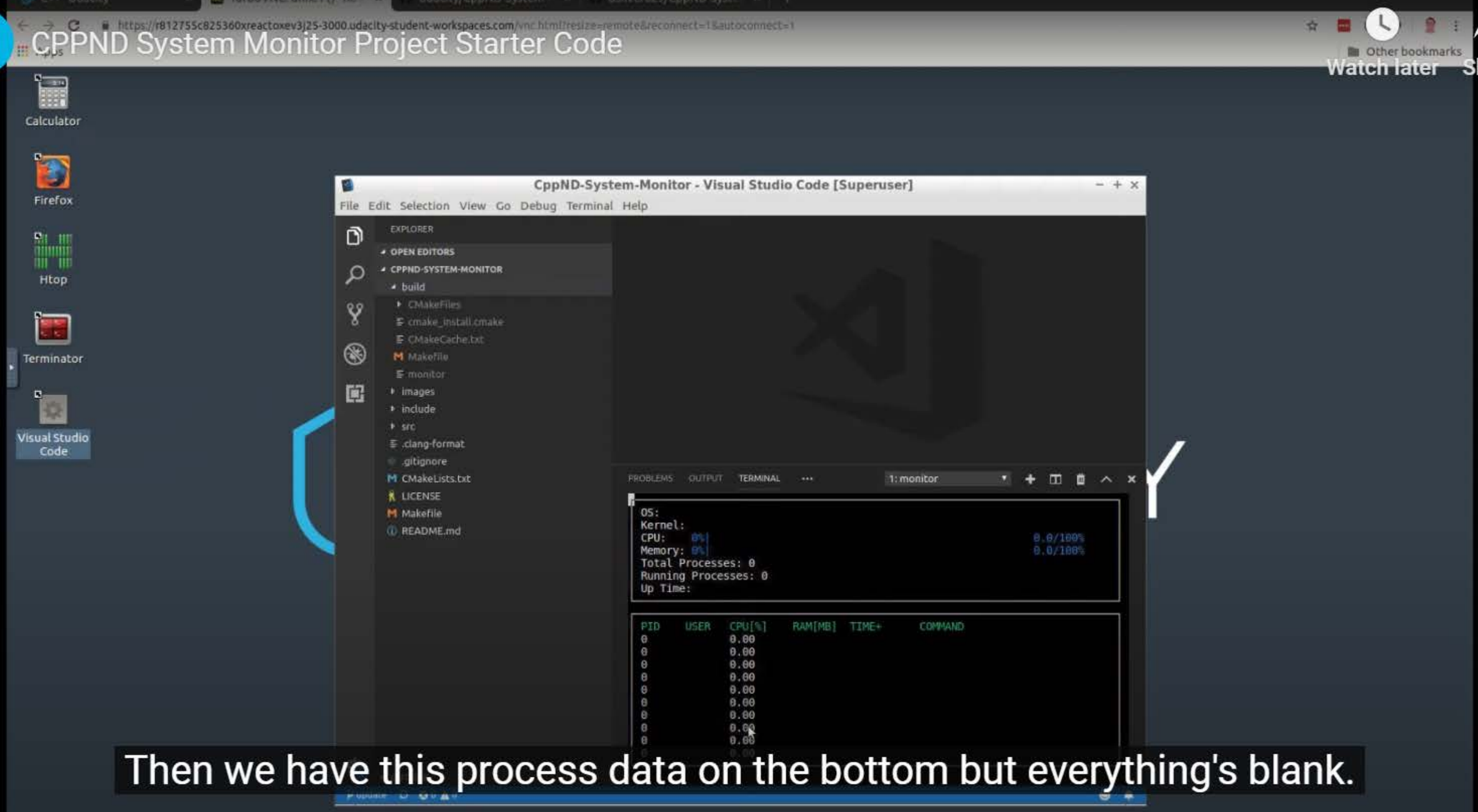
I'm going to hit "Control Shift P" which brings up the command palette.

0:35 / 3:46

CPPND System Monitor Project Starter Code



2:59 / 3:46



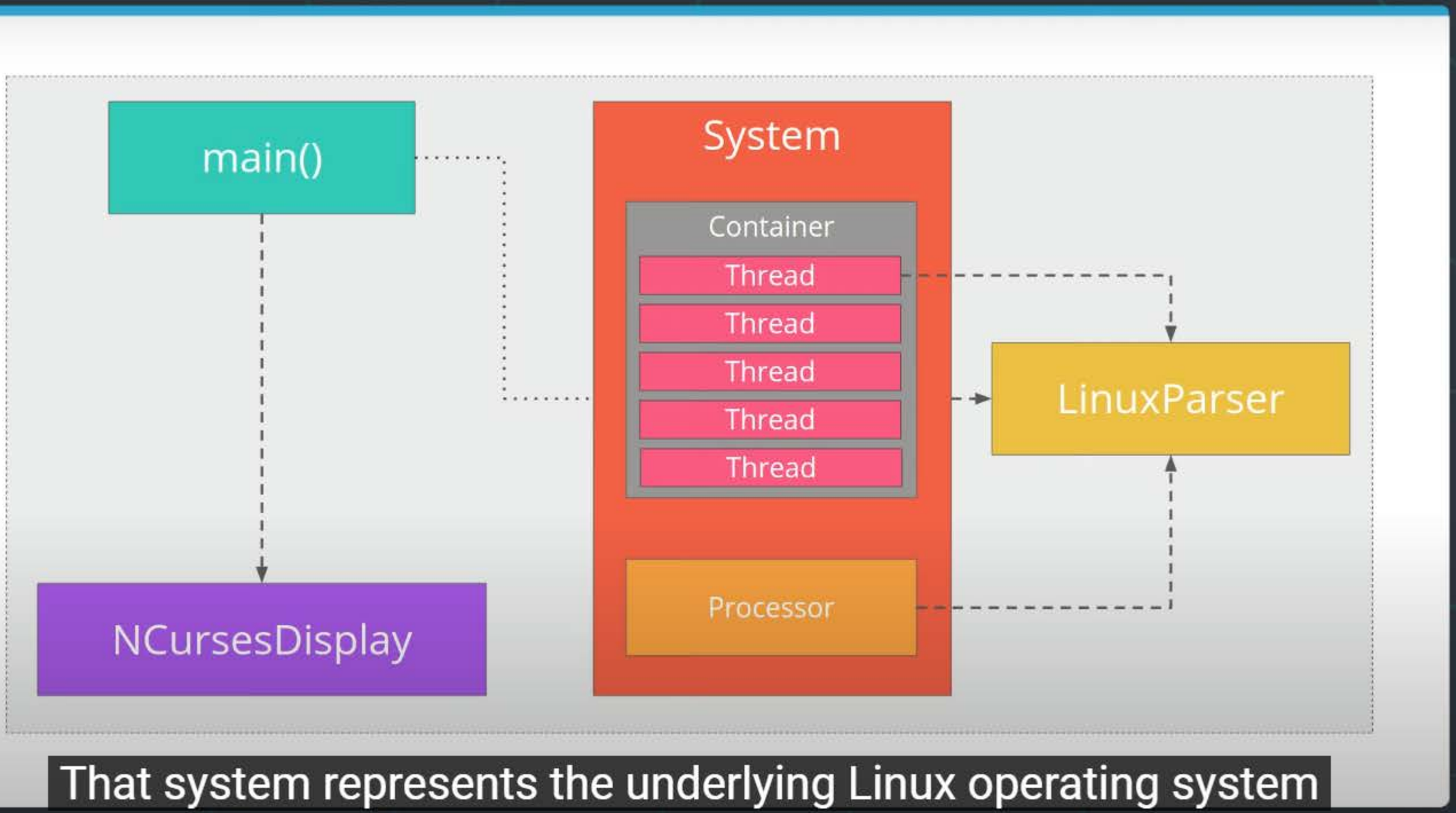
Then we have this process data on the bottom but everything's blank.


Structure
Watch later
Share
Project Structure
System Monitor
Let's look at how this project is structured.
0:02 / 2:54
YouTube

Project Structure

- Conceptual Objects
- Code Organization
- Starter Code
- Semantic Markers

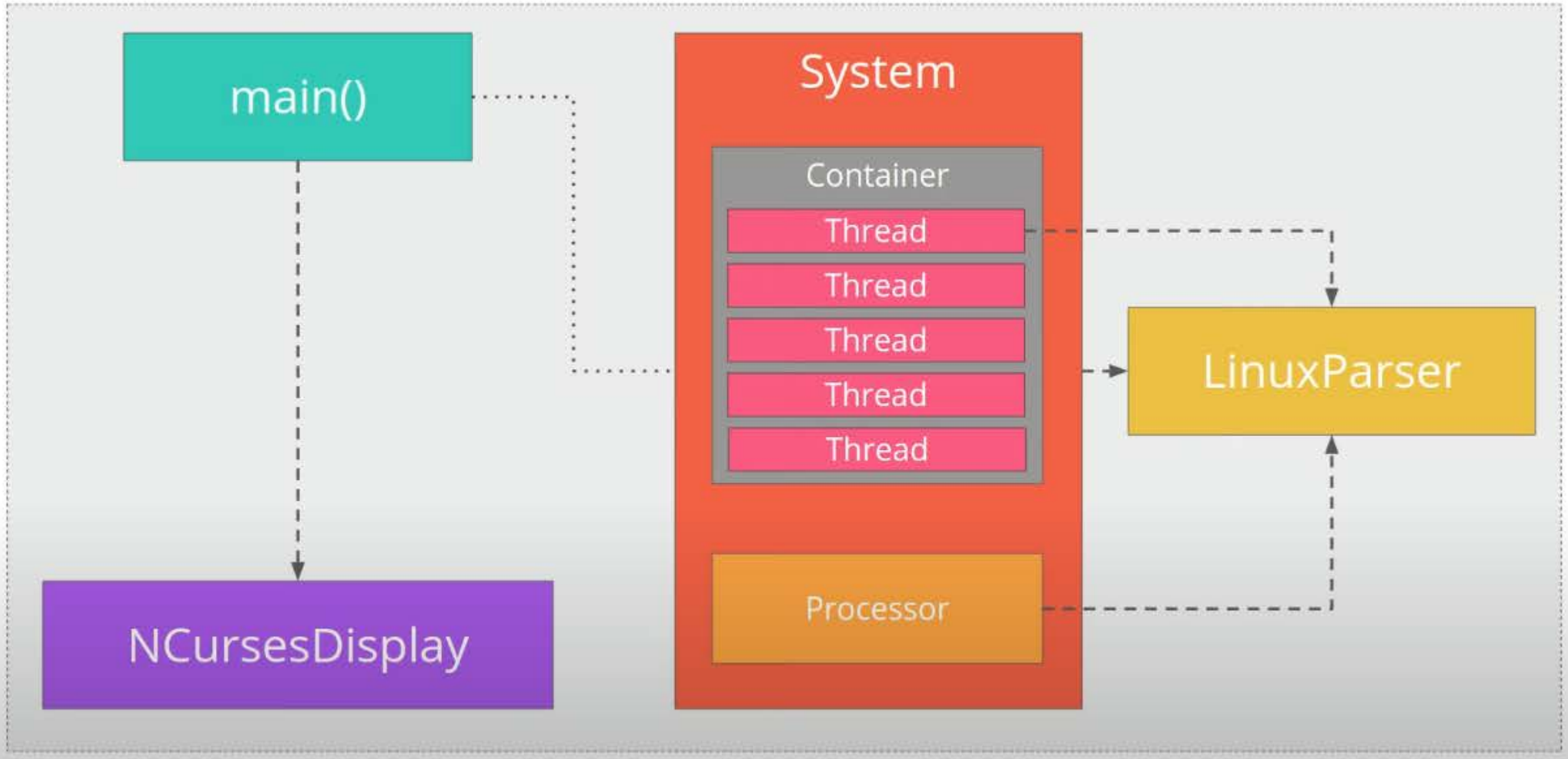
So let's start with the conceptual objects.



That system represents the underlying Linux operating system

Structure

Watch later Share

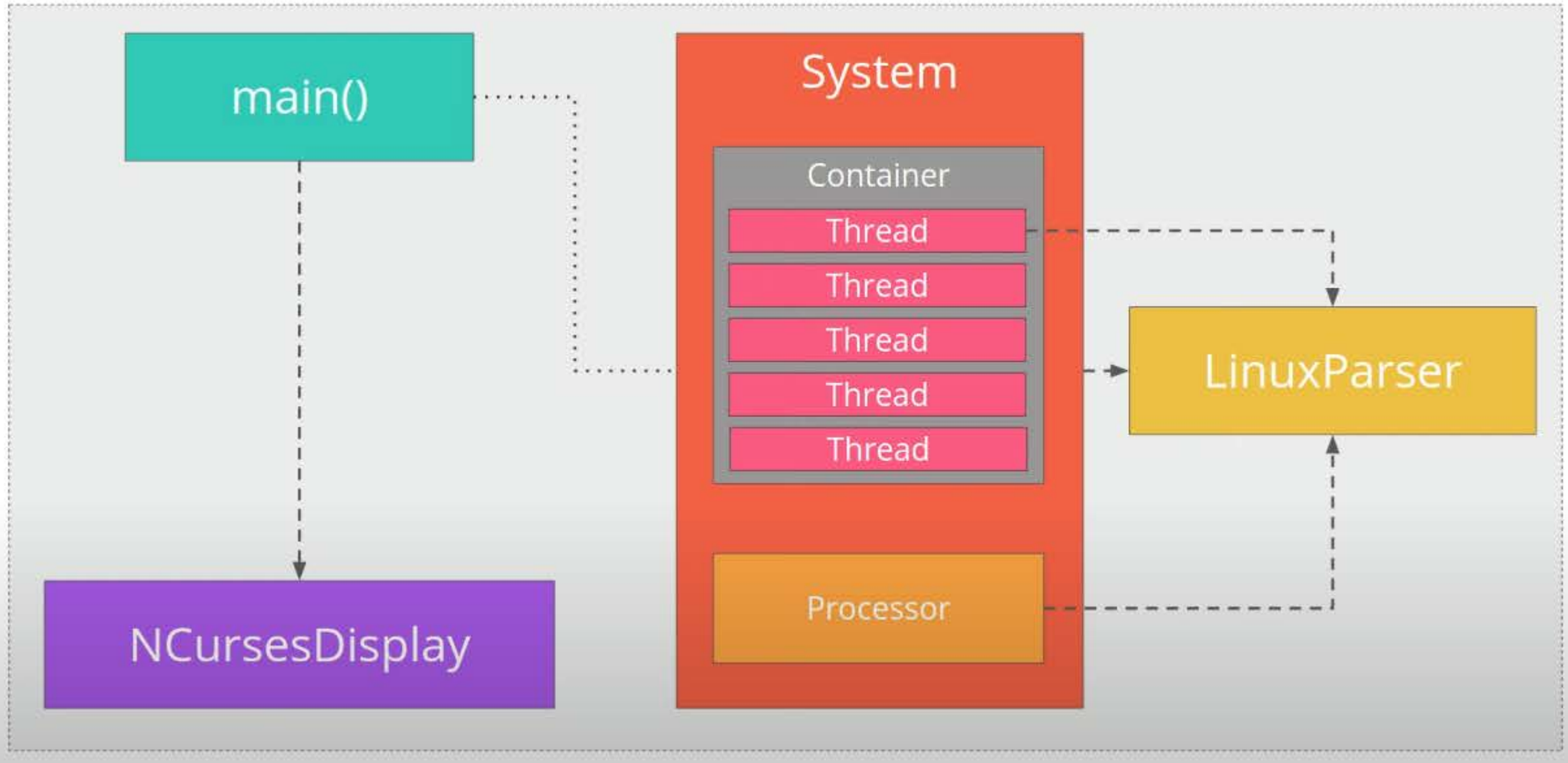


or the computer itself you might say.

1:09 / 2:54

Structure

Watch later Share

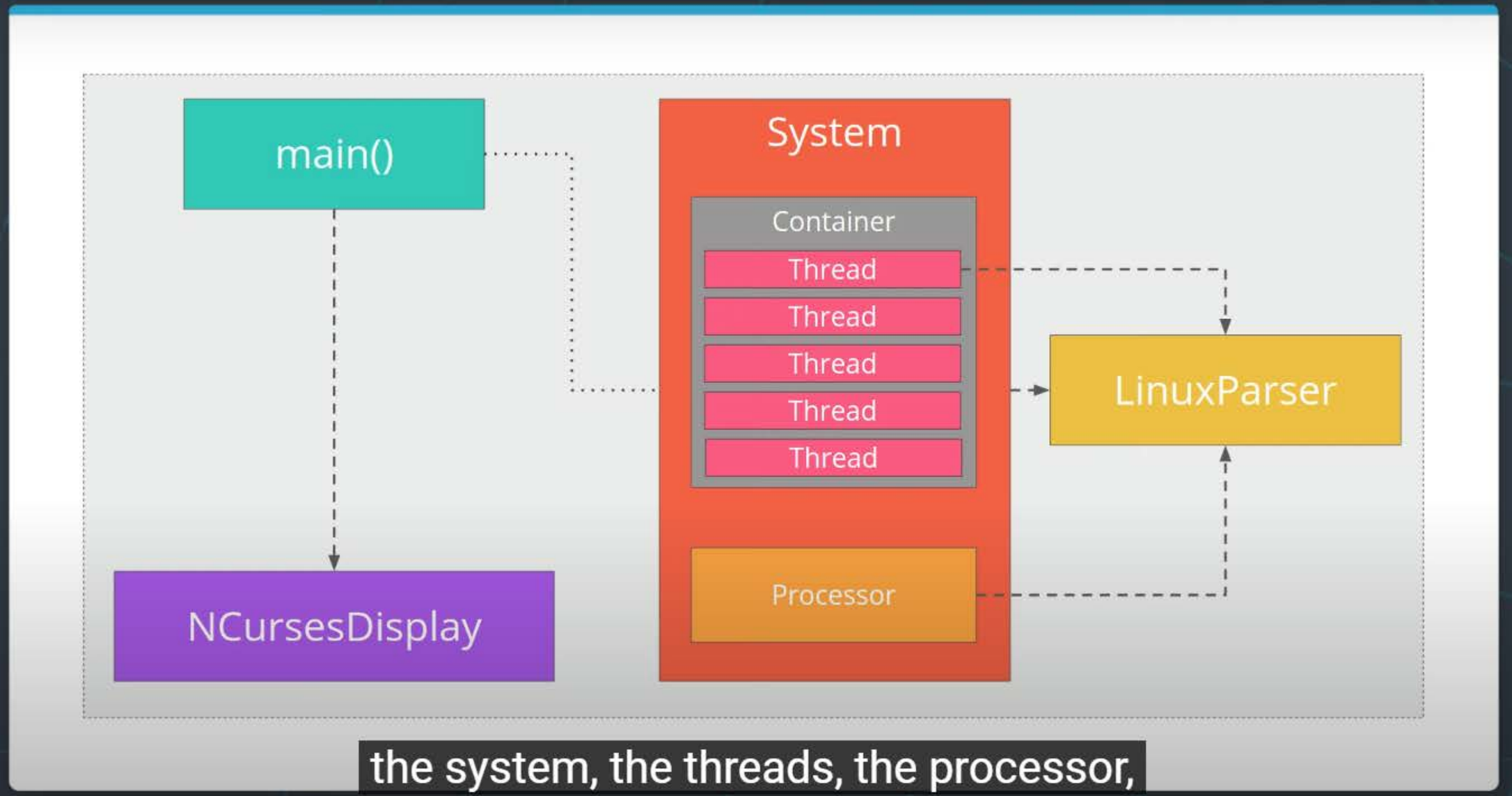


All of these different objects,

1:35 / 2:54

Structure

Watch later Share

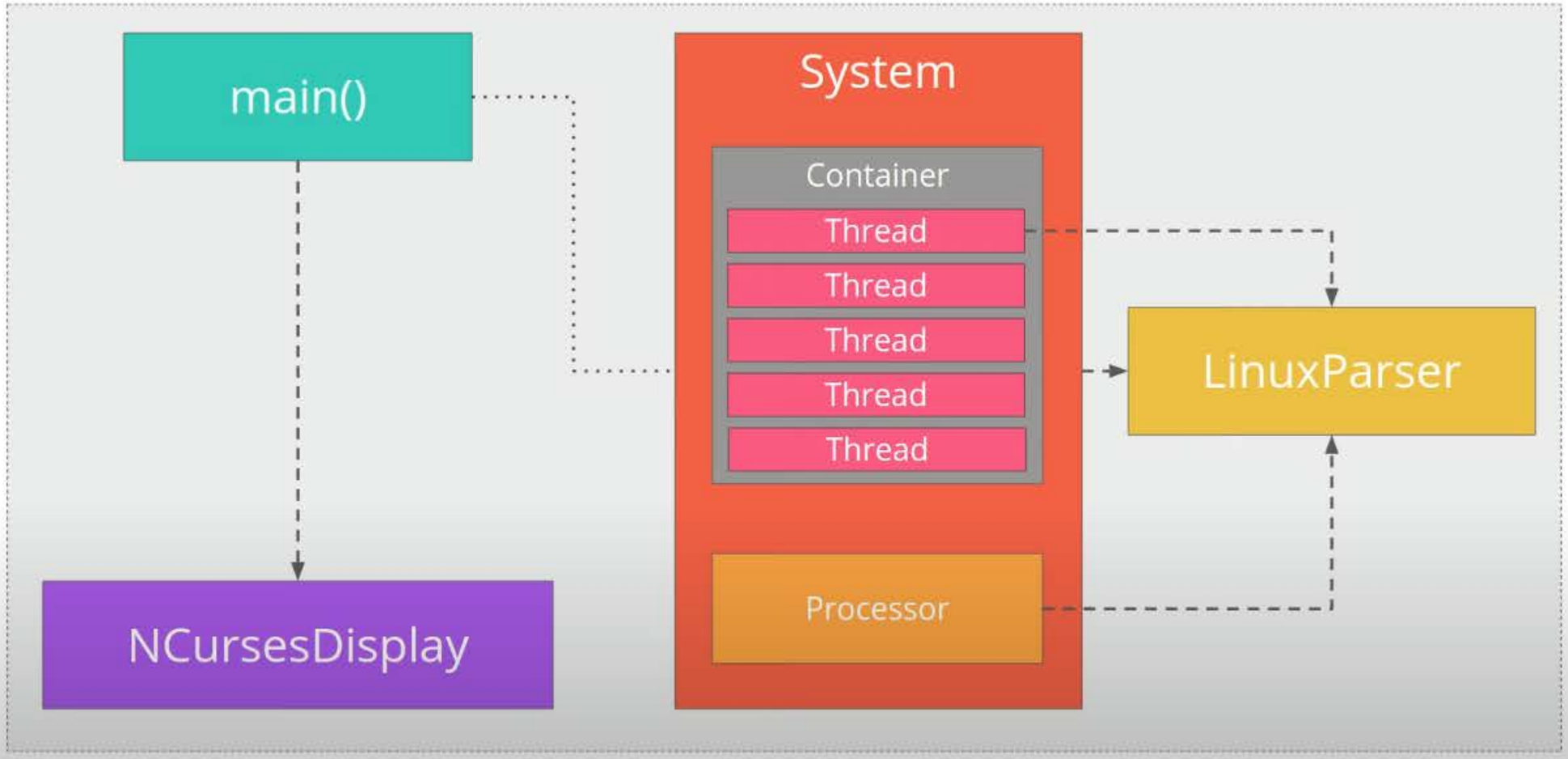


the system, the threads, the processor,

1:36 / 2:54

Structure

Watch later Share

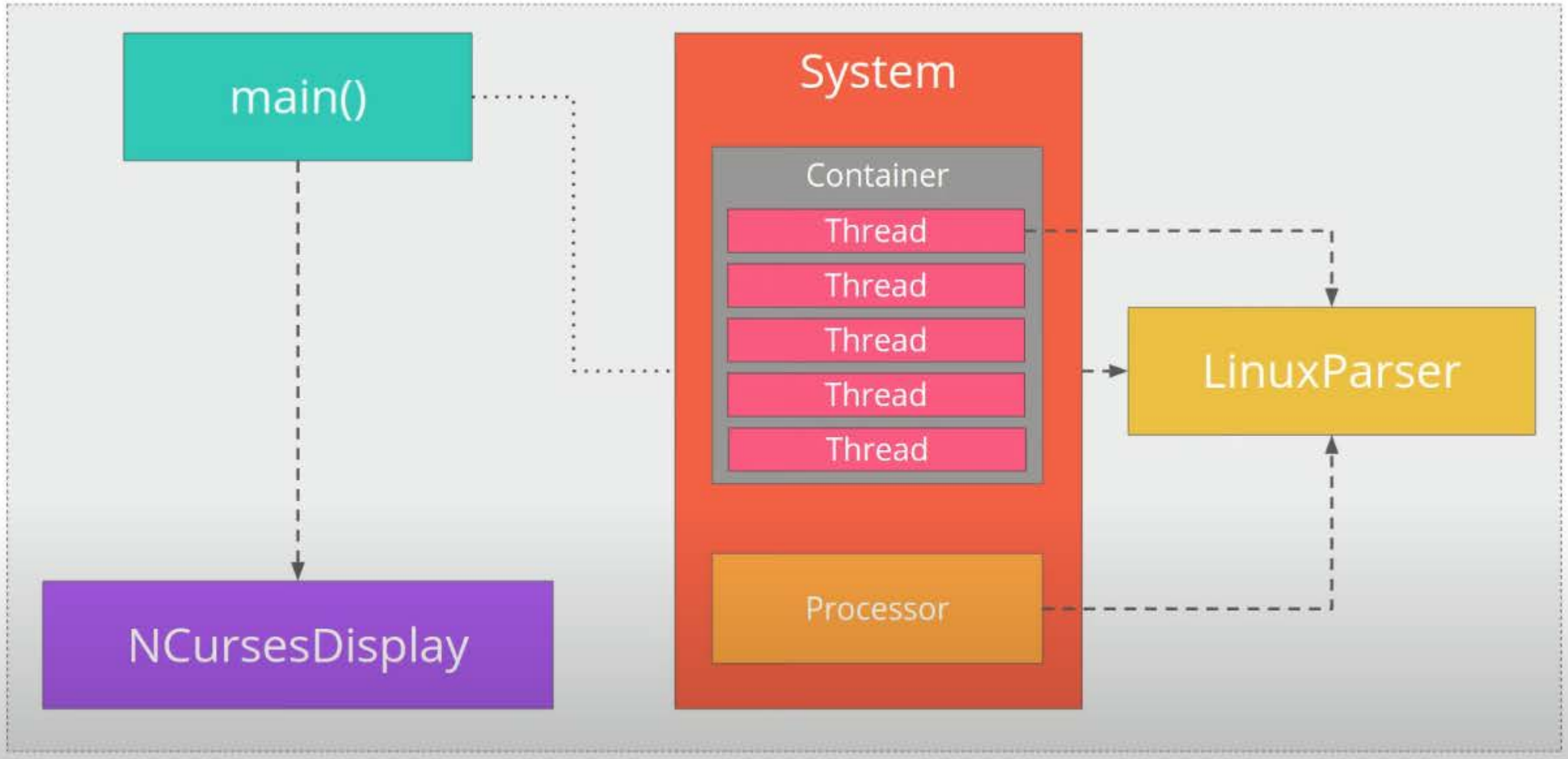


they all call these functions that exist in

1:39 / 2:54

Structure

Watch later Share

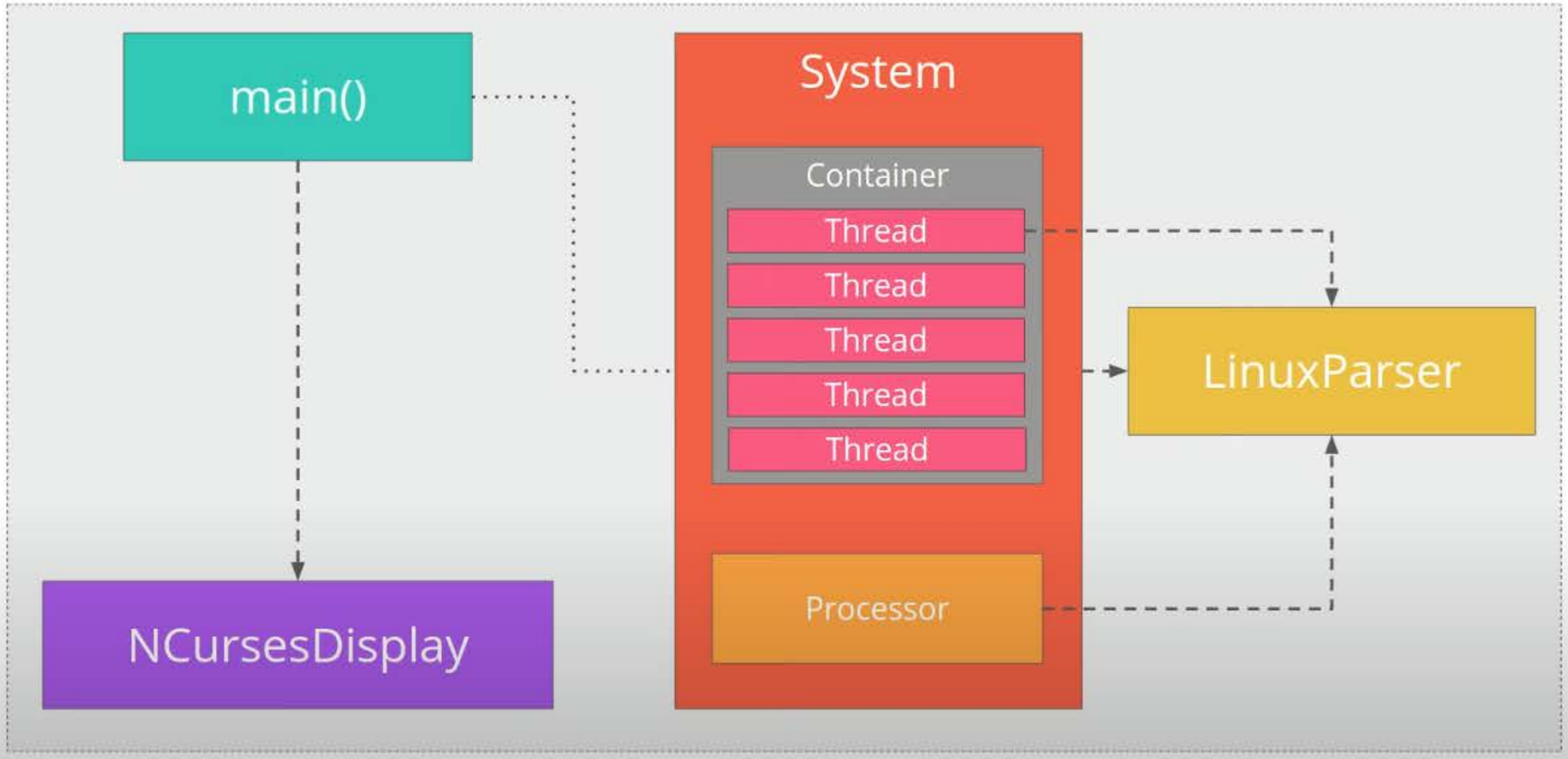


the Linux parser namespace and those functions go out

1:43 / 2:54

Structure

Watch later Share

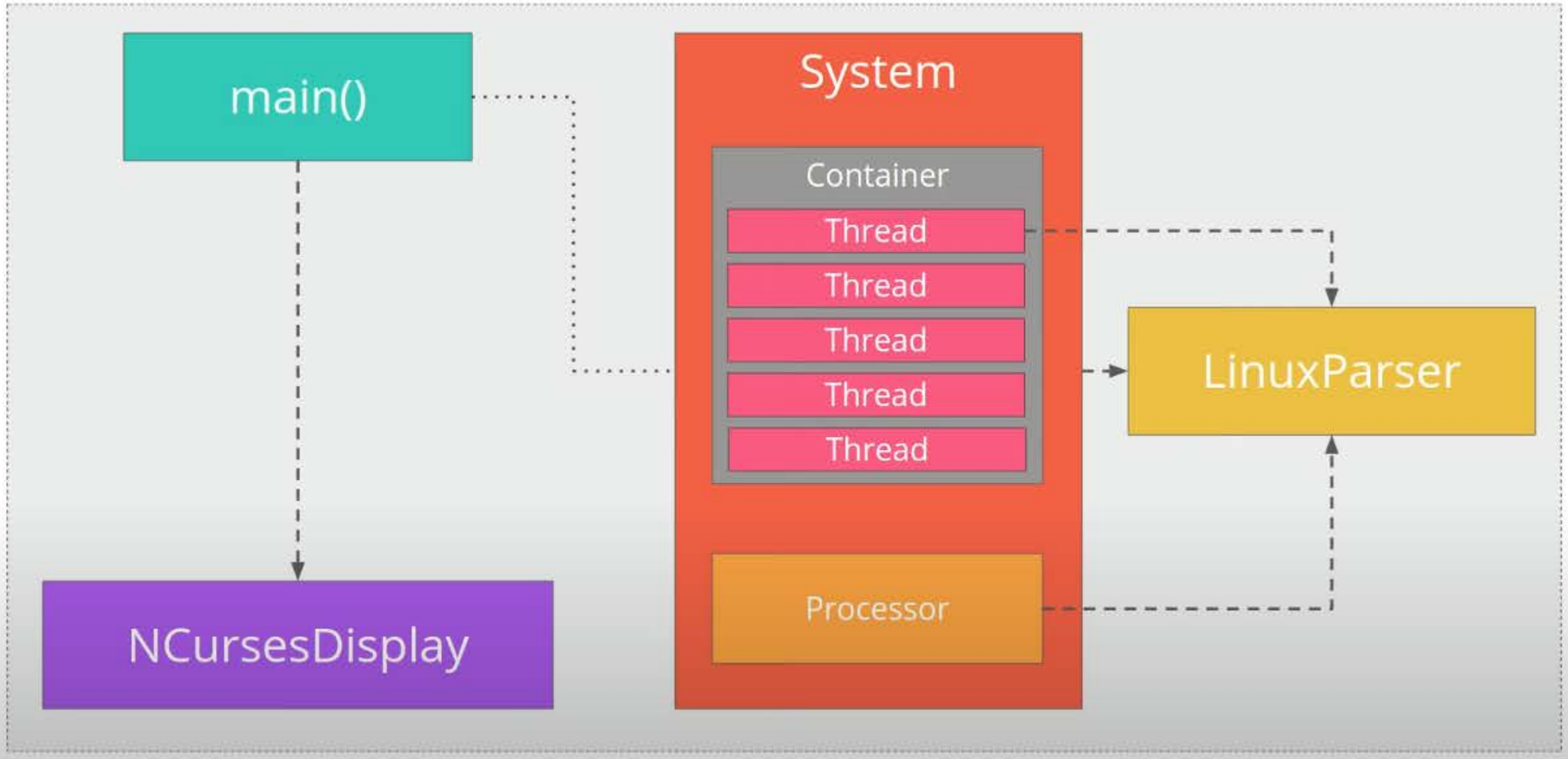


and parse data from the Linux file system

1:47 / 2:54

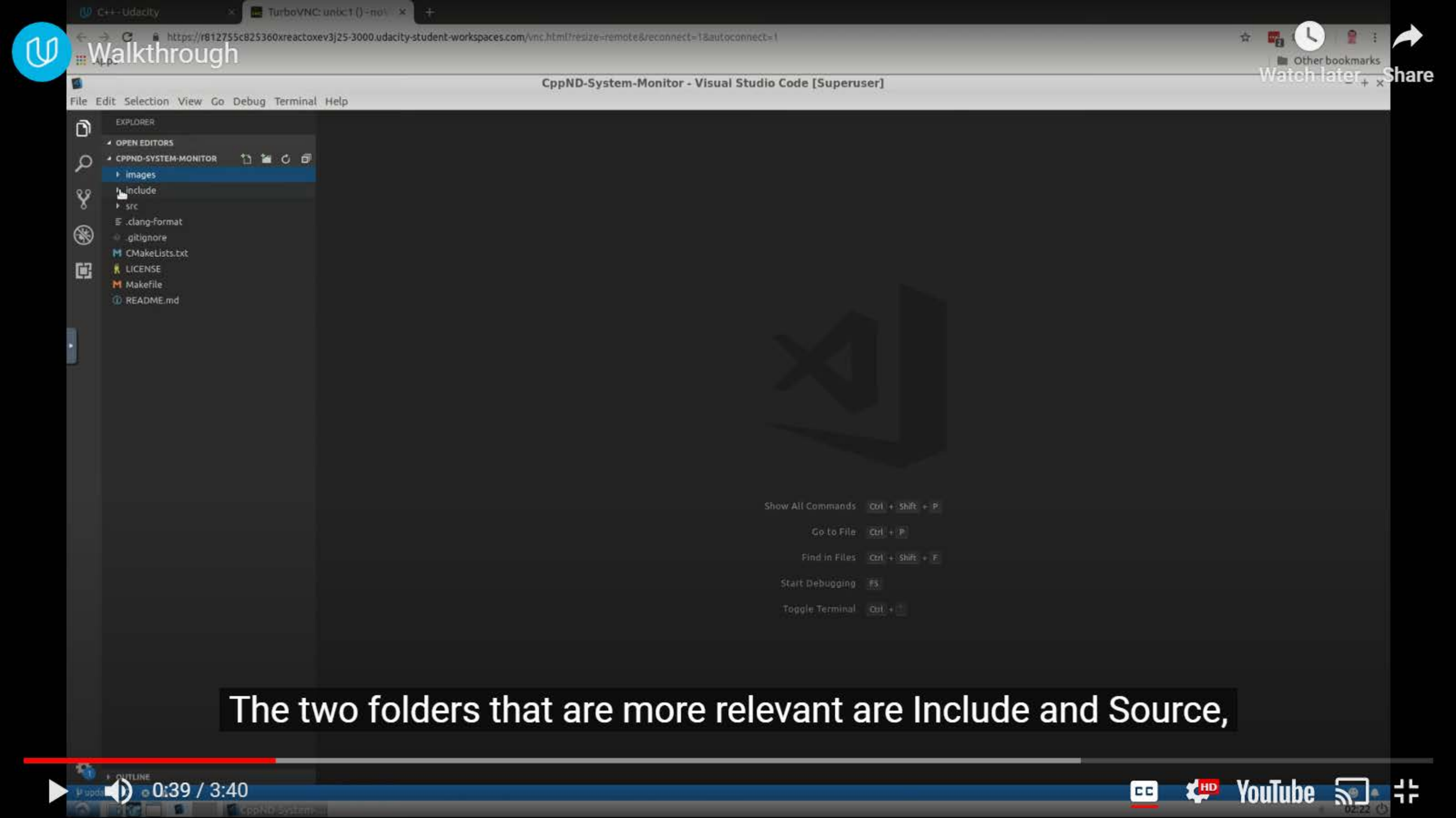
Structure

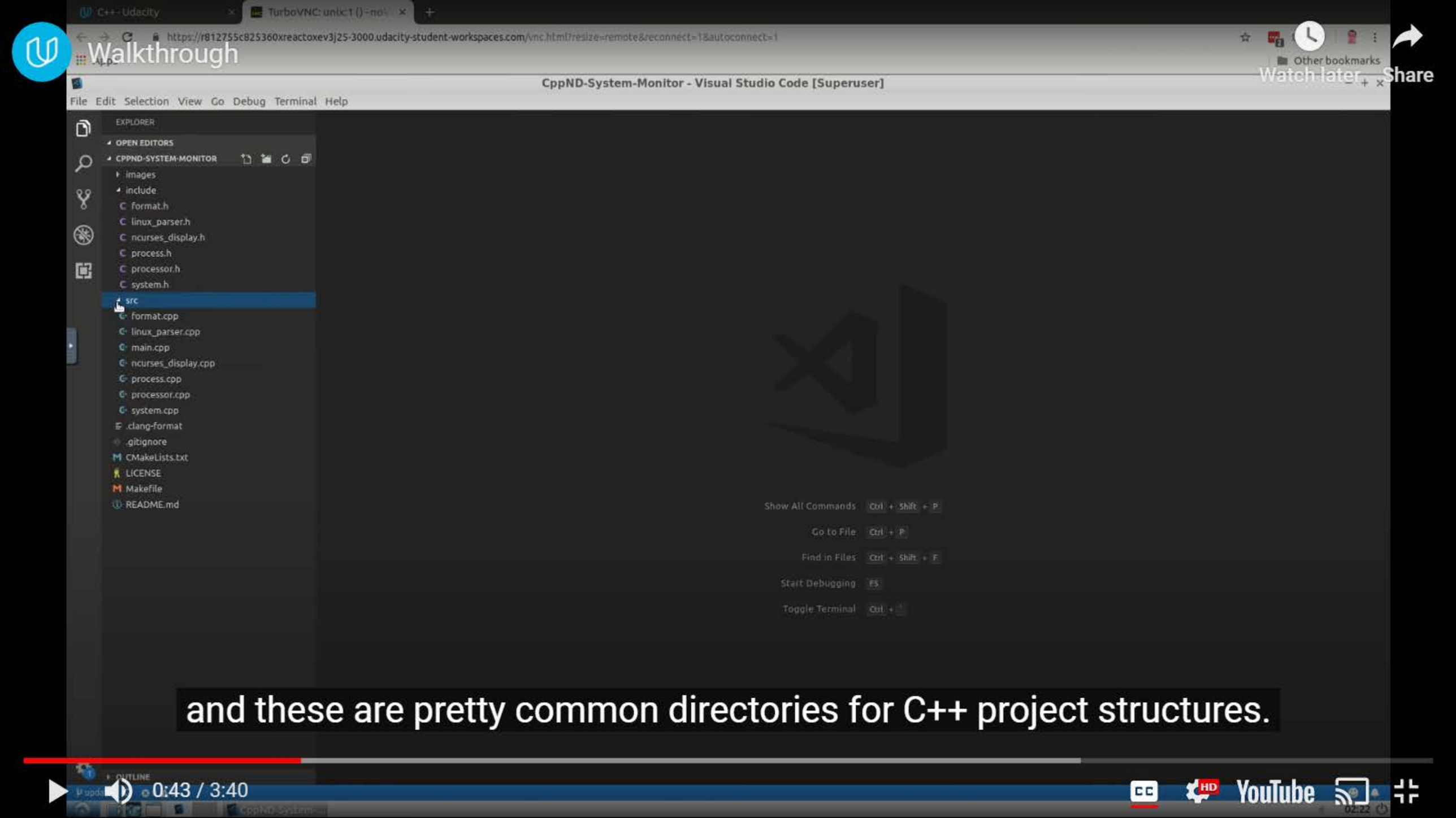
Watch later Share



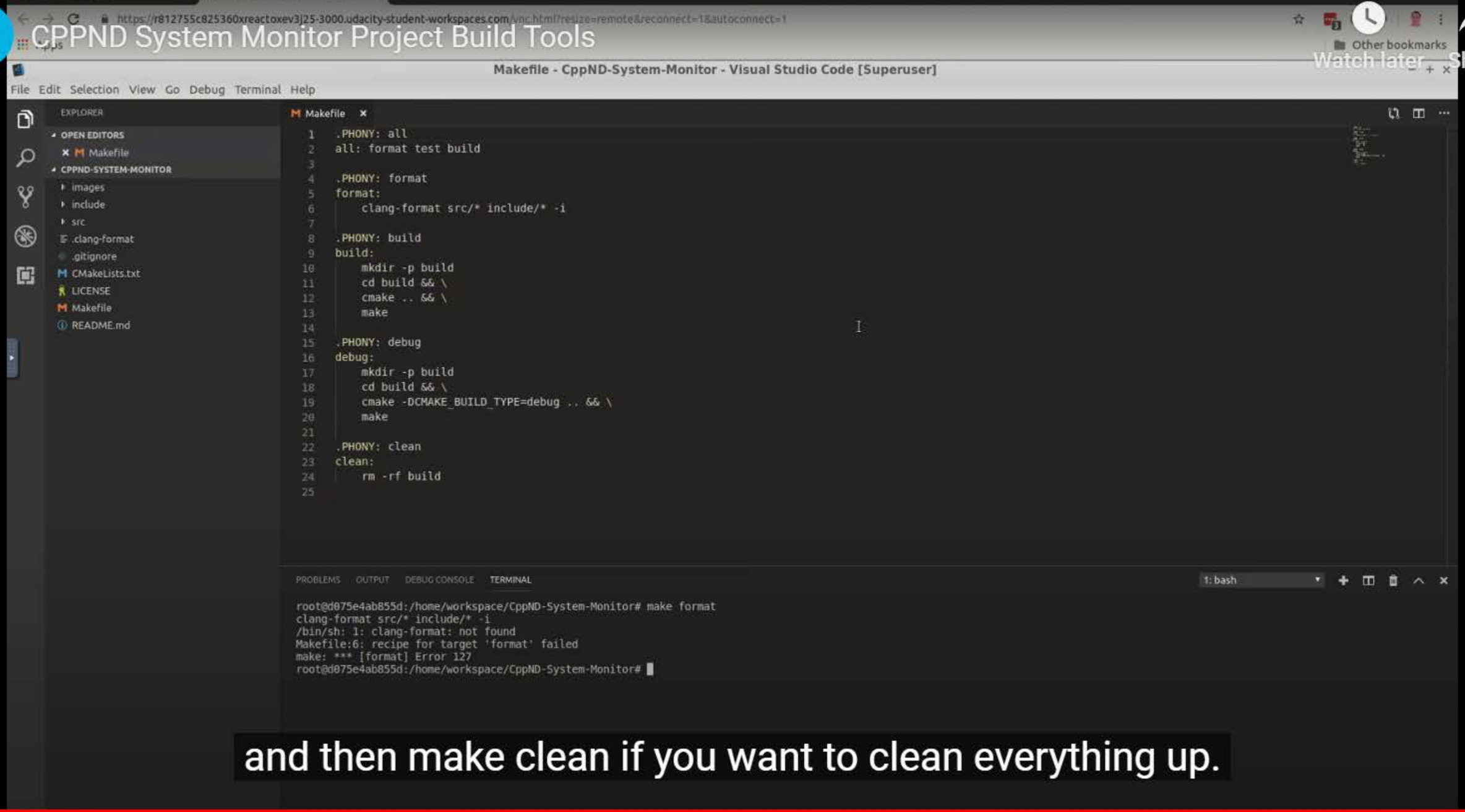
that provide information about the status of all these different objects.

1:51 / 2:54


Walkthrough
Watch later
Share
CppND-System-Monitor - Visual Studio Code [Superuser]
File Edit Selection View Go Debug Terminal Help
EXPLORER
OPEN EDITORS
CPPND-SYSTEM-MONITOR
images
include
src
.clang-format
.gitignore
CMakeLists.txt
LICENSE
Makefile
README.md
Show All Commands Ctrl + Shift + P
Go to File Ctrl + P
Find in Files Ctrl + Shift + F
Start Debugging F5
Toggle Terminal Ctrl +
The two folders that are more relevant are Include and Source,
0:39 / 3:40
YouTube

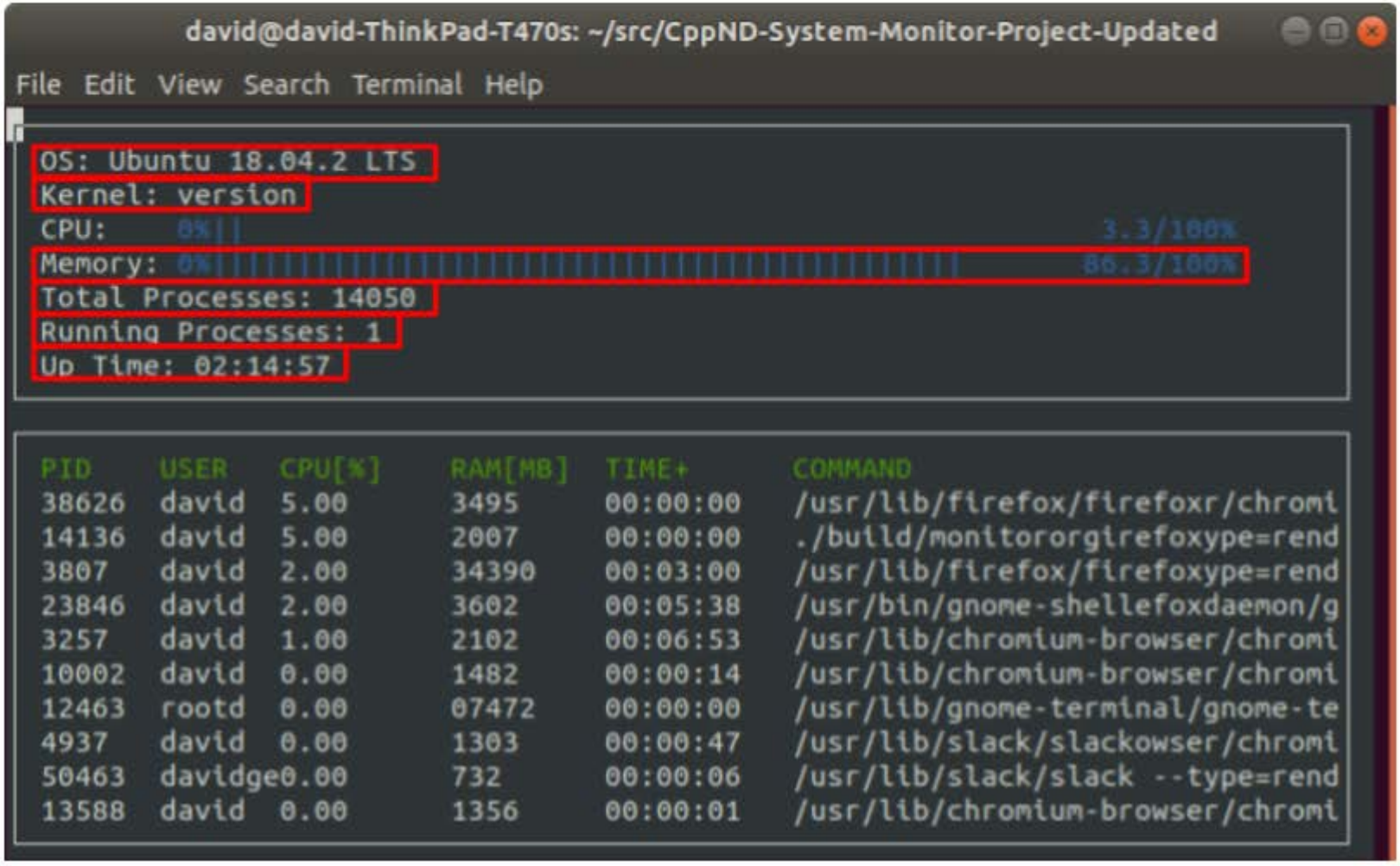

Walkthrough
Watch later
Share
CppND-System-Monitor - Visual Studio Code [Superuser]
File Edit Selection View Go Debug Terminal Help
EXPLORER
OPEN EDITORS
CPPND-SYSTEM-MONITOR
images
include
format.h
linux_parser.h
ncurses_display.h
process.h
processor.h
system.h
src
format.cpp
linux_parser.cpp
main.cpp
ncurses_display.cpp
process.cpp
processor.cpp
system.cpp
.clang-format
.gitignore
CMakeLists.txt
LICENSE
Makefile
README.md
Show All Commands Ctrl + Shift + P
Go to File Ctrl + P
Find in Files Ctrl + Shift + F
Start Debugging F5
Toggle Terminal Ctrl + `
and these are pretty common directories for C++ project structures.
0:43 / 3:40
YouTube

CPPND System Monitor Project Build Tools



and then make clean if you want to clean everything up.

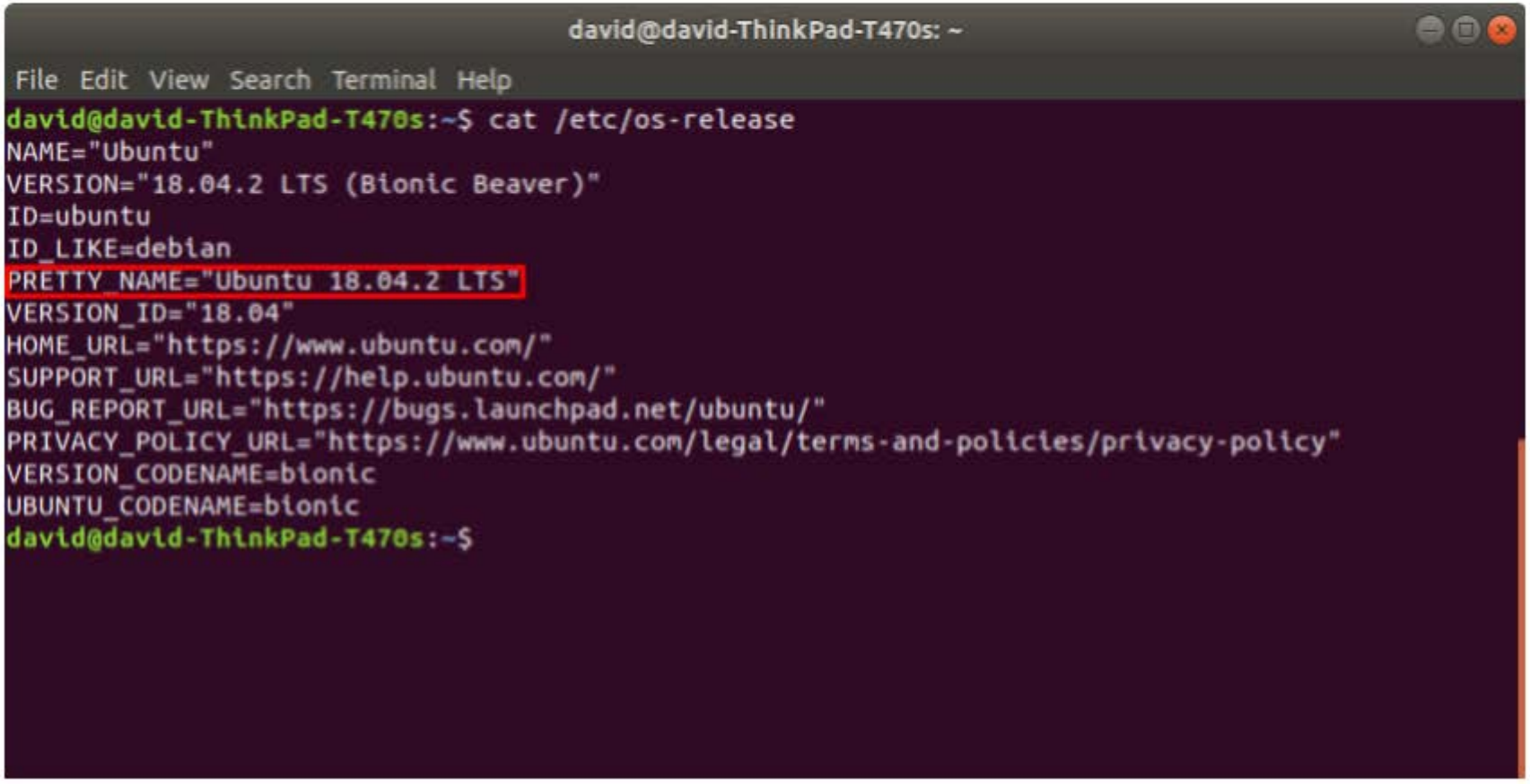
System Data


david@david-ThinkPad-T470s: ~/src/CppND-System-Monitor-Project-Updated
File Edit View Search Terminal Help
OS: Ubuntu 18.04.2 LTS
Kernel: version
CPU: 0%|| 3.3/100%
Memory: 0%|||||||||||||||||||||||||||||||||||||||||||||||||| 86.3/100%
Total Processes: 14050
Running Processes: 1
Up Time: 02:14:57
PID USER CPU[%] RAM[MB] TIME+ COMMAND
38626 david 5.00 3495 00:00:00 /usr/lib/firefox/firefoxr/chromi
14136 david 5.00 2007 00:00:00 ./build/monitororgirefoxype=rend
3807 david 2.00 34390 00:03:00 /usr/lib/firefox/firefoxype=rend
23846 david 2.00 3602 00:05:38 /usr/bin/gnome-shellefoxdaemon/g
3257 david 1.00 2102 00:06:53 /usr/lib/chromium-browser/chromi
10002 david 0.00 1482 00:00:14 /usr/lib/chromium-browser/chromi
12463 rootd 0.00 07472 00:00:00 /usr/lib/gnome-terminal/gnome-te
4937 david 0.00 1303 00:00:47 /usr/lib/slack/slackowser/chromi
50463 davidge0.00 732 00:00:06 /usr/lib/slack/slack --type=rend
13588 david 0.00 1356 00:00:01 /usr/lib/chromium-browser/chromi

Linux stores a lot of system data in files within the `/proc` directory. Most of the data that this project requires exists in those files.

Operating System

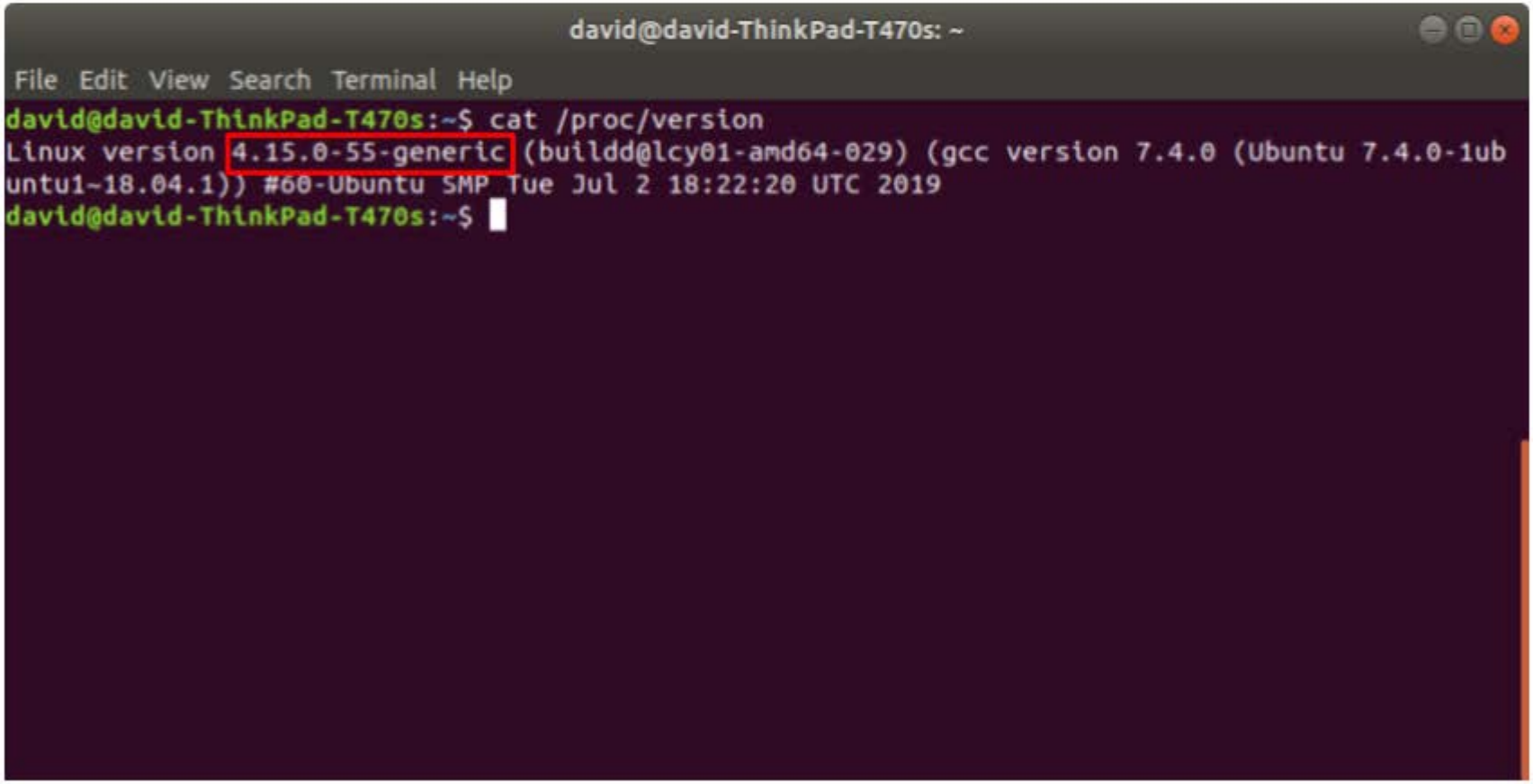
Information about the operating system exists outside of the `/proc` directory, in the `/etc/os-release` file.



There are several strings from which to choose here, but the most obvious is the value specified by "PRETTY_NAME".

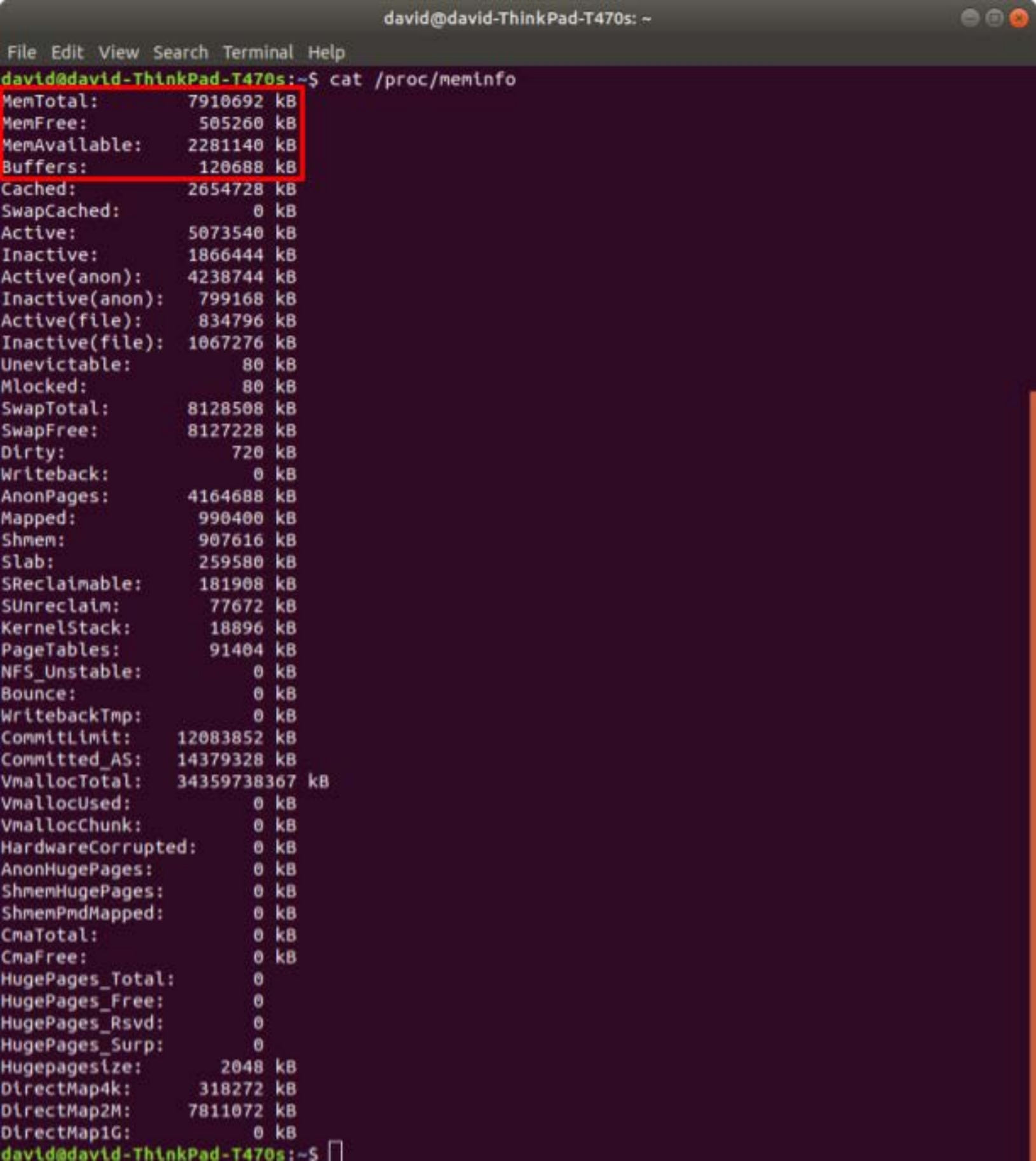
Kernel

Information about the kernel exists `/proc/version` file.



Memory Utilization

Information about memory utilization exists in the `/proc/meminfo` file.


david@david-ThinkPad-T470s: ~
File Edit View Search Terminal Help
david@david-ThinkPad-T470s:~$ cat /proc/meminfo
MemTotal: 7910692 kB
MemFree: 505260 kB
MemAvailable: 2281140 kB
Buffers: 120688 kB
Cached: 2654728 kB
SwapCached: 0 kB
Active: 5073540 kB
Inactive: 1866444 kB
Active(anon): 4238744 kB
Inactive(anon): 799168 kB
Active(file): 834796 kB
Inactive(file): 1067276 kB
Unevictable: 80 kB
Mlocked: 80 kB
SwapTotal: 8128508 kB
SwapFree: 8127228 kB
Dirty: 720 kB
Writeback: 0 kB
AnonPages: 4164688 kB
Mapped: 990400 kB
Shmem: 907616 kB
Slab: 259580 kB
SReclaimable: 181908 kB
SUnreclaim: 77672 kB
KernelStack: 18896 kB
PageTables: 91404 kB
NFS_Unstable: 0 kB
Bounce: 0 kB
WritebackTmp: 0 kB
CommitLimit: 12083852 kB
Committed_AS: 14379328 kB
VmallocTotal: 34359738367 kB
VmallocUsed: 0 kB
VmallocChunk: 0 kB
HardwareCorrupted: 0 kB
AnonHugePages: 0 kB
ShmemHugePages: 0 kB
ShmemPmdMapped: 0 kB
CmaTotal: 0 kB
CmaFree: 0 kB
HugePages_Total: 0
HugePages_Free: 0
HugePages_Rsvd: 0
HugePages_Surp: 0
Hugepagesize: 2048 kB
DirectMap4k: 318272 kB
DirectMap2M: 7811072 kB
DirectMap1G: 0 kB
david@david-ThinkPad-T470s:~$

There are a variety of ways to use this data to calculate memory utilization.

Hisham H. Muhammad, the author of htop, wrote a Stack Overflow answer about how htop calculates memory utilization from the data in `/proc/meminfo`.

Use the formula that makes the most sense to you!

Total Processes

Information about the total number of processes on the system exists in the `/proc/meminfo` file.

david@david-ThinkPad-T470s: ~

File Edit View Search Terminal Help

```
david@david-ThinkPad-T470s:~$ cat /proc/stat
cpu  655593 2758 162862 4287232 4331 0 5766 0 0 0
cpu0 164172 638 38553 1074498 1050 0 1443 0 0 0
cpu1 167790 669 45798 1062618 1039 0 1428 0 0 0
cpu2 160407 720 38843 1075686 1159 0 1899 0 0 0
cpu3 163223 730 39666 1074428 1082 0 996 0 0 0
intr 22516429 8 11 0 0 0 0 0 0 1 6015 0 0 511 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 241194 6442 45216 4
6593 57239 56692 62 348010 1484659 874 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ctxt 70733117
btime 1564258364
processes 11102
procs_running 15
procs_blocked 0
softirq 9494964 1410773 3052424 195 41960 6414 0 30032 2957711 0 1995455
david@david-ThinkPad-T470s:~$
```

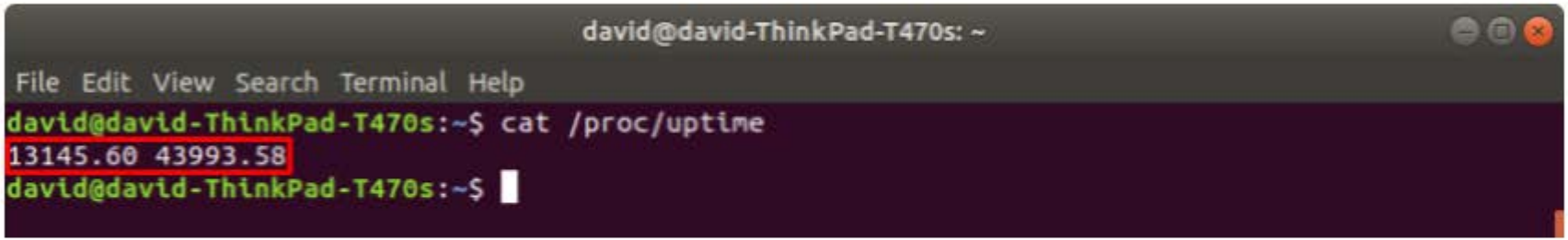
Running Processes

Information about the number of processes on the system that are currently running exists in the `/proc/meminfo` file.

```
david@david-ThinkPad-T470s: ~
File Edit View Search Terminal Help
david@david-ThinkPad-T470s:~$ cat /proc/stat
cpu  655593 2758 162862 4287232 4331 0 5766 0 0 0
cpu0 164172 638 38553 1074498 1050 0 1443 0 0 0
cpu1 167790 669 45798 1062618 1039 0 1428 0 0 0
cpu2 160407 720 38843 1075686 1159 0 1899 0 0 0
cpu3 163223 730 39666 1074428 1082 0 996 0 0 0
intr 22516429 8 11 0 0 0 0 0 0 1 6015 0 0 511 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 241194 6442 45216 4
6593 57239 56692 62 348010 1484659 874 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ctxt 70733117
btime 1564258364
processes 11102
procs_running 15
procs_blocked 0
softirq 9494964 1410773 3052424 195 41960 6414 0 30032 2957711 0 1995455
david@david-ThinkPad-T470s:~$
```

Up Time

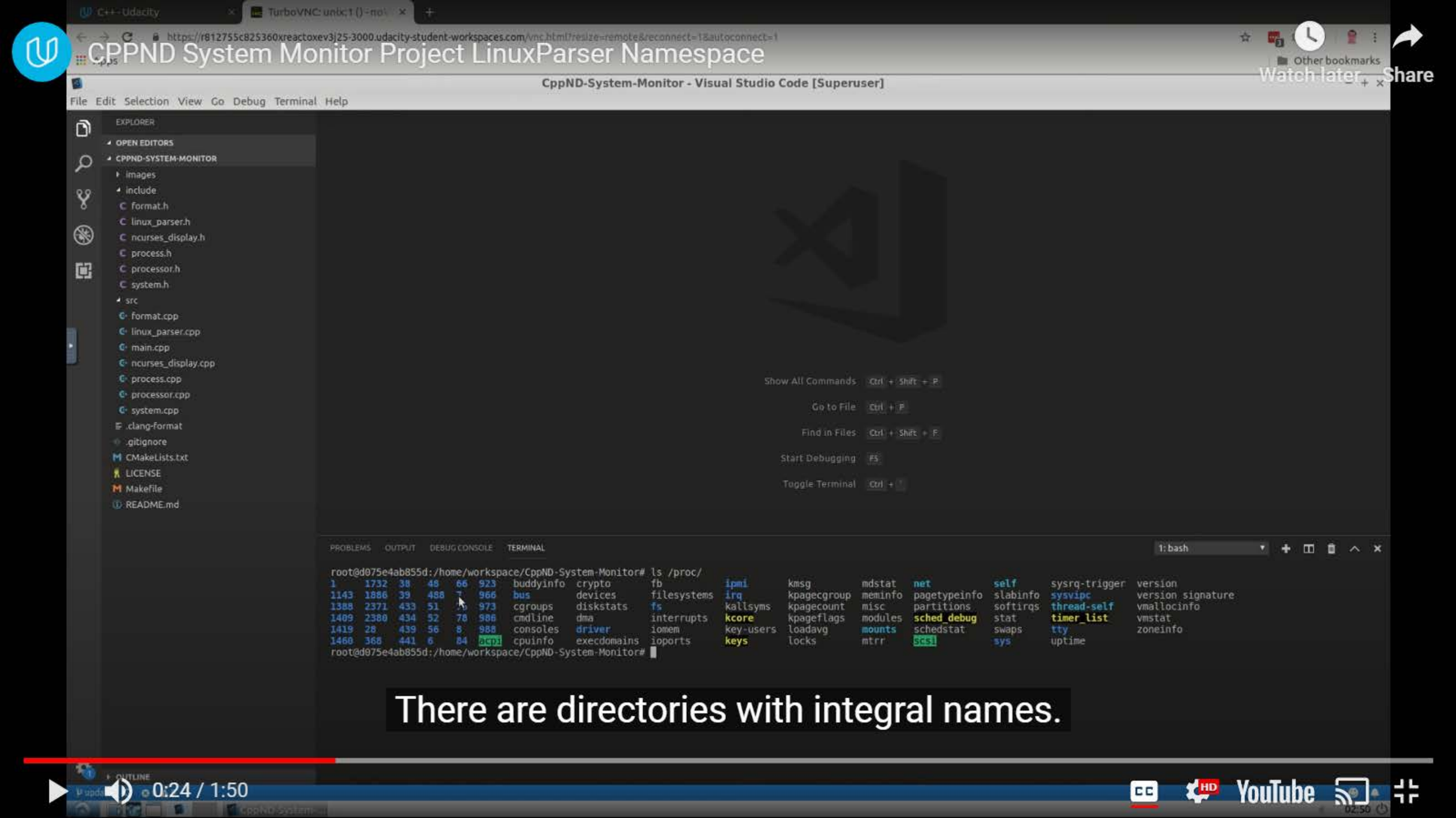
Information about system up time exists in the `/proc/uptime` file.



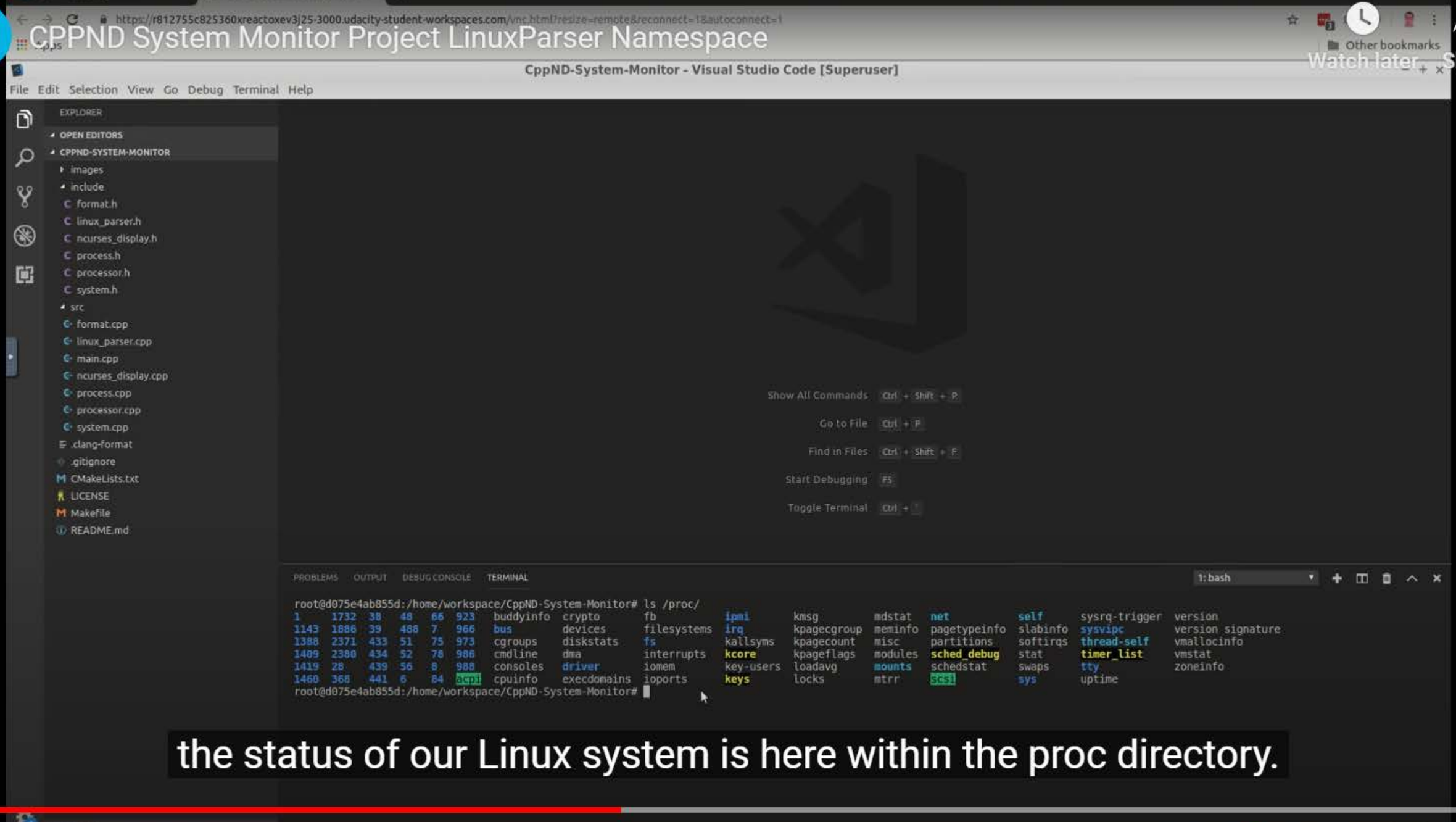
> *This file contains two numbers (values in seconds): the uptime of the system (including time spent in suspend) and the amount of time spent in the idle process.*

From the *man* page for `proc`

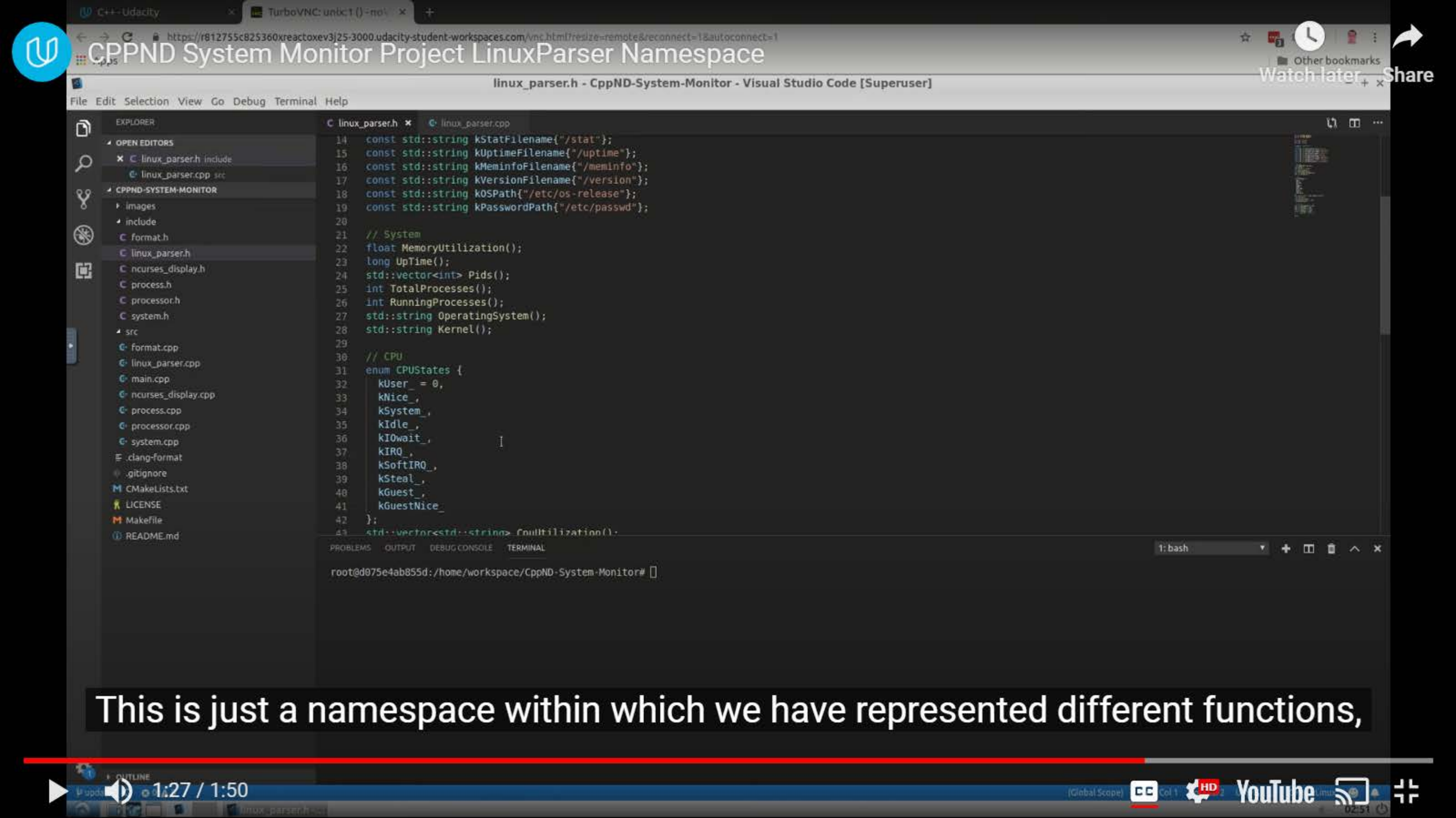
CPPND System Monitor Project LinuxParser Namespace



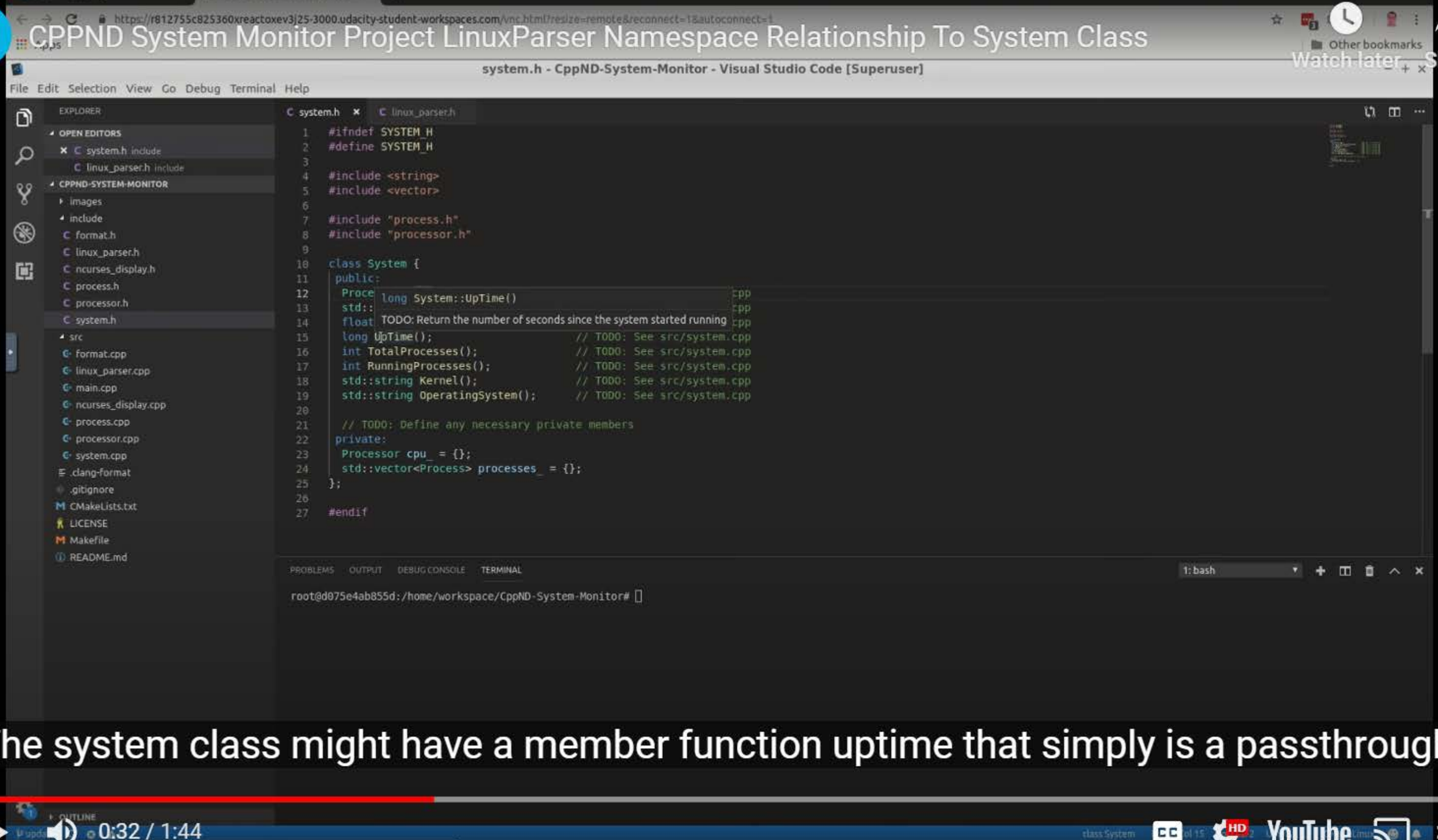
CPPND System Monitor Project LinuxParser Namespace



the status of our Linux system is here within the proc directory.


CPPND System Monitor Project LinuxParser Namespace
Watch later
Share
linux_parser.h - CppND-System-Monitor - Visual Studio Code [Superuser]
File Edit Selection View Go Debug Terminal Help
EXPLORER
OPEN EDITORS
linux_parser.h include
linux_parser.cpp src
CPPND-SYSTEM-MONITOR
images
include
format.h
linux_parser.h
ncurses_display.h
process.h
processor.h
system.h
src
format.cpp
linux_parser.cpp
main.cpp
ncurses_display.cpp
process.cpp
processor.cpp
system.cpp
.clang-format
.gitignore
CMakeLists.txt
LICENSE
Makefile
README.md
const std::string kStatFilename{"/stat"};
const std::string kUptimeFilename{"/uptime"};
const std::string kMeminfoFilename{"/meminfo"};
const std::string kVersionFilename{"/version"};
const std::string kOSPath{"/etc/os-release"};
const std::string kPasswordPath{"/etc/passwd"};
// System
float MemoryUtilization();
long UpTime();
std::vector<int> Pids();
int TotalProcesses();
int RunningProcesses();
std::string OperatingSystem();
std::string Kernel();
// CPU
enum CPUStates {
kUser_ = 0,
kNice_,
kSystem_,
kIdle_,
kIOwait_,
kIRQ_,
kSoftIRQ_,
kSteal_,
kGuest_,
kGuestNice_
};
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
root@d075e4ab855d:/home/workspace/CppND-System-Monitor#
This is just a namespace within which we have represented different functions,
1:27 / 1:50

CPPND System Monitor Project LinuxParser Namespace Relationship To System Class

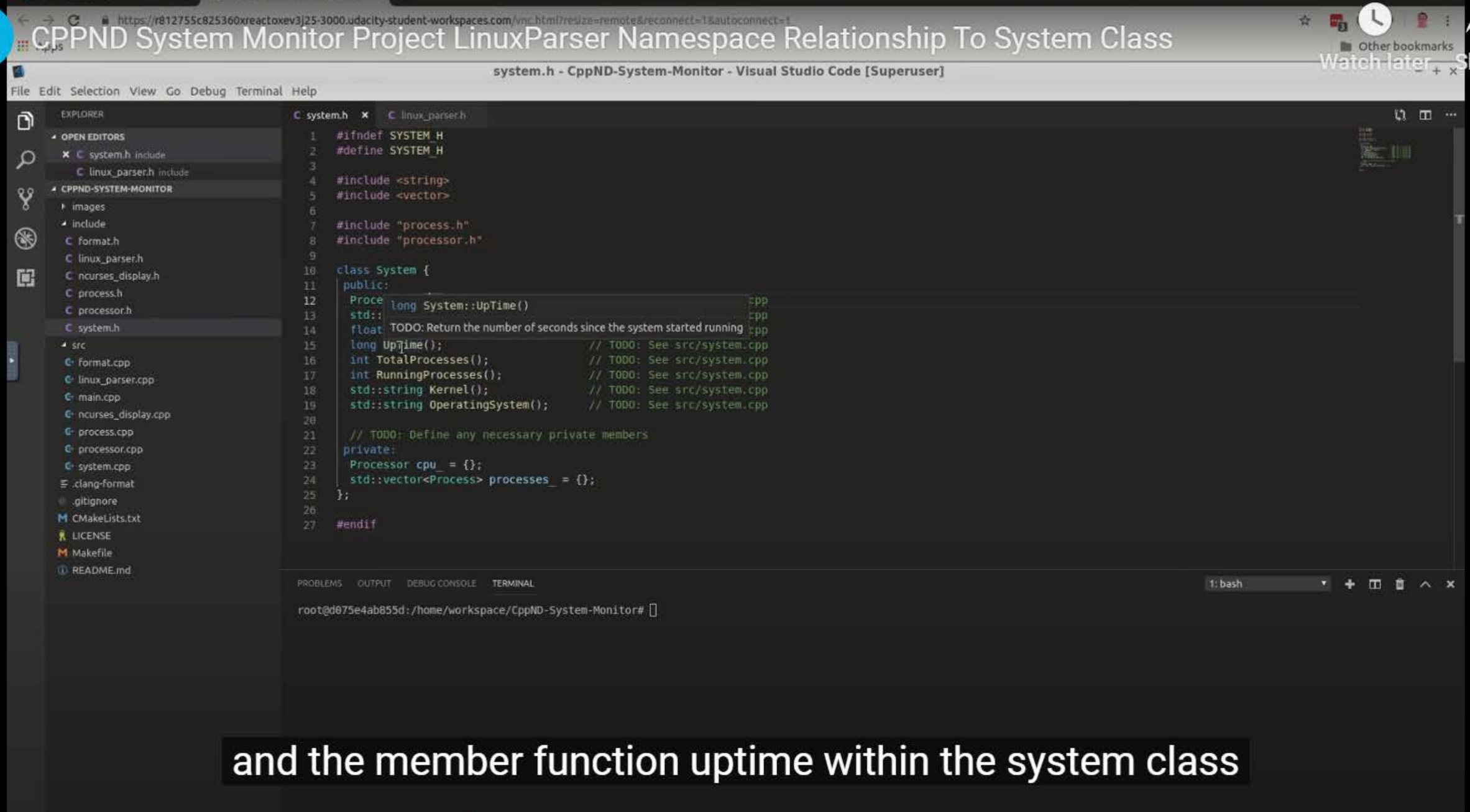



```
#ifndef SYSTEM_H
#define SYSTEM_H

#include <string>
#include <vector>

#include "process.h"
#include "processor.h"

class System {
 public:
  Proce
  std::
  float
  long UpTime();                // TODO: See src/system.cpp
  int TotalProcesses();         // TODO: See src/system.cpp
  int RunningProcesses();       // TODO: See src/system.cpp
  std::string Kernel();         // TODO: See src/system.cpp
  std::string OperatingSystem();// TODO: See src/system.cpp

  // TODO: Define any necessary private members
 private:
  Processor cpu_ = {};
  std::vector<Process> processes_ = {};
};

#endif
```

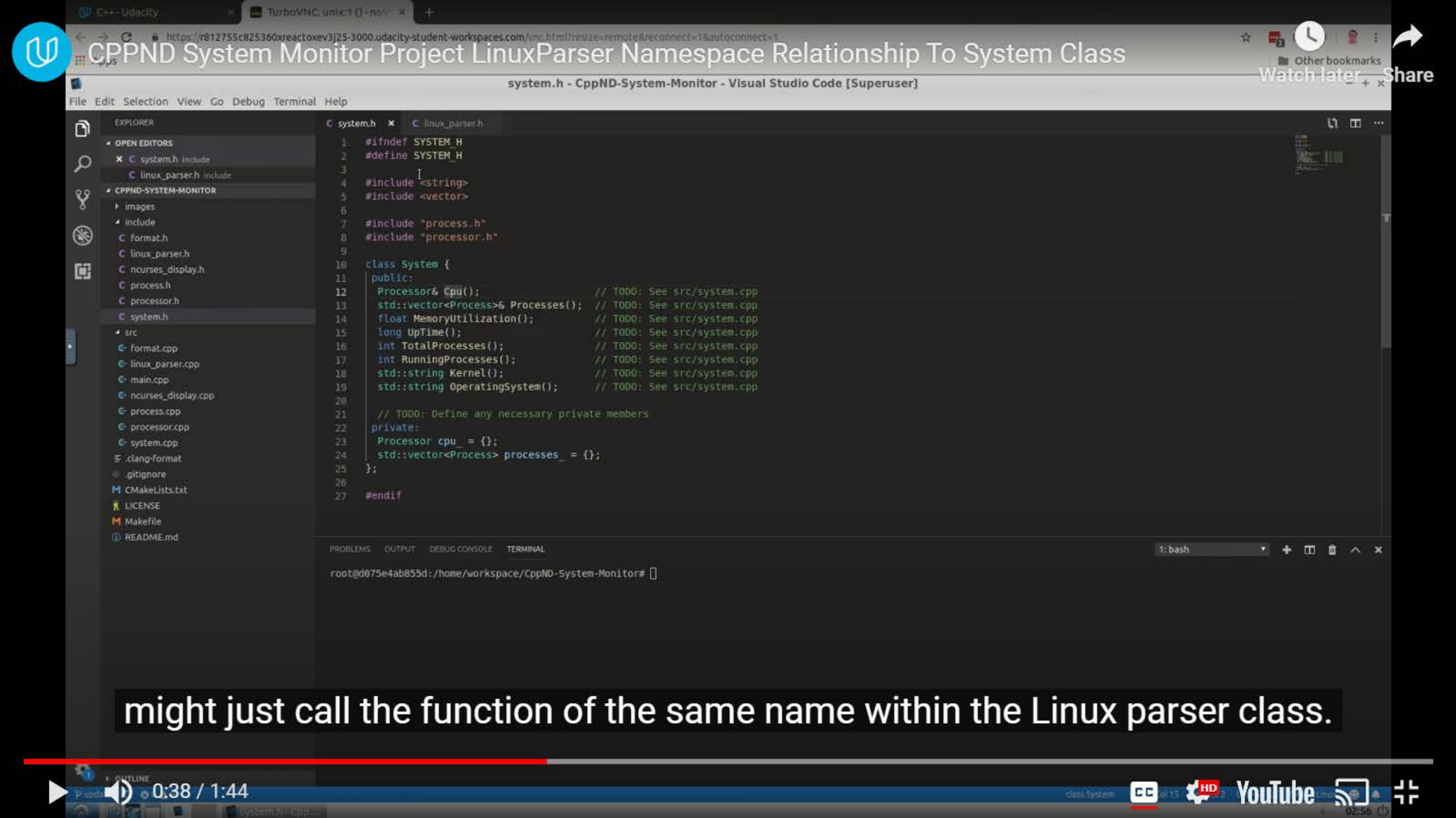
The system class might have a member function uptime that simply is a passthrough,

0:32 / 1:44

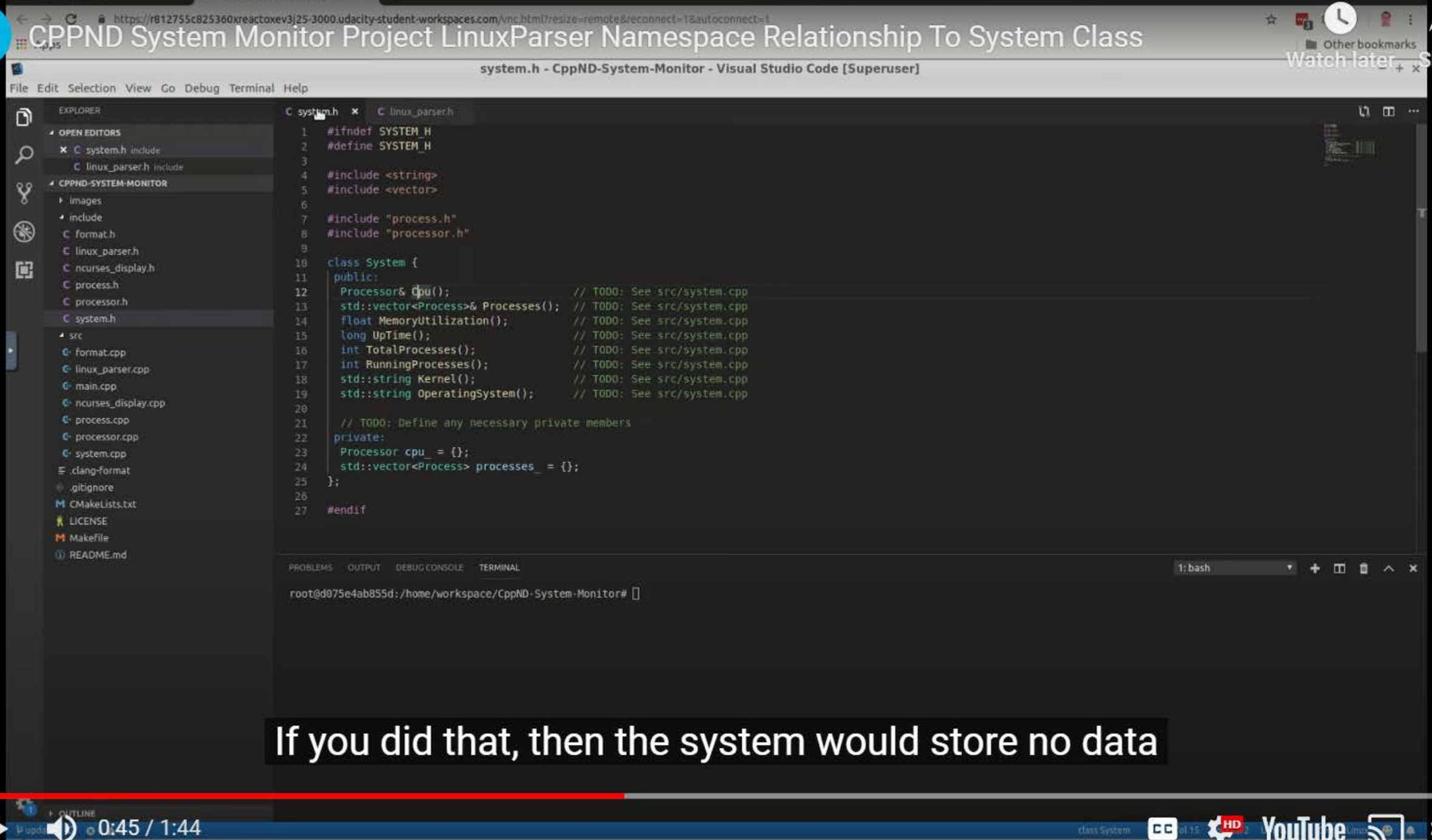
CPPND System Monitor Project LinuxParser Namespace Relationship To System Class



and the member function uptime within the system class

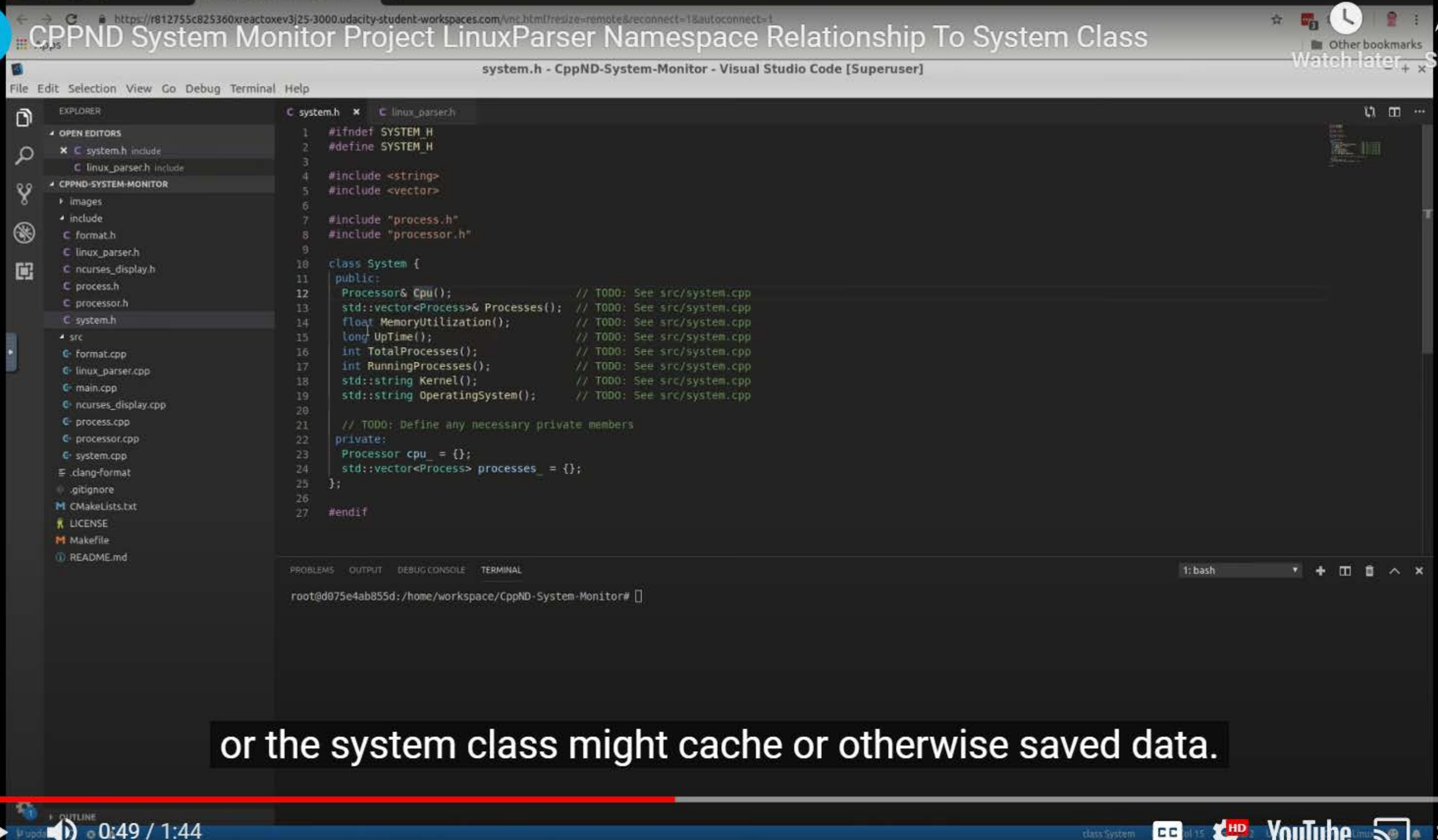

CPPND System Monitor Project LinuxParser Namespace Relationship To System Class
Watch later
Share
system.h - CppND-System-Monitor - Visual Studio Code [Superuser]
File Edit Selection View Go Debug Terminal Help
EXPLORER
OPEN EDITORS
system.h include
linux_parser.h include
CPPND-SYSTEM-MONITOR
images
include
format.h
linux_parser.h
ncurses_display.h
process.h
processor.h
system.h
src
format.cpp
linux_parser.cpp
main.cpp
ncurses_display.cpp
process.cpp
processor.cpp
system.cpp
.clang-format
.gitignore
CMakeLists.txt
LICENSE
Makefile
README.md
system.h
linux_parser.h
#ifndef SYSTEM_H
#define SYSTEM_H
#include <string>
#include <vector>
#include "process.h"
#include "processor.h"
class System {
public:
Processor& Cpu(); // TODO: See src/system.cpp
std::vector<Process>& Processes(); // TODO: See src/system.cpp
float MemoryUtilization(); // TODO: See src/system.cpp
long UpTime(); // TODO: See src/system.cpp
int TotalProcesses(); // TODO: See src/system.cpp
int RunningProcesses(); // TODO: See src/system.cpp
std::string Kernel(); // TODO: See src/system.cpp
std::string OperatingSystem(); // TODO: See src/system.cpp
// TODO: Define any necessary private members
private:
Processor cpu_ = {};
std::vector<Process> processes_ = {};
};
#endif
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
root@d075e4ab855d:/home/workspace/CppND-System-Monitor#
might just call the function of the same name within the Linux parser class.
0:38 / 1:44

CPPND System Monitor Project LinuxParser Namespace Relationship To System Class



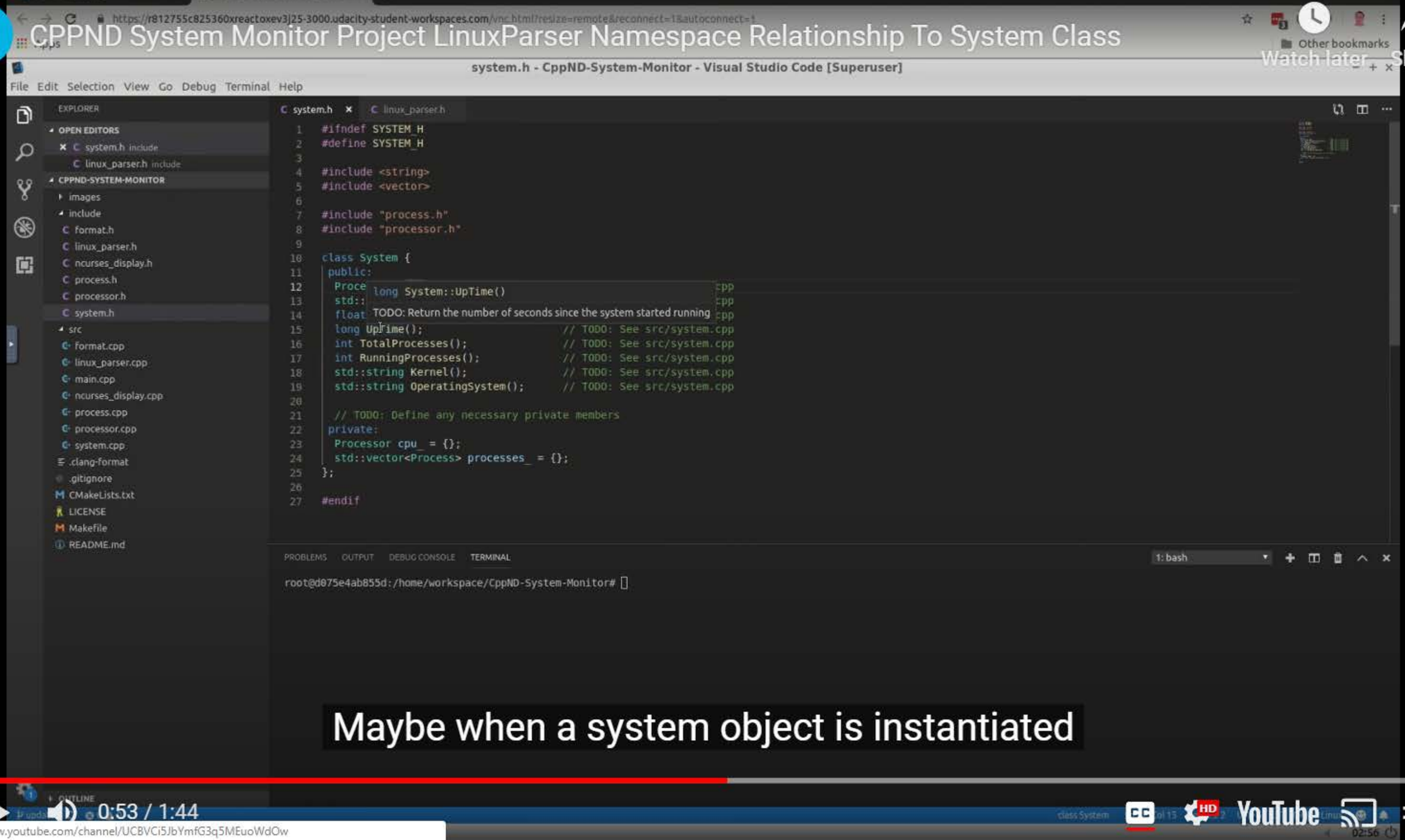
If you did that, then the system would store no data

CPPND System Monitor Project LinuxParser Namespace Relationship To System Class



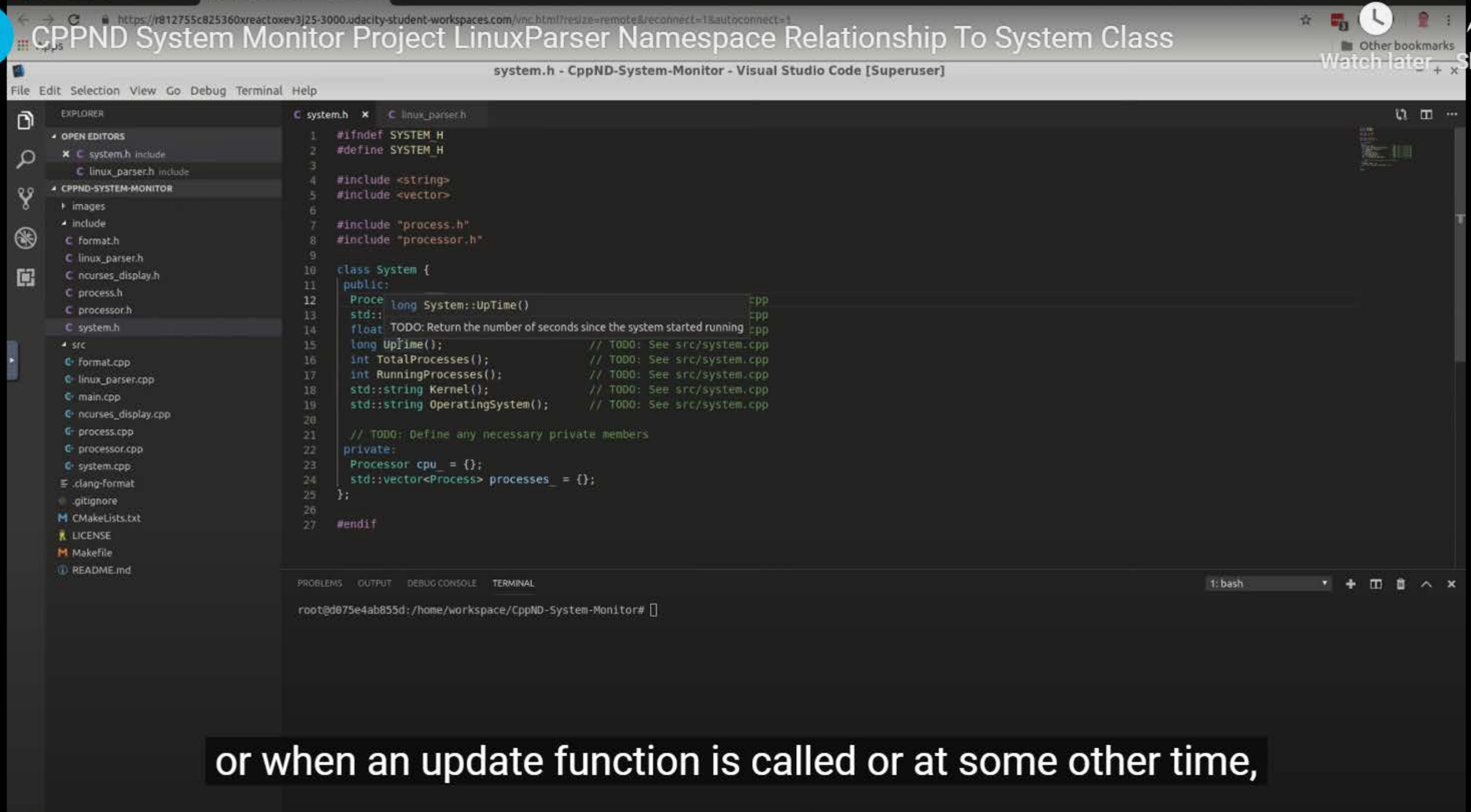
or the system class might cache or otherwise saved data.

CPPND System Monitor Project LinuxParser Namespace Relationship To System Class



Maybe when a system object is instantiated

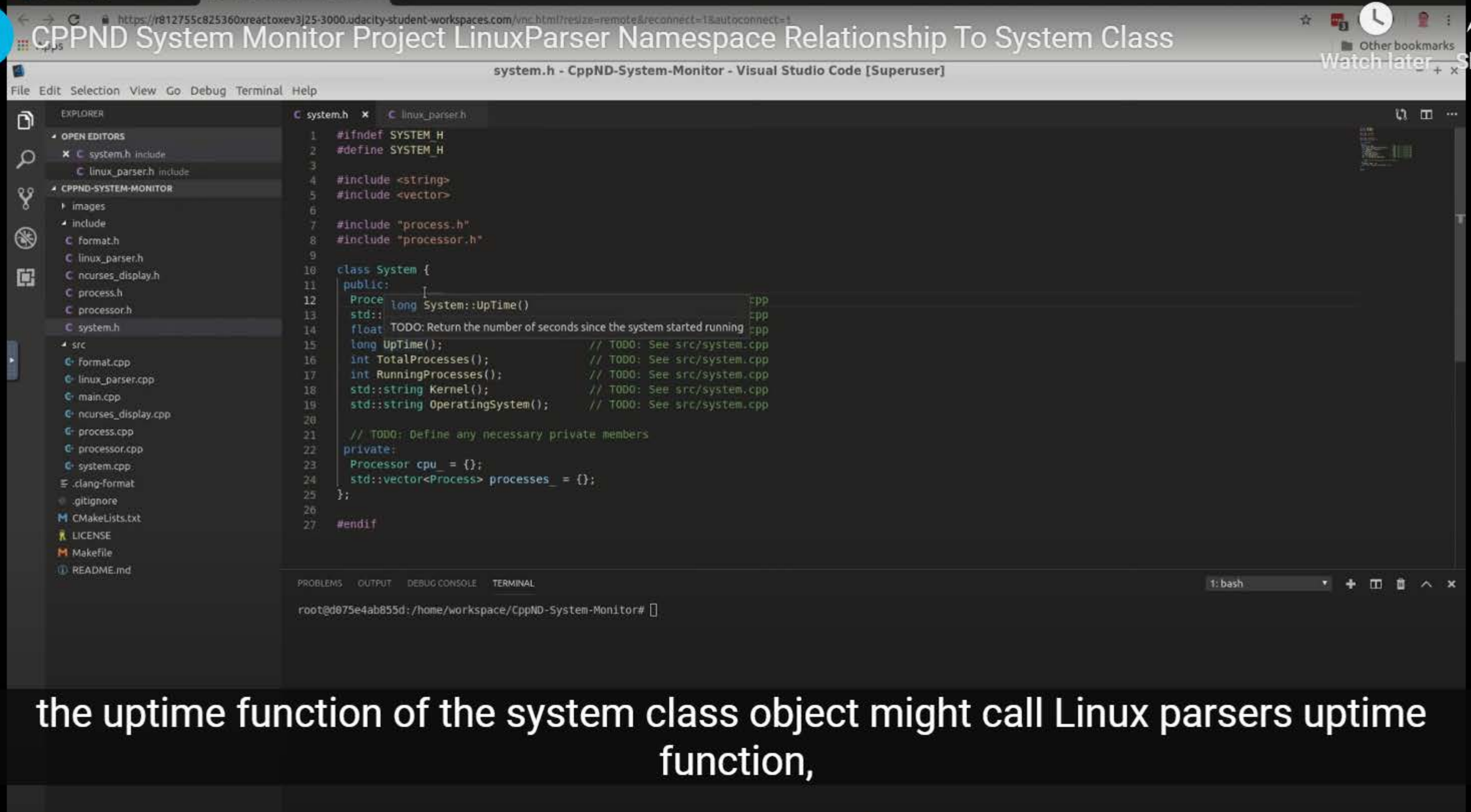
CPPND System Monitor Project LinuxParser Namespace Relationship To System Class



or when an update function is called or at some other time,

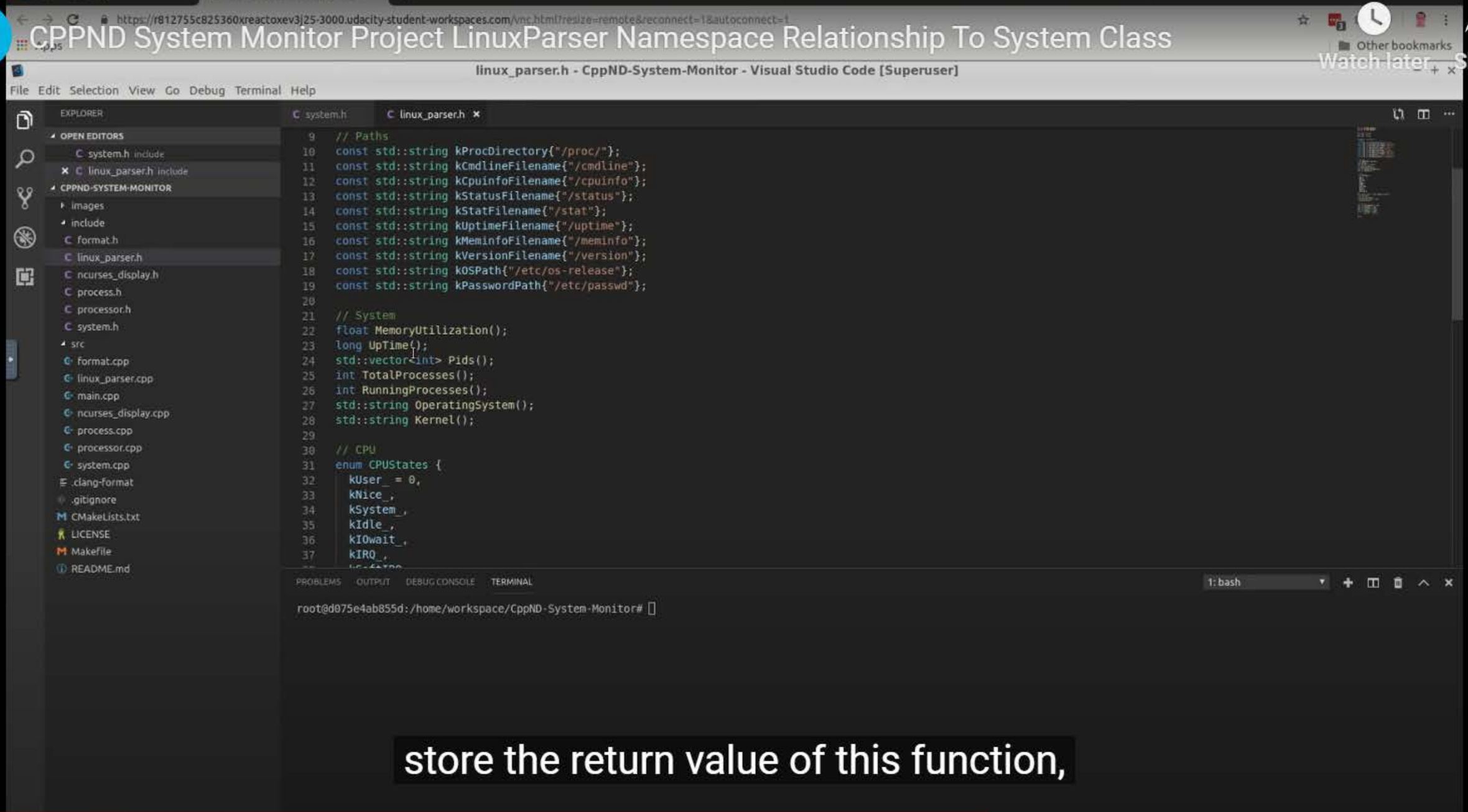
0:56 / 1:44

CPPND System Monitor Project LinuxParser Namespace Relationship To System Class



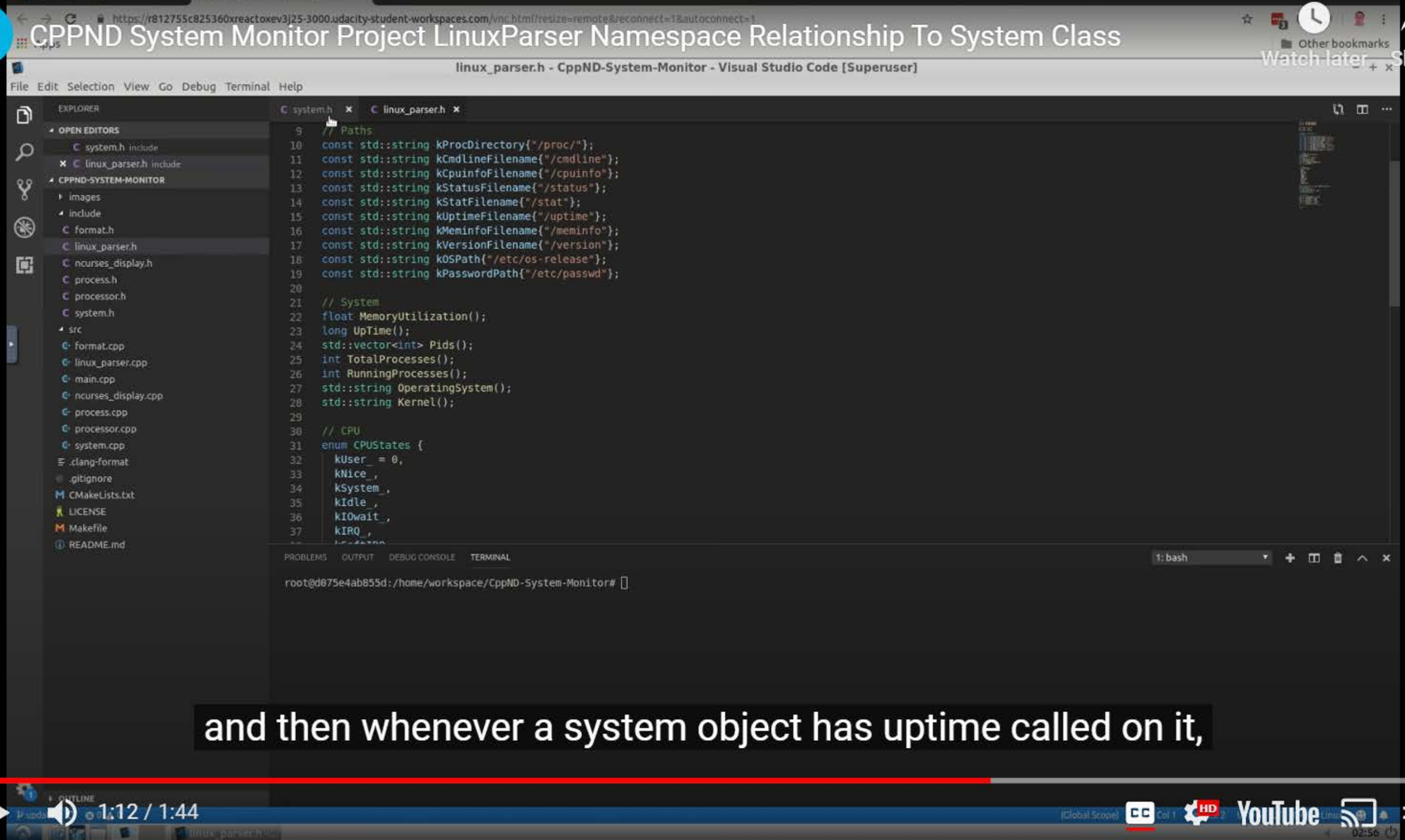
the uptime function of the system class object might call Linux parsers uptime function,

CPPND System Monitor Project LinuxParser Namespace Relationship To System Class



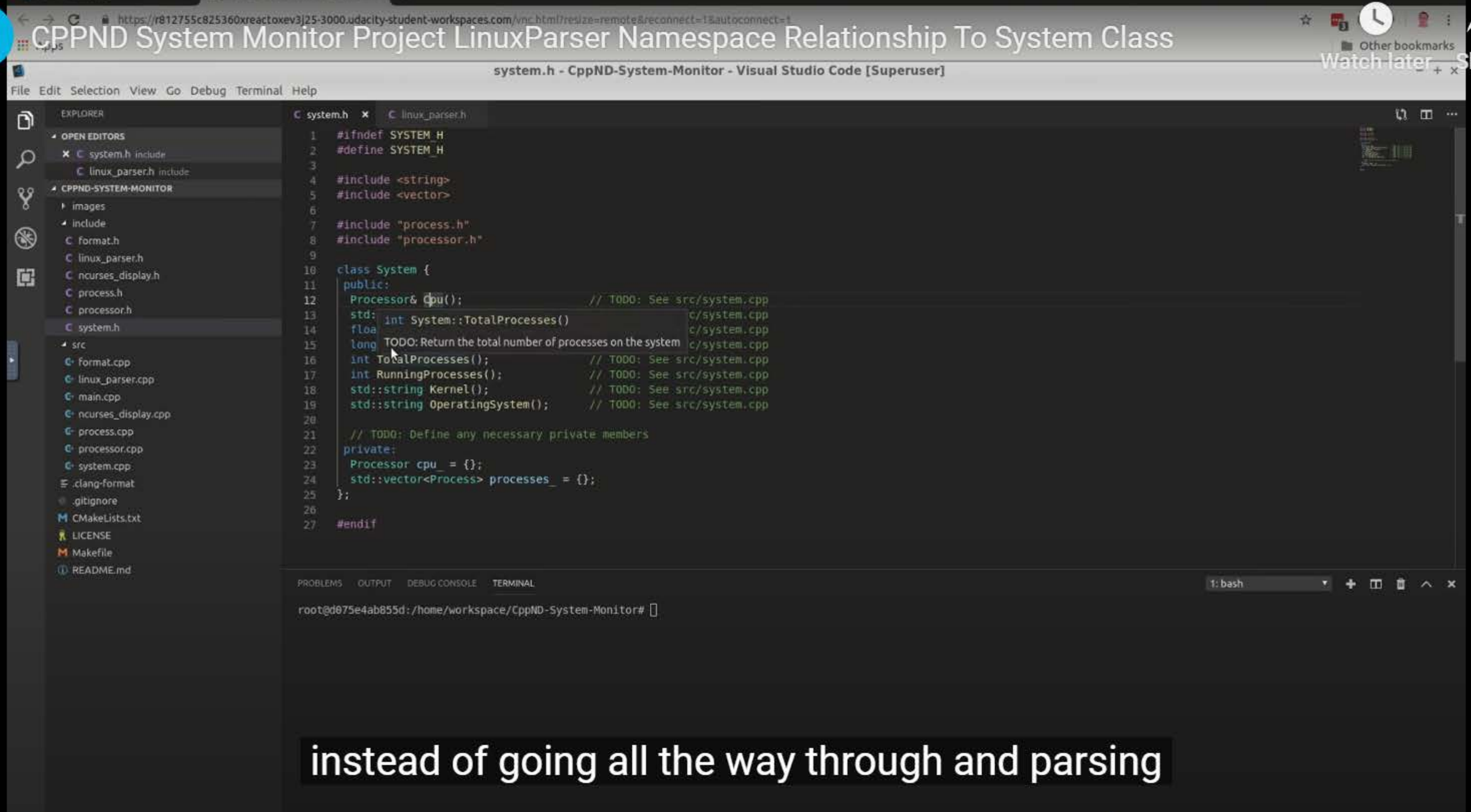
store the return value of this function,

CPPND System Monitor Project LinuxParser Namespace Relationship To System Class

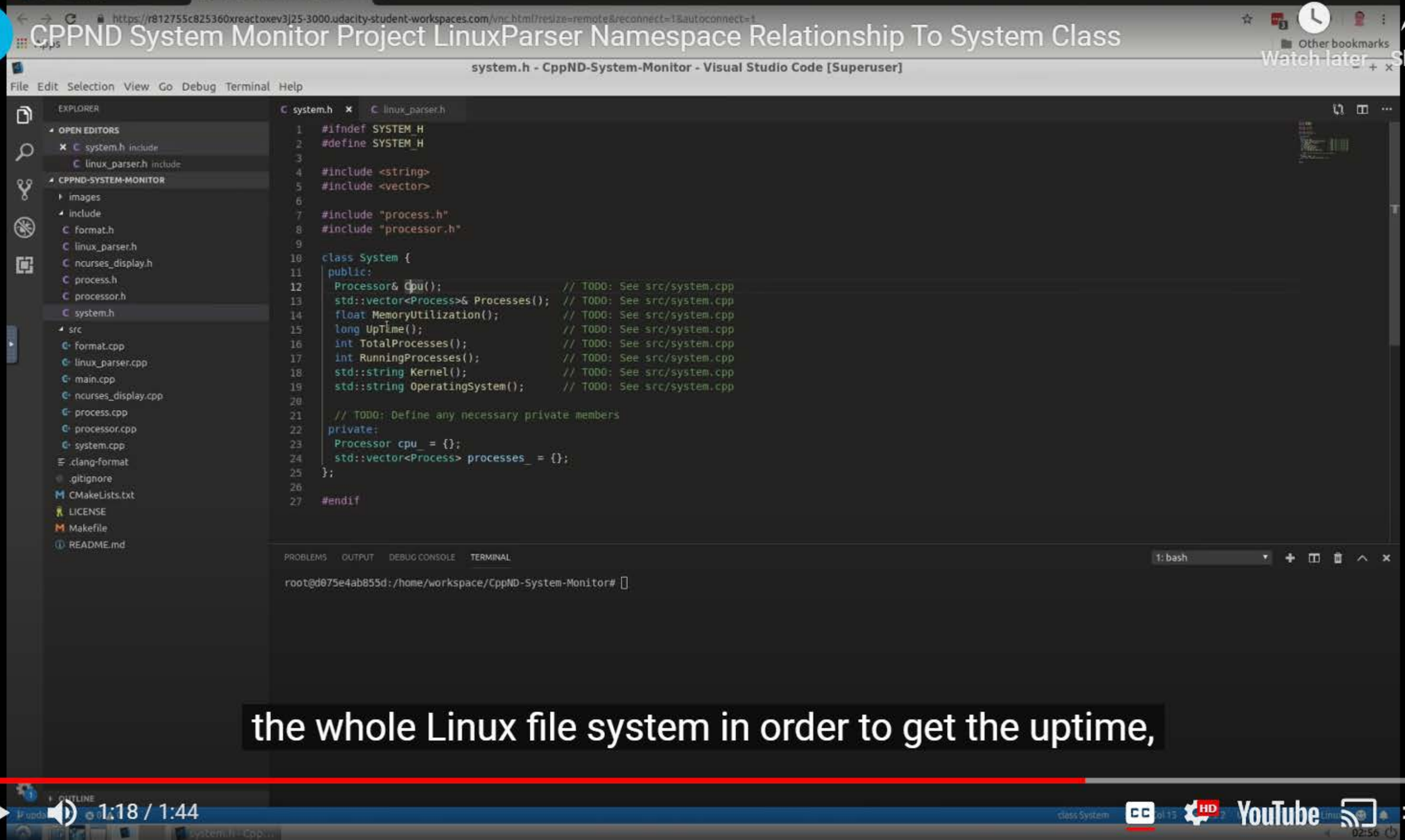


and then whenever a system object has uptime called on it,

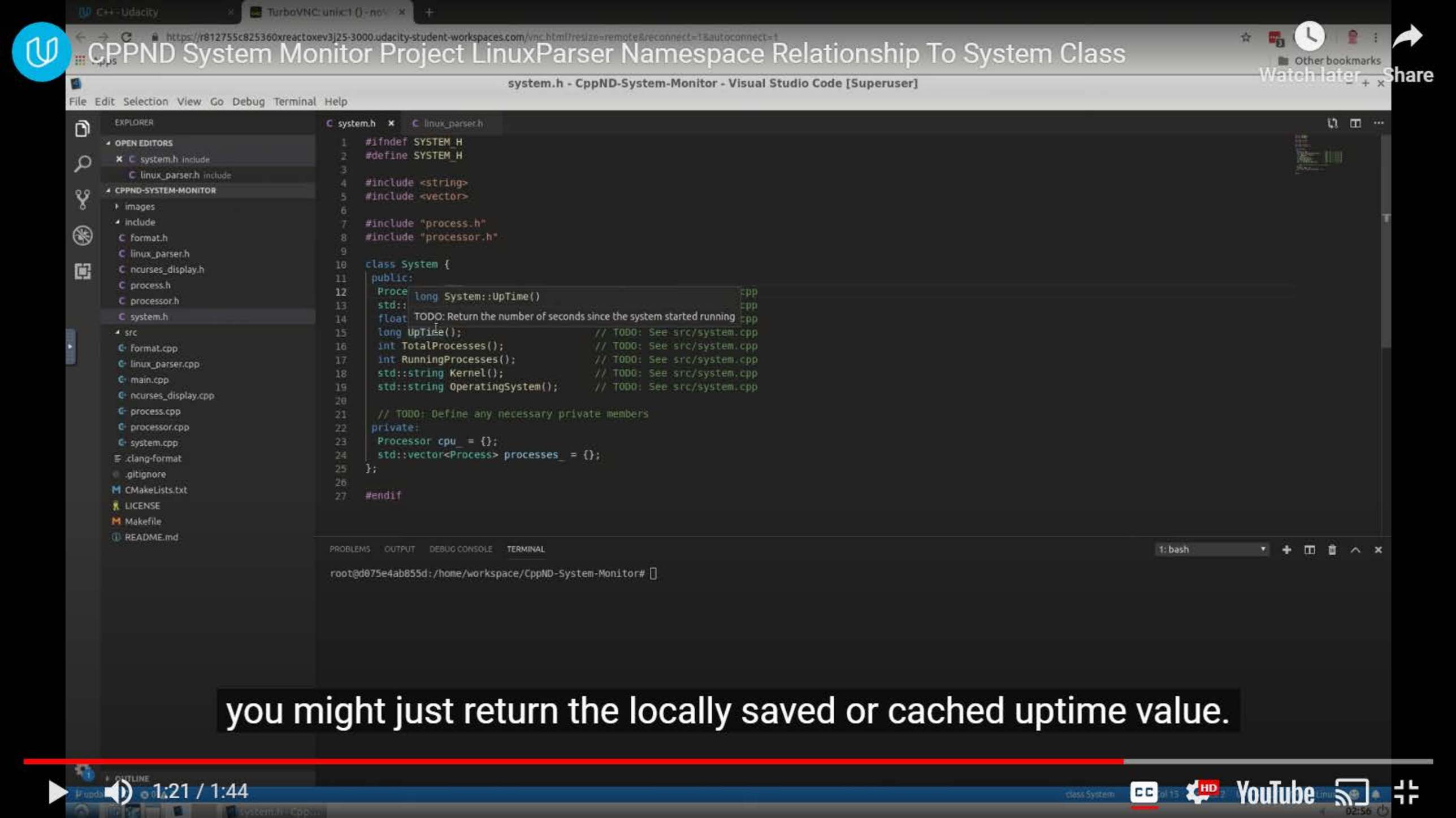
CPPND System Monitor Project LinuxParser Namespace Relationship To System Class

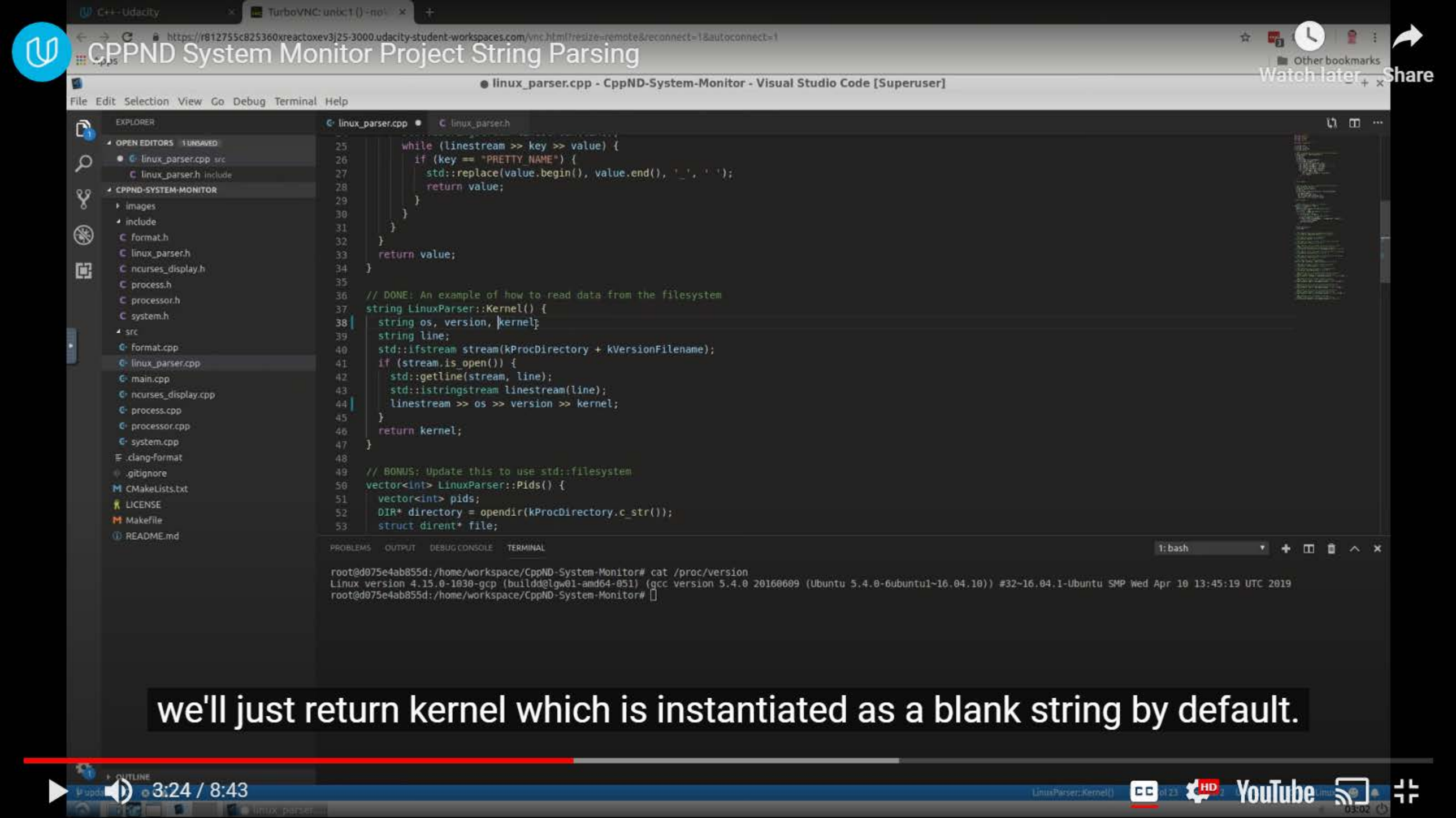


instead of going all the way through and parsing

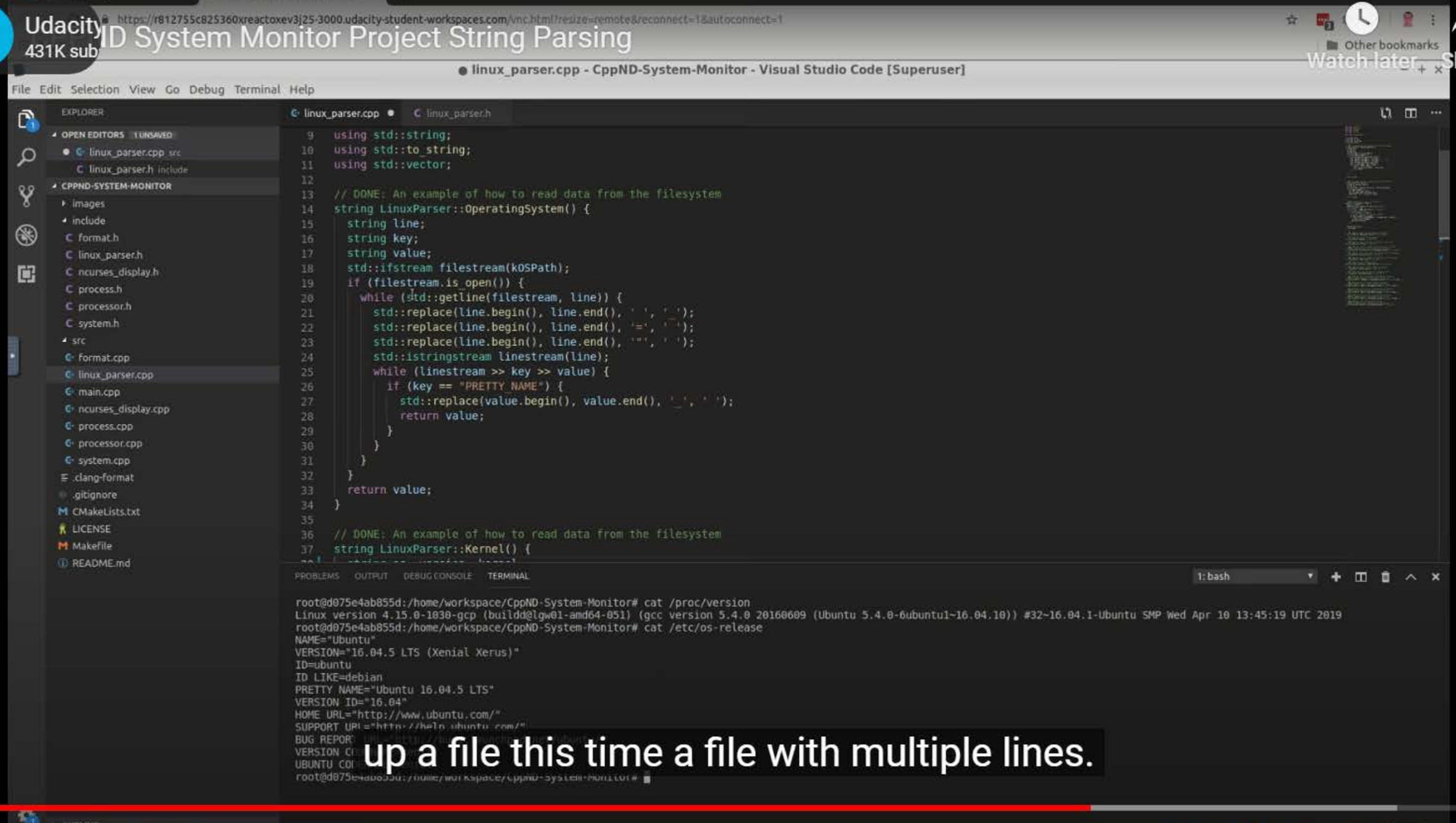


the whole Linux file system in order to get the uptime,


CPPND System Monitor Project LinuxParser Namespace Relationship To System Class
Watch later
Share
system.h - CppND-System-Monitor - Visual Studio Code [Superuser]
File Edit Selection View Go Debug Terminal Help
EXPLORER
OPEN EDITORS
system.h include
linux_parser.h include
CPPND-SYSTEM-MONITOR
images
include
format.h
linux_parser.h
ncurses_display.h
process.h
processor.h
system.h
src
format.cpp
linux_parser.cpp
main.cpp
ncurses_display.cpp
process.cpp
processor.cpp
system.cpp
.clang-format
.gitignore
CMakeLists.txt
LICENSE
Makefile
README.md
long System::UpTime()
TODO: Return the number of seconds since the system started running
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
root@d075e4ab855d:/home/workspace/CppND-System-Monitor#
you might just return the locally saved or cached uptime value.
1:21 / 1:44

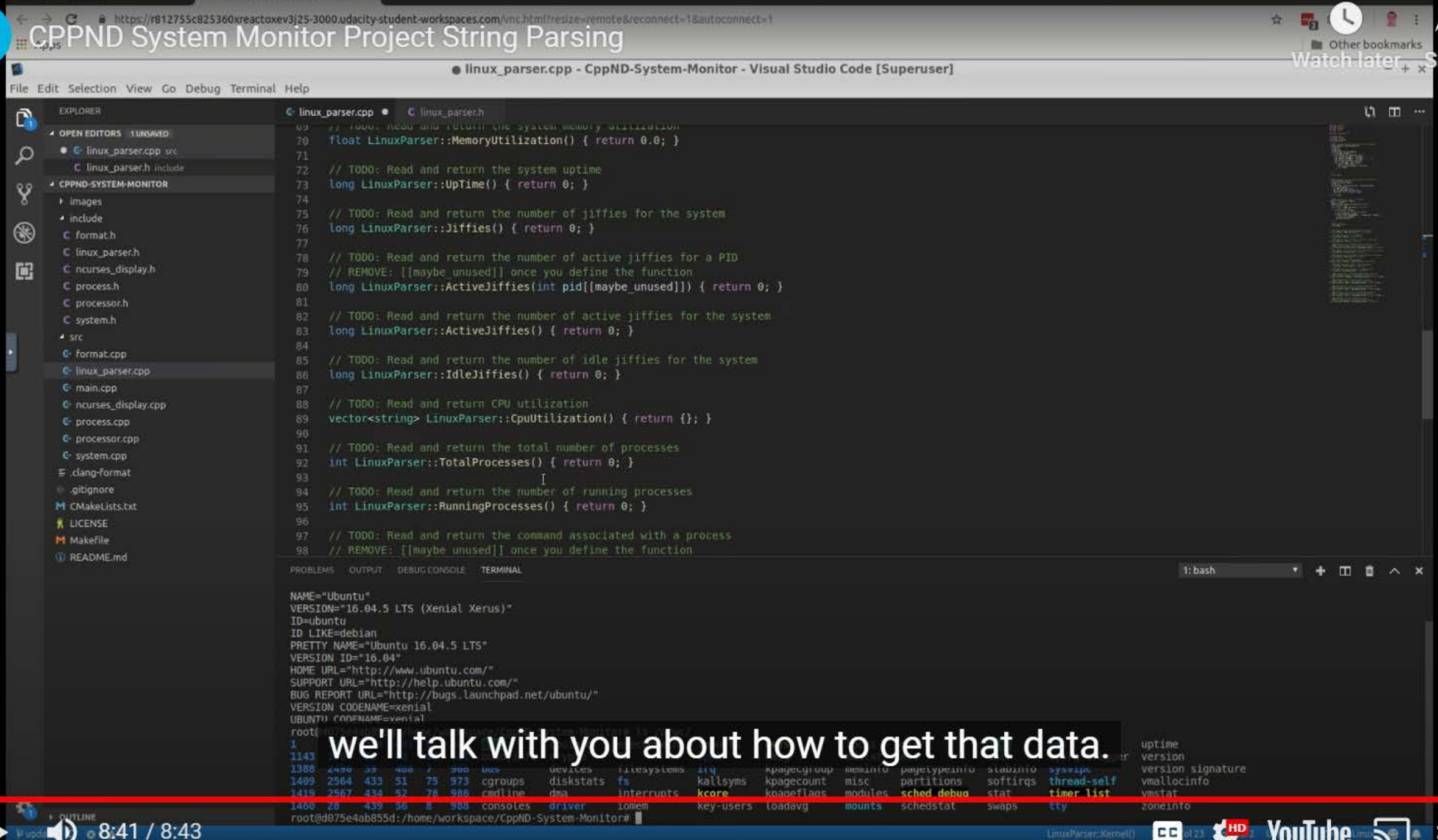

CPPND System Monitor Project String Parsing
Watch later
Share
linux_parser.cpp - CppND-System-Monitor - Visual Studio Code [Superuser]
File Edit Selection View Go Debug Terminal Help
EXPLORER
OPEN EDITORS 1 UNSAVED
linux_parser.cpp src
linux_parser.h include
CPPND-SYSTEM-MONITOR
images
include
format.h
linux_parser.h
ncurses_display.h
process.h
processor.h
system.h
src
format.cpp
linux_parser.cpp
main.cpp
ncurses_display.cpp
process.cpp
processor.cpp
system.cpp
.clang-format
.gitignore
CMakeLists.txt
LICENSE
Makefile
README.md
OUTLINE
while (linestream >> key >> value) {
if (key == "PRETTY_NAME") {
std::replace(value.begin(), value.end(), '_', ' ');
return value;
}
}
}
}
return value;
}
// DONE: An example of how to read data from the filesystem
string LinuxParser::Kernel() {
string os, version, kernel;
string line;
std::ifstream stream(kProcDirectory + kVersionFilename);
if (stream.is_open()) {
std::getline(stream, line);
std::istringstream linestream(line);
linestream >> os >> version >> kernel;
}
return kernel;
}
// BONUS: Update this to use std::filesystem
vector<int> LinuxParser::Pids() {
vector<int> pids;
DIR* directory = opendir(kProcDirectory.c_str());
struct dirent* file;
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
root@d075e4ab855d:/home/workspace/CppND-System-Monitor# cat /proc/version
Linux version 4.15.0-1030-gcp (buildd@lgw01-amd64-051) (gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.10)) #32~16.04.1-Ubuntu SMP Wed Apr 10 13:45:19 UTC 2019
root@d075e4ab855d:/home/workspace/CppND-System-Monitor#
we'll just return kernel which is instantiated as a blank string by default.
3:24 / 8:43

ID System Monitor Project String Parsing



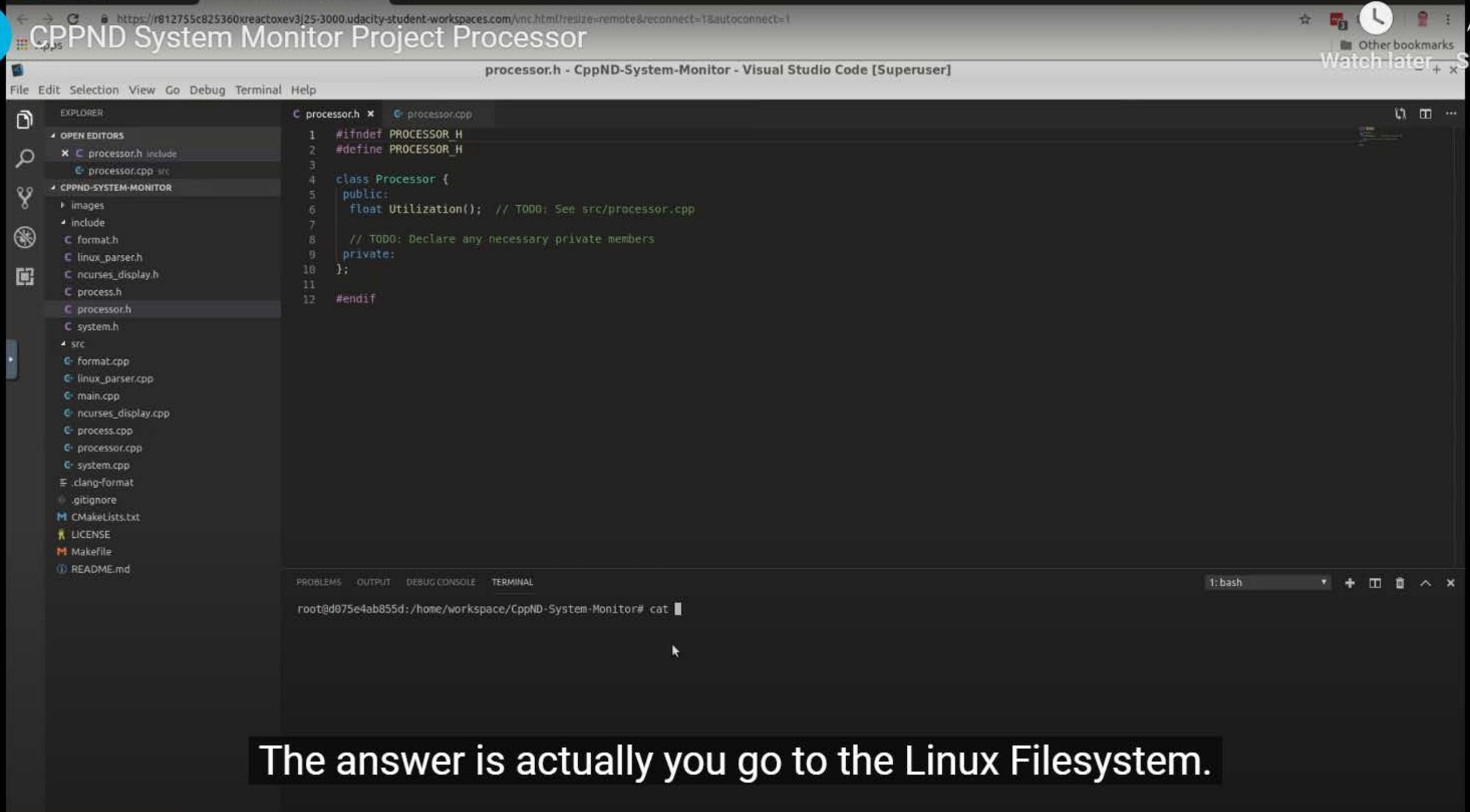
up a file this time a file with multiple lines.

CPPND System Monitor Project String Parsing

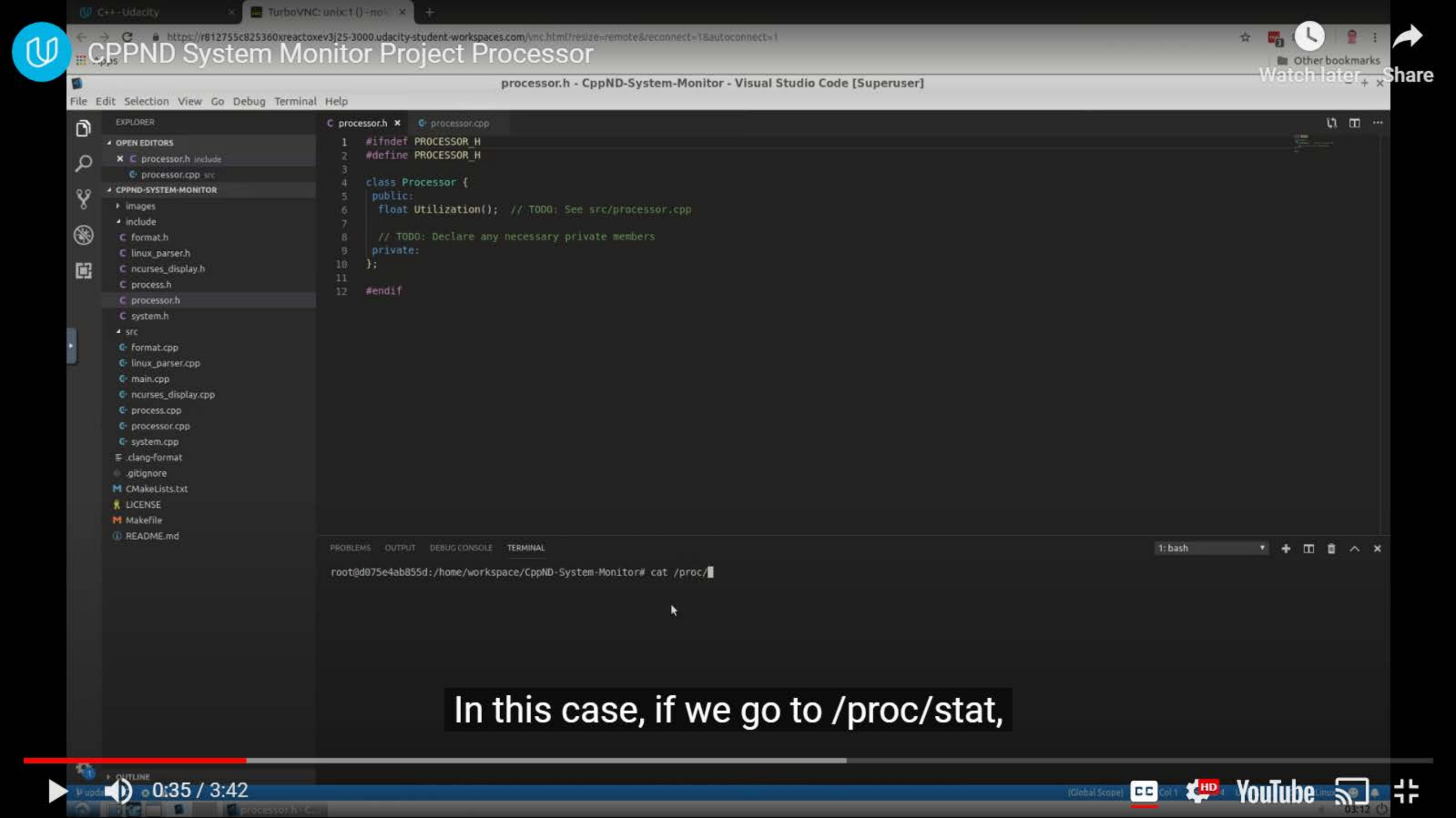


we'll talk with you about how to get that data.

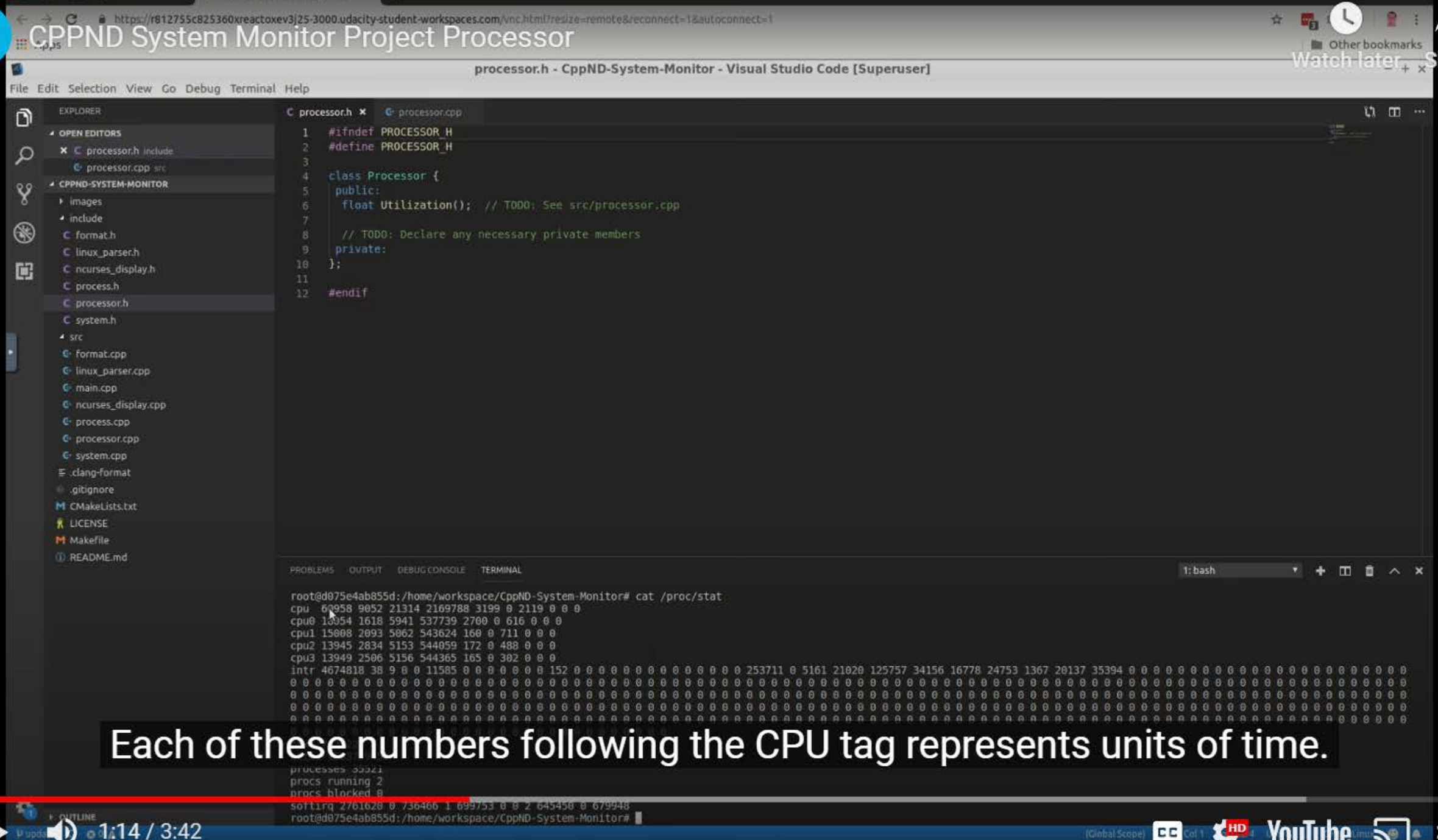
CPPND System Monitor Project Processor



The answer is actually you go to the Linux Filesystem.


CPPND System Monitor Project Processor
Watch later
Share
processor.h - CppND-System-Monitor - Visual Studio Code [Superuser]
File Edit Selection View Go Debug Terminal Help
EXPLORER
OPEN EDITORS
processor.h include
processor.cpp src
CPPND-SYSTEM-MONITOR
images
include
format.h
linux_parser.h
ncurses_display.h
process.h
processor.h
system.h
src
format.cpp
linux_parser.cpp
main.cpp
ncurses_display.cpp
process.cpp
processor.cpp
system.cpp
.clang-format
.gitignore
CMakeLists.txt
LICENSE
Makefile
README.md
processor.h
processor.cpp
#ifndef PROCESSOR_H
#define PROCESSOR_H
class Processor {
public:
float Utilization(); // TODO: See src/processor.cpp
// TODO: Declare any necessary private members
private:
};
#endif
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
root@d075e4ab855d:/home/workspace/CppND-System-Monitor# cat /proc/
In this case, if we go to /proc/stat,
0:35 / 3:42

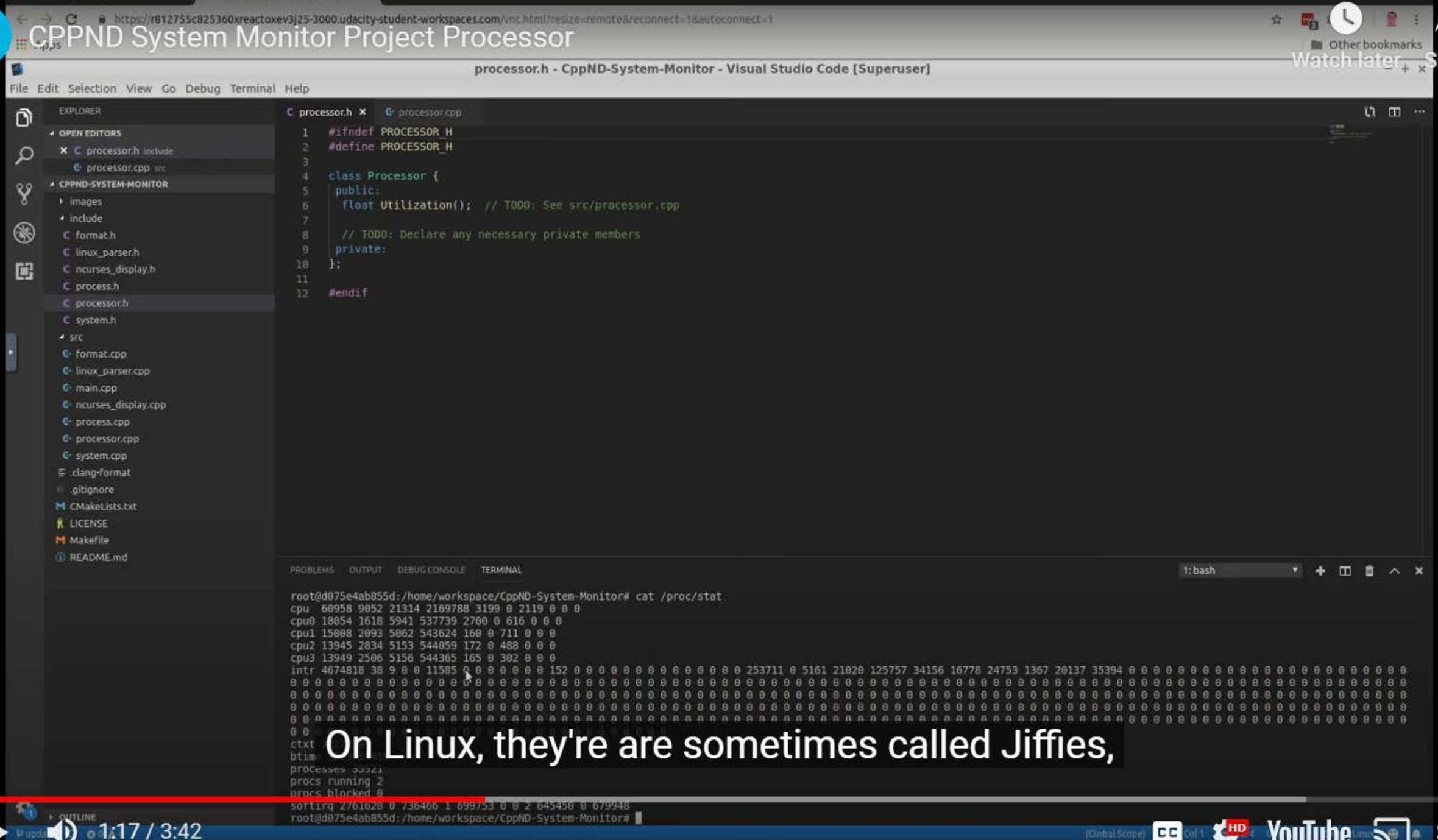
CPPND System Monitor Project Processor



Each of these numbers following the CPU tag represents units of time.

1:14 / 3:42

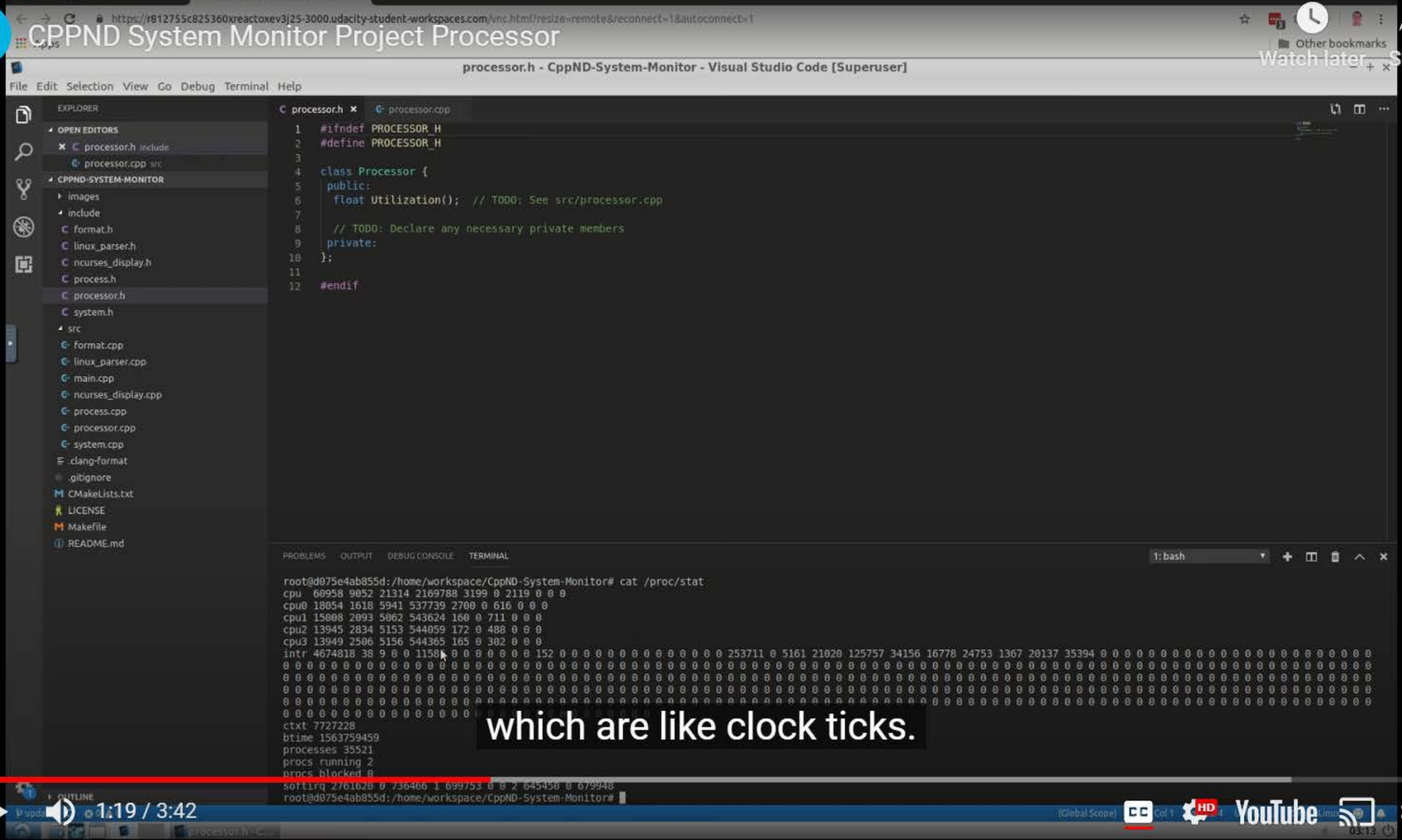
CPPND System Monitor Project Processor



On Linux, they're are sometimes called Jiffies,

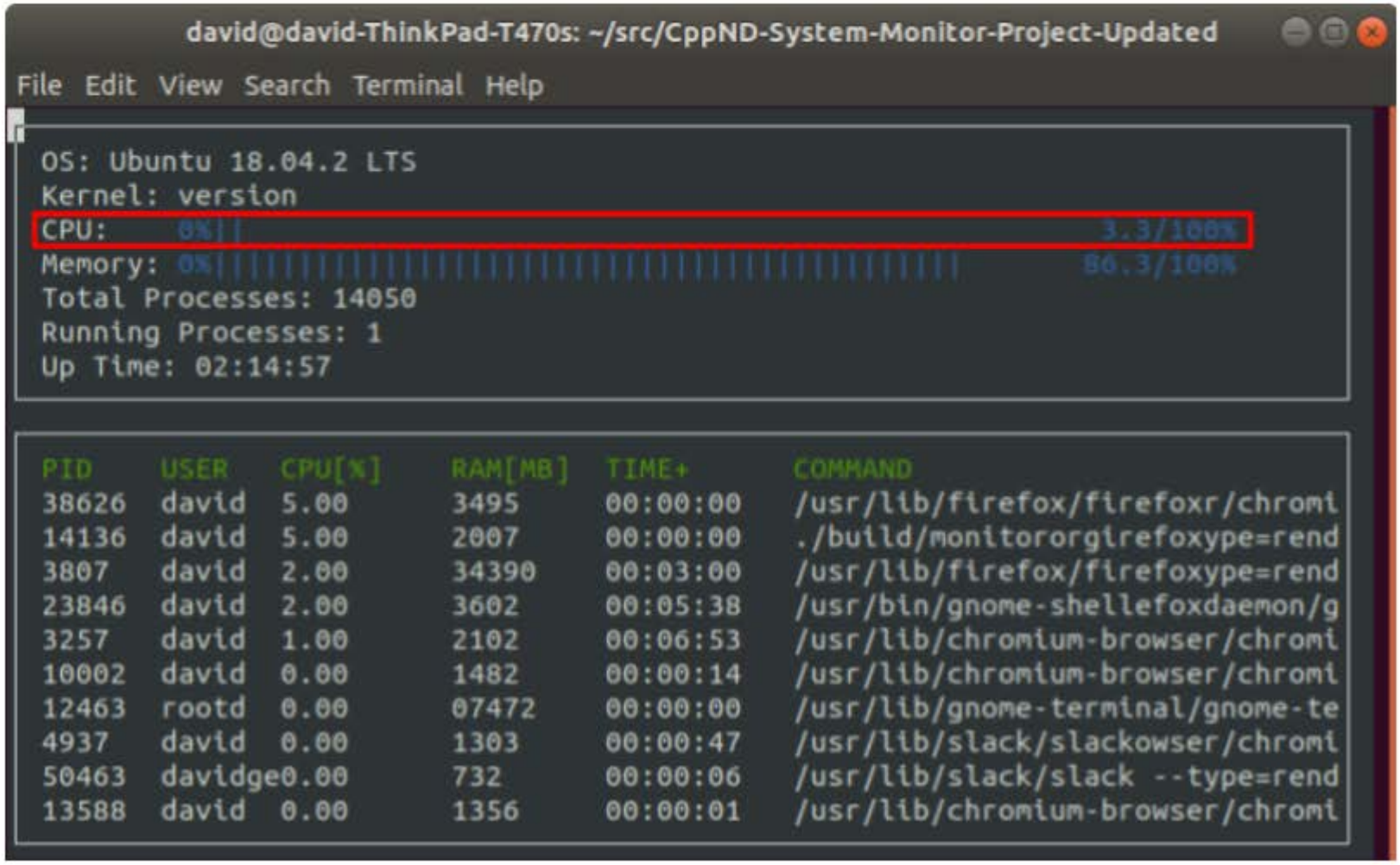
1:17 / 3:42

CPPND System Monitor Project Processor



which are like clock ticks.

Processor Data



Linux stores processor utilization data within the `/proc/stat` file.

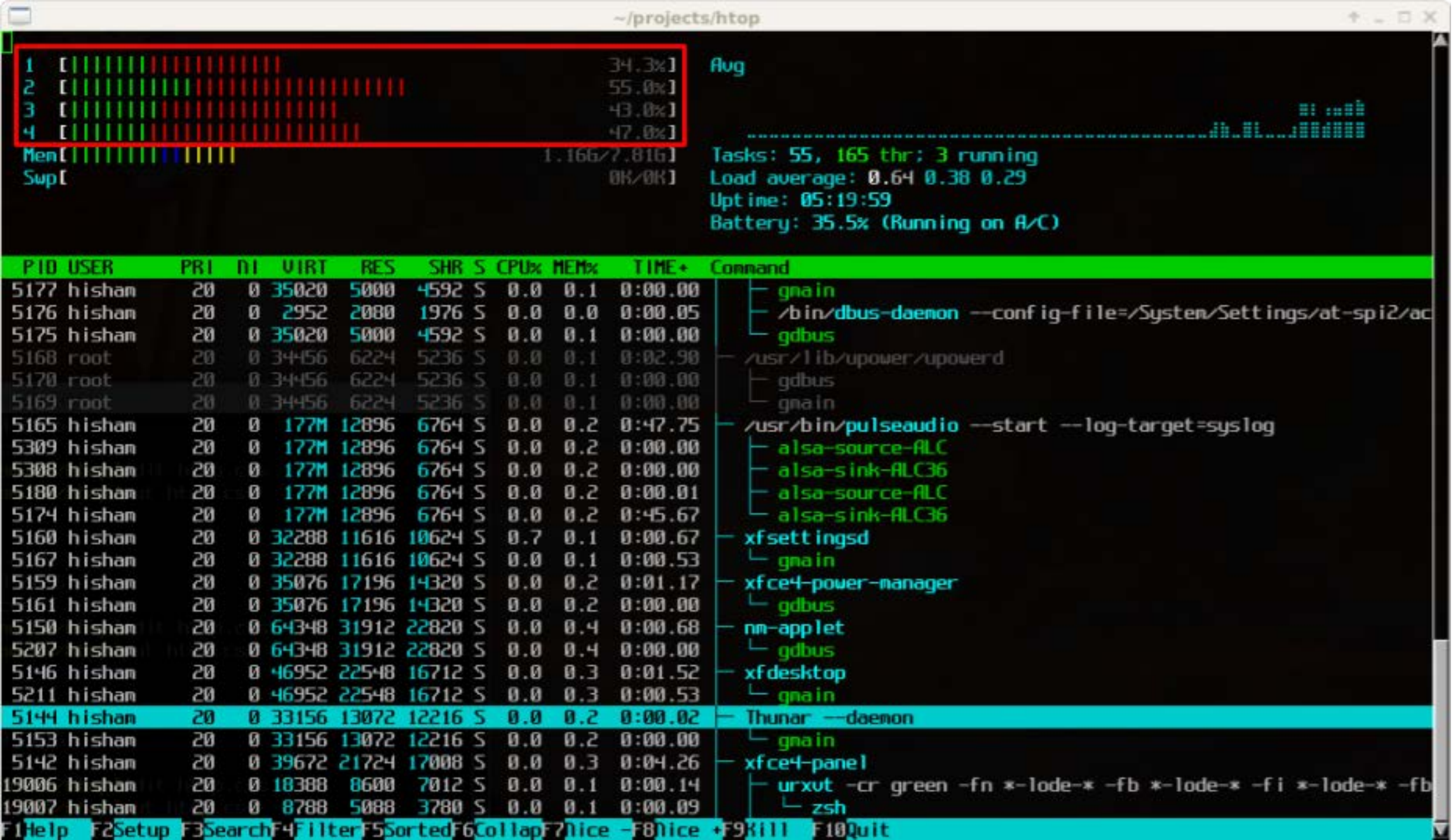
david@david-ThinkPad-T470s: ~

File Edit View Search Terminal Help

```
david@david-ThinkPad-T470s:~$ cat /proc/stat
cpu  655593 2758 162862 4287232 4331 0 5766 0 0 0
cpu0 164172 638 38553 1074498 1050 0 1443 0 0 0
cpu1 167790 669 45798 1062618 1039 0 1428 0 0 0
cpu2 160407 720 38843 1075686 1159 0 1899 0 0 0
cpu3 163223 730 39666 1074428 1082 0 996 0 0 0
intr 22516429 8 11 0 0 0 0 0 0 1 6015 0 0 511 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 241194 6442 45216 4
6593 57239 56692 62 348010 1484659 874 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ctxt 70733117
btime 1564258364
processes 11102
procs_running 15
procs_blocked 0
softirq 9494964 1410773 3052424 195 41960 6414 0 30032 2957711 0 1995455
david@david-ThinkPad-T470s:~$
```

This data is more complex than most of the other data necessary to complete this project.

For example, `/proc/stat` contains aggregate processor information (on the "cpu" line) and individual processor information (on the "cpu0", "cpu1", etc. lines). Indeed, htop displays utilization information for each individual processor.


~/projects/htop
1 [34.3%]
2 [55.0%]
3 [43.0%]
4 [47.0%]
Men[1.16G/7.81G]
Swp[0K/0K]
Avg
Tasks: 55, 165 thr; 3 running
Load average: 0.64 0.38 0.29
Uptime: 05:19:59
Battery: 35.5% (Running on A/C)
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
5177 hisham 20 0 35020 5000 4592 S 0.0 0.1 0:00.00 gnain
5176 hisham 20 0 2952 2080 1976 S 0.0 0.0 0:00.05 /bin/dbus-daemon --config-file=/System/Settings/at-spi2/ac
5175 hisham 20 0 35020 5000 4592 S 0.0 0.1 0:00.00 gdbus
5168 root 20 0 34456 6224 5236 S 0.0 0.1 0:02.90 /usr/lib/upower/upowerd
5170 root 20 0 34456 6224 5236 S 0.0 0.1 0:00.00 gdbus
5169 root 20 0 34456 6224 5236 S 0.0 0.1 0:00.00 gnain
5165 hisham 20 0 177M 12896 6764 S 0.0 0.2 0:47.75 /usr/bin/pulseaudio --start --log-target=syslog
5309 hisham 20 0 177M 12896 6764 S 0.0 0.2 0:00.00 alsa-source-ALC
5308 hisham 20 0 177M 12896 6764 S 0.0 0.2 0:00.00 alsa-sink-ALC36
5180 hisham 20 0 177M 12896 6764 S 0.0 0.2 0:00.01 alsa-source-ALC
5174 hisham 20 0 177M 12896 6764 S 0.0 0.2 0:45.67 alsa-sink-ALC36
5160 hisham 20 0 32288 11616 10624 S 0.7 0.1 0:00.67 xfsettingsd
5167 hisham 20 0 32288 11616 10624 S 0.0 0.1 0:00.53 gnain
5159 hisham 20 0 35076 17196 14320 S 0.0 0.2 0:01.17 xfce4-power-manager
5161 hisham 20 0 35076 17196 14320 S 0.0 0.2 0:00.00 gdbus
5150 hisham 20 0 64348 31912 22820 S 0.0 0.4 0:00.68 nm-applet
5207 hisham 20 0 64348 31912 22820 S 0.0 0.4 0:00.00 gdbus
5146 hisham 20 0 46952 22548 16712 S 0.0 0.3 0:01.52 xfdesktop
5211 hisham 20 0 46952 22548 16712 S 0.0 0.3 0:00.53 gnain
5144 hisham 20 0 33156 13072 12216 S 0.0 0.2 0:00.02 Thunar --daemon
5153 hisham 20 0 33156 13072 12216 S 0.0 0.2 0:00.00 gnain
5142 hisham 20 0 39672 21724 17008 S 0.0 0.3 0:04.26 xfce4-panel
19006 hisham 20 0 18388 8600 7012 S 0.0 0.1 0:00.14 urxvt -cr green -fn *-lode-* -fb *-lode-* -fi *-lode-* -fb
19007 hisham 20 0 8788 5088 3780 S 0.0 0.1 0:00.09 zsh
F1Help F2Setup F3SearchF4FilterF5SortedF6CollapF7Nice -F8Nice +F9Kill F10Quit

For this project, however, you only need to display aggregate CPU information, which you can find on the "cpu" line of `/proc/stat`.

If you would like to add individual processor information to your system monitor project, go for it!

Data

`/proc/stat` contains 10 integer values for each processor. The Linux source code documents each of these numbers:

The very first "cpu" line aggregates the numbers in all of the other "cpuN" lines. These numbers identify the amount of time the CPU has spent performing different kinds of work. Time units are in USER_HZ (typically hundredths of a second). The meanings of the columns are as follows, from left to right:

- *user: normal processes executing in user mode*
- *nice: niced processes executing in user mode*
- *system: processes executing in kernel mode*
- *idle: twiddling thumbs*
- *iowait: In a word, iowait stands for waiting for I/O to complete. But there are several problems:*
    1. *Cpu will not wait for I/O to complete, iowait is the time that a task is waiting for I/O to complete. When cpu goes into idle state for outstanding task io, another task will be scheduled on this CPU.*
    2. *In a multi-core CPU, the task waiting for I/O to complete is not running on any CPU, so the iowait of each CPU is difficult to calculate.*
    3. *The value of iowait field in /proc/stat will decrease in certain conditions. So, the iowait is not reliable by reading from /proc/stat.*
- *irq: servicing interrupts*
- *softirq: servicing softirqs*
- *steal: involuntary wait*
- *guest: running a normal guest*
- *guest_nice: running a niced guest*

Even once you know what each of these numbers represents, it's still a challenge to determine exactly how to use these figures to calculate processor utilization. This guide and this StackOverflow post are helpful.

Measurement Interval

Once you've parsed `/proc/stat` and calculated the processor utilization, you've got what you need for this project. Congratulations!

However, when you run your system monitor, you might notice that the process utilization seems very stable. Too stable.

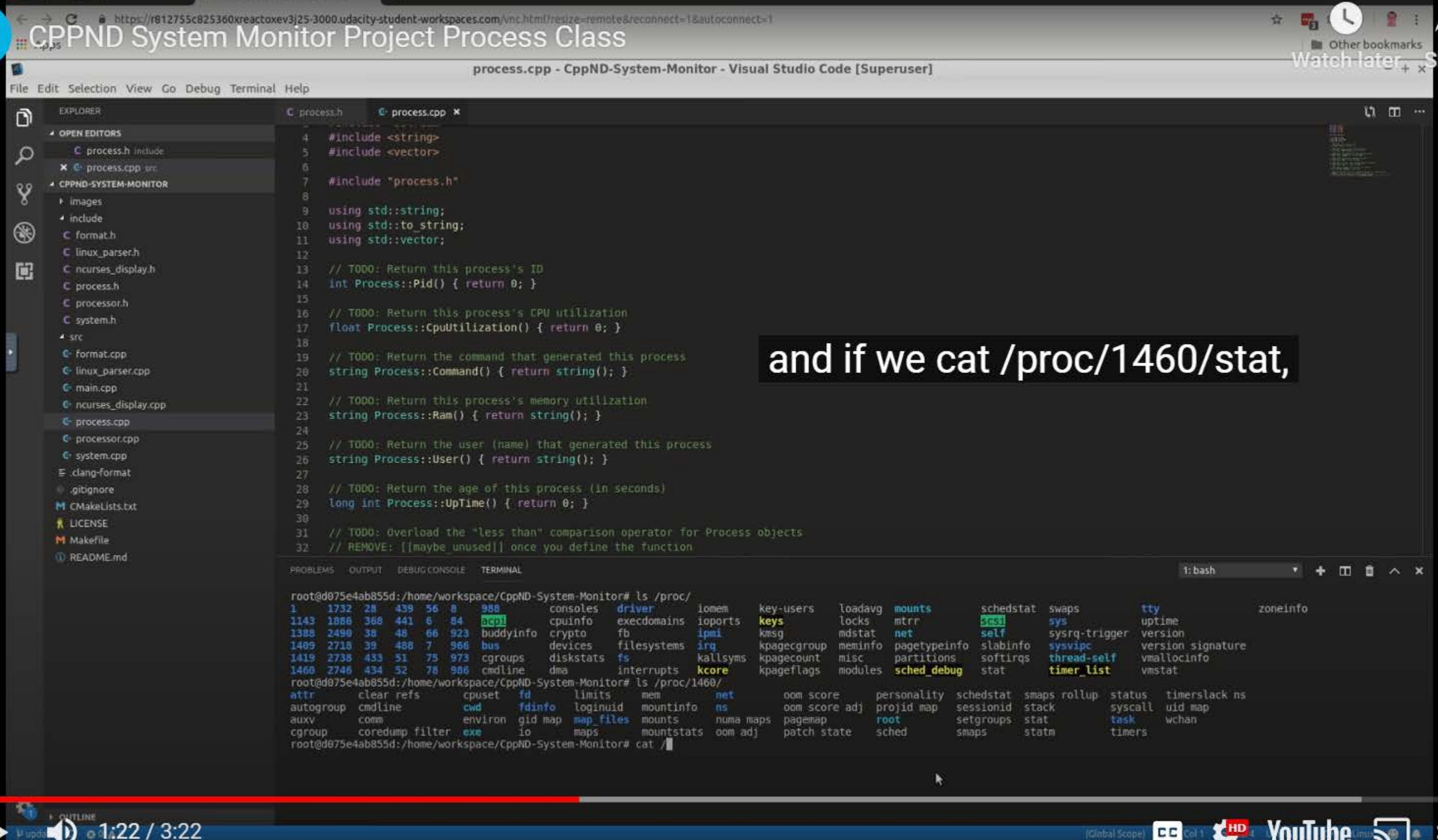
That's because the processor data in `/proc/stat` is measured since boot. If the system has been up for a long time, a temporary interval of even extreme system utilization is unlikely to change the long-term average statistics very much. This means that the processor could be red-lining *right now* but the system monitor might still show a relatively underutilized processor, if the processor has spent most of the time since boot in an idle state.

You might want to update the system monitor to report the current utilization of the processor, rather than the long-term average utilization since boot. You would need to measure the difference in system utilization between two points in time relatively close to the present. A formula like:

Δ active time units / Δ total time units

Consider this a bonus challenge that is not required to pass the project.

CPPND System Monitor Project Process Class



CPPND System Monitor Project Process Class

process.cpp - CppND-System-Monitor - Visual Studio Code [Superuser]

```
#include <string>
#include <vector>

#include "process.h"

using std::string;
using std::to_string;
using std::vector;

// TODO: Return this process's ID
int Process::Pid() { return 0; }

// TODO: Return this process's CPU utilization
float Process::CpuUtilization() { return 0; }

// TODO: Return the command that generated this process
string Process::Command() { return string(); }

// TODO: Return this process's memory utilization
string Process::Ram() { return string(); }

// TODO: Return the user (name) that generated this process
string Process::User() { return string(); }

// TODO: Return the age of this process (in seconds)
long int Process::UpTime() { return 0; }

// TODO: Overload the "less than" comparison operator for Process objects
// REMOVE: [[maybe_unused]] once you define the function
```

```
root@d075e4ab855d:/home/workspace/CppND-System-Monitor# ls /proc/
1     1732  28   439  56  8    988        consoles    driver      iomem      key-users   loadavg  mounts       schedstat  swaps        tty              zoneinfo
1143  1886  368  441  6   84   acpi       cpuinfo     execdomains ioports    keys        locks    mtrr         scsi       sys          uptime
1388  2490  38   48   66  923  buddyinfo  crypto      fb          ipmi       kmsg        mdstat   net          self       sysrq-trigger version
1409  2718  39   488  7   966  bus        devices     filesystems irq        kpagecgroup meminfo  pagetypeinfo slabinfo   sysvipc      version signature
1419  2738  433  51   75  973  cgroups    diskstats   fs          kallsyms   kpagecount  misc     partitions   softirqs   thread-self  vmallocinfo
1460  2746  434  52   78  986  cmdline    dma         interrupts  kcore      kpageflags  modules  sched_debug  stat       timer_list   vmstat
root@d075e4ab855d:/home/workspace/CppND-System-Monitor# ls /proc/1460/
attr        clear_refs      cpuset  fd      limits     mem        net         oom_score      personality  schedstat  smaps_rollup  status   timerslack_ns
autogroup   cmdline         cwd     fdinfo  loginuid   mountinfo  ns          oom_score_adj  projid_map   sessionid  stack         syscall  uid_map
auxv        comm            environ gid_map map_files  mounts     numa_maps   pagemap        root         setgroups  stat          task     wchan
cgroup      coredump_filter exe     io      maps       mountstats oom_adj     patch_state    sched        smaps      statm         timers
root@d075e4ab855d:/home/workspace/CppND-System-Monitor#
1460 (Microsoft.VSCod) S 1419 1 1 0 -1 ... 233 1265242112 27183 18446744073709551615 4194304 16177008 140722712463216 0 0 0 0
 16777216 16386 0 0 0 17 1 0 0 0 0 0 ... 140722712471744 140722712471836 140722712473500 0
root@d075e4ab855d:/home/workspace/CppND-System-Monitor# cat /proc/1460/status
```

If we look at /status,

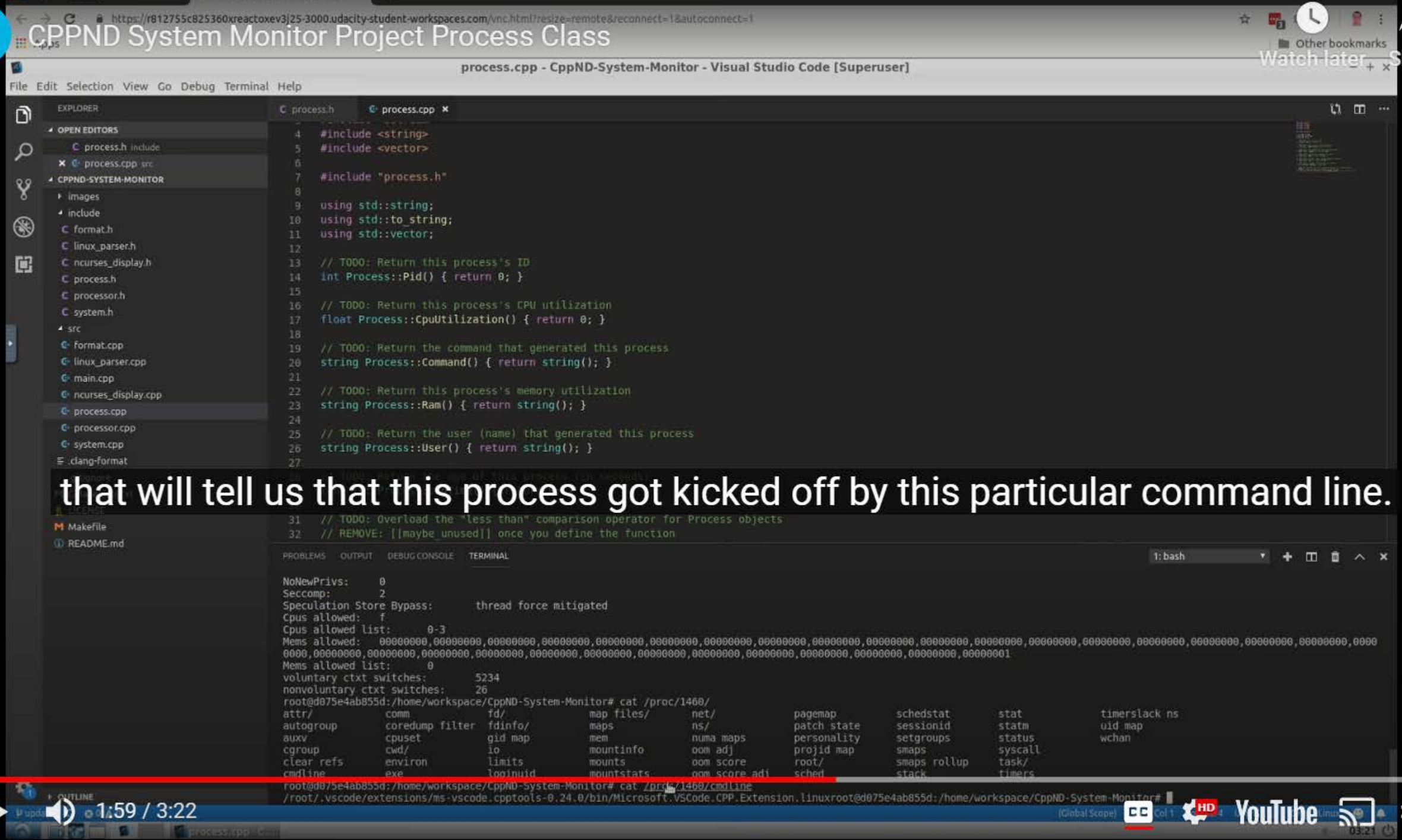
1:38 / 3:22

Watch later Share

Other bookmarks

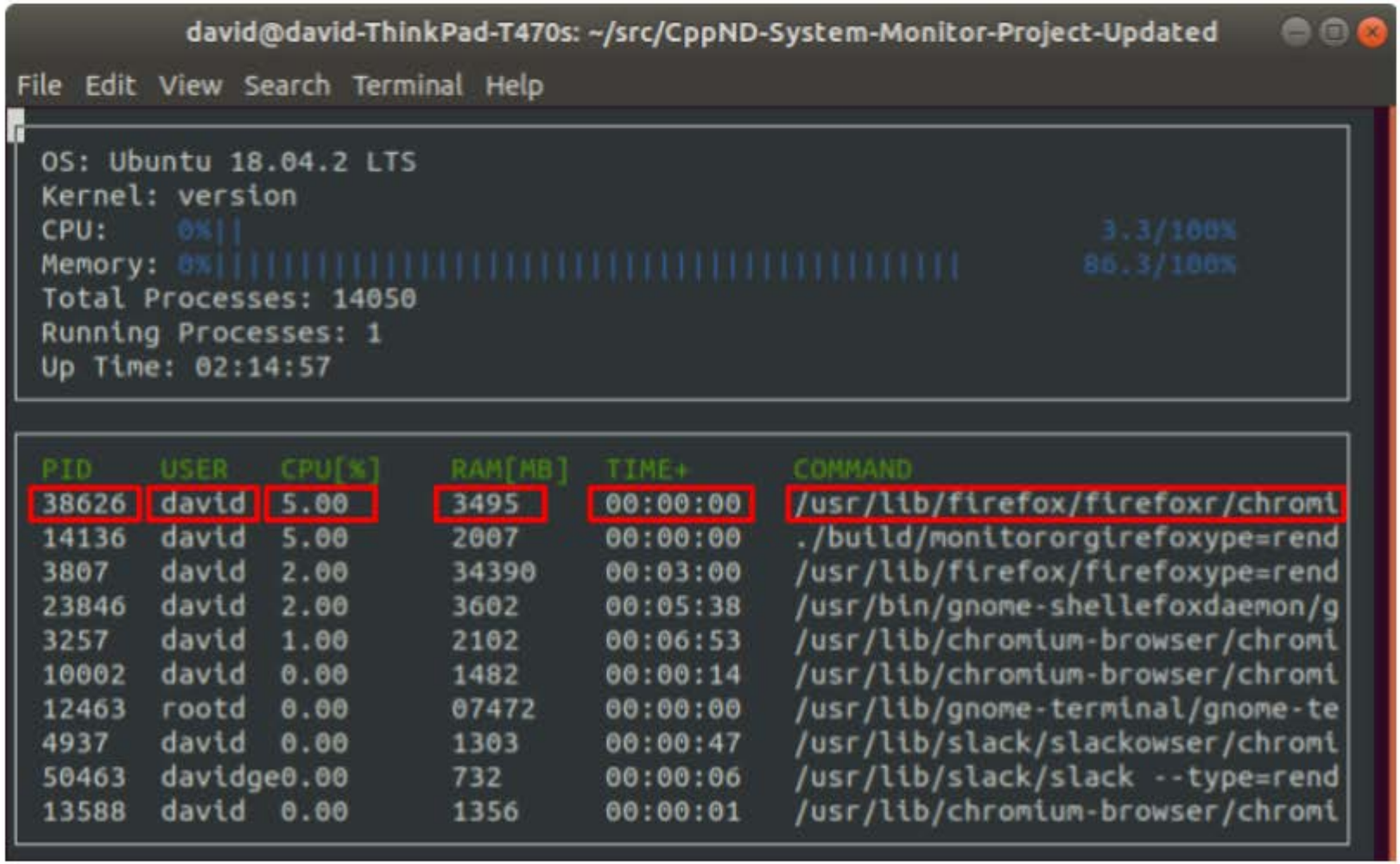
https://www.youtube.com/channel/UCBVCi5JbYmfG3q5MEuoWdOw

CPPND System Monitor Project Process Class

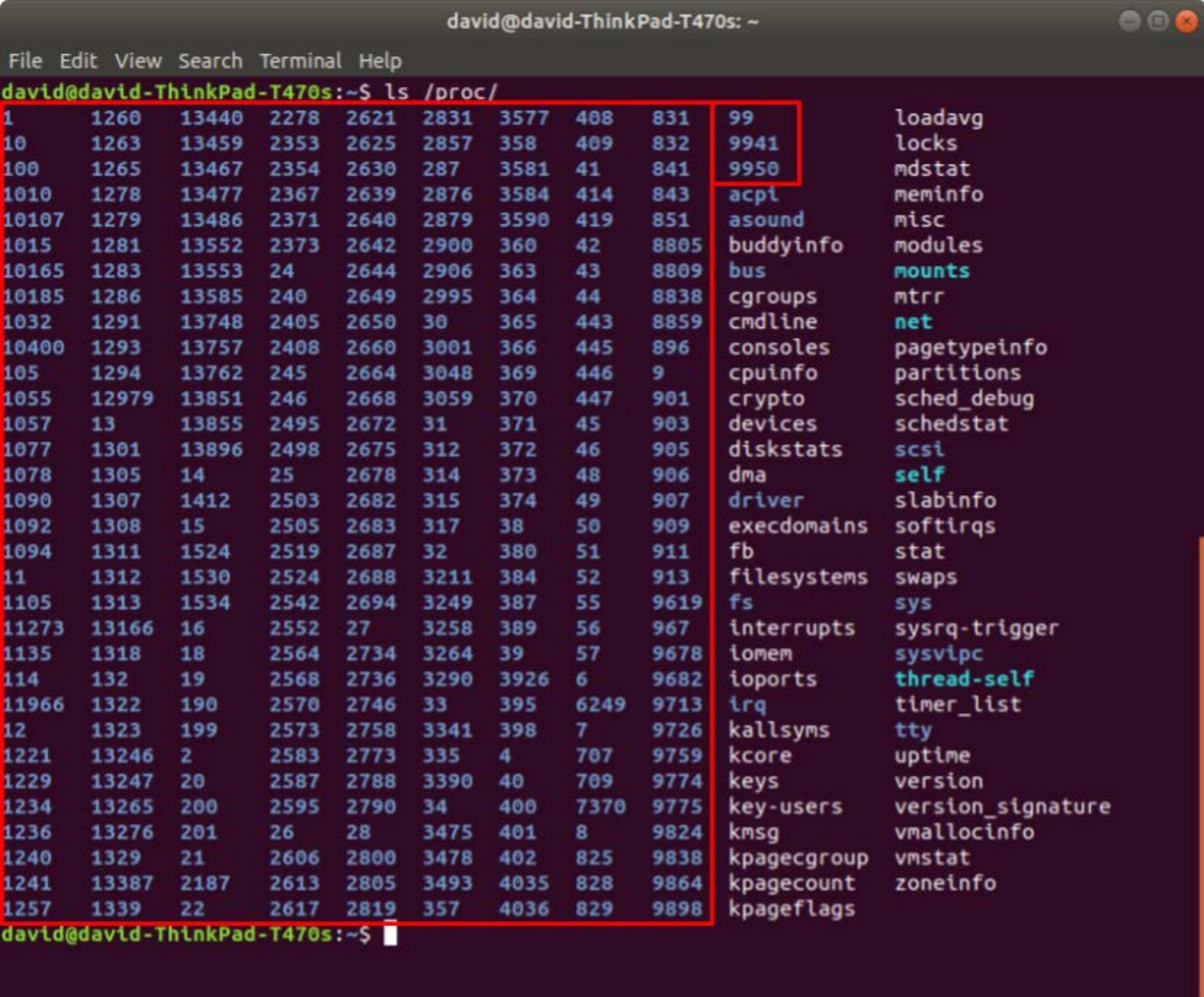


that will tell us that this process got kicked off by this particular command line.

Process Data



Linux stores data about individual processes in files within subdirectories of the `/proc` directory. Each subdirectory is named for that particular process's identifier number. The data that this project requires exists in those files.


david@david-ThinkPad-T470s: ~
File Edit View Search Terminal Help
david@david-ThinkPad-T470s:~$ ls /proc/
1 1260 13440 2278 2621 2831 3577 408 831 99 loadavg
10 1263 13459 2353 2625 2857 358 409 832 9941 locks
100 1265 13467 2354 2630 287 3581 41 841 9950 mdstat
1010 1278 13477 2367 2639 2876 3584 414 843 acpi meminfo
10107 1279 13486 2371 2640 2879 3590 419 851 asound misc
1015 1281 13552 2373 2642 2900 360 42 8805 buddyinfo modules
10165 1283 13553 24 2644 2906 363 43 8809 bus mounts
10185 1286 13585 240 2649 2995 364 44 8838 cgroups mtrr
1032 1291 13748 2405 2650 30 365 443 8859 cmdline net
10400 1293 13757 2408 2660 3001 366 445 896 consoles pagetypeinfo
105 1294 13762 245 2664 3048 369 446 9 cpuinfo partitions
1055 12979 13851 246 2668 3059 370 447 901 crypto sched_debug
1057 13 13855 2495 2672 31 371 45 903 devices schedstat
1077 1301 13896 2498 2675 312 372 46 905 diskstats scsi
1078 1305 14 25 2678 314 373 48 906 dma self
1090 1307 1412 2503 2682 315 374 49 907 driver slabinfo
1092 1308 15 2505 2683 317 38 50 909 execdomains softirqs
1094 1311 1524 2519 2687 32 380 51 911 fb stat
11 1312 1530 2524 2688 3211 384 52 913 filesystems swaps
1105 1313 1534 2542 2694 3249 387 55 9619 fs sys
11273 13166 16 2552 27 3258 389 56 967 interrupts sysrq-trigger
1135 1318 18 2564 2734 3264 39 57 9678 iomem sysvipc
114 132 19 2568 2736 3290 3926 6 9682 ioports thread-self
11966 1322 190 2570 2746 33 395 6249 9713 irq timer_list
12 1323 199 2573 2758 3341 398 7 9726 kallsyms tty
1221 13246 2 2583 2773 335 4 707 9759 kcore uptime
1229 13247 20 2587 2788 3390 40 709 9774 keys version
1234 13265 200 2595 2790 34 400 7370 9775 key-users version_signature
1236 13276 201 26 28 3475 401 8 9824 kmsg vmallocinfo
1240 1329 21 2606 2800 3478 402 825 9838 kpagecgroup vmstat
1241 13387 2187 2613 2805 3493 4035 828 9864 kpagecount zoneinfo
1257 1339 22 2617 2819 357 4036 829 9898 kpageflags
david@david-ThinkPad-T470s:~$

PID

The process identifier (PID) is accessible from the `/proc` directory. Typically, all of the subdirectories of `/proc` that have integral names correspond to processes. Each integral name corresponds to a process ID.

Parsing directory names with C++ is tricky, so we have provided in the project starter code a pre-implemented function to capture the PIDs.

User

Each process has an associated user identifier (UID), corresponding to the process owner. This means that determining the process owner requires two steps:

1. Find the UID associated with the process
2. Find the user corresponding to that UID

The UID for a process is stored in `/proc/[PID]/status`.

david@david-ThinkPad-T470s: ~

File Edit View Search Terminal Help

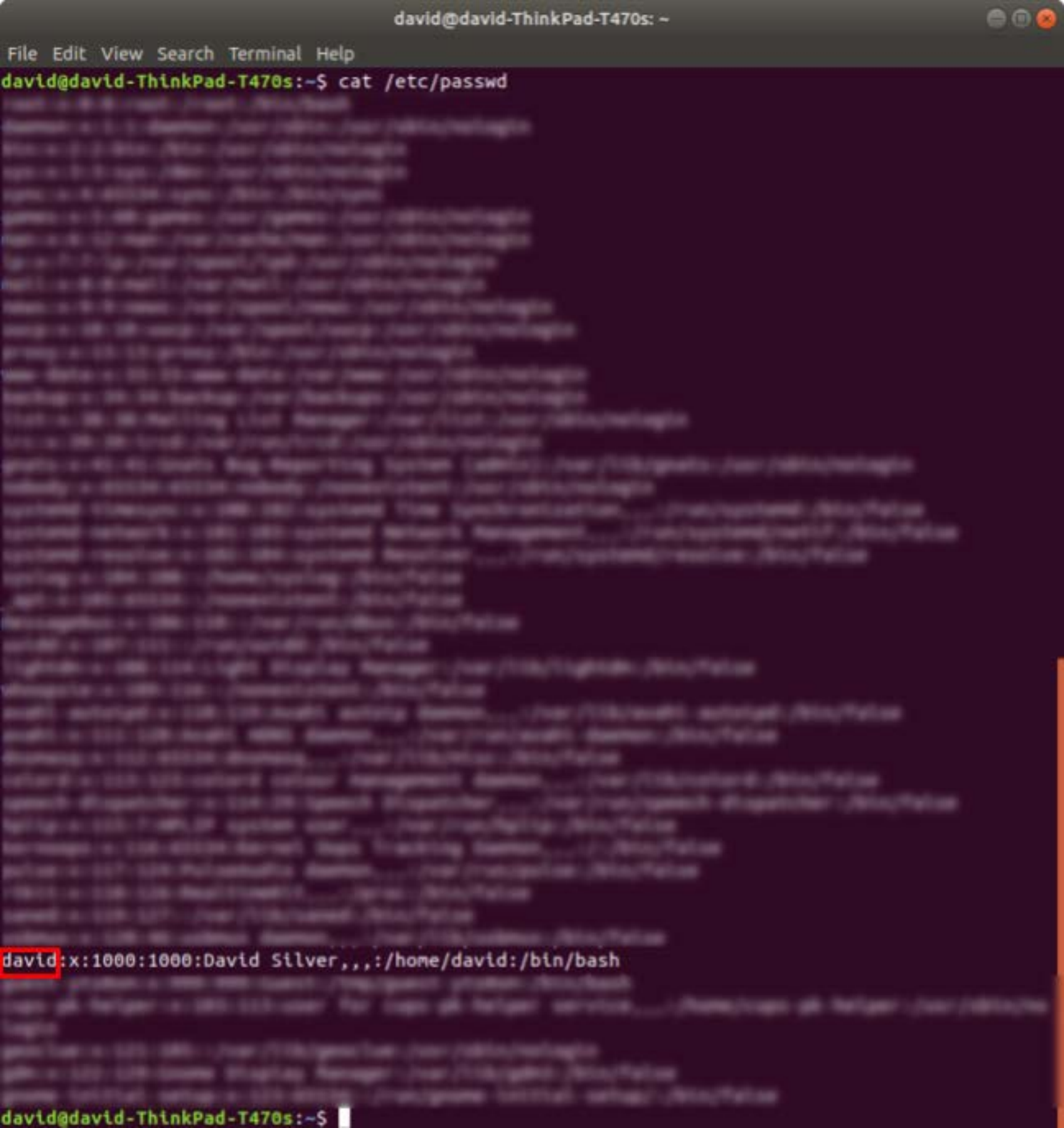
```
david@david-ThinkPad-T470s:~$ cat /proc/2879/status
Name:   chromium-browse
Umask:  0002
State:  S (sleeping)
Tgid:   2879
Ngid:   0
Pid:    2879
PPid:   2876
TracerPid:      0
Uid:    1000    1000    1000    1000
Gid:    1000    1000    1000    1000
FDSize: 64
Groups: 4 24 27 30 46 113 128 1000
NStgid: 2879    3
NSpid:  2879    3
NSpgid: 2371    0
NSsid:  2371    0
VmPeak:   457704 kB
VmSize:   457704 kB
VmLck:         0 kB
VmPin:         0 kB
VmHWM:     13040 kB
VmRSS:     12972 kB
RssAnon:            9408 kB
RssFile:            3564 kB
RssShmem:              0 kB
VmData:     5200 kB
VmStk:       132 kB
VmExe:    161612 kB
VmLib:     58804 kB
VmPTE:       512 kB
VmSwap:        0 kB
HugetlbPages:          0 kB
CoreDumping:    0
Threads:        1
SigQ:   0/30496
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000000000
SigIgn: 0000000000001002
SigCgt: 0000000180010000
CapInh: 0000000000000000
CapPrm: 0000000000200000
CapEff: 0000000000200000
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
NoNewPrivs:     1
Seccomp:        0
Speculation_Store_Bypass:       thread vulnerable
Cpus_allowed:   f
Cpus_allowed_list:      0-3
Mems_allowed:   00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000
000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000
,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00
000000,00000000,00000001
Mems_allowed_list:      0
voluntary_ctxt_switches:        1034
nonvoluntary_ctxt_switches:     345
david@david-ThinkPad-T470s:~$
```

The *man* page for `proc` contains a "/proc/[pid]/status" section that describes this file.

For the purposes of this project, you simply need to capture the first integer on the "Uid:" line.

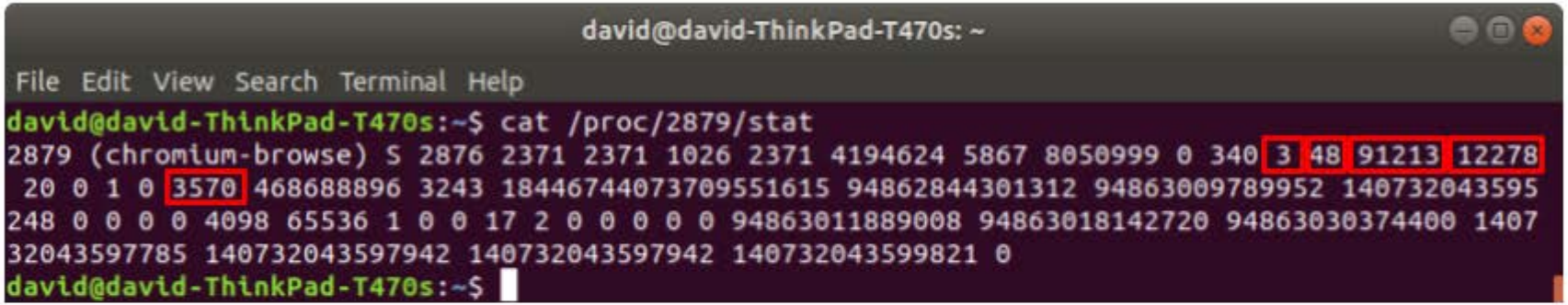
Username

`/etc/passwd` contains the information necessary to match the UID to a username.


david@david-ThinkPad-T470s: ~
File Edit View Search Terminal Help
david@david-ThinkPad-T470s:~$ cat /etc/passwd
david:x:1000:1000:David Silver,,,:/home/david:/bin/bash
david@david-ThinkPad-T470s:~$

Processor Utilization

Linux stores the CPU utilization of a process in the `/proc/[PID]/stat` file.



Much like the calculation of aggregate processor utilization, half the battle is extracting the relevant data from the file, and the other half of the battle is figuring out how to use those numbers to calculate processor utilization.

The "/proc/[pid]/stat" section of the `proc` *man page* describes the meaning of the values in this file. This StackOverflow answer explains how to use this data to calculate the process's utilization.

As with the calculation of aggregate processor utilization, it is sufficient for this project to calculate the average utilization of each process since the process launched. If you would like to extend your project to calculate a more current measurement of process utilization, we encourage you to do that!

david@david-ThinkPad-T470s: ~

File Edit View Search Terminal Help

```
david@david-ThinkPad-T470s:~$ cat /proc/2879/status
Name:   chromium-browse
Umask:  0002
State:  S (sleeping)
Tgid:   2879
Ngid:   0
Pid:    2879
PPid:   2876
TracerPid:      0
Uid:    1000    1000    1000    1000
Gid:    1000    1000    1000    1000
FDSize: 64
Groups: 4 24 27 30 46 113 128 1000
NStgid: 2879    3
NSpid:  2879    3
NSpgid: 2371    0
NSsid:  2371    0
VmPeak:   457704 kB
VmSize:   457704 kB
VmLck:         0 kB
VmPin:         0 kB
VmHWM:     13040 kB
VmRSS:     12972 kB
RssAnon:            9408 kB
RssFile:            3564 kB
RssShmem:              0 kB
VmData:     5200 kB
VmStk:       132 kB
VmExe:    161612 kB
VmLib:     58804 kB
VmPTE:       512 kB
VmSwap:        0 kB
HugetlbPages:          0 kB
CoreDumping:    0
Threads:        1
SigQ:   0/30496
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000000000
SigIgn: 0000000000001002
SigCgt: 0000000180010000
CapInh: 0000000000000000
CapPrm: 0000000000200000
CapEff: 0000000000200000
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
NoNewPrivs:     1
Seccomp:        0
Speculation_Store_Bypass:       thread vulnerable
Cpus_allowed:   f
Cpus_allowed_list:      0-3
Mems_allowed:   00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000
000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000
,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,00
000000,00000000,00000001
Mems_allowed_list:      0
voluntary_ctxt_switches:        1034
nonvoluntary_ctxt_switches:     345
david@david-ThinkPad-T470s:~$
```

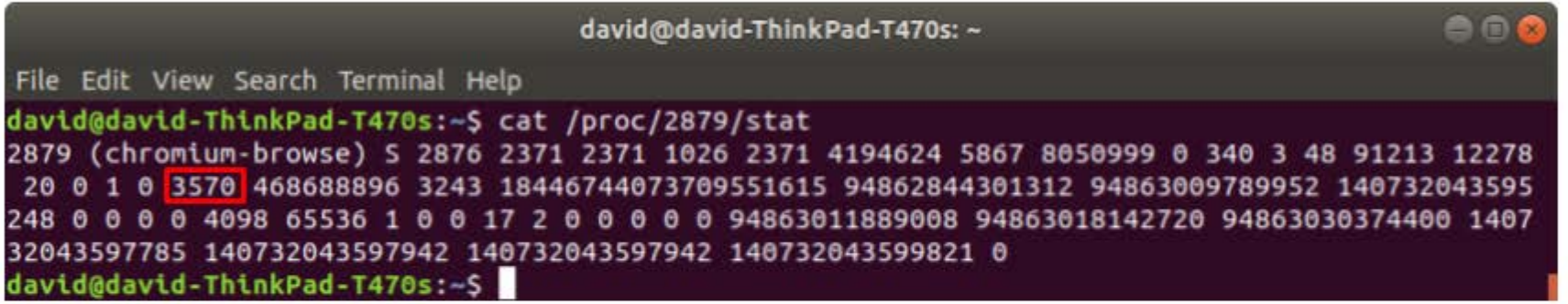
Memory Utilization

Linux stores memory utilization for the process in `/proc/[pid]/status`.

In order to facilitate display, consider converting the memory utilization into megabytes.

Up Time

Linux stores the process up time in `/proc/[pid]/stat`.



The "/proc/[pid]/stat" section of the `proc` *man page* describes each of the values in this file.

> *(22) starttime %llu*
>
> *The time the process started after system boot. In kernels before Linux 2.6, this value was expressed in jiffies. Since Linux 2.6, the value is expressed in clock ticks (divide by sysconf(_SC_CLK_TCK)).*

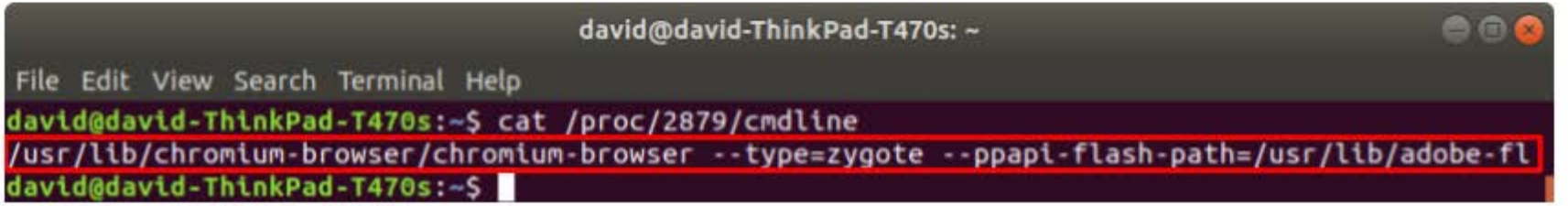
Note that the "starttime" value in this file is measured in "clock ticks". In order to convert from "clock ticks" to seconds, you must:

- #include <unistd.h>
- divide the "clock ticks" value by `sysconf(_SC_CLK_TCK)`

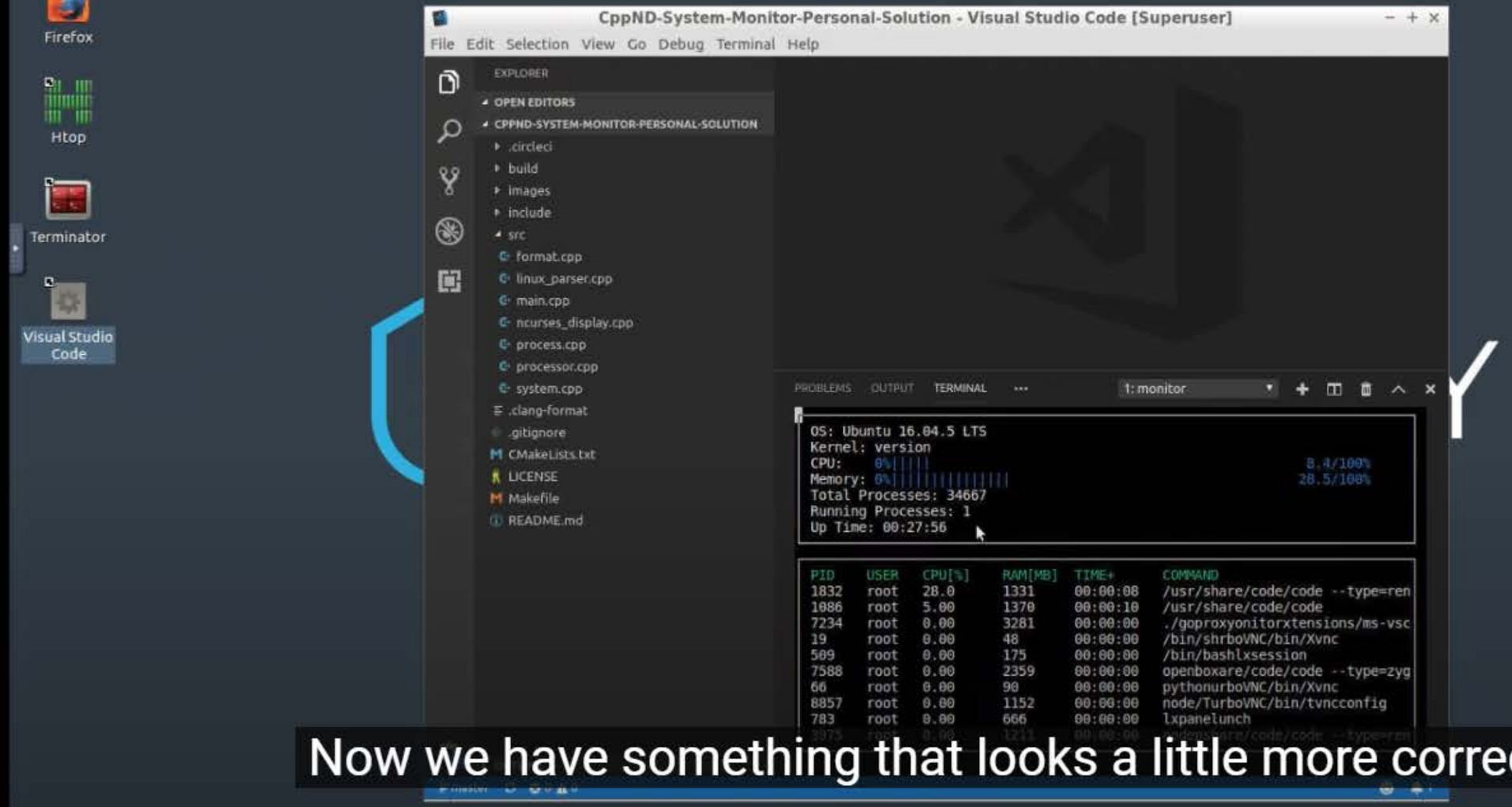
Once you have converted the time value to seconds, you can use the `Format::Time()` function from the project starter code to display the seconds in a "HH:MM:SS" format.

Command

Linux stores the command used to launch the function in the `/proc/[pid]/cmdline` file.



CPPND System Monitor Finished Project



Now we have something that looks a little more correct.

2:10 / 3:06

Udacity Workspace

You are welcome to use this Udacity Workspace to complete the project. Or you can use your own Linux development environment, if you have one.

GitHub Repo

The starter code is available on GitHub: https://github.com/udacity/CppND-System-Monitor

You can clone the starter code repository, either into the Udacity Workspace or into your own development environment, by running:

```
git clone https://github.com/udacity/CppND-System-Monitor
```

PROJECT SPECIFICATION

Process Monitor

Basic Requirements

CRITERIA	MEETS SPECIFICATIONS
The student will be able to organize code in a project structure.	The program must build an executable system monitor.
The student will be able to write warning-free code.	The program must build without generating compiler warnings.
The student will be able to create a working project.	The system monitor must run continuously without error, until the user terminates the program.
The student will be able to organize code using object oriented programming principles.	The project should be organized into appropriate classes.

System Requirements

CRITERIA	MEETS SPECIFICATIONS
The student will be able to extract and display basic data about the system.	The system monitor program should list at least the operating system, kernel version, total number of processes, number of running processes, and up time.
The student will be able to use composition.	The System class should be composed of at least one other class.

Processor Requirements

CRITERIA	MEETS SPECIFICATIONS
The student will be able to read and display data about the CPU.	The system monitor should display the CPU utilization.

Process Requirements

CRITERIA	MEETS SPECIFICATIONS
The student will be able to read and display the processes on the system.	The system monitor should display a partial list of processes running on the system.
The student will be able to display data about individual processes.	The system monitor should display the PID, user, CPU utilization, memory utilization, up time, and command for each process.

Suggestions to Make Your Project Stand Out!

Calculate CPU utilization dynamically, based on recent utilization
Sort processes based on CPU or memory utilization
Make the display interactive
Restructure the program to use abstract classes (interfaces) and pure virtual functions
Port the program to another operating system

CppND-System-Monitor

Starter code for System Monitor Project is provided on GitHub:
https://github.com/udacity/CppND-System-Monitor-Project-Updated

Follow along with the classroom lesson to complete the project!

Udacity Linux Workspace

Udacity provides a browser-based Linux Workspace for students.

You are welcome to develop this project on your local machine, and you are not required to use the Udacity Workspace. However, the Workspace provides a convenient and consistent Linux development environment we encourage you to try.

ncurses

ncurses is a library that facilitates text-based graphical output in the terminal. This project relies on ncurses for display output.

Within the Udacity Workspace, `.student_bashrc` automatically installs ncurses every time you launch the Workspace.

If you are not using the Workspace, install ncurses within your own Linux environment: `sudo apt install libncurses5-dev libncursesw5-dev`

Make

This project uses Make. The Makefile has four targets:

- `build` compiles the source code and generates an executable
- `format` applies ClangFormat to style the source code
- `debug` compiles the source code and generates an executable, including debugging symbols
- `clean` deletes the `build/` directory, including all of the build artifacts

Rubric

Before you start the project, read the project rubric.

Mentor

We suggest you schedule a check in call with your mentor before you start this project. Your mentor can help you develop a plan to successfully complete the project.

Instructions

1. Clone the project repository:
   `git clone https://github.com/udacity/CppND-System-Monitor-Project-Updated.git`
2. Build the project: `make build`
3. Run the resulting executable: `./build/monitor`
4. Follow along with the lesson.
5. Implement the `System`, `Process`, and `Processor` classes, as well as functions within the `LinuxParser` namespace.
6. Verify that your submission meets all of the criteria in the project rubric.
7. Submit!