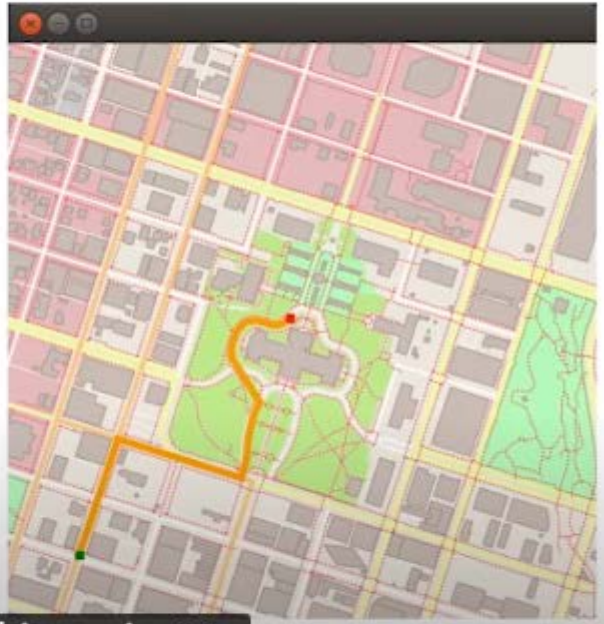
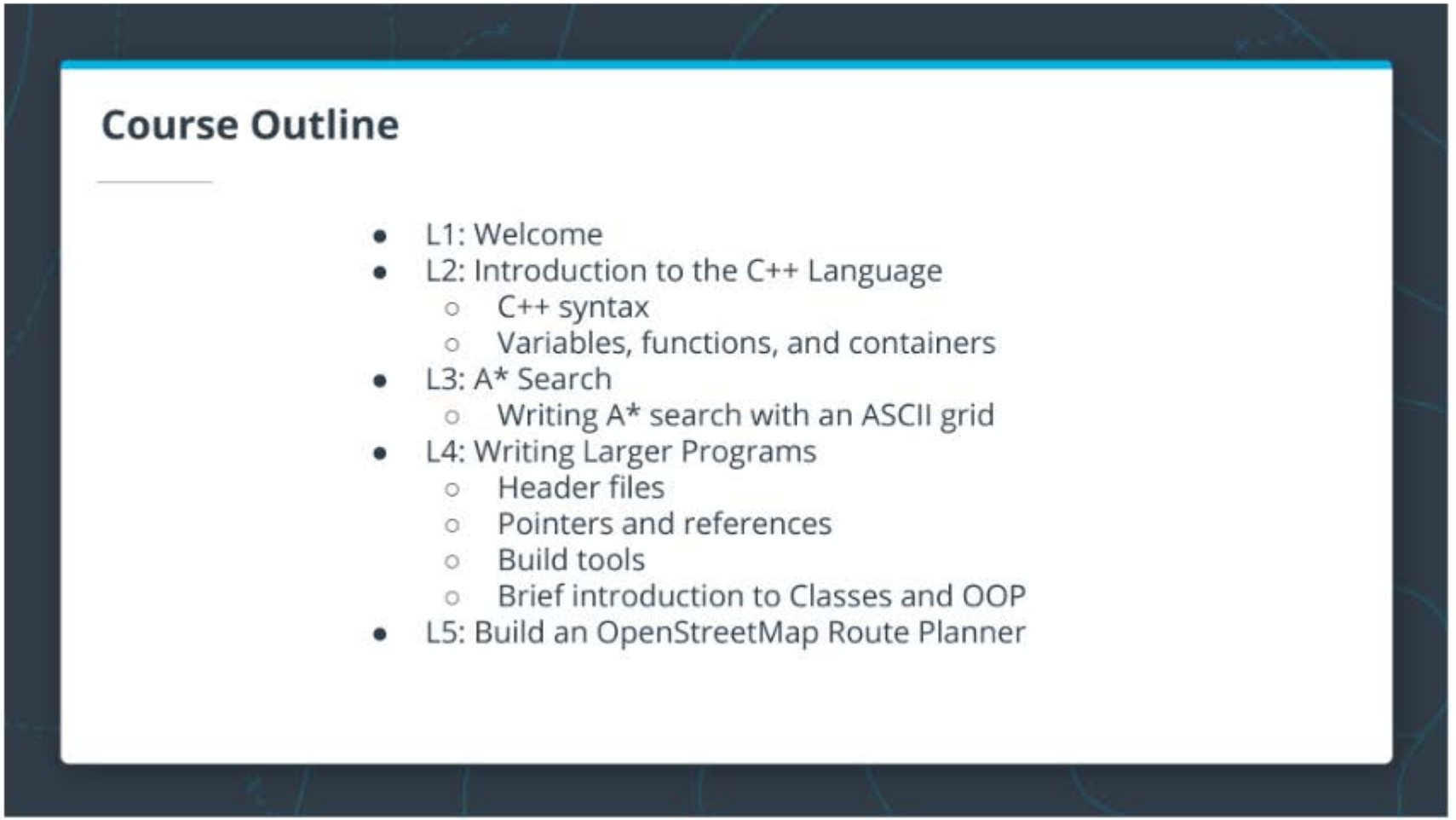


You'll be able to show this project to

Course Outline

Welcome to the first course of the C++ Nanodegree! In addition to this lesson, there are four other lessons (L2 - L4) and a project (L5). The general course outline is as follows:



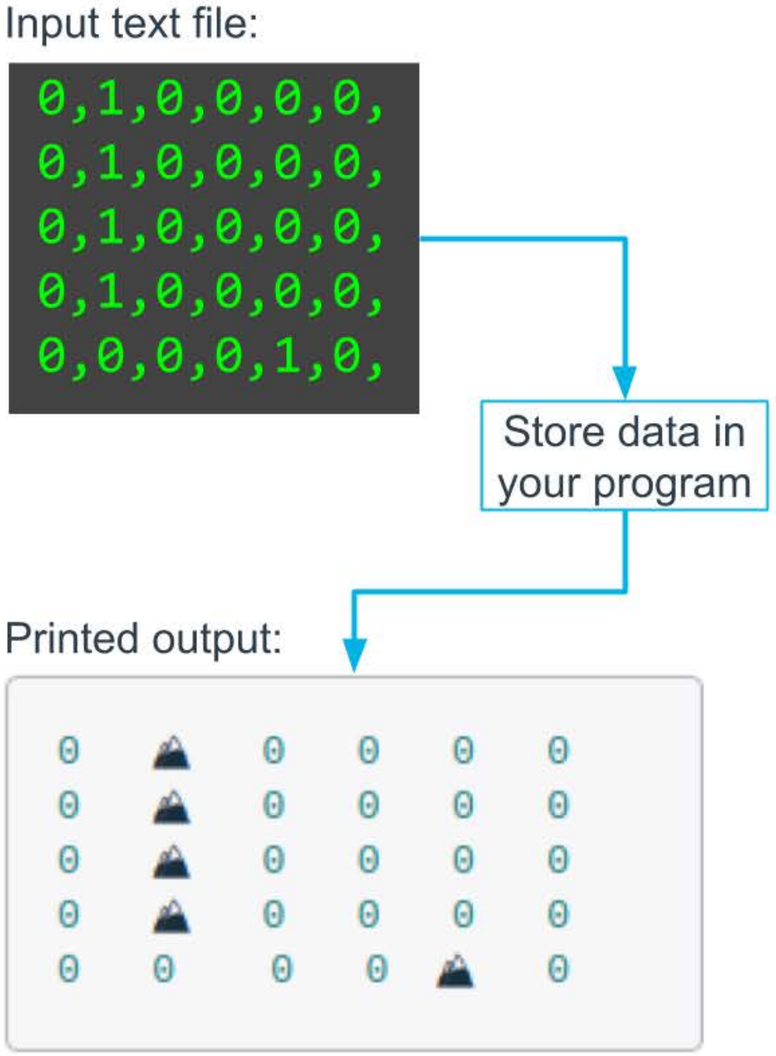
We can't wait for you to get started!


and you need to handle the complexity,
0:37 / 0:41
YouTube


as well as performance.
0:39 / 0:41
YouTube

Lesson Outline

- Getting Started:
  - Write and run your first C++ program
  - Send output to the console
- Variables and Containers:
  - Variable types
  - Vectors
  - Using auto
- Functions and Control Structures:
  - For loops
  - Functions
  - If statements and while loops
- Data Input
  - Read data from a file
  - Parse data and process strings
- Defining your own types with Enums



Write and Run Your First C++ Program

Now that you have a sense of *what* you will be building in this lesson, you can learn about the tools that you will use. To get started, it helps to know a little bit about the C++ programming language. C++ is a *compiled* language; there is a separate program - the compiler - that converts your code to an executable program that the computer can run. This means that running a new C++ program is normally a two step process:

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

C++ Main()

In C++, every program contains a `main` function which is executed automatically when the program is run. Every part of a C++ program is run directly or indirectly from `main`, and the most basic program that will compile in C++ is just a `main` function with nothing else.

`main()` should return an integer (an `int` in C++), which indicates if the program exited successfully. This is specified in code by writing the return type, followed by the `main` function name, followed by empty arguments:

```
int main()
```

The body of the `main()`, which comes after the `main` function name and arguments, is enclosed in curly brackets: `{` and `}`. In this exercise, you will write the smallest possible C++ program, which is a `main` function with empty body. If you have trouble, have a look at the `solution.cpp` file in the workspace below.

Remember that you can compile and run your program with the following:

1. To compile, use the following command: `g++ main.cpp`
2. To run, use: `./a.out`

jupyter C++ Output and Language Basics (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted C++17

Run Markdown

First Code Example

The next cell contains the first example of code that might be included in a typical C++ program. Hover your cursor over each line of the code and then click play to hear an explanation, or have a look at the **Review** section below.

In [ ]:
```
#include <iostream>
using std::cout;

int main() {
    cout << "Hello!" << "\n";
}
```

Run Code

```
root@d76c7486f76c:/home/workspace#
```

Reset

Review

`#include <iostream>`

- The `#include` is a preprocessor command which is executed before the code is compiled. It searches for the `iostream` header file and pastes its contents into the program. `iostream` contains the declarations for the input/output stream objects.

`using std::cout;`

- Namespaces are a way in C++ to group identifiers (names) together. They provide context for identifiers to avoid naming collisions. The `std` namespace is the namespace used for the standard library.
- The `using` command adds `std::cout` to the global scope of the program. This way you can use `cout` in your code instead of having to write `std::cout`.
- `cout` is an output stream you will use to send output to the notebook or to a terminal, if you are using one.
- Note that the second two lines in the example end with a semicolon `;`. Coding statements end with a semicolon in C++. The `#include` statement is a preprocessor command, so it doesn't need one.

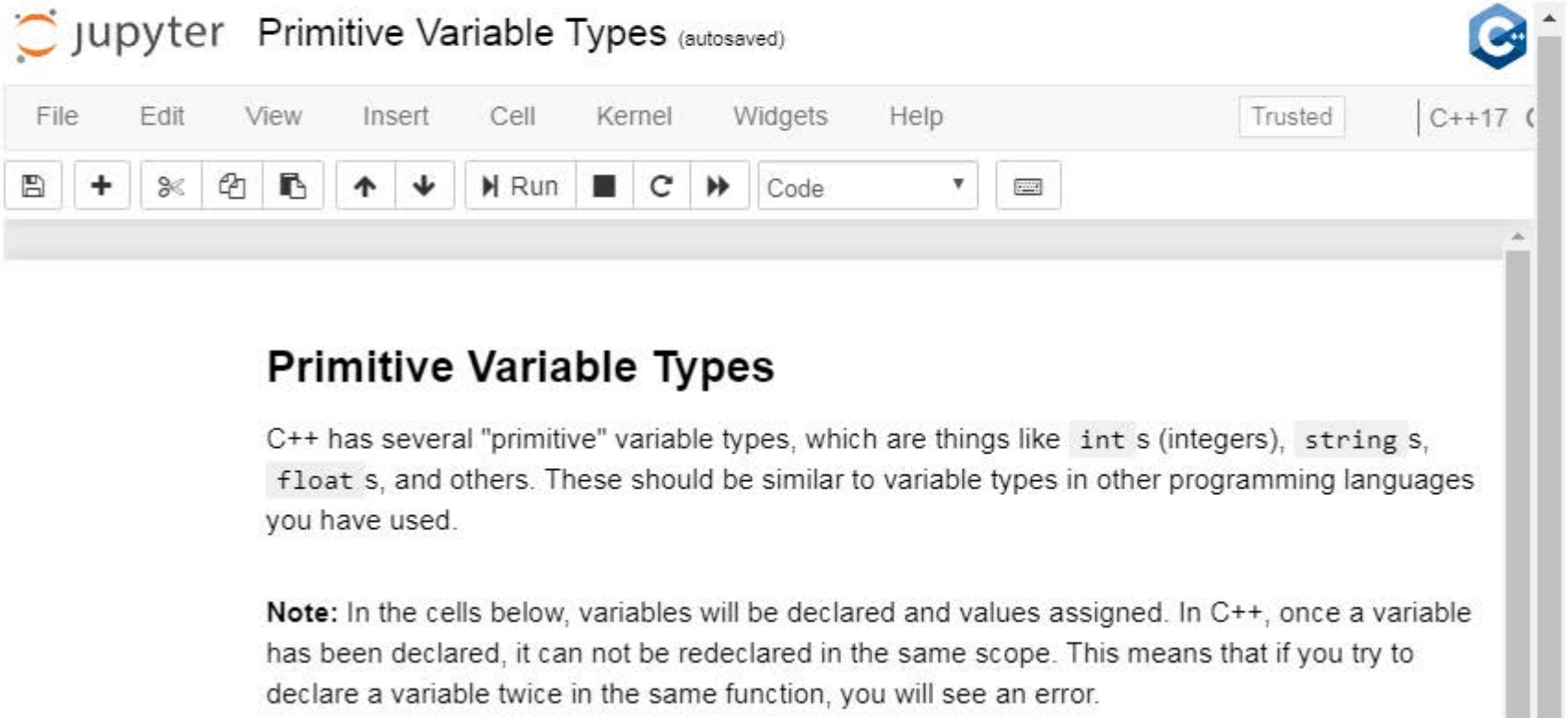
`cout << "Hello!" << "\n";`

- In this line, the code is using cout to send output to the notebook. The `<<` operator is the stream insertion operator, and it writes what's on the right side of the operator to the left side. So in this case, `"Message here"` is written to the output stream `cout`.

Bjarne Introduces C++ Types

C++ uses variables, just as in nearly every other programming language. Unlike some other languages, however, in C++ each variable has a fixed type. When a new variable is "declared", or introduced in a program, the program author must (usually) specify the variable type in the declaration.

In the notebook below, you will learn about some of the fundamental types in C++.



Primitive Variable Types

In [ ]:

```cpp
#include <iostream>
#include <string>
using std::cout;

int main() {
    // Declaring and initializing an int variable.
    int a = 9;

    // Declaring a string variable without initializing right away.
    std::string b;

    // Initializing the string b.
    b = "Here is a string";

    cout << a << "\n";
    cout << b << "\n";
}
```

Run Code | See Explanation

```
root@6bfd41b682e7:/home/workspace# g++ -std=c++17 ./code/type_example.cp
p && ./a.out
9
Here is a string
root@6bfd41b682e7:/home/workspace#
```

Reset

Practice

Practice declaring an `int` with the name `j` in the cell below and assing the value `10` to `j`.

In [ ]:
```cpp
#include <iostream>
#include <string>
using std::cout;

int main() {
    // Declare and initialize j here.
    int j = 10;
    cout<<j<<"\n";
}
```

Run Code | Show Solution

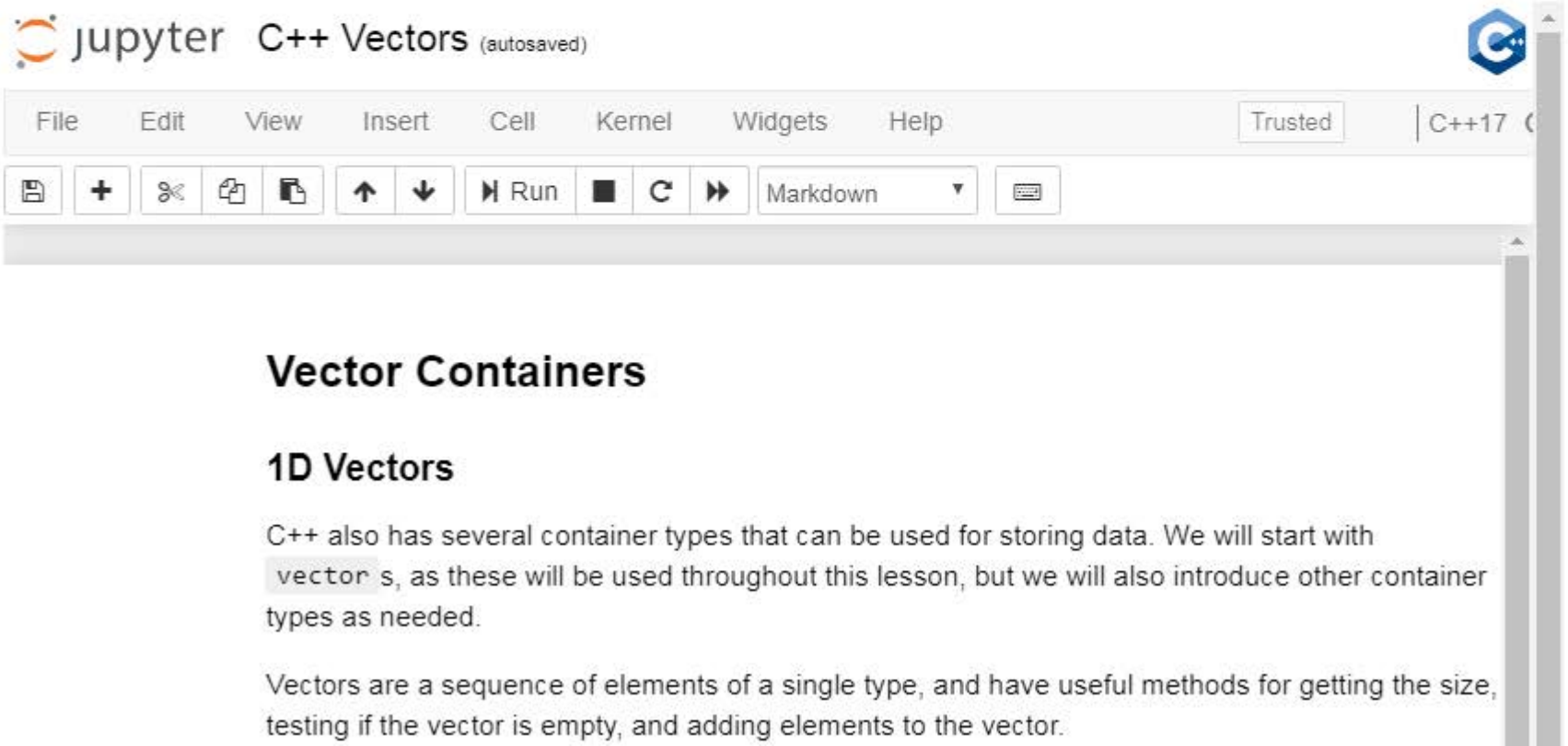
```
root@6bfd41b682e7:/home/workspace# g++ -std=c++17 ./code/type_example_2.
cpp && ./a.out
10
root@6bfd41b682e7:/home/workspace#
```

Reset

What is a Vector?

In the previous concept, you learned about some of the primitive types that C++ offers, including `string`s and `int`s, and you learned how to store these types in your program. In this concept, you will learn about one of the most common data structures in C++: the `vector`.

In the notebook below, you will learn how to declare and store a vector containing primitive types, and you will also get some practice with 2D vectors, which you will be using in A* search.



In [ ]:

```cpp
#include <iostream>
#include <vector>
using std::vector;
using std::cout;

int main() {
    // Three ways of declaring and initializing vectors.
    vector<int> v_1{0, 1, 2};
    vector<int> v_2 = {3, 4, 5};
    vector<int> v_3;
    v_3 = {6};
    cout << "Everything worked!" << "\n";
}
```

Run Code | See Explanation

```
root@7366c7d7d985:/home/workspace# g++ -std=c++17 ./code/vect_example.cp
p && ./a.out
Everything worked!
root@7366c7d7d985:/home/workspace#
```

Reset

2D Vectors

Unfortunately, there isn't a built-in way to print vectors in C++ using `cout`. You will learn how to access vector elements and you will write your own function to print vectors later. For now, you can see how vectors are created and stored. Below, you can see how to nest vectors to create 2D containers.

In [ ]:
```cpp
#include <iostream>
#include <vector>
using std::vector;
using std::cout;

int main() {
    // Creating a 2D vector.
    vector<vector<int>> v {{1,2}, {3,4}};
    cout << "Great! A 2D vector has been created." << "\n";
}
```

Run Code | See Explanation

```
root@7366c7d7d985:/home/workspace# g++ -std=c++17 ./code/vect_example_2.
cpp && ./a.out
Great! A 2D vector has been created.
root@7366c7d7d985:/home/workspace#
```

Reset

Practice

Practice declaring a `vector<int>` in the cell below, and assign the value `{6, 7, 8}`.

In [ ]:

```cpp
#include <iostream>
#include <vector>
using std::vector;
using std::cout;

int main() {
    // Declare and initialize a vector v here.
    vector<int> v{6, 7, 8};
    for(int i: v){
        cout<< i <<" ";
    }
    cout<<"\n";
}
```

Run Code | Show Solution

```
6 7 8
root@7366c7d7d985:/home/workspace#
```

Reset

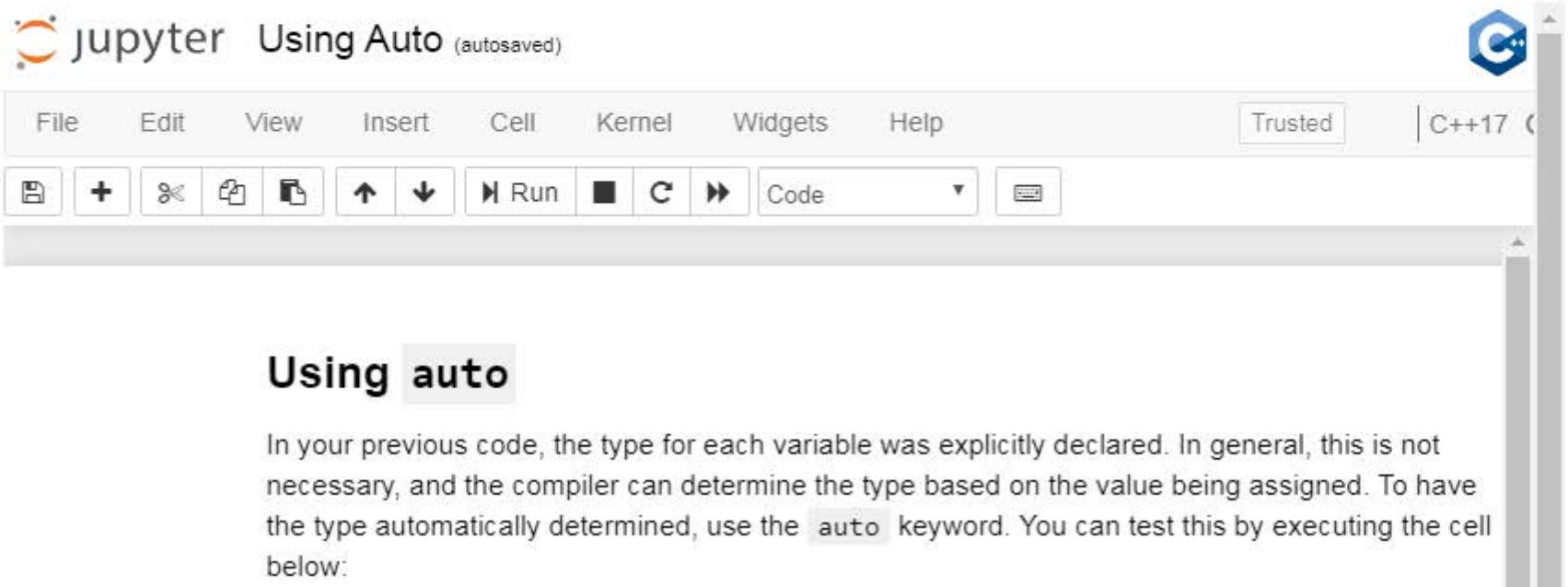

Comments should be used to say things that you cannot do well in code.
0:12 / 0:59
YouTube

You may have noticed comments in some of the code up until this point. C++ provides two kinds of comments:

```
// You can use two forward slashes for single line comments.

/*
For longer comments, you can enclose the text with an opening
slash-star and closing star-slash.
*/
```

You have now seen how to store basic types and vectors containing those types. As you practiced declaring variables, in each case you indicated the type of the variable. It is possible for C++ to do automatic type inference, using the `auto` keyword.

Have a look at the notebook below to see how this works.



In [ ]:
```
#include <iostream>
#include <vector>
using std::vector;
using std::cout;

int main() {
    auto i = 5;
    auto v_6 = {1, 2, 3};
    cout << "Variables declared and initialized without explicitly stating type
}
```

Run Code | See Explanation

```
root@7366c7d7d985:/home/workspace# g++ -std=c++17 ./code/auto_example.cp
p && ./a.out
Variables declared and initialized without explicitly stating type!
root@7366c7d7d985:/home/workspace#
```

Reset

It is helpful to manually declare the type of a variable if you want the variable type to be clear for reader of your code, or if you want to be explicit about the number precision being used; C++ has several number types with different levels of precision, and this precision might not be clear from the value being assigned.

Practice

Practice using `auto` to declare and initialize a vector `v` with the value `{7, 8, 9, 10}`. If you have trouble, **click here** for help.

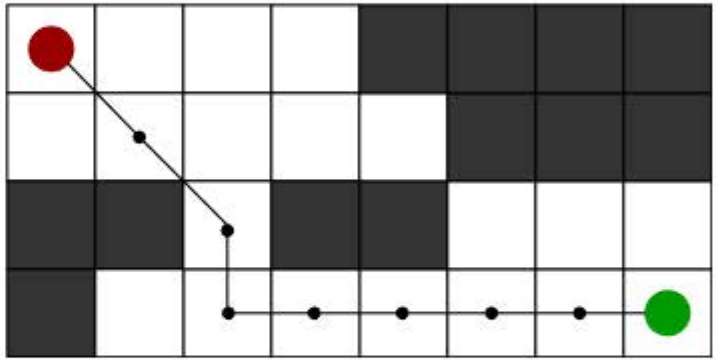
In [ ]:
```
#include <iostream>
#include <vector>
using std::vector;
using std::cout;

int main() {
    // Declare and initialize v using auto here.
    auto v = {7, 8, 9, 10};
    for(auto i : v){
        cout<< i << " ";
    }
    cout<<"\n";
}
```

Run Code | Show Solution

```
root@7366c7d7d985:/home/workspace# g++ -std=c++17 ./code/auto_example_2.
cpp && ./a.out
7 8 9 10
root@7366c7d7d985:/home/workspace#
```

Reset

Store a Grid in Your Program



In order to write the A* search algorithm, you will need a grid or "board" to search through. We'll be working with this board throughout the remaining exercises, and we'll start by storing a hard-coded board in the main function. In later exercises, you will write code to read the board from a file.

To Complete This Exercise:

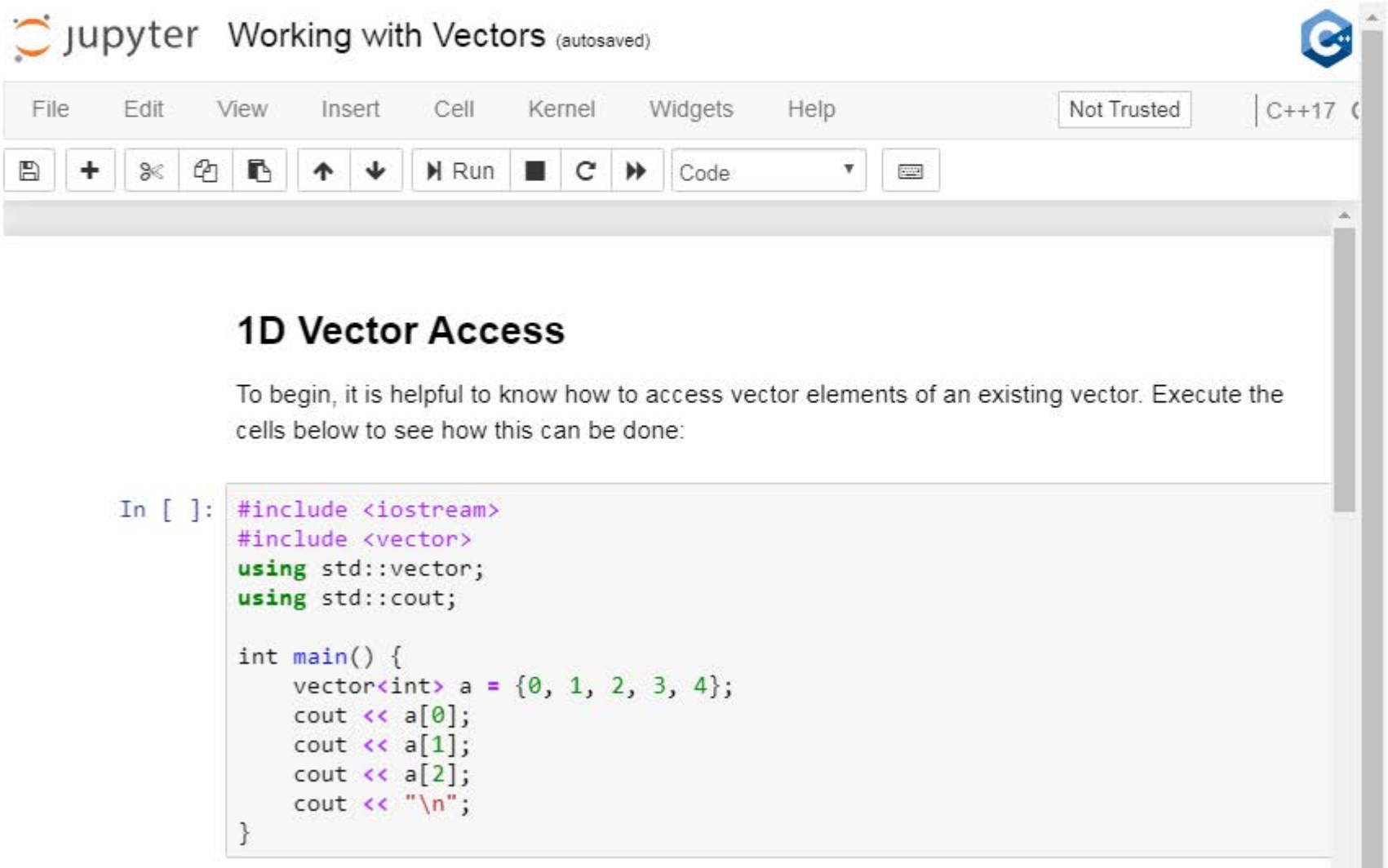
1. In the `main` function, declare a variable `board` as a vector of vectors of ints: `vector<vector<int>>`.
2. Assign this data to the board variable:

```
{{0, 1, 0, 0, 0, 0},
{0, 1, 0, 0, 0, 0},
{0, 1, 0, 0, 0, 0},
{0, 1, 0, 0, 0, 0},
{0, 0, 0, 0, 1, 0}}
```

Note: you will need to include the `vector` library, just as `iostream` is included. You will also need to use the namespace `std::vector` if you want to write `vector` rather than `std::vector` in your code.

This exercise will be ungraded, but if you get stuck, you can find the solution in `solution.cpp`. Finally, if you feel a little crowded in the editor below and need more space to work, you can click the "Expand" button in the lower left corner.

You declared and initialized vectors in a previous notebook, but in order for the vector to be useful, you will need to be able to retrieve the vector elements. You will learn about vector access in this notebook, along with some other useful vector features.



Run Code

```
root@51d75109a273:/home/workspace# g++ -std=c++17 ./code/printing_ex_1.c
pp && ./a.out
012
root@51d75109a273:/home/workspace#
```

Reset

Great! Now try accessing some of the elements of vector `a` yourself in the cell bellow:

In [ ]:

```cpp
#include <iostream>
#include <vector>
using std::vector;
using std::cout;

int main() {
    vector<int> a = {0, 1, 2, 3, 4};
    // Add some code here to access and print elements of a.
    for(int i: a){
        cout<< i;
    }
    cout << "\n" << a[10];
    cout << "\n";
}
```

Run Code | Show Solution

```
pp && ./a.out
01234
0
root@51d75109a273:/home/workspace#
```

Reset

If you tried to access the elements of `a` using an out-of-bound index, you might have noticed that there is no error or exception thrown. If you haven't seen this already, try the following code in the cell above to see what happens:

```
cout << a[10];
```

In this case, *the behavior is undefined*, so you can not depend on a certain value to be returned. Be careful about this! In a later lesson where you will learn about exceptions, we will discuss other ways to access vector elements that don't fail silently with out-of-range indices.

2D Vector Access

In the previous exercise, you stored a 2D vector - a `vector<vector<int>>`. The syntax for accessing elements of a 2D vector is very similar to accessing in a 1D vector. In the second cell below, try accessing an element of `b`. If you get stuck, click the solution button for help.

In [ ]:

```cpp
#include <iostream>
#include <vector>
using std::vector;
using std::cout;

int main() {
    vector<vector<int>> b = {{1, 1, 2, 3},
                             {2, 1, 2, 3},
                             {3, 1, 2, 3}};
    cout<< b[2][3] << "\n";
}
```

Run Code | Show Solution

```
root@51d75109a273:/home/workspace# g++ -std=c++17 ./code/printing_ex_3.cpp && ./a.out
3
root@51d75109a273:/home/workspace#
```

Reset

Getting a Vector's Length

1D Vector Length

One method of a `vector` object that will be useful in the next code exercise is the `.size()` method. This returns the length of the vector. Execute the cell below to see how this can be used:

In [ ]:
```
#include <iostream>
#include <vector>
using std::vector;
using std::cout;

int main() {
    vector<int> a = {0, 1, 2, 3, 4};

    // Print the length of vector a to the console.
    cout << a.size() << "\n";
}
```

Run Code

```
root@51d75109a273:/home/workspace# g++ -std=c++17 ./code/printing_ex_4.cpp && ./a.out
5
root@51d75109a273:/home/workspace#
```

Reset

2D Vector Length

For the `vector<vector<int>> b` defined above, try to get the size of one of the inner vectors - this should be 4. If you have trouble, click the button below for some help.

In [ ]:
```
#include <iostream>
#include <vector>
using std::vector;
using std::cout;

int main() {

    vector<vector<int>> b = {{1, 1, 2, 3},
                             {2, 1, 2, 3},
                             {3, 1, 2, 3}};
    // Print the length of an inner vector of b here.
    cout << b[0].size() <<"\n";
    cout << b[1].size() <<"\n";
    cout << b[2].size() <<"\n";
}
```

Run Code | Show Solution

```
pp && ./a.out
4
4
4
root@51d75109a273:/home/workspace#
```

Reset

Nice work! You now know a little more about C++ vectors. After learning about for loops, you should be well prepared for the upcoming code exercises.

Just as in other languages you've worked with, C++ has both `for` loops and `while` loops. You will learn about `for` loops in the notebook below, and you will see `while` loops later in the course.

File Not Trusted C++17

For Loop with an Index Variable

A simple `for` loop using an index variable has the following syntax. Click the button below for an explanation of the different parts.

In [ ]:
```
#include <iostream>
using std::cout;

int main() {
    for (int i=0; i < 5; i++) {
      cout << i << "\n";
    }
}
```

Run Code See Explanation

```
1
2
3
4
root@75c5f07c8383:/home/workspace#
```

Reset

The Increment Operator

If you haven't seen the `++` operator before, this is the *post-increment operator*, and it is where the `++` in the name "C++" comes from. The operator increments the value of `i`.

There is also a *pre-increment operator* which is used before a variable, as well as *pre* and *post decrement* operators: `--`. The difference between *pre* and *post* lies in what value is returned by the operator when it is used.

You will only use the *post-increment operator* `i++` for now, but if you are curious, click below for an explanation of the code:

In [ ]:
```
#include <iostream>
using std::cout;

int main() {
    auto i = 1;

    // Post-increment assigns i to c and then increments i.
    auto c = i++;

    cout << "Post-increment example:" << "\n";
    cout << "The value of c is: " << c << "\n";
    cout << "The value of i is: " << i << "\n";
    cout << "\n";

    // Reset i to 1.
    i = 1;

    // Pre-increment increments i, then assigns to c.
    c = ++i;
```

```
    cout << "Pre-increment example:" << "\n";
    cout << "The value of c is: " << c << "\n";
    cout << "The value of i is: " << i << "\n";
    cout << "\n";

    // Decrement i;
    i--;
    cout << "Decrement example:" << "\n";
    cout << "The value of i is: " << i << "\n";
}
```

Run Code | See Explanation

```
The value of i is: 1
root@75c5f07c8383:/home/workspace# g++ -std=c++17 ./code/printing_ex_7.cpp && ./a.out
```

Reset

Practice

Before you learn how to write a `for` loop using an iterator, practice writing a for loop that prints values from `-3` through `10` in the cell below. Don't forget to assign an initial value (like 0) to your index variable!

In [ ]:
```cpp
#include <iostream>
using std::cout;

int main() {
    // Add your code here.
    for(int i=-3; i<11; i++){
        cout<< i <<" ";
    }
    cout<<"\n";
}
```

Run Code | Show Solution

```
 && ./a.out
-3 -2 -1 0 1 2 3 4 5 6 7 8 9 10
root@75c5f07c8383:/home/workspace#
```

Reset

For Loop with a Container

C++ offers several ways to iterate over containers. One way is to use an index-based loop as above. Another way is using a "range-based loop", which you will see frequently in the rest of this course. See the following code for an example of how this works:

In [ ]:

```cpp
#include <iostream>
#include <vector>
using std::cout;
using std::vector;

int main() {
    // Add your code here.
    vector<int> a {1, 2, 3, 4, 5};
    for (int i: a) {
        cout << i << "\n";
    }
}
```

Run Code | See Explanation

```
2
3
4
5
root@75c5f07c8383:/home/workspace#
```

Reset

Challenge

In the next cell, try to write a double range-based for loop that prints all of the entries of the 2D vector `b`. If you get stuck, click on the solution button for an explanation.

In [ ]:

```cpp
#include <iostream>
#include <vector>
using std::cout;
using std::vector;

int main() {
    // Add your code here.
    vector<vector<int>> b {{1, 2},
                           {3, 4},
                           {5, 6}};

    // Write your double loop here.
    for (vector<int> i: b) {
        for (int j: i){
            cout<<j<<" ";
        }
        cout<<"\n";
    }
}
```

Run Code | Show Solution

```
root@75c5f07c8383:/home/workspace# g++ -std=c++17 ./code/for_loop_ex_3.cpp && ./a.out
```

Reset

Functions

The last thing you will need to learn in order to complete the next exercise is how to write a function. Fortunately, you have seen a function before when you wrote `main()`!

When a function is declared and defined in a single C++ file, the basic syntax is as follows:

```
return_type FunctionName(parameter_list) {
  // Body of function here.
}
```

On to the Exercise

Excellent work! You have learned quite a lot in the last few concepts, including:

- Accessing elements of a vector and getting the vector's size.
- How `for` loops work in C++, using iterators and range-based loops.
- Increment (and decrement) operators.
- How to write your own functions.

In the next exercise, you will write two `for` loops to print the contents of a 2D vector so you will be able to print the grid in your project!

Print the Board

Now that you have a board stored in your program, you'll need a way to print it out so you can display the results of your project. In this exercise, you will add a `PrintBoard` function to print the board one row at a time. When you are done, the printed output should look like:

```
010000
010000
010000
010000
000010
```

To Complete This Exercise:

- Write a `void PrintBoard` function. The function should accept the board as an argument. The function should print each row of the board with a newline `"\n"`.
- When you have written your `PrintBoard` function, call it from within `main()` to print the board.

Now that you are able to print the board in your program, you will make the program more flexible by reading the board from a file. This will allow you to run your program with different board files to see the results.

Before you can read the contents of a file into your program, you'll need to learn the syntax for just a couple more parts of the C++ language: `if` statements and `while` loops.

Great! Now that you have some practice with `if` and `while` statements, you are ready to read data from a file into your C++ program. In the following notebook and the next exercise, you are going to write code to read a file, line by line. Have a look below for step-by-step instructions on how to do this.

CODE: Read the Board from a File

SEND FEEDBACK

Until now, the board has been declared and initialized in the `main()` function. As discussed in the previous notebook, you will need a function to read the board in from another file in order to make the program a little more flexible and user-friendly.

The first step in this process will be to write a `ReadBoardFile` function that reads in the file and prints each line to `cout`. The output should look like the `1.board` file, which can be opened in the editor below:

```
0,1,0,0,0,0,
0,1,0,0,0,0,
0,1,0,0,0,0,
0,1,0,0,0,0,
0,0,0,0,1,0,
```

Now that the board is being read into your program line by line, you will want to process each line and store the data, rather than just streaming it to `cout`. There are many ways to do this in C++, but we will focus on `istringstream` from the `<sstream>` header file.

Trusted | C++17

Streaming `int`s from a `string` with istringstream

In C++ strings can be streamed into temporary variables, similarly to how files can be streamed into strings. Streaming a string allows us to work with each character individually.

One way to stream a string is to use an input string stream object `istringstream` from the `<sstream>` header.

Once an `istringstream` object has been created, parts of the string can be streamed and stored using the "extraction operator": `>>`. The extraction operator will read until whitespace is reached or until the stream fails. Execute the following code to see how this works:

In [ ]:
```
#include <iostream>
#include <sstream>
#include <string>

using std::istringstream;
using std::string;
using std::cout;
```

```cpp
int main ()
{
    string a("1 2 3");

    istringstream my_stream(a);

    int n;
    my_stream >> n;
    cout << n << "\n";
}
```

Compile & Execute | Explain

```
root@e12b17110ba6:/home/workspace# g++ -std=c++17 ./code/string_st
ream.cpp && ./a.out
1
root@e12b17110ba6:/home/workspace#
```

Reset

The `istringstream` object can also be used as a boolean to determine if the last extraction operation failed - this happens if there wasn't any more of the string to stream, for example. If the stream still has more characters, you are able to stream again. See the following code for an example of using the `istringstream` this way:

In [ ]:

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

using std::istringstream;
using std::string;
using std::cout;

int main()
{
    string a("1 2 3");

    istringstream my_stream(a);

    int n;

    // Testing to see if the stream was successful and printing results.
    while (my_stream) {
        my_stream >> n;
        if (my_stream) {
            cout << "That stream was successful: " << n << "\n";
        }
        else {
            cout << "That stream was NOT successful!" << "\n";
        }
    }
}
```

Compile & Execute | Explain

```
That stream was successful: 1
That stream was successful: 2
That stream was successful: 3
That stream was NOT successful!
root@e12b17110ba6:/home/workspace#
```

Reset

The extraction operator `>>` writes the stream to the variable on the right of the operator and returns the `istringstream` object, so the entire expression `my_stream >> n` is an `istringstream` object and can be used as a boolean! Because of this, a common way to use `istringstream` is to use the entire extraction expression in a while loop as follows:

In [ ]:
```
#include <iostream>
#include <sstream>
#include <string>

using std::istringstream;
using std::string;
using std::cout;

int main () {
    string a("1 2 3");

    istringstream my_stream(a);

    int n;

    while (my_stream >> n) {
      cout << "That stream was successful: " << n << "\n";
    }
    cout << "The stream has failed." << "\n";

}
```

Compile & Execute | Explain

```
tream_3.cpp && ./a.out
That stream was successful: 1
That stream was successful: 2
That stream was successful: 3
The stream has failed.
root@e12b17110ba6:/home/workspace#
```

Reset

Strings with Mixed Types

In the stream example above, the string contained only whitespaces and characters which could be converted to `int` s. If the string has mixed types, more care is needed to process the string. In the following example, the type `char` is used, which is a type that can hold only a single ASCII character.

In [ ]:

```
#include <iostream>
#include <sstream>
#include <string>

using std::istringstream;
using std::string;
using std::cout;

int main()
{
    string b("1,2,3");

    istringstream my_stream(b);

    char c;
    int n;
```

```
    while (my_stream >> n >> c) {
      cout << "That stream was successful:" << n << " " << c << "\n";
    }
    cout << "The stream has failed." << "\n";
}
```

Compile & Execute | Explain

```
ream_example_4.cpp && ./a.out
That stream was successful:1 ,
That stream was successful:2 ,
The stream has failed.
root@e12b17110ba6:/home/workspace#
```

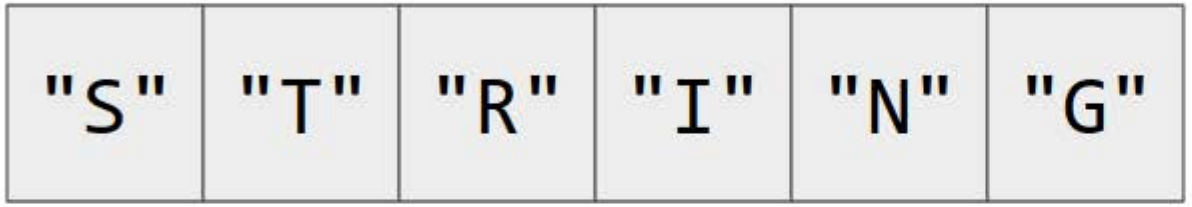
Reset

In that example, notice that the `3` was not printed! The expression:

```
my_stream >> n >> c
```

tried to stream an `int` followed by a `char`. Since there was no `char` after the `3`, the stream failed and the `while` loop exited.

Parse Lines from the File



Now that you are able to read a board line by line from a file, you will want to parse these lines and store them in a `vector<int>`. In this exercise, you will focus on a helper function to do this string parsing.

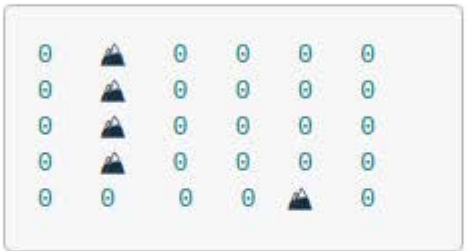
We have included a test in the `main` to ensure that the function is working correctly. If everything works, you should see:

```
---------------------------------------------------
TestParseLine: passed
---------------------------------------------------
```

Use the ParseLine Function

Great! With the `ParseLine` function complete, you can now use it in the `ReadBoardFile` to process each line of the incoming file. In this exercise, you will do just that: follow the TODOs in the code below, updating both `ReadBoardFile` and the `main` function to finish the processing of an external board file.

Formatting the Printed Board



In the previous exercises, you stored and printed the board as a `vector<vector<int>>`, where only two states were used for each cell: `0` and `1`. This is a great way to get started, but as the program becomes more complicated, there will be more than two possible states for each cell. Additionally, it would be nice to print the board in a way that clearly indicates open areas and obstacles, just as the board is printed above.

To do this clearly in your code, you will learn about and use something called an `enum`. An `enum`, short for enumerator, is a way to define a type in C++ with values that are restricted to a fixed range. For an explanation and examples, see the notebook below.

Formatting the Printed Board

```
0   0   0   0   0   0
0   0   0   0   0   0
0   0   0   0   0   0
0   0   0   0   0   0
0   0    0   0  0   0
```

The board will eventually have more than two cell states as the program becomes more complicated, and it would be nice to add formatting to the printed output of the board to ensure readability as the number of board states increases.

To accommodate more board states and facilitate print formatting, we have provided the `State` enum with enumerator values `kEmpty` and `kObstacle`. In this exercise, you will write a `CellString` function which converts each `State` to an appropriate string. In the next exercise, we will update the program to use the `enum` values and `CellString` function.

Store the Board Using the State Enum

```
0   0   0   0   0   0
0   0   0   0   0   0
0   0   0   0   0   0
0   0   0   0   0   0
0   0    0   0  0   0
```

Fantastic work! Now that you have a way to print the `State` `enum` values, you will be able to modify your program to use `State` values in the board exclusively. To do this, you will need to modify the return types and variable types in several places of the code. Don't worry, as we have clearly marked these with a `TODO` in each part of the code.

After this exercise, you will have completed the first part of this lesson, and you will begin coding the main A* search algorithm!

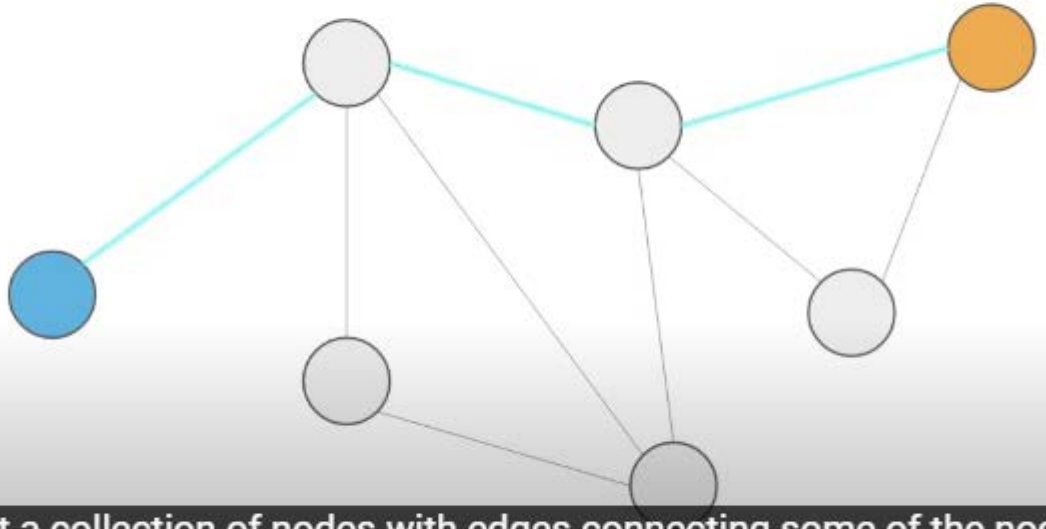
Lesson Recap

- Compile and run a simple C++ program
- Send output to the terminal
- Variables and containers
  - Variable types
  - Vectors
  - auto
- Functions and control structures
  - Conditionals
  - Loops
  - Functions
- Data Input
  - Read data from a file
  - Parse data and process strings
- Define your own types with enums

Lesson Recap

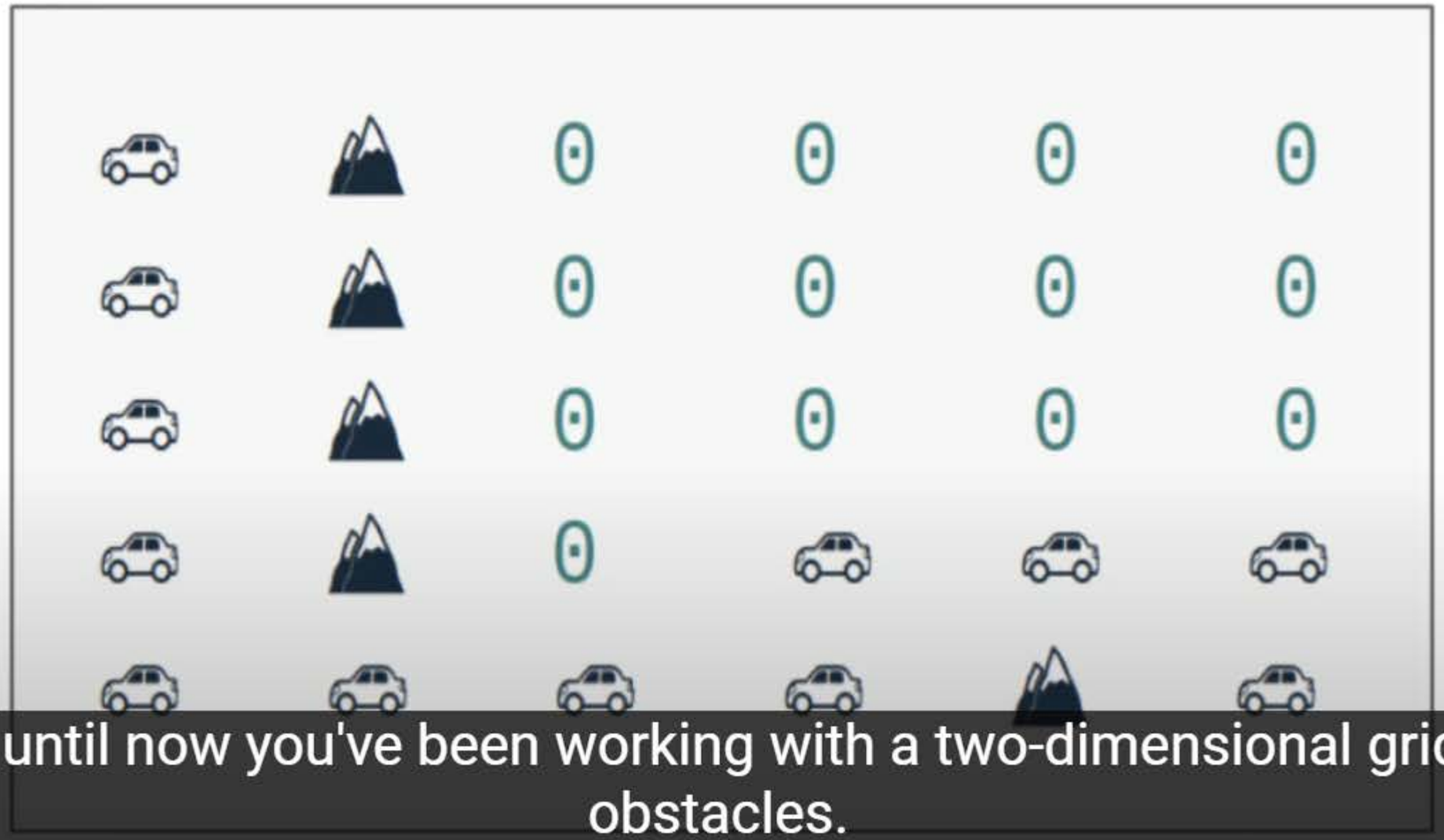
You defined your own types using an enum to add ascii character formatting to the output.

A* Search



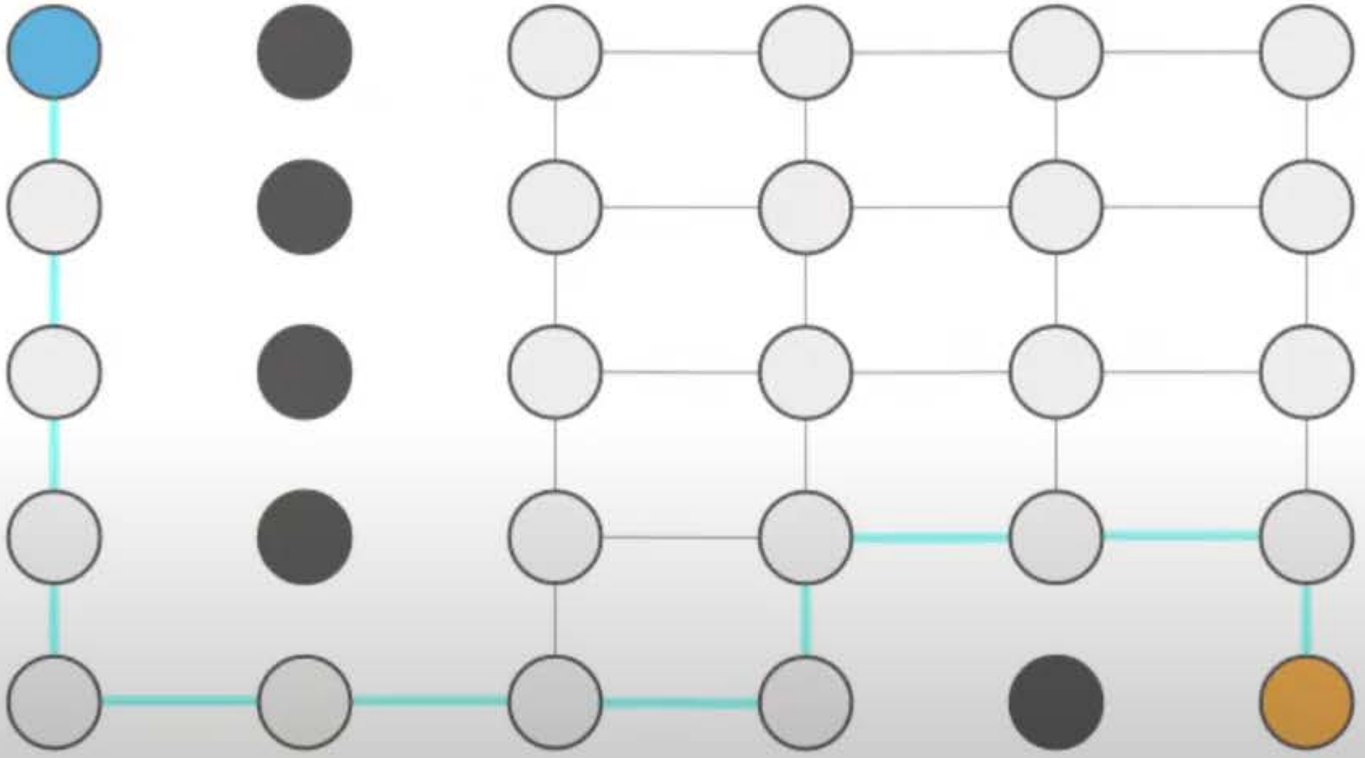
just a collection of nodes with edges connecting some of the nodes.

A* Search



In fact, up until now you've been working with a two-dimensional grid that has obstacles.

A* Search



edges connecting each pair of nodes that aren't separated by an obstacle.

A* Search

Input text file:

```
0,1,0,0,0,0,
0,1,0,0,0,0,
0,1,0,0,0,0,
0,1,0,0,0,0,
0,0,0,0,1,0,
```

Store data in your program

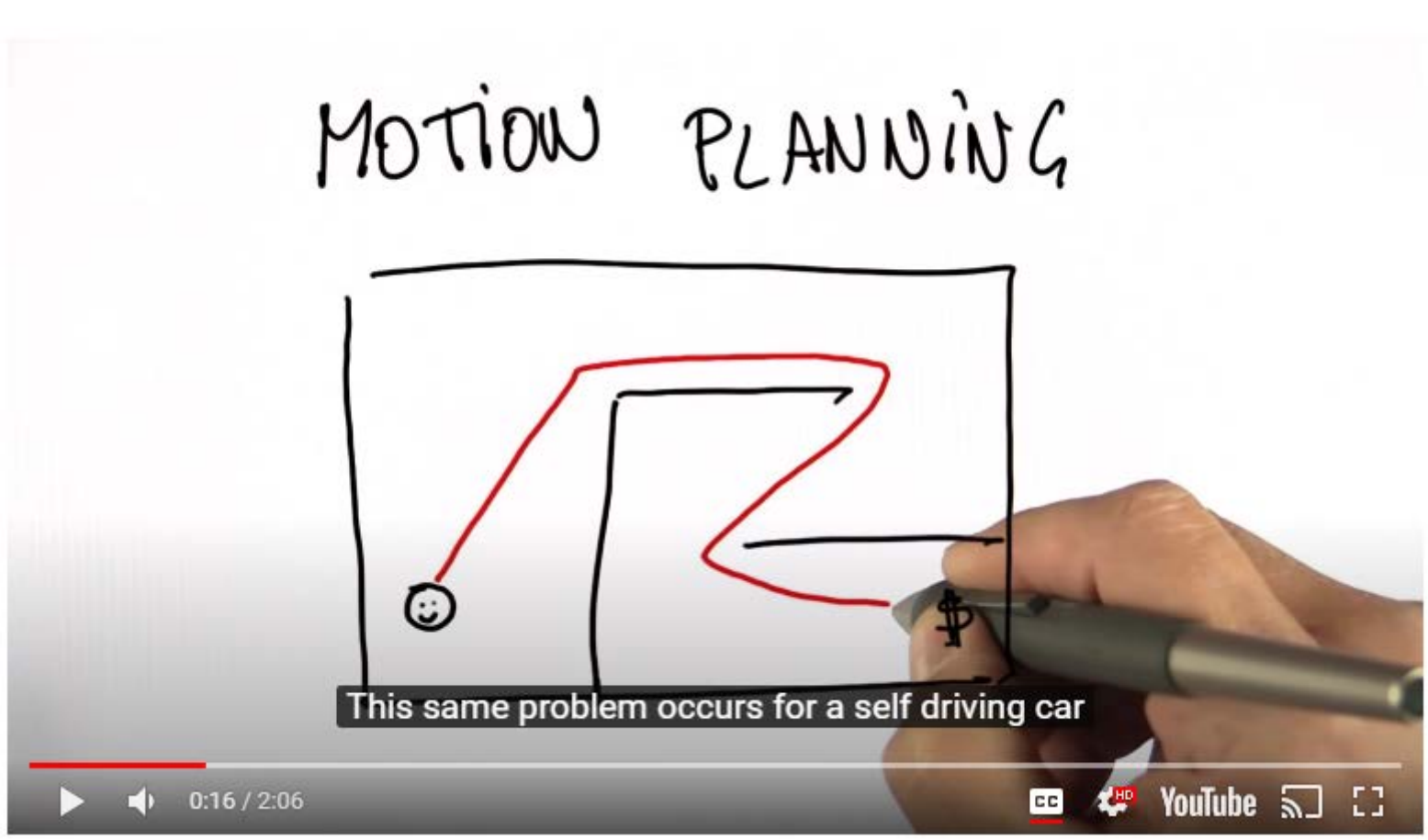
Find a path using A* search

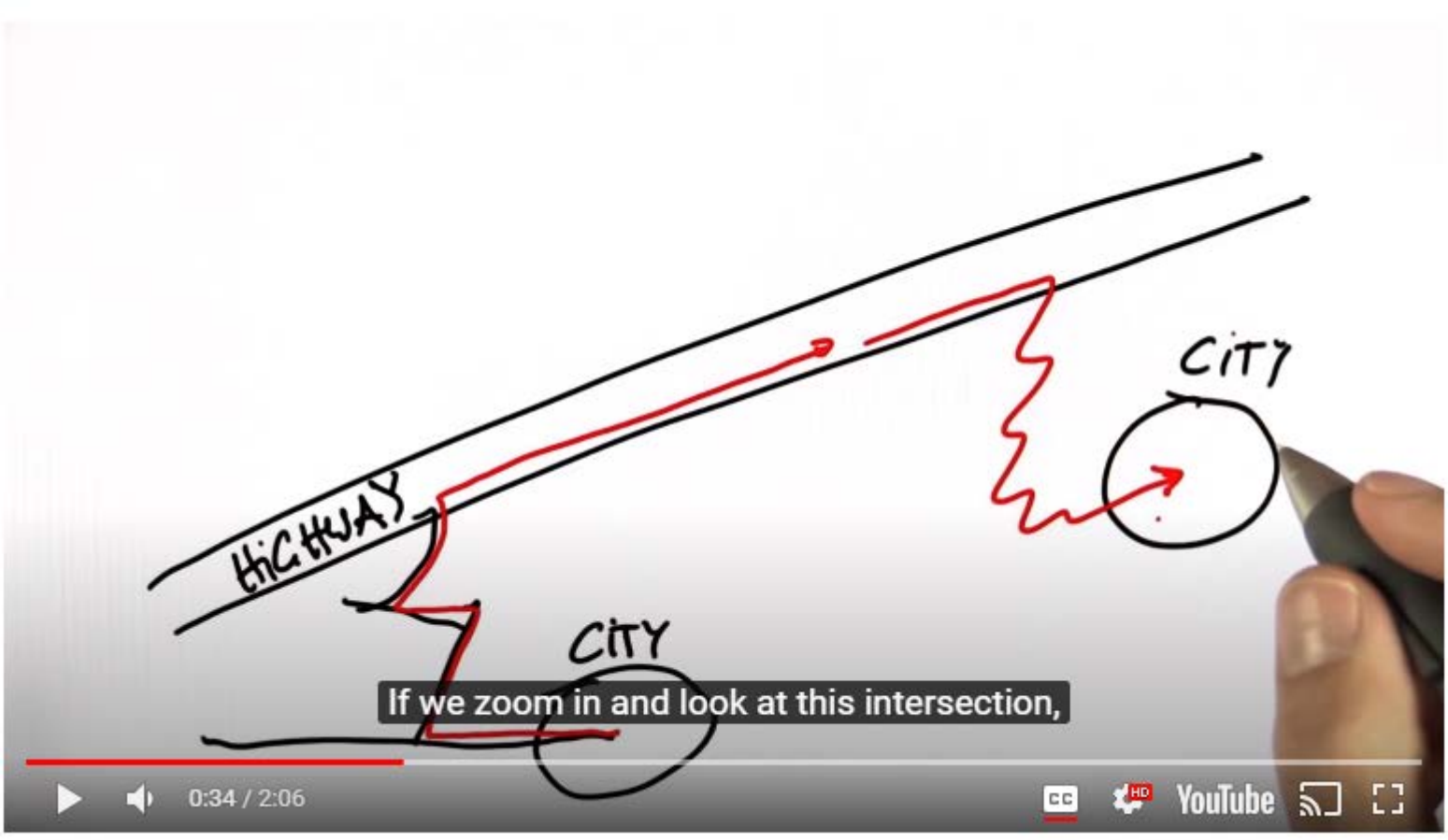
Printed output:

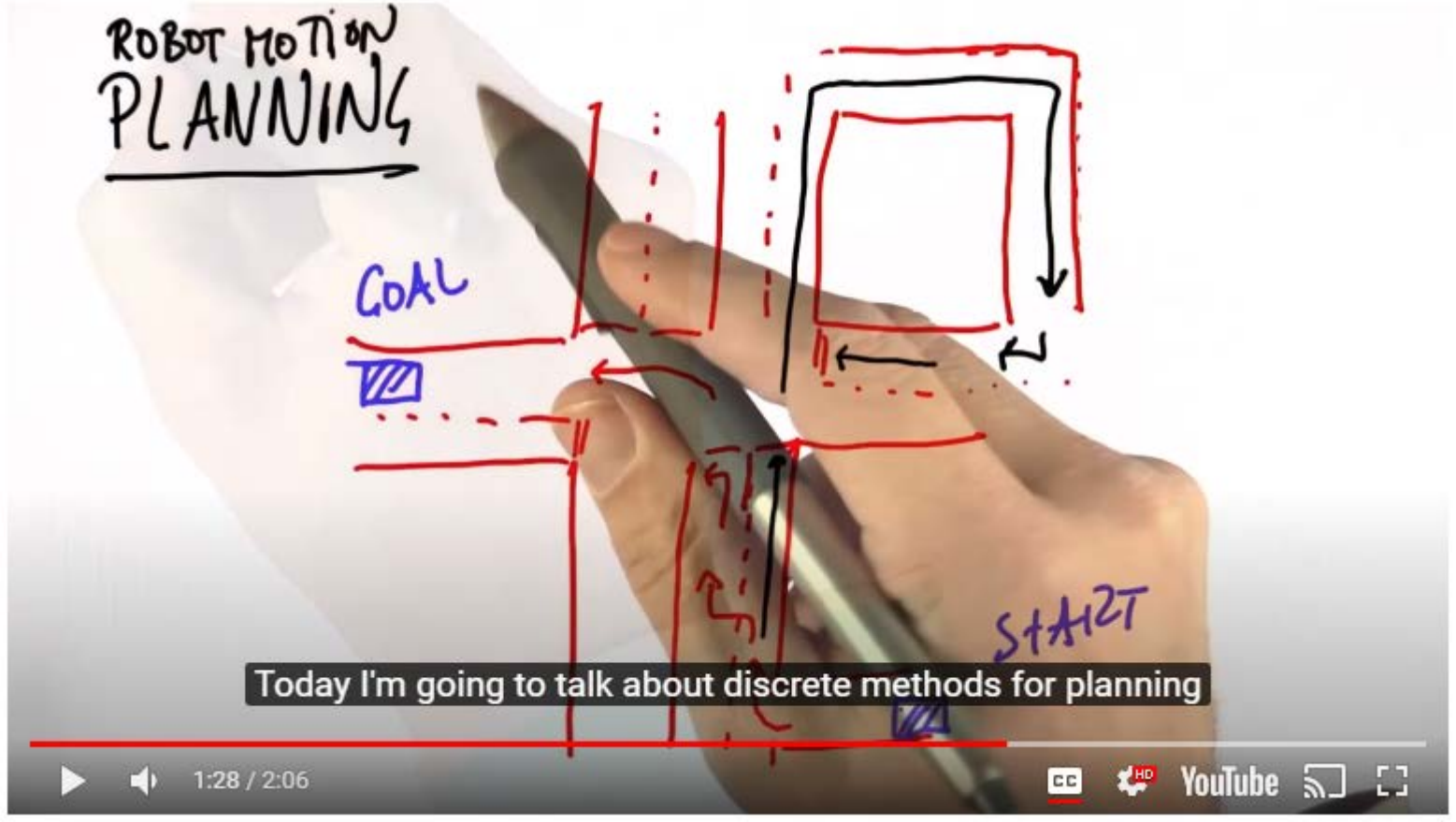
and that will complete your program.

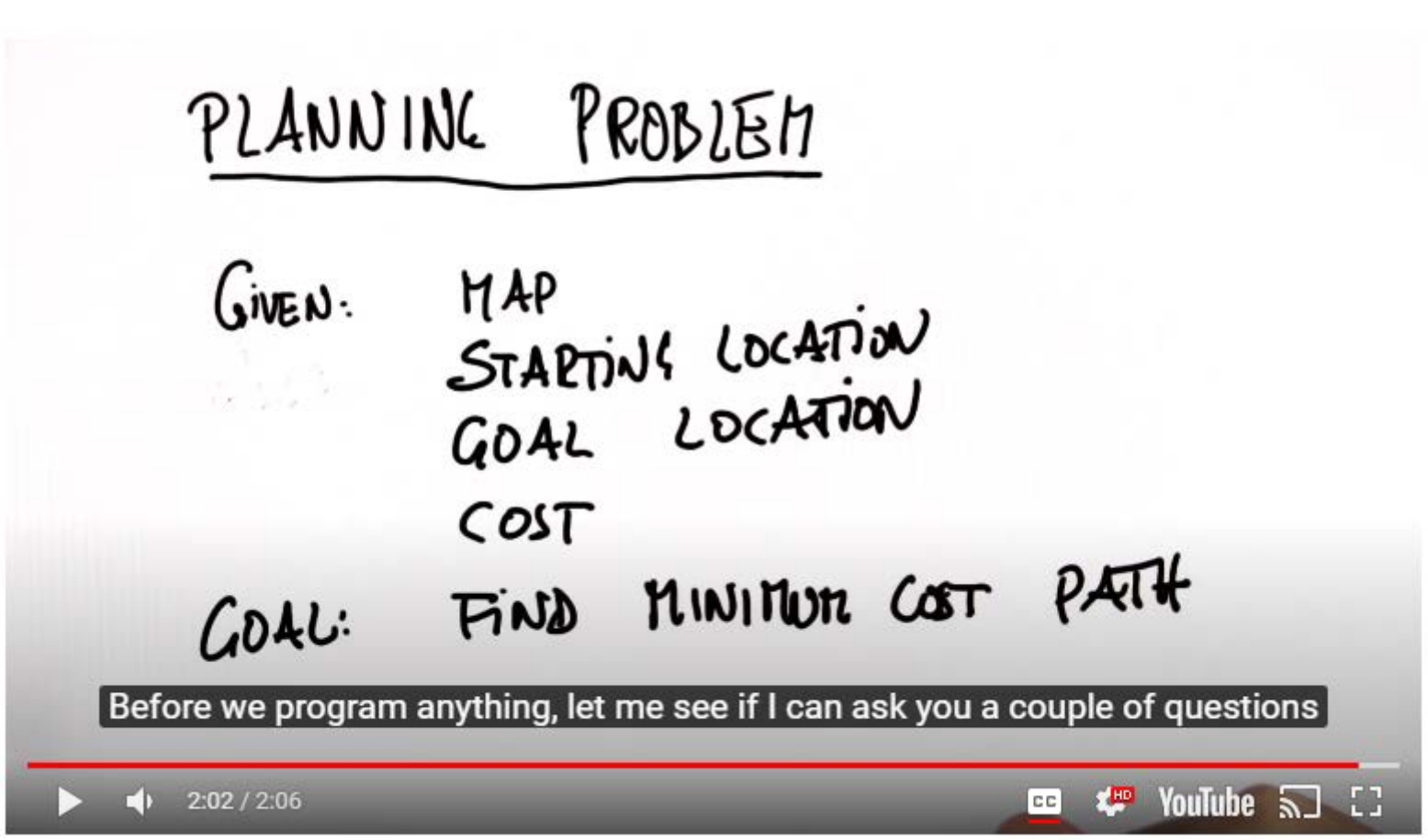
Lesson Outline

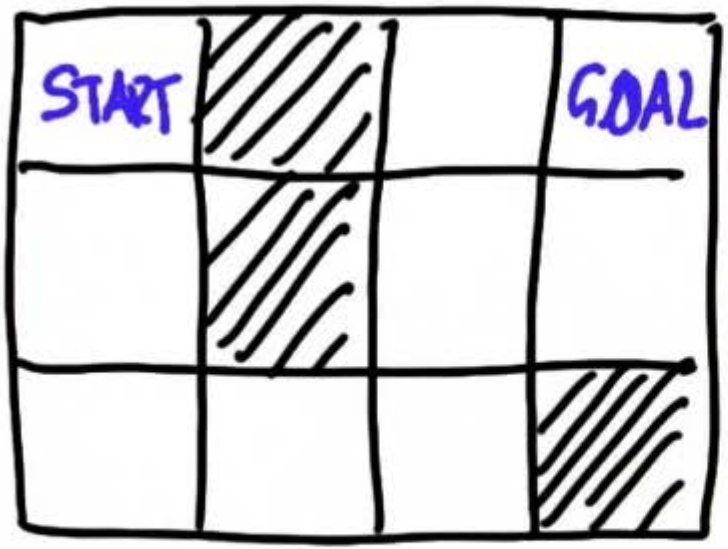
- Introduction To Search
  - Sebastian Thrun will explain the A* search algorithm
- Program A* Search
  - Algorithm will be broken down into several exercises for you
  - Algorithm summary and structure of code for exercises
- C++ Features
  - Pass-by-reference
  - const and constexpr
  - Arrays

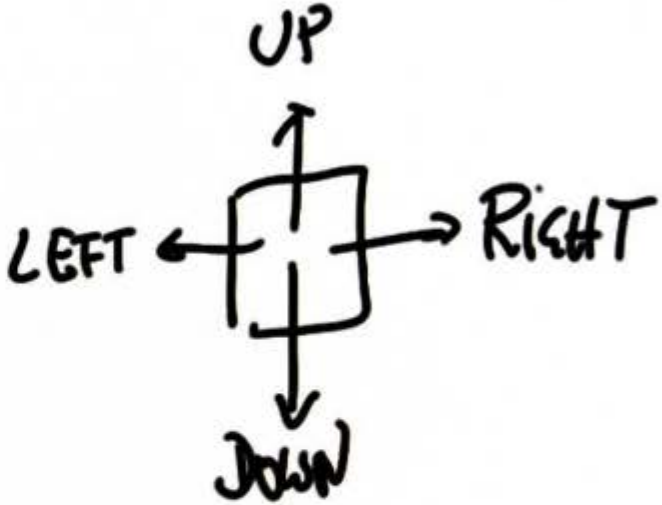

MOTION PLANNING
This same problem occurs for a self driving car
0:16 / 2:06
YouTube

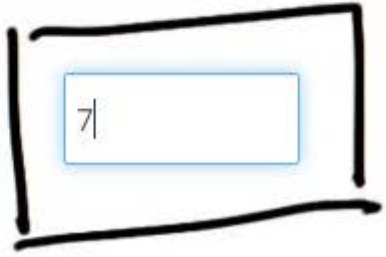

HIGHWAY
CITY
CITY
If we zoom in and look at this intersection,
0:34 / 2:06
YouTube


ROBOT MOTION
PLANNING
GOAL
START
Today I'm going to talk about discrete methods for planning
1:28 / 2:06
YouTube

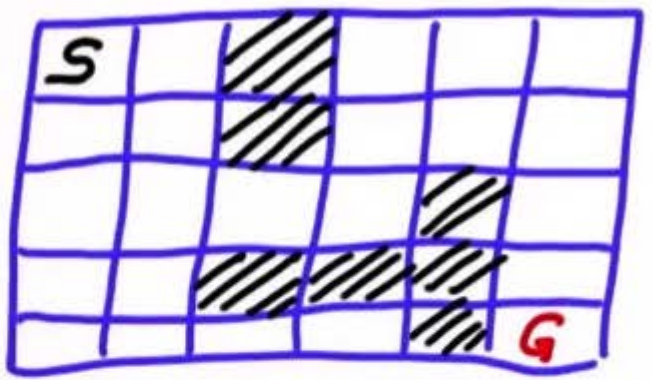

PLANNING PROBLEM
GIVEN:
MAP
STARTING LOCATION
GOAL LOCATION
COST
GOAL:
FIND MINIMUM COST PATH
Before we program anything, let me see if I can ask you a couple of questions
2:02 / 2:06
YouTube


START
GOAL


UP
LEFT
RIGHT
DOWN


7

SEARCH - PATH PLANNING



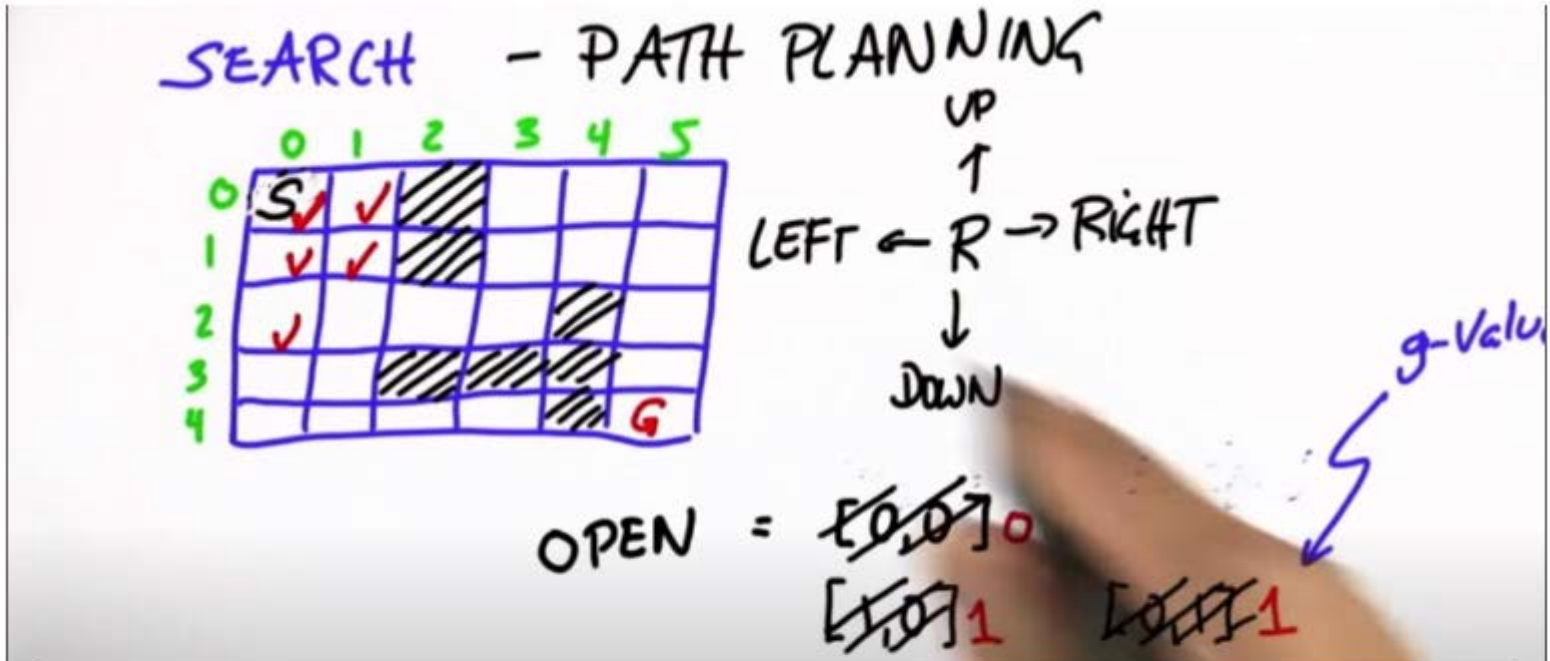
UP

LEFT ← R → RIGHT

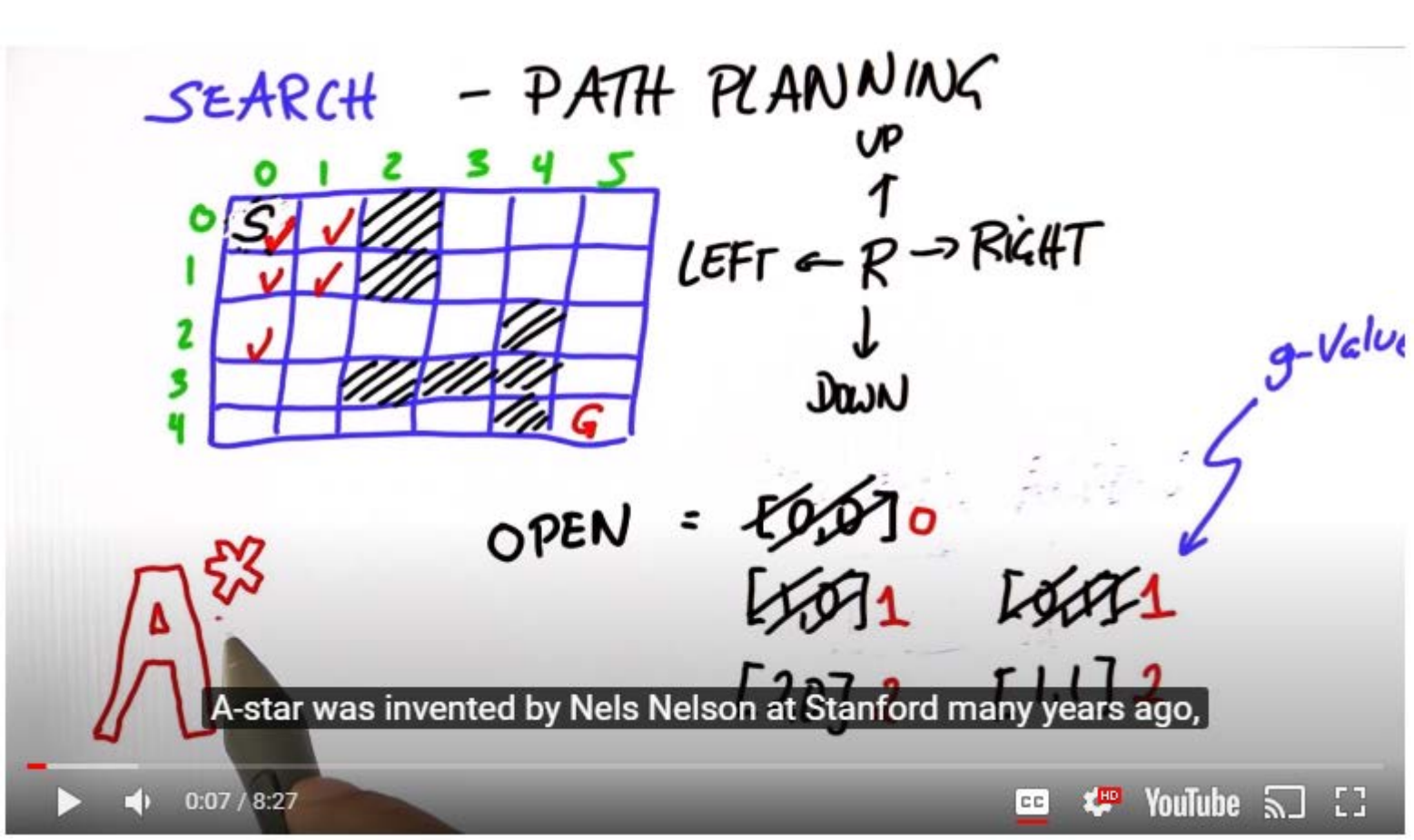
DOWN

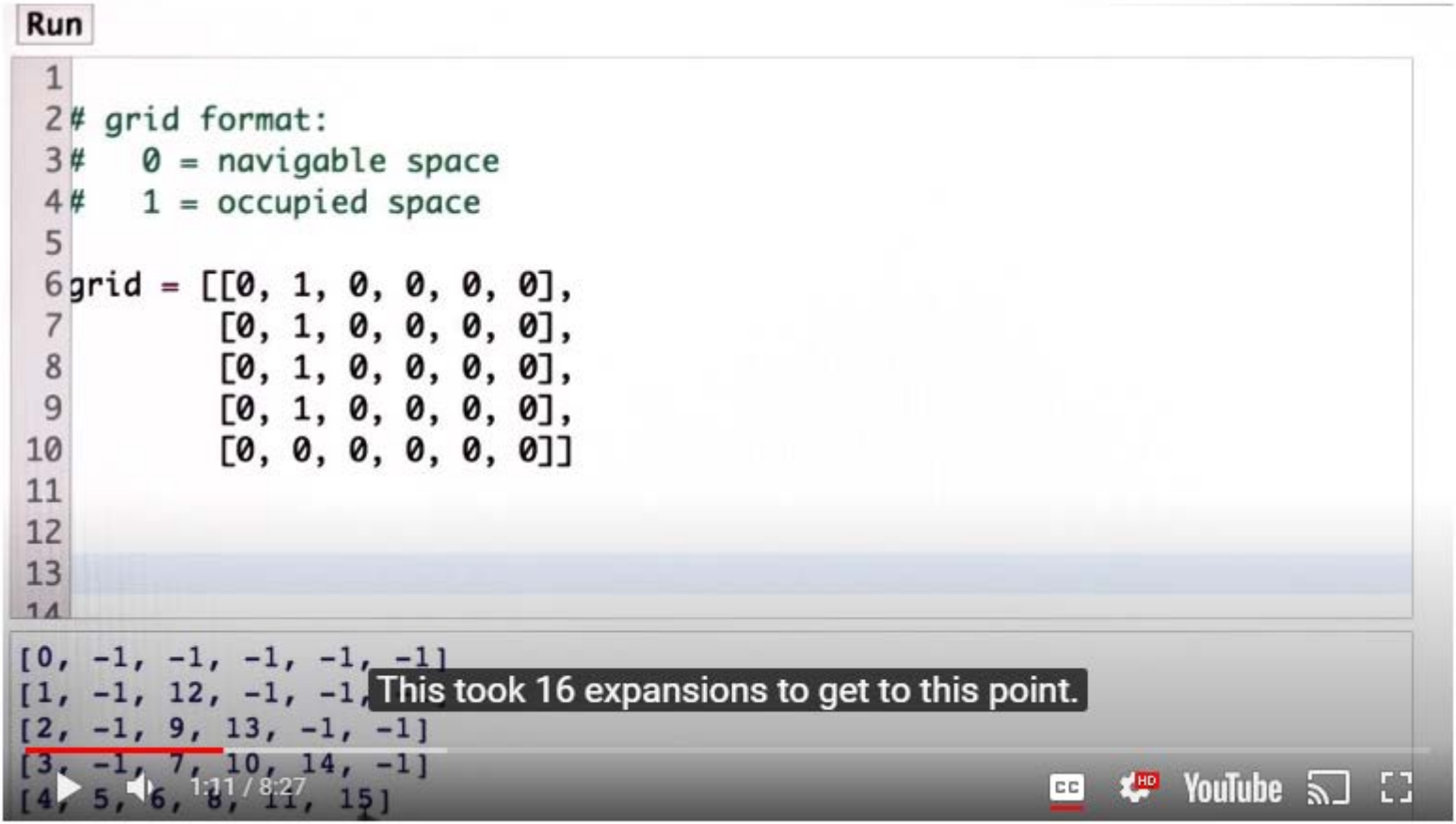
HOW MANY ACTIONS

11



What I want you to do is, to implement a piece of code that implements what I just did.


SEARCH - PATH PLANNING
0 1 2 3 4 5
S
G
UP
LEFT
R
RIGHT
DOWN
g-Value
OPEN =
A*
A-star was invented by Nels Nelson at Stanford many years ago,
0:07 / 8:27
YouTube


Run
# grid format:
#   0 = navigable space
#   1 = occupied space
grid = [[0, 1, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0]]
[0, -1, -1, -1, -1, -1]
[1, -1, 12, -1, -1,
This took 16 expansions to get to this point.
[2, -1, 9, 13, -1, -1]
[3, -1, 7, 10, 14, -1]
[4, 5, 6, 8, 11, 15]
1:11 / 8:27
YouTube

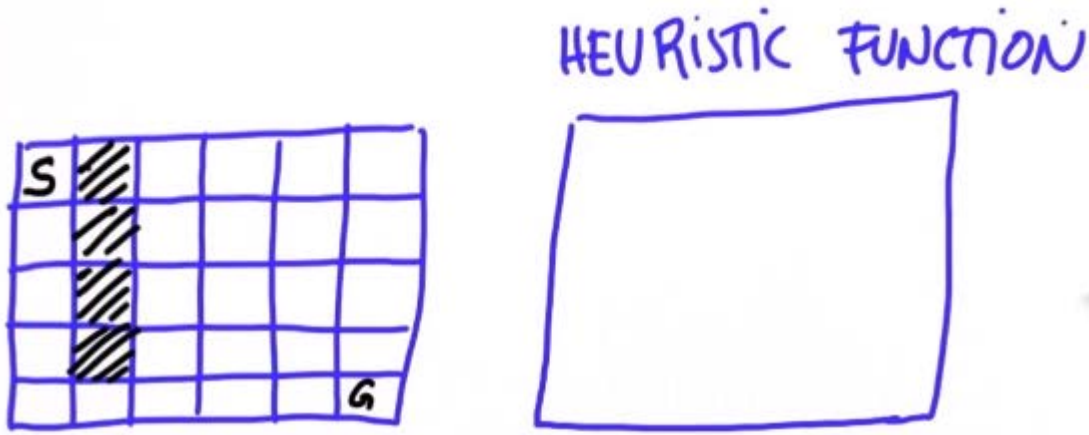
Run

```
1
2 # grid format:
3 #   0 = navigable space
4 #   1 = occupied space
5
6 grid = [[0, 1, 0, 0, 0, 0],
7         [0, 1, 0, 0, 0, 0],
8         [0, 1, 0, 0, 0, 0],
9         [0, 1, 0, 0, 0, 0],
10        [0, 0, 0, 0, 0, 0]]
11
12
13
14
```

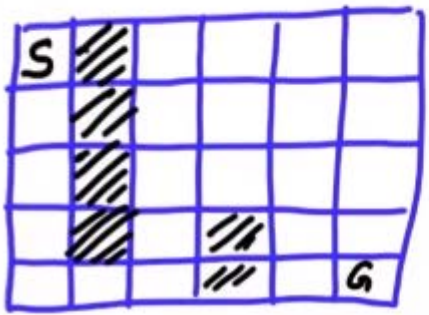
What we now find it only takes 10 expansions to get to this point, zero to nine over here.

```
[1, -1, -1, -1, -1, -1]
[2, -1, -1, -1, -1, -1]
[3, -1, -1, -1, -1, -1]
[4, 5, 6, 7, 8, 9]
```

1:20 / 8:27



A-star uses a so called heuristic function, which is a function that has to be set up.

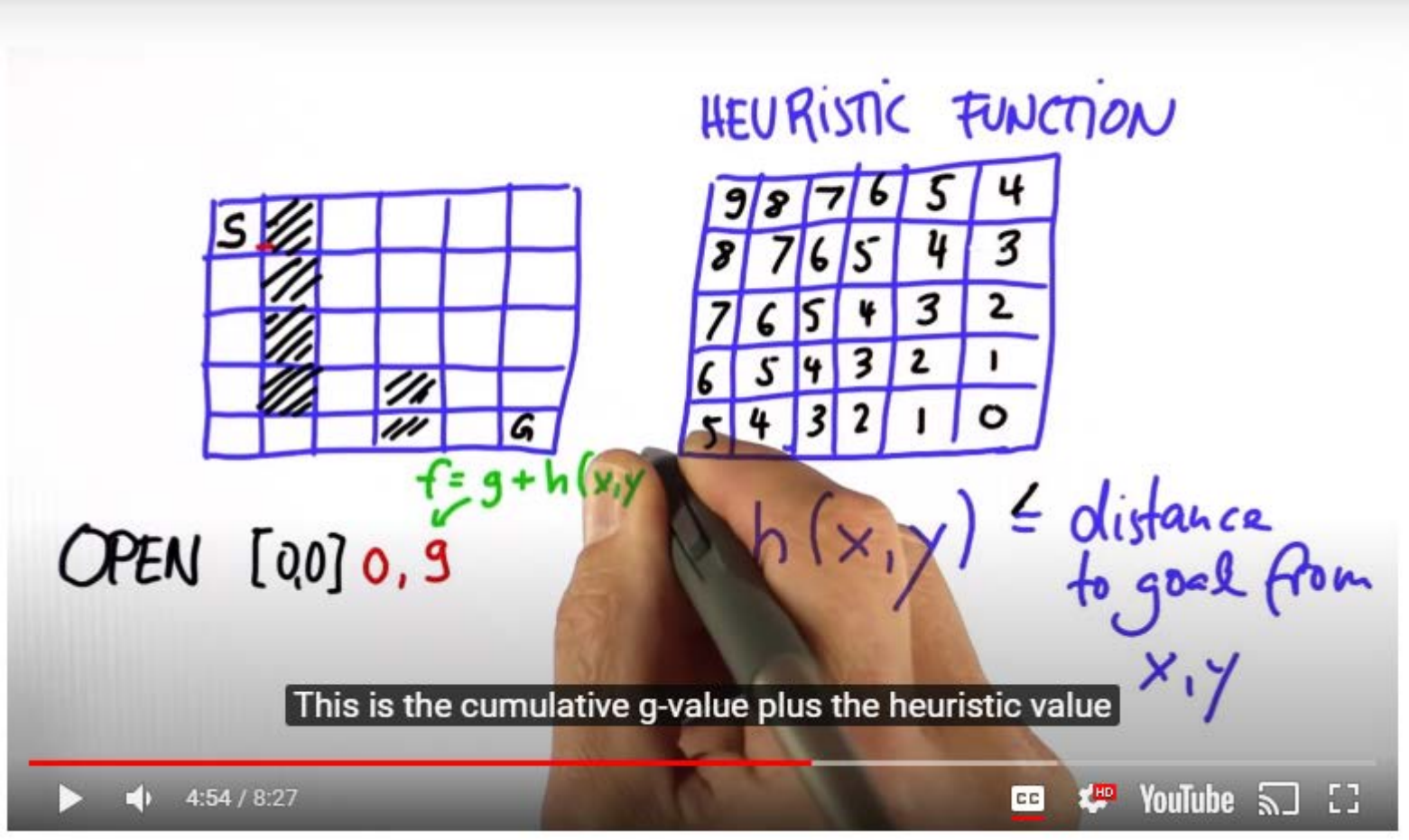


HEURISTIC FUNCTION

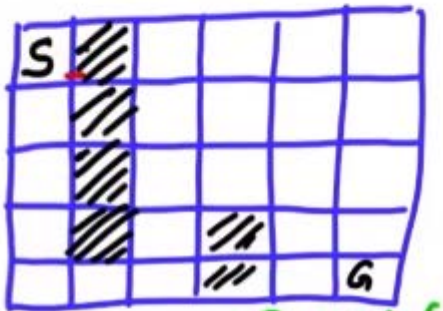
9	8	7	6	5	4
8	7	6	5	4	3
7	6	5	4	3	2
6	5	4	3	2	1
5	4	3	2	1	0

$h(x,y) \leq$ distance to goal from x,y

It has many, many value heuristic function including setting everything to zero,


HEURISTIC FUNCTION
S
G
9 8 7 6 5 4
8 7 6 5 4 3
7 6 5 4 3 2
6 5 4 3 2 1
5 4 3 2 1 0
f = g + h(x,y
OPEN [0,0] 0, 9
h(x,y) ≤ distance to goal from x,y
This is the cumulative g-value plus the heuristic value
4:54 / 8:27
YouTube

HEURISTIC FUNCTION



9	8	7	6	5	4
8	7	6	5	4	3
7	6	5	4	3	2
6	5	4	3	2	1
5	4	3	2	1	0

$f = g + h(x,y)$

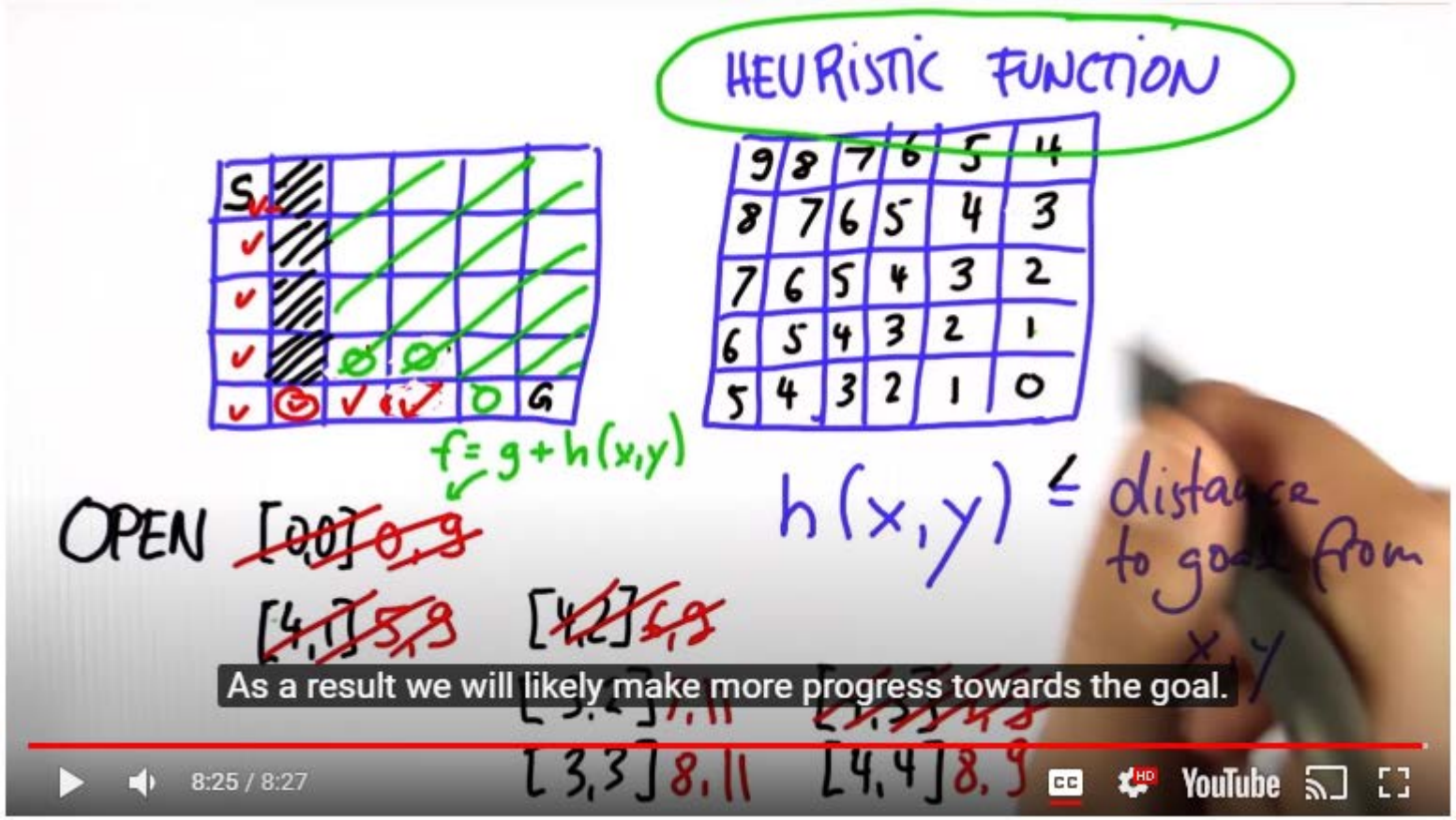
$h(x,y) \leq$ distance to goal from

OPEN

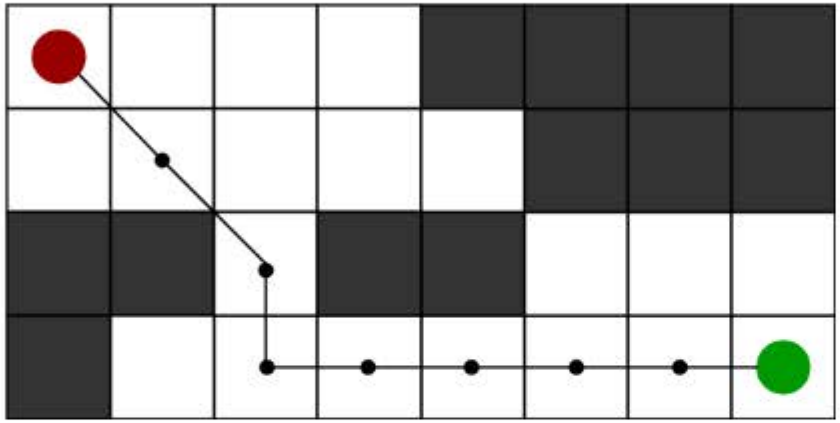
If I now e element with the lowest f-value and not the lowest g-value.

2:04

5:02 / 8:27


HEURISTIC FUNCTION
S
G
9 8 7 6 5 4
8 7 6 5 4 3
7 6 5 4 3 2
6 5 4 3 2 1
5 4 3 2 1 0
f = g + h(x,y)
OPEN [0,0] 0,9
[4,1] 5,9
[4,2] 6,9
[3,3] 8,11
[4,4] 8,9
h(x,y) ≤ distance to goal from x,y
As a result we will likely make more progress towards the goal.
8:25 / 8:27
YouTube

A* Search



A* Overview

In the previous lesson, Sebastian described how a general path search worked between two cells on a board, and you wrote C++ code to implement the board. In the next video, Sebastian will describe an improved way of searching, using an algorithm called A* search. This is the algorithm you will use for the implementation of your project.

After the video, there is an additional pseudocode outline of the A* algorithm that you will be following as you work through each exercise. Don't worry about remembering it all now, as the exercises will guide you through each step!

Summary and A* Pseudocode

This algorithm described by Sebastian is very similar to other search algorithms you may have seen before, such as breadth-first search, except for the additional step of computing a heuristic and using that heuristic (in addition to the cost) to find the next node.

The following is psuedocode for the algorithm described in the video above. Although the pseudocode shows the complete algorithm in a single function, we will split parts of the algorithm into separate functions in this lesson so you can implement them step-by-step in a sequence of exercises:

> ***Search****(grid, initial_point, goal_point) :*
>
> 1. *Initialize an empty list of open nodes.*
> 2. *Initialize a starting node with the following:*
>
>   - *x and y values given by initial_point.*
>   - *g = 0, where g is the cost for each move.*
>   - *h given by the heuristic function (a function of the current coordinates and the goal).*
>
> 3. *Add the new node to the list of open nodes.*
> 4. ***while** the list of open nodes is nonempty:*
>
>   1. *Sort the open list by f-value*
>   2. *Pop the optimal cell (called the current cell).*
>   3. *Mark the cell's coordinates in the grid as part of the path.*
>   4. ***if** the current cell is the goal cell:*
>
>   - *return the grid.*

5. ***else***, *expand the search to the current node's neighbors. This includes the following steps:*

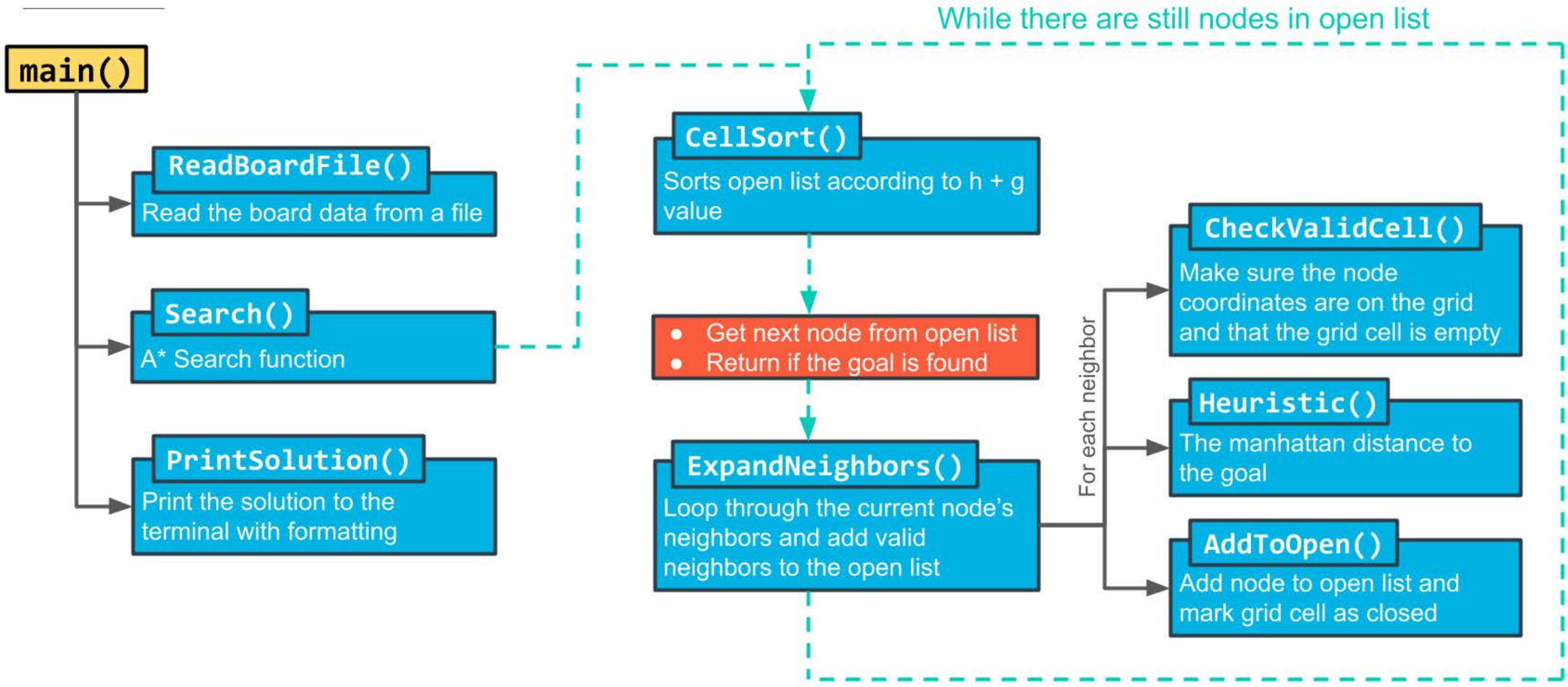
    - *Check each neighbor cell in the grid to ensure that the cell is empty: it hasn't been closed and is not an obstacle.*
    - *If the cell is empty, compute the cost (g value) and the heuristic, and add to the list of open nodes.*
    - *Mark the cell as closed.*

5. *If you exit the while loop because the list of open nodes is empty, you have run out of new nodes to explore and haven't found a path.*

Summary

The A* algorithm finds a path from the start node to the end node by checking for open neighbors of the current node, computing a heuristic for each of the neighbors, and adding those neighbors to the list of open nodes to explore next. The next node to explore is the one with the lowest total cost + heuristic (g + h). This process is repeated until the end is found, as long as there are still open nodes to explore.

A* Code Structure:



Lesson Code Structure

The next video provides an overview of the code that you will be implementing throughout the rest of the lesson. Be sure you have carefully reviewed the pseudocode from the previous concept so you'll have a better understanding of each of the functions described in the video.

Quiz

Below are the steps from the `while` loop in the A* pseudocode you saw previously:

> **while** the list of open nodes is nonempty:
>
> 1. Sort the open list by f-value
> 2. Pop the optimal cell (called the *current* cell).
> 3. Mark the cell's coordinates in the grid as part of the path.
> 4. **if** the *current* cell is the goal cell:
>   - return the *grid*.
> 5. **else**, expand the search to the *current* node's neighbors. This includes the following steps:
>   - Check each neighbor cell in the *grid* to ensure that the cell is empty: it hasn't been closed and is not an obstacle.
>   - If the cell is empty, compute the cost (g value) and the heuristic, and add to the list of open nodes.
>   - Mark the cell as closed.

In the quiz below, match the steps to the appropriate function from the A* code structure diagram.

QUIZ QUESTION

Match the steps in the `while` loop of the pseudocode above to the functions in the code structure diagram where the code would be implemented.

Submit to check your answer choices!

FUNCTIONS	STEPS
`CellSort()`	1
`ExpandNeighbors()`	5
`Search()`	4

SUBMIT

Summary

The code for the A* search algorithm has been broken down into the following functions:

- `CellSort()` - sorts the open list according to the sum of h + g
- `ExpandNeighbors()` - loops through the current node's neighbors and calls appropriate functions to add neighbors to the open list
- `CheckValidCell()` - ensures that the potential neighbor coordinates are on the grid and that the cell is open
- `Heuristic()` - computes the distance to the goal
- `AddToOpen()` - adds the node to the open list and marks the grid cell as closed

You will be implementing these functions along with a few other small helper functions throughout the rest of this lesson to complete the ASCII A* search program.

Starting A* Search

To get started with writing the A* search algorithm, you will first add a `Search` function stub that accepts and returns the appropriate variable types.

In this quiz, you will write a `Heuristic` function that will be used to guide the A* search. In general, any admissible function can be used for the heuristic, but for this project, you will write one that takes a pair of 2D coordinates on the grid and returns the Manhattan Distance from one coordinate to the other.

Pass by Reference

In the previous exercises, you've written functions that accept and return various kinds of objects. However, in all of the functions you've written so far, the objects returned by the function are different from the objects provided to the function. In other words, when the function is called on some data, a copy of that data is made, and the function operates on a copy of the data instead of the original data. This is referred to as pass by value, since only a copy of the values of an object are passed to the function, and not the actual objects itself.

To see how to use a function to operate directly on a given object, have a look at the notebook below.

On to the Exercises!

In the next exercises you will write some functions that require passing variables by reference. This will allow helper functions in the A* search program to modify the state of the board without having to copy the entire board, for instance.

Note that if you've encountered references before in C++, you are aware that they can be used in many other scenarios, aside from just passing variables to functions. In the next lessons, you will learn about references more generally, along with closely related *pointers*. However, you now know enough to finish the mini-project for the first half of the course!

Since you are well prepared with all you need at this point, this will be the last notebook in this half of the course. Good luck!

CODE: Adding Nodes to the Open Vector

SEND FEEDBACK

As you've seen from Sebastian's explanation of A* search, the search algorithm keeps a list of potential board cells to search through. In this implementation of A*, we will refer to a board cell along with it's `g` and `h` values as a *node*. In other words, each node will consist of the following values which are needed for the A* algorithm:

- an **x** coordinate,
- a **y** coordinate,
- the **g** value (or *cost*) that has accumulated up to that cell,
- the **h** value for the cell, given by the heuristic function.

In the code, nodes will be implemented with the type `vector<int>`, and should have the form `{x, y, g, h}` for `int`s x, y, g, and h. Also, the open list will be implemented as a C++ vector (of type `vector<vector<int>>`). The goal in this exercise is for you to write a helper function for your A* Search which will add nodes to the open vector and mark them as visited in the grid.

Fantastic work so far! In the last few coding exercises, you've been writing helper functions that will be used in the A* search. While there are a few more helper functions that still need to be written, in this exercise, you will begin implementing the body of the `Search` function. In particular, you will take the arguments that are passed to the search function, get the x, y, g, and h values for the first node, and then add the first node to the open vector.

We have provided the empty vector of open nodes, `open`, in the `Search` function for you to use.

Before you can use the vector of open nodes to expand the A* search, you will first need to be able to sort the vector. Since the vector contains nodes `{x, y, g, h}`, and there is no standard library function to sort these types of vectors, you will begin by writing a function which compares two nodes to determine their order.

This function is a helper function for the `CellSort()` function you will write later, so it is not shown on the code structure diagram.

Great work so far! Now on to some of the core functionality of the A* search algorithm. A* search works by sorting the open list using the f-value, and using the node with the lowest f-value as the next node in the search. This process continues until the goal node has been found or the open list runs out of nodes to use for searching.

In this exercise, you will implement the primary `while` loop in the algorithm which carries out the process described above:

Nice work, you are almost done with your program! The last part of the A* algorithm to be implemented is the part that adds neighboring nodes to the open vector. In order to expand your A* search from the current node to neighboring nodes, you first will need to check that neighboring grid cells are not closed, and that they are not an obstacle. In this exercise, you will write a function `CheckValidCell` that does exactly this.

Constants

In *A Tour of C++*, Bjarne Stroustrup writes:

> C++ supports two notions of immutability:
>
> - `const`: meaning roughly " I promise not to change this value."...The compiler enforces the promise made by `const`....
> - `constexpr`: meaning roughly "to be evaluated at compile time." This is used primarily to specify constants...

You have now reached the final step of the A* algorithm! You are ready to expand your A* search to neighboring nodes and add valid neighbors to the open vector. In this exercise, you will write an `ExpandNeighbors` function that takes care of this functionality for you.

Arrays

In the previous exercise, we included an array of directional deltas for convenience:

```
// directional deltas
const int delta[4][2]{{-1, 0}, {0, -1}, {1, 0}, {0, 1}};
```

Arrays are a lower level data structure than vectors, and can be slightly more efficient, in terms of memory and element access. However, this efficiency comes with a price. Unlike vectors, which can be extended with more elements, arrays have a fixed length. Additionally, arrays may require careful memory management, depending how they are used.

The example in the project code is a good use case for an array, as it was not intended to be changed during the execution of the program. However, a vector would have worked there as well.

Adding a Start and End to the Board

Excellent work! Your project is essentially complete, and the A* search algorithm is fully functional. To wrap things up, there is one modification that can be made to the project to make the printout slightly clearer. At this point, your program should print the following:

```
🚗  □  0  0  0  0
🚗  □  0  0  0  0
🚗  □  0  0  0  0
🚗  □  0  🚗 🚗 🚗
🚗  🚗 🚗 🚗 □  🚗
```

This is fantastic, but it isn't clear where the beginning and end of the path are. In this exercise, you will add a `□` for the beginning of the path, and a `🏁` for the end.

Lesson Recap

- Introduction to vehicle motion planning
  - Sebastian Thrun explained the A* search algorithm
- Wrote your own A* search
  - Different parts were broken down over a series of exercises
- Learned about new C++ language features, including
  - Passing variables to a function by reference instead of value
  - C++ const and constexpr
  - C++ Arrays

Lesson Outline

- Header files
  - Using headers to break a single file into multiple files
- Build systems
  - CMake and Make
- Tools for writing larger programs:
  - References
  - Pointers
  - Maps
  - Classes and object-oriented programming in C++

Header files, or `.h` files, allow related function, method, and class declarations to be collected in one place. The corresponding definitions can then be placed in `.cpp` files. The compiler considers a header declaration a "promise" that the definition will be found later in the code, so if the compiler reaches a function that hasn't been defined yet, it can continue on compiling until the definition is found. This allows functions to be defined (and declared) in arbitrary order.

In the previous concept, you saw how header files could be useful for separating definitions from declarations, so that you don't need to be too careful about the order in which functions are defined. Using header files is typically the first step in creating a multi-file program. In this concept, you will learn about using multiple `.cpp` and `.h` files in a program - how to compile all the files together, and how to ensure the code from one file can be used in another.

CMake and Make

In the previous notebook, you saw how example code could be split into multiple `.h` and `.cpp` files, and you used `g++` to build all of the files together. For small projects with a handful of files, this works well. But what would happen if there were hundreds, or even thousands, of files in the project? You could type the names of the files at the command line each time, but there tools to make this easier.

Many larger C++ projects use a build system to manage all the files during the build process. The build system allows for large projects to be compiled with a few commands, and build systems are able to do this in an efficient way by only

recompiling files that have been changed.

In this workspace you will learn about

- Object files: what actually happens when you run `g++`.
- How to use object files to compile only a single file at a time. If you have many files in a project, this will allow you can compile only files that have changed and need to be re-compiled.
- How to use `cmake` (and `make`), a build system which is popular in large C++ projects.CMake will simplify the process of building project and re-compiling only the changed files.

Click to go to the next page to see how this works!

Object Files

When you compile a project with `g++`, `g++` actually performs several distinct tasks:

1. The preprocessor runs and executes any statement beginning with a hash symbol: `#`, such as `#include` statements. This ensures all code is in the correct location and ready to compile.
2. Each file in the source code is compiled into an "object file" (a `.o` file). Object files are platform-specific machine code that will be used to create an executable.
3. The object files are "linked" together to make a single executable. In the examples you have seen so far, this executable is `a.out`, but you can specify whatever name you want.

It is possible to have `g++` perform each of the steps separately by using the `-c` flag.
For example,

```
g++ -c main.cpp
```

will produce a `main.o` file, and that file can be converted to an executable with

```
g++ main.o
```

Your Turn

Try to compile `main.cpp` to an object file using the commands from the previous page. In the terminal to the right:

1. Compile the `main.cpp` file to an object file using the `-c` flag. You can list the files in the directory with `ls`.

   After compiling, you should see a `main.o` in the directory (along with other notebook files).
2. Convert the file to an executable with `g++`.
3. Run the executable with `./a.out`.

main.cpp

```cpp
#include <iostream>
using std::cout;

int main()
{
    cout << "Hello!" << "\n";
}
```

root@e3c141c7d4a1: /home/w

```
root@e3c141c7d4a1:/home/workspace# ls
CMake and Make.ipynb  cmake_example  cmake.jpg  main.cpp  multiple_files_example
root@e3c141c7d4a1:/home/workspace# g++ -c main.cpp
root@e3c141c7d4a1:/home/workspace# ls
CMake and Make.ipynb  cmake_example  cmake.jpg  main.cpp  main.o  multiple_files_example
root@e3c141c7d4a1:/home/workspace# g++ main.o
root@e3c141c7d4a1:/home/workspace# ls
a.out  CMake and Make.ipynb  cmake_example  cmake.jpg  main.cpp  main.o  multiple_files_example
root@e3c141c7d4a1:/home/workspace# ./a.out
Hello!
root@e3c141c7d4a1:/home/workspace#
```

CMake and Make

Compiling One File of Many, Step 1

In the previous example, you compiled a single source code file to an object file, and that object file was then converted into an executable.

Now you are going to try the same process with multiple files. Navigate to the `multiple_files_example` directory in the terminal to the right. This directory should have the `increment_and_sum` and `vect_add_one` files from a previous Notebook. Try compiling with the commands below:

```
root@abc123defg:/home/workspace/multiple_files_example# g++ -c *.cpp
root@abc123defg:/home/workspace/multiple_files_example# g++ *.o
root@abc123defg:/home/workspace/multiple_files_example# ./a.out
```

Here, the `*` operator is a wildcard, so any matching file is selected. If you compile and run these files together, the executable should print:

> *The total is: 14*

main.cpp | vect_add_one.h | vect_add_one.cpp | increment_and_sum | increment_and_sum

```cpp
#include <iostream>
#include <vector>
#include "increment_and_sum.h"
using std::vector;
using std::cout;

int main()
{
    vector<int> v{1, 2, 3, 4};
    int total = IncrementAndComputeVectorSum(v);
    cout << "The total is: " << total << "\n";
}
```

root@e3c141c7d4a1: /home/v

```
root@e3c141c7d4a1:/home/workspace/multiple_files_example# g++ -c *.cpp
root@e3c141c7d4a1:/home/workspace/multiple_files_example# ls
increment_and_sum.cpp  increment_and_sum.o  main.o           vect_add_one.h
increment_and_sum.h    main.cpp            vect_add_one.cpp  vect_add_one.o
root@e3c141c7d4a1:/home/workspace/multiple_files_example# g++ *.o
root@e3c141c7d4a1:/home/workspace/multiple_files_example# ls
a.out                  increment_and_sum.h  main.cpp  vect_add_one.cpp  vect_add_one.o
increment_and_sum.cpp  increment_and_sum.o  main.o    vect_add_one.h
root@e3c141c7d4a1:/home/workspace/multiple_files_example# ./a.out
The total is: 14
root@e3c141c7d4a1:/home/workspace/multiple_files_example#
```

Compiling One File of Many, Step 2

But what if you make changes to your code and you need to re-compile? In that case, you can compile only the file that you changed, and you can use the existing object files from the unchanged source files for linking.

Try changing the code in `/multiple_files_example/main.cpp` to have different numbers in the vector and save the file with `CTRL-s`.

When you have done that, re-compile just `main.cpp` by running:

```
root@abc123defg:/home/workspace/multiple_files_example# g++ -c main.cpp
root@abc123defg:/home/workspace/multiple_files_example# g++ *.o
root@abc123defg:/home/workspace/multiple_files_example# ./a.out
```

Compiling just the file you have changed saves time if there are many files and compilation takes a long time. However, the process above is tedious when using many files, especially if you don't remember which ones you have modified.

For larger projects, it is helpful to use a build system which can compile exactly the right files for you and take care of linking

main.cpp | vect_add_one.h | vect_add_one.cpp | increment_and_sum | increment_and_sum

```cpp
#include <iostream>
#include <vector>
#include "increment_and_sum.h"
using std::vector;
using std::cout;

int main()
{
    vector<int> v{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
    int total = IncrementAndComputeVectorSum(v);
    cout << "The total is: " << total << "\n";
}
```

root@e3c141c7d4a1: /home/\

```
root@e3c141c7d4a1:/home/workspace/multiple_files_example#
```

you can compile only the file that you changed, and you can use the existing object files from the unchanged source files for linking.

Try changing the code in `/multiple_files_example/main.cpp` to have different numbers in the vector and save the file with `CTRL-s`.

When you have done that, re-compile just `main.cpp` by running:

```
root@abc123defg:/home/workspace/multiple_files_example# g++ -c main.cpp
root@abc123defg:/home/workspace/multiple_files_example# g++ *.o
root@abc123defg:/home/workspace/multiple_files_example# ./a.out
```

Compiling just the file you have changed saves time if there are many files and compilation takes a long time. However, the process above is tedious when using many files, especially if you don't remember which ones you have modified.

For larger projects, it is helpful to use a build system which can compile exactly the right files for you and take care of linking.

On the next page, we'll introduce a cross-platform build system that you'll be using in several of the projects in this Nanodegree program.

main.cpp | vect_add_one.h | vect_add_one.cpp | increment_and_sum | increment_and_sum

```
1  #include <iostream>
2  #include <vector>
3  #include "increment_and_sum.h"
4  using std::vector;
5  using std::cout;
6
7  int main()
8  {
9      vector<int> v{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
10     int total = IncrementAndComputeVectorSum(v);
11     cout << "The total is: " << total << "\n";
12 }
```

root@e3c141c7d4a1: /home/\

```
root@e3c141c7d4a1:/home/workspace/multiple_files_example# g++ -c main.cpp
root@e3c141c7d4a1:/home/workspace/multiple_files_example# g++ *.o
root@e3c141c7d4a1:/home/workspace/multiple_files_example# ./a.out
The total is: 135
root@e3c141c7d4a1:/home/workspace/multiple_files_example#
```

CMake and Make

CMake is an open-source, platform-independent build system. CMake uses text documents, denoted as `CMakeLists.txt` files, to manage build environments, like make. A comprehensive tutorial on CMake would require an entire course, but you can learn the basics of CMake here, so you'll be ready to use it in the upcoming projects.

CMakeLists.txt

`CMakeList.txt` files are simple text configuration files that tell CMake how to build your project. There can be multiple `CMakeLists.txt` files in a project. In fact, one `CMakeList.txt` file can be included in each directory of the project, indicating how the files in that directory should be built.

These files can be used to specify the locations of necessary packages, set build flags and environment variables, specify build target names and locations, and other actions.

In the next few pages of this workspace, you are going to create a basic `CMakeLists.txt` file to build a small project.


```
1 cmake_minimum_required(VERSION 3.5.1)
2
3 set(CMAKE_CXX_STANDARD 14)
4
5 project(ExampleProject)
6
7 add_executable(example src/main.cpp src/vect_add_one.cpp src/increment_and_sum.cpp)
```

root@e3c141c7d4a1: /home/v

```
root@e3c141c7d4a1:/home/workspace/multiple_files_example# g++ -c main.cpp
root@e3c141c7d4a1:/home/workspace/multiple_files_example# g++ *.o
root@e3c141c7d4a1:/home/workspace/multiple_files_example# ./a.out
The total is: 135
root@e3c141c7d4a1:/home/workspace/multiple_files_example#
```

CMake Step 1

In the terminal to the right, navigate to the `cmake_example` folder. This folder should contain a simple CMake project, including:

- An empty `CMakeLists.txt` file
- A `src` directory with the `increment_and_sum` and `vect_add_one` files from a previous Notebook

The `CMakeLists.txt` file should be open in the tabs on the right, along with the files from the `src` directory. You will write a basic `CMakeLists.txt` file to build all of these project files into an executable.

The first lines that you'll want in your `CMakeLists.txt` are lines that specifies the minimum versions of cmake and C++ required to build the project. Add the following lines to your `CMakeLists.txt` and save the file:

```
cmake_minimum_required(VERSION 3.5.1)

set(CMAKE_CXX_STANDARD 14)
```

These lines set the minimum cmake version required to 3.5.1 and set the

1


```
root@e3c141c7d4a1:/home/workspace/cmake_example#
```

In the terminal to the right, navigate to the `cmake_example` folder. This folder should contain a simple CMake project, including:

- An empty `CMakeLists.txt` file
- A `src` directory with the `increment_and_sum` and `vect_add_one` files from a previous Notebook

The `CMakeLists.txt` file should be open in the tabs on the right, along with the files from the `src` directory. You will write a basic `CMakeLists.txt` file to build all of these project files into an executable.

The first lines that you'll want in your `CMakeLists.txt` are lines that specifies the minimum versions of cmake and C++ required to build the project. Add the following lines to your `CMakeLists.txt` and save the file:

```
cmake_minimum_required(VERSION 3.5.1)

set(CMAKE_CXX_STANDARD 14)
```

These lines set the minimum cmake version required to 3.5.1 and set the environment variable `CMAKE_CXX_STANDARD` so CMake uses C++ 14. On your own computer, if you have a recent `g++` compiler, you could use C++ 17 instead.


```
1 cmake_minimum_required(VERSION 3.5.1)
2
3 set(CMAKE_CXX_STANDARD 14)
```



```
root@e3c141c7d4a1:/home/workspace/cmake_example#
```

CMake and Make

CMake Step 2

CMake requires that we name the project, so you should choose a name for the project and then add the following line to `CMakeLists.txt`:

```
project(<your_project_name>)
```

You can choose any name you want, but be sure to change `<your_project_name>` to the actual name of the project!

CMakeLists.t | main.cpp | vect_add_on | vect_add_on | increment_a | increment_a | SolutionCMa

```
1 cmake_minimum_required(VERSION 3.5.1)
2
3 set(CMAKE_CXX_STANDARD 14)
4
5 project(<Cython>)
```

root@e3c141c7d4a1: /home/v

```
root@e3c141c7d4a1:/home/workspace/cmake_example#
```

CMake and Make

CMake Step 3

Next, we want to add an executable to this project. You can do that with the `add_executable` command by specifying the executable name, along with the locations of all the source files that you will need. CMake has the ability to automatically find source files in a directory, but for now, you can just specify each file needed:

```
add_executable(your_executable_name  path_to_file_1  path_to_file_2 ...)
```

Hint: The source files you need are the *three* .cpp files in the `src/` directory. You can specify the path relative to the `CMakeLists.txt` file, so `src/main.cpp` would work, for example.

CMakeLists.t | main.cpp | vect_add_on | vect_add_on | increment_a | increment_a | SolutionCMa

```
1  cmake_minimum_required(VERSION 3.5.1)
2
3  set(CMAKE_CXX_STANDARD 14)
4
5  project(<Cython>)
6
7  add_executable(exe src/main.cpp src/vect_add_one.cpp src/increment_and_sum.cpp)
```

root@e3c141c7d4a1: /home/v

```
root@e3c141c7d4a1:/home/workspace/cmake_example#
```

CMake and Make

CMake Step 4

A typical CMake project will have a `build` directory in the same place as the top-level `CMakeLists.txt`. Make a `build` directory in the `/home/workspace/cmake_example` folder:

```
root@abc123defg:/home/workspace/cmake_example# mkdir build
root@abc123defg:/home/workspace/cmake_example# cd build
```

From within the build directory, you can now run CMake as follows:

```
root@abc123defg:/home/workspace/cmake_example/build# cmake ..
root@abc123defg:/home/workspace/cmake_example/build# make
```

The first line directs the `cmake` command at the top-level `CMakeLists.txt` file with `..`. This command uses the `CMakeLists.txt` to configure the project and create a `Makefile` in the `build` directory.

In the second line, `make` finds the `Makefile` and uses the instructions in the

CMakeLists.t | main.cpp | vect_add_on | vect_add_on | increment_a | increment_a | SolutionCMa

```
1 cmake_minimum_required(VERSION 3.5.1)
2
3 set(CMAKE_CXX_STANDARD 14)
4
5 project(<Cython>)
6
7 add_executable(exe src/main.cpp src/vect_add_one.cpp src/increment_and_sum.cpp)
```

root@e3c141c7d4a1: /home/v

```
root@e3c141c7d4a1:/home/workspace/cmake_example#
```

/home/workspace/cmake_example folder:

```
root@abc123defg:/home/workspace/cmake_example# mkdir build
root@abc123defg:/home/workspace/cmake_example# cd build
```

From within the build directory, you can now run CMake as follows:

```
root@abc123defg:/home/workspace/cmake_example/build# cmake ..
root@abc123defg:/home/workspace/cmake_example/build# make
```

The first line directs the `cmake` command at the top-level `CMakeLists.txt` file with `..`. This command uses the `CMakeLists.txt` to configure the project and create a `Makefile` in the `build` directory.

In the second line, `make` finds the `Makefile` and uses the instructions in the `Makefile` to build the project.

If CMake and Make are successful, you should see something like the following output in the terminal:

Image

CMakeLists.t | main.cpp | vect_add_on | vect_add_on | increment_a | increment_a | SolutionCMa

```
1 cmake_minimum_required(VERSION 3.5.1)
2
3 set(CMAKE_CXX_STANDARD 14)
4
5 project(<Cython>)
6
7 add_executable(exe src/main.cpp src/vect_add_one.cpp src/increment_and_sum.cpp)
```

root@e3c141c7d4a1: /home/v

```
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/workspace/cmake_example/build
root@e3c141c7d4a1:/home/workspace/cmake_example/build# make
Scanning dependencies of target exe
[ 25%] Building CXX object CMakeFiles/exe.dir/src/main.cpp.o
[ 50%] Building CXX object CMakeFiles/exe.dir/src/vect_add_one.cpp.o
[ 75%] Building CXX object CMakeFiles/exe.dir/src/increment_and_sum.cpp.o
[100%] Linking CXX executable exe
[100%] Built target exe
root@e3c141c7d4a1:/home/workspace/cmake_example/build# ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  exe  Makefile
root@e3c141c7d4a1:/home/workspace/cmake_example/build#
```

CMake and Make

CMake Step 5

If everything has worked correctly, you should now be able to run your executable from the build folder:

```
root@abc123defg:/home/workspace/cmake_example/build# ./your_executable_name
```

This executable should print:

> *The total is: 14*

just as it did in the previous workspace.

If you don't remember the name of your executable, the last line of the `make` output should tell you:

```
[100%] Built target <your_executable_name>
```


CMakeLists.t | main.cpp | vect_add_on | vect_add_on | increment_a | increment_a | SolutionCMa

```
cmake_minimum_required(VERSION 3.5.1)

set(CMAKE_CXX_STANDARD 14)

project(<Cython>)

add_executable(exe src/main.cpp src/vect_add_one.cpp src/increment_and_sum.cpp)
```

root@e3c141c7d4a1: /home/v

```
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/workspace/cmake_example/build
root@e3c141c7d4a1:/home/workspace/cmake_example/build# make
Scanning dependencies of target exe
[ 25%] Building CXX object CMakeFiles/exe.dir/src/main.cpp.o
[ 50%] Building CXX object CMakeFiles/exe.dir/src/vect_add_one.cpp.o
[ 75%] Building CXX object CMakeFiles/exe.dir/src/increment_and_sum.cpp.o
[100%] Linking CXX executable exe
[100%] Built target exe
root@e3c141c7d4a1:/home/workspace/cmake_example/build# ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  exe  Makefile
root@e3c141c7d4a1:/home/workspace/cmake_example/build# ./exe
The total is: 14
root@e3c141c7d4a1:/home/workspace/cmake_example/build#
```

CMake and Make

CMake Step 6

Now that your project builds correctly, try modifying one of the files. When you are ready to run the project again, you'll only need to run the `make` command from the build folder, and only that file will be compiled again. Try it now!

In general, CMake only needs to be run once for a project, unless you are changing build options (e.g. using different build flags or changing where you store your files).

`Make` will be able to keep track of which files have changed and compile only those that need to be compiled before building.

Note: If you do re-run CMake, or if you are having problems with your build, *it can be helpful to delete your build directory and start from scratch. Otherwise, some environment variables may not be reset correctly.*

CMakeLists.t | main.cpp | vect_add_on | vect_add_on | increment_a | increment_a | SolutionCMa

```
cmake_minimum_required(VERSION 3.5.1)

set(CMAKE_CXX_STANDARD 14)

project(<Cython>)

add_executable(exe src/main.cpp src/vect_add_one.cpp src/increment_and_sum.cpp)
```

root@e3c141c7d4a1: /home/v

```
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/workspace/cmake_example/build
root@e3c141c7d4a1:/home/workspace/cmake_example/build# make
Scanning dependencies of target exe
[ 25%] Building CXX object CMakeFiles/exe.dir/src/main.cpp.o
[ 50%] Building CXX object CMakeFiles/exe.dir/src/vect_add_one.cpp.o
[ 75%] Building CXX object CMakeFiles/exe.dir/src/increment_and_sum.cpp.o
[100%] Linking CXX executable exe
[100%] Built target exe
root@e3c141c7d4a1:/home/workspace/cmake_example/build# ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  exe  Makefile
root@e3c141c7d4a1:/home/workspace/cmake_example/build# ./exe
The total is: 14
root@e3c141c7d4a1:/home/workspace/cmake_example/build#
```

CMake and Make

CMake Review

Excellent work! You've now written a basic `CMakeLists.txt` file that builds a small project for you.

CMake is a build system that uses text files named `CMakeLists.txt` to configure and build your project. Once the `CMakeLists.txt` is written, you only need the `cmake` and `make` commands to build your project again and again, so it is very convenient to use!

Coming up, we will provide the `CMakeLists.txt` for your course project, and as you will see, it creates two executables and links several external libraries, so it will be a bit longer than the one you've just created. However, you should now have a better understanding of the mechanics of CMake, and you are ready to start experimenting with CMake on your own projects.

CMakeLists.t | main.cpp | vect_add_on | vect_add_on | increment_a | increment_a | SolutionCMa

```
cmake_minimum_required(VERSION 3.5.1)

set(CMAKE_CXX_STANDARD 14)

project(ExampleProject)

add_executable(example src/main.cpp src/vect_add_one.cpp src/increment_and_sum.cpp)
```

root@e3c141c7d4a1: /home/v

```
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/workspace/cmake_example/build
root@e3c141c7d4a1:/home/workspace/cmake_example/build# make
Scanning dependencies of target exe
[ 25%] Building CXX object CMakeFiles/exe.dir/src/main.cpp.o
[ 50%] Building CXX object CMakeFiles/exe.dir/src/vect_add_one.cpp.o
[ 75%] Building CXX object CMakeFiles/exe.dir/src/increment_and_sum.cpp.o
[100%] Linking CXX executable exe
[100%] Built target exe
root@e3c141c7d4a1:/home/workspace/cmake_example/build# ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  exe  Makefile
root@e3c141c7d4a1:/home/workspace/cmake_example/build# ./exe
The total is: 14
root@e3c141c7d4a1:/home/workspace/cmake_example/build#
```

References

You have seen references used previously, in both pass-by-reference for functions, and in a range-based `for` loop example that used references to modify a vector. As you write larger C++ programs, you will find references useful in a variety of situations. In this short notebook, you will see a few more examples of references to solidify your knowledge.

Pointers have traditionally been a stumbling block for many students learning C++, but they do not need to be!

> *A C++ pointer is just a variable that stores the memory address of an object in your program.*

That is the most important thing to understand and remember about pointers - they essentially keep track of *where* a variable is stored in the computer's memory.

In the previous lessons, you implemented A* search in a single file without using C++ pointers, except in `CellSort` code that was provided for you; a C++ program can be written without using pointers extensively (or at all). However, pointers give you better control over how your program uses memory. However, much like the pass-by-reference example that you saw previously, it can often be far more efficient to perform an operation with a pointer to an object than performing the same operation using the object itself.

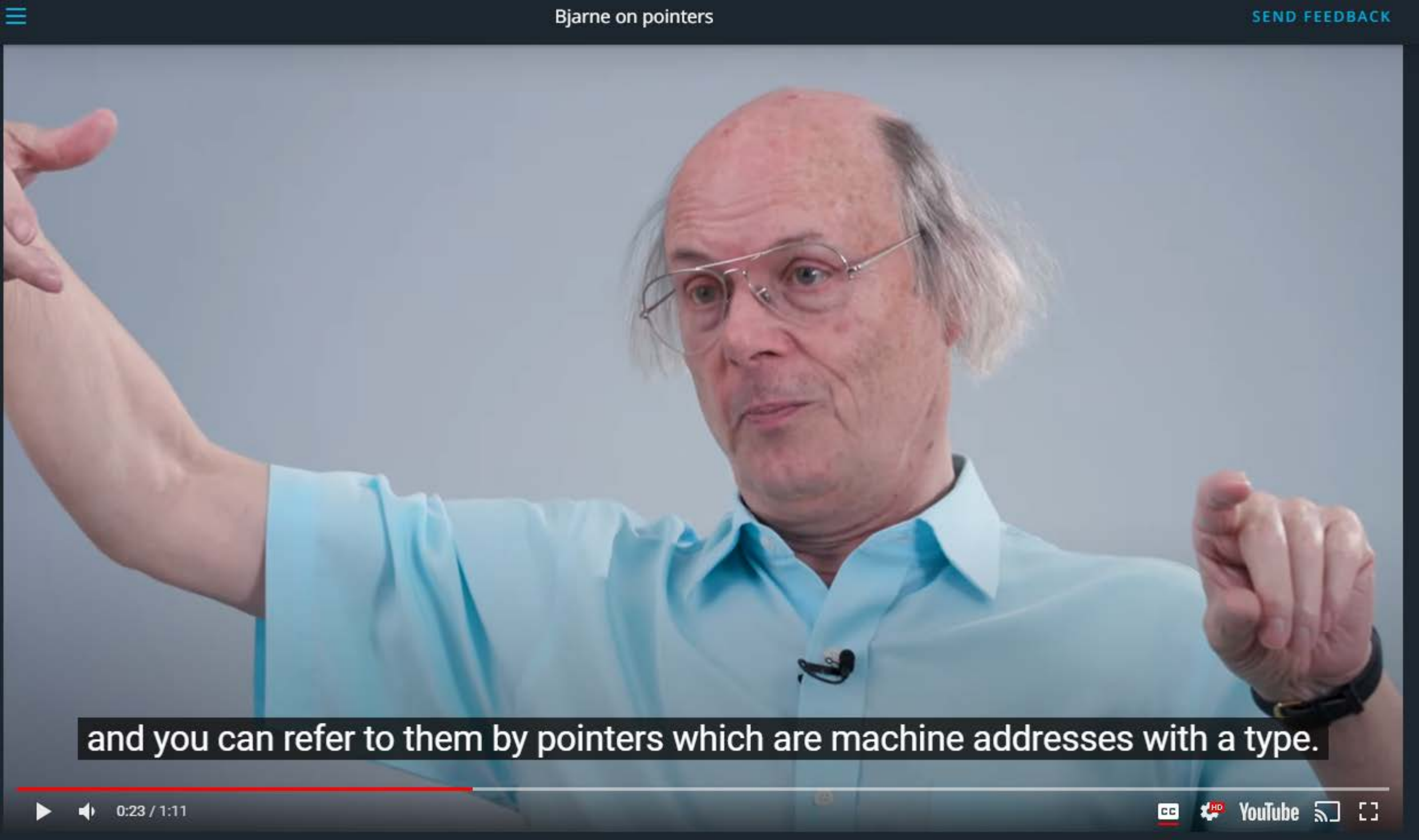
Pointers are an extremely important part of the C++ language, and as you are exposed to more C++ code, you will certainly encounter them. In this notebook, you will become familiar with basic pointers so you get comfortable with the syntax, and you will be ready to use them in the course project code.

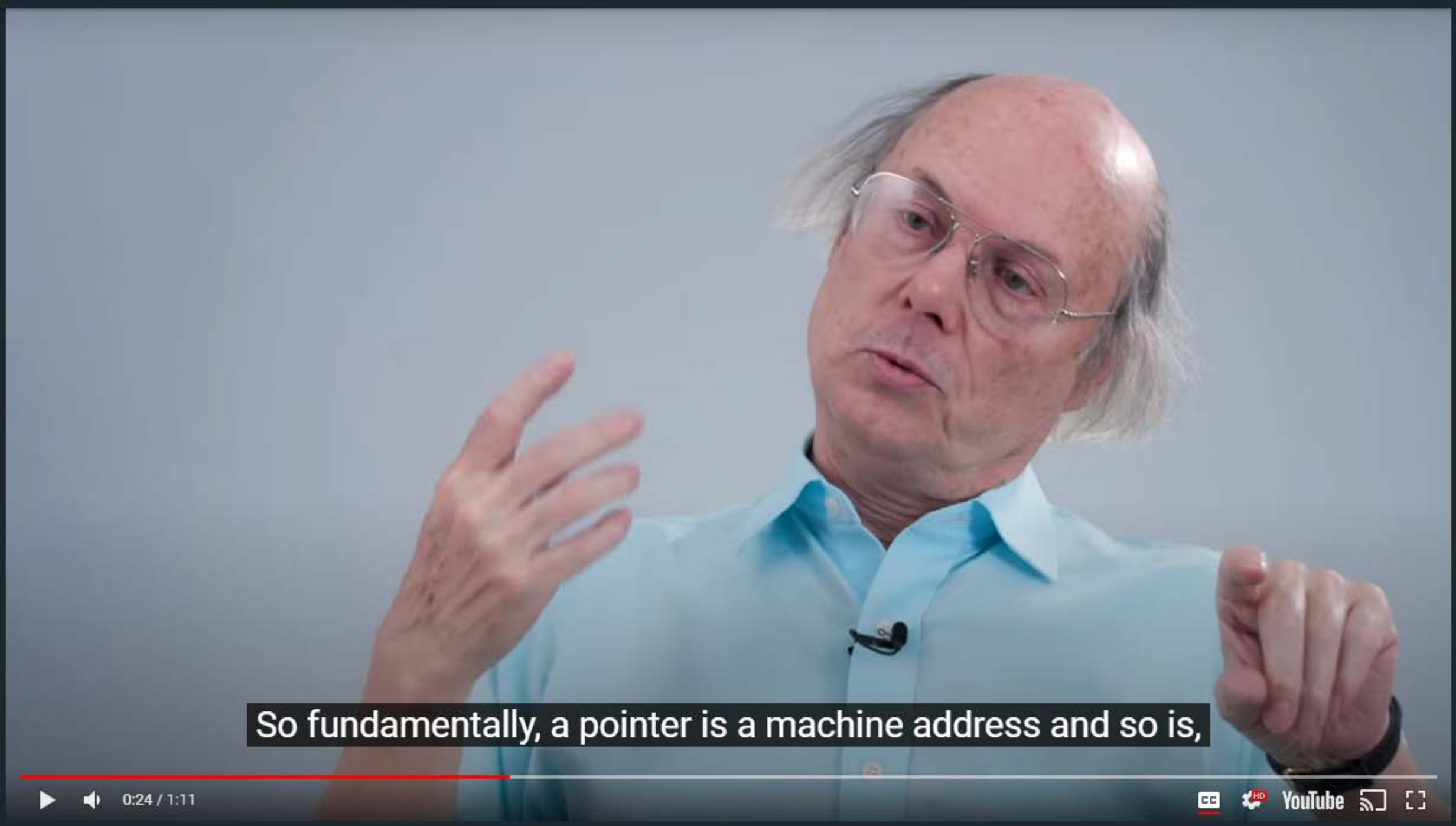
Pointers Continued

SEND FEEDBACK

In the previous concept, you were introduced to `int` pointers, and you learned the syntax for creating a pointer and retrieving an object from a pointer.

In the notebook below, you will see how to create pointers to other object types, and you will learn about how to use pointers with functions.


Bjarne on pointers
SEND FEEDBACK
and you can refer to them by pointers which are machine addresses with a type.
0:23 / 1:11
YouTube


So fundamentally, a pointer is a machine address and so is,
0:24 / 1:11
YouTube


by the way, a reference.
0:29 / 1:11
YouTube


It's a machine address with a type associated with it at compile time.
1:03 / 1:11
YouTube

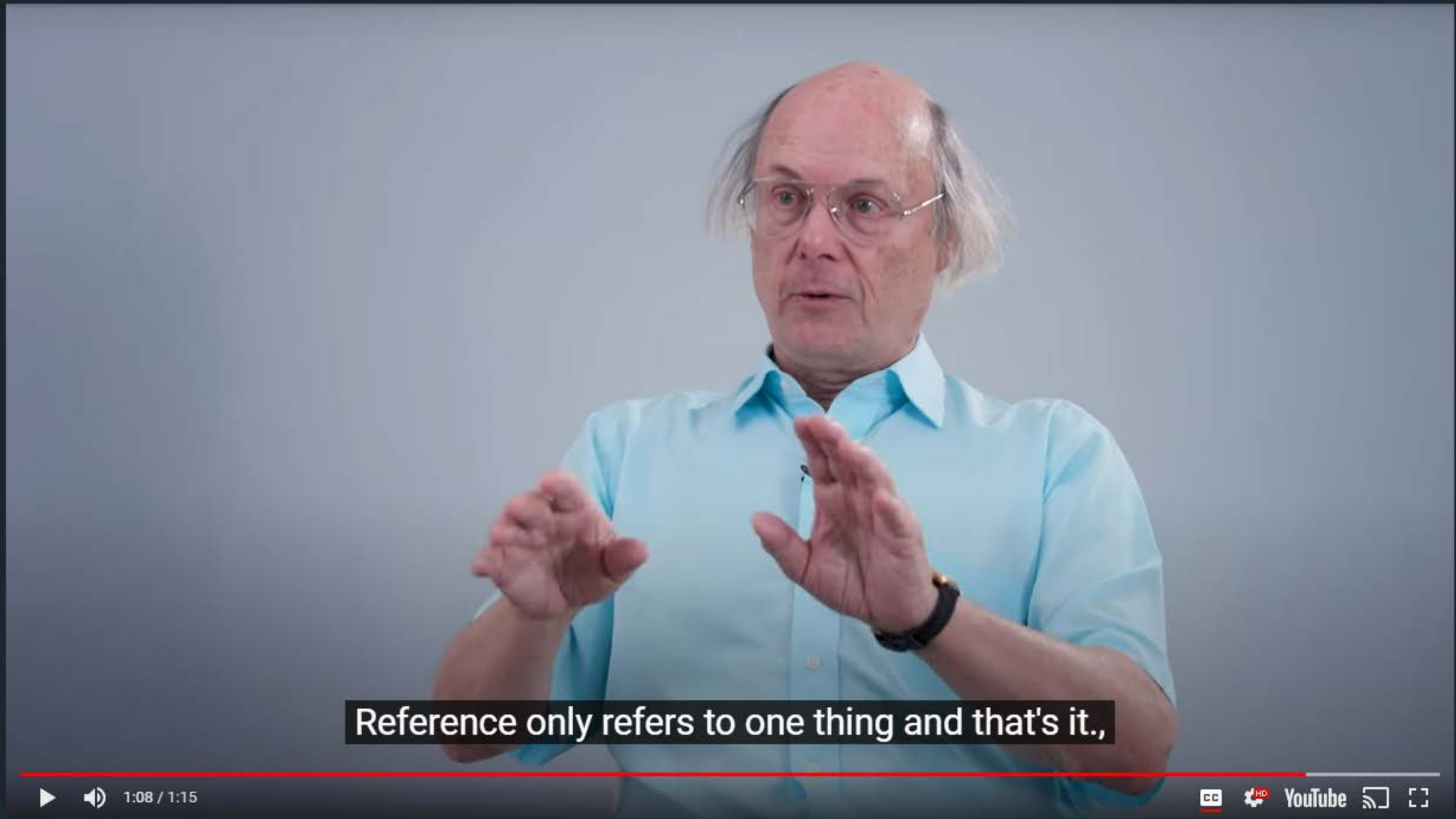
References vs Pointers

Pointers and references can have similar use cases in C++. As seen previously both references and pointers can be used in pass-by-reference to a function. Additionally, they both provide an alternative way to access an existing variable: pointers through the variable's address, and references through another name for that variable. But what are the differences between the two, and when should each be used? The following list summarizes some of the differences between pointers and references, as well as when each should be used:

References	Pointers
References must be initialized when they are declared. This means that a reference will always point to data that was intentionally assigned to it.	Pointers can be declared without being initialized, which is dangerous. If this happens mistakenly, the pointer could be pointing to an arbitrary address in memory, and the data associated with that address could be meaningless, leading to undefined behavior and difficult-to-resolve bugs.
References can not be null. This means that a reference should point to meaningful data in the program.	Pointers can be null. In fact, if a pointer is not initialized immediately, it is often best practice to initialize to `nullptr`, a special type which indicates that the pointer is null.
When used in a function for pass-by-reference, the reference can be used just as a variable of the same type would be.	When used in a function for pass-by-reference, a pointer must be dereferenced in order to access the underlying object.

References are generally easier and safer than pointers. As a decent rule of thumb, references should be used in place of pointers when possible.

However, there are times when it is not possible to use references. One example is object initialization. You might like one object to store a reference to another object. However, if the other object is not yet available when the first object is created, then the first object will need to use a pointer, not a reference, since a reference cannot be null. The reference could only be initialized once the other object is created.


Reference only refers to one thing and that's it.,
1:08 / 1:15
YouTube

So far in this course you have seen container data structures, like the `vector` and the `array`. Additionally, you have used classes in your code for this project. Container data structures are fantastic for storing ordered data, and classes are useful for grouping related data and functions together, but neither of these data structures is optimal for storing associated data.

Dictionary Example

A map (alternatively **hash table**, hash map, or dictionary) is a data structure that uses *key/value* pairs to store data, and provides efficient lookup and insertion of the data. The name "dictionary" should provide an excellent idea of how these work, since a dictionary is a real life example of a map. Here is a slightly edited entry from **www.dictionary.com** defining the word "word":

> word
>
> - a unit of language, consisting of one or more spoken sounds or their written representation, that functions as a principal carrier of meaning.
> - speech or talk: to express one's emotion in words.
> - a short talk or conversation: "Marston, I'd like a word with you."
> - an expression or utterance: a word of warning.

Data Representation

If you were to store this data in your program, you would probably want to be able to quickly look up the definitions using the *key* "word". With a map, a vector of definitions could be stored as the *value* corresponding to the "word" key:

Key `string`	Value `vector<string>`
`"word"`	`<"a unit of language, consisting of one or more spoken sounds or their written representation, that functions as a principal carrier of meaning.", "speech or talk: to express one's emotion in words.", "a short talk or conversation: 'Marston, I'd like a word with you.'", "an expression or utterance: a word of warning.">'`

In the following notebook, you will learn how to use an `unordered_map`, which is the C++ standard library implementation of a map. Although C++ has several different implementations of map data structures which are similar, `unordered_map` is the structure that you will use in your project.

If you are taking this course, you have probably used object-oriented programming (OOP) previously in another language. If it's been a while since you've used OOP, OOP is a style of coding that collects related data (object *attributes*) and functions (object *methods*) together to form a single data structure, called an *object*. This allows that collection of attributes and methods to be used repeatedly in your program without code repetition.

In C++ the attributes and methods that make up an object are specified in a code *class*, and each object in the program is an *instance* of that class.

This concept is intended to provide you with the basic syntax for writing classes in C++. In this Foundations course, you will not need to write your own classes for the project, but you will be modifying existing classes in the code. You will be writing your own classes in the next course of this Nanodegree: Object-Oriented Programming.

Inheritance

It is possible for a class to use methods and attributes from another class using class *inheritance*. For example, if you wanted to make a `Sedan` class with additional attributes or methods not found in the generic `Car` class, you could create a `Sedan` class that inherited from the `Car` by using the colon notation:

```
class Sedan : public Car {
    // Sedan class declarations/definitions here.
};
```

By doing this, each `Sedan` class instance will have access to any of the *public* methods and attributes of `Car`. In the code above, these are `IncrementDistance()` and `PrintCarData()`. You can add additional features to the `Sedan` class as well. In the example above, `Car` is often referred to as the *parent* class, and `Sedan` as the *child* or *derived* class.

A full discussion of inheritance is beyond the scope of this course, but you will encounter it briefly in the project code later. In the project code, the classes are set up to inherit from existing classes of an open source code project. You won't need to use inheritance otherwise, but keep in mind that your classes can use all of the public methods and attributes of their parent class.

This Pointer

When working with classes it is often helpful to be able to refer to the current class instance or object. For example, given the following `Car` class from a previous lesson, the `IncrementDistance()` method implicitly refers to the current `Car` instance's `distance` attribute:

```
// The Car class
class Car {
  public:
    // Method to print data.
    void PrintCarData() {
        cout << "The distance that the " << color << " car " << number << " has traveled is: " << distanc
    }

    // Method to increment the distance travelled.
    void IncrementDistance() {
        distance++;
    }

    // Class/object attributes
    string color;
    int distance = 0;
    int number;
};
```

It is possible to make this explicit in C++ by using the `this` pointer, which points to the current class instance. Using `this` can sometimes be helpful to add clarity to more complicated code:

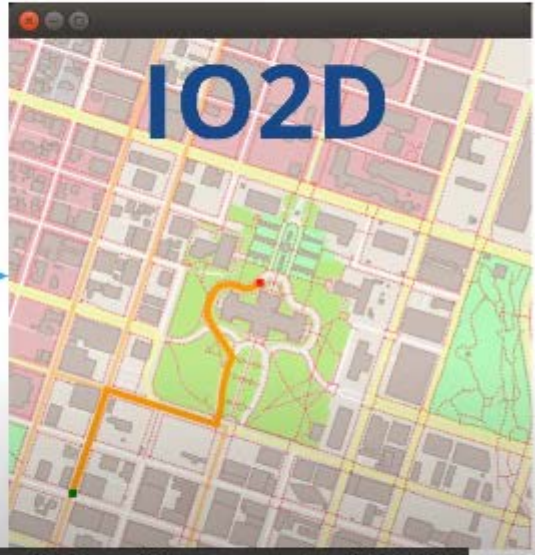
```
// The Car class
class Car {
  public:
    // Method to print data.
    void PrintCarData() {
        cout << "The distance that the " << this->color << " car " << this->number << " has traveled is:
    }

    // Method to increment the distance travelled.
    void IncrementDistance() {
        this->distance++;
    }

    // Class/object attributes
    string color;
    int distance = 0;
    int number;
};
```

Note: you may see `this` used in some code in the remainder of the course.

Lesson Recap

- Header files
  - Used headers to break a single file into multiple files
- Build system
  - CMake and Make
- Tools for writing larger programs:
  - References
  - Pointers
  - Maps
  - Classes and object-oriented programming in C++

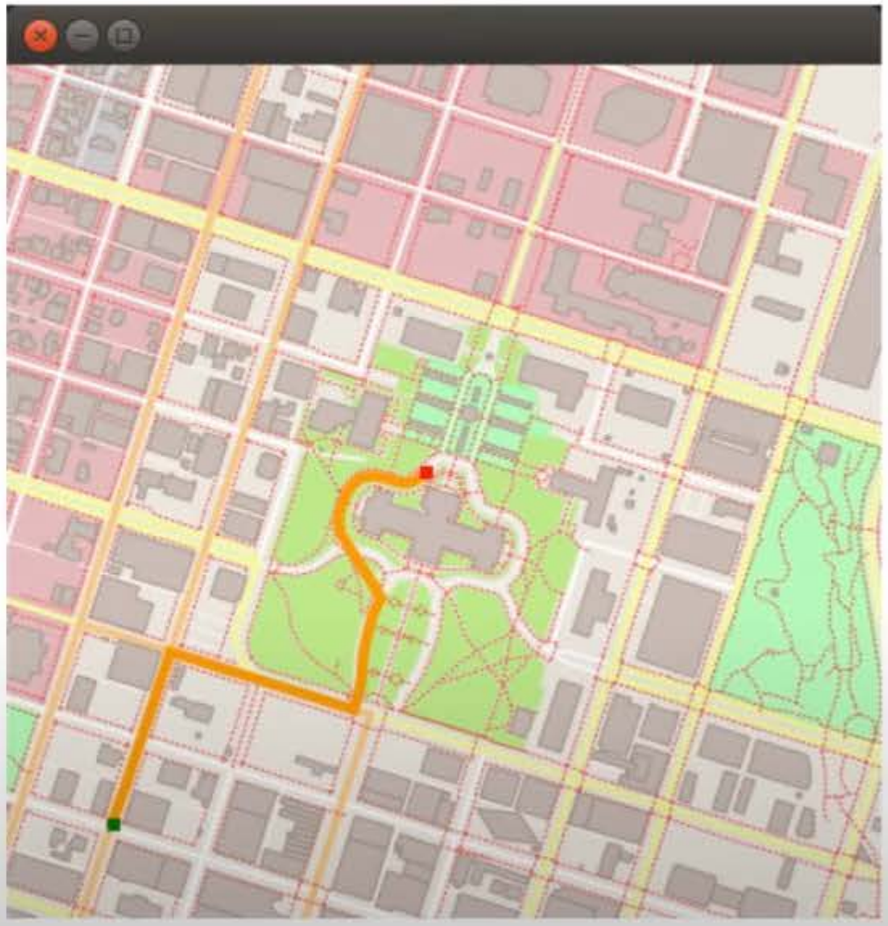
```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap
0.0.2">
 <bounds minlat="54.0889580"
minlon="12.2487570" maxlat="54.0913900"
maxlon="12.2524800"/>
 <node id="298884269" lat="54.0901746"
lon="12.2482632" user="SvenHRO"
uid="46882" visible="true" version="1"
changeset="676636"
timestamp="2008-09-21T21:37:45Z"/>
 <node id="261728686" lat="54.0906309"
lon="12.2441924" user="PikoWinter"
uid="36744" visible="true" version="1"
changeset="323878"
timestamp="2008-05-03T13:39:23Z"/>
 <node id="1831881213" version="1"
```



the project code we'll be using a 2D rendering library called IO2D.

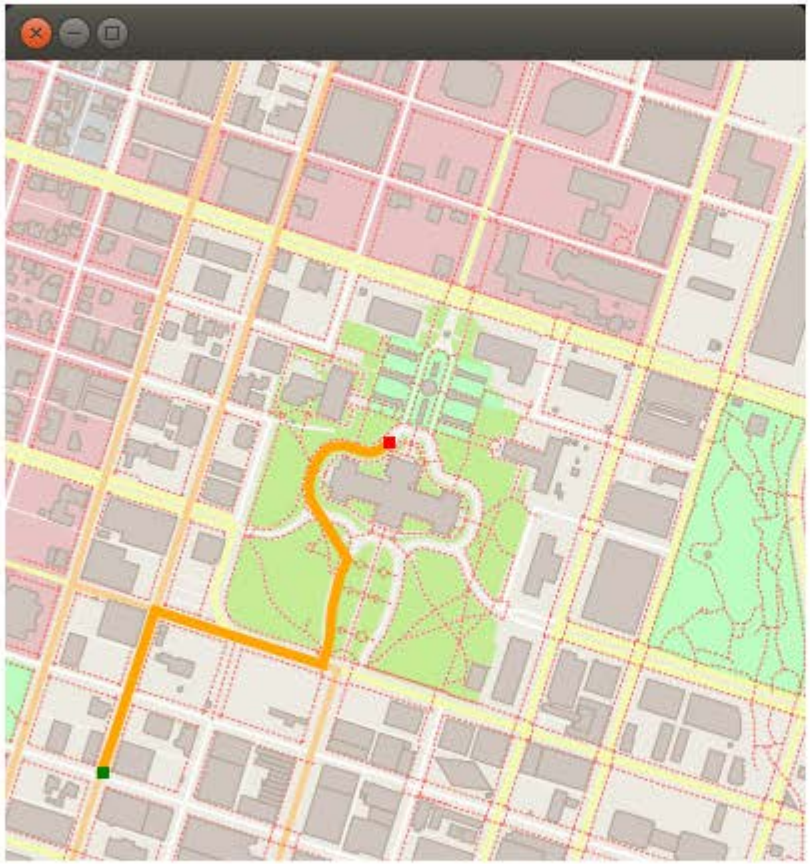
Lesson Outline

- Overview of the OpenStreetMap Project
  - Data format
  - Element types
- Building and running the project
- Testing your code
- Project code overview
  - Class diagram
- Project starter repository structure
  - Files and folders
- The `src` directory
  - Code walkthroughs
- Steps to complete the project
- Project submission instructions and works



there'll be project submission instructions and workspace.

Welcome to the course project! In this project, you will create a route planner that plots a path between two points on a map using real map data from the OpenStreeMap project. When you are finished with the project, your output should look like the image below:



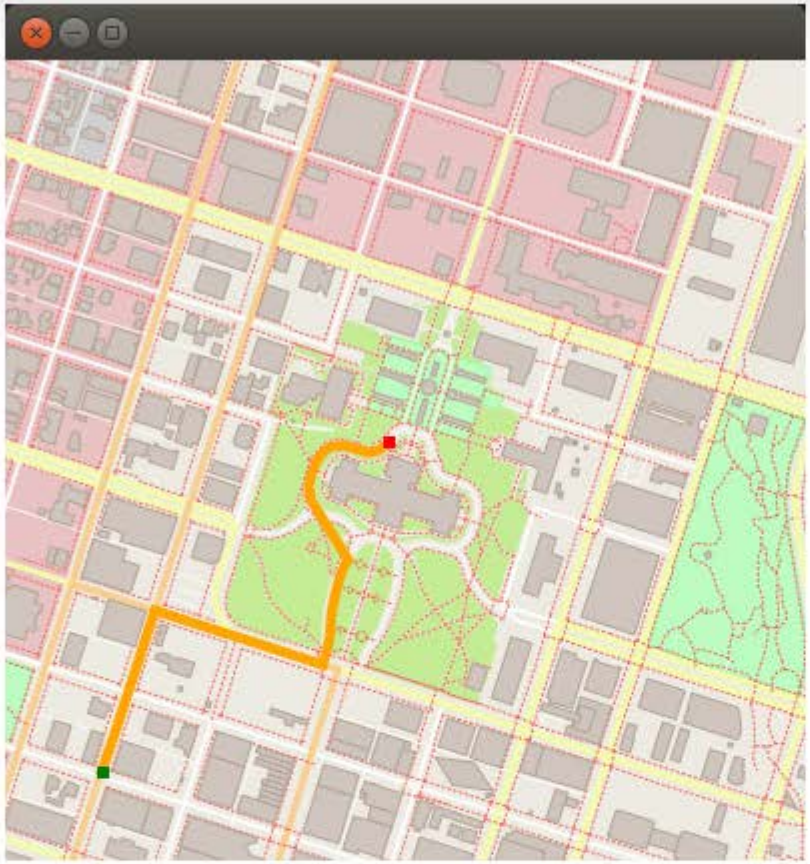
A path rendered between two points on a map.

This starting code for this project comes from a map rendering example in the Github repo for the 2D Graphics Library, IO2D. In your project, you will be extending the code to search for and display a path between two points on the map.

The OpenStreetMap Project

In the next classroom concepts, you will write a course project in C++ using real map data and A* search to find a path between two points, just as you might see in a desktop or mobile mapping application. The project you will will write will be using data from the OpenStreetMap project.

The OpenStreetMap project is an open source, collaborative endeavor to create free, user-generated maps of every part of the world. These maps are similar to the maps you might use in Google Maps or the Apple Maps app on your phone, but they are completely generated by individuals who volunteer to perform ground surveys of their local environment.



Example OpenStreetMap with plotted path

OpenStreetMap Data

OpenStreetMap data can come in several different formats. The data that is used for this project comes in the form of an OSM XML file (.osm file), and we have provided a sample from the OpenStreetMap Wiki below. Although you may not have worked with this type of data before, have a look at the data below and try to guess which element types from the next quiz are present in the data. We are confident you can figure this out by carefully studying the data!

QUESTION 1 OF 2

Have a look at the element types in the XML map data below. Which of the following types do you see in the data?

- node
- way
- relation

SUBMIT

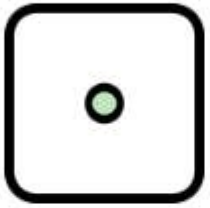
```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.0.2">
 <bounds minlat="54.0889580" minlon="12.2487570" maxlat="54.0913900" maxlon="12.2524800"/>
 <node id="298884269" lat="54.0901746" lon="12.2482632" user="SvenHRO" uid="46882" visible="true" version
 <node id="261728686" lat="54.0906309" lon="12.2441924" user="PikoWinter" uid="36744" visible="true" vers
 <node id="1831881213" version="1" changeset="12370172" lat="54.0900666" lon="12.2539381" user="lafkor" u
  <tag k="name" v="Neu Broderstorf"/>
  <tag k="traffic_sign" v="city_limit"/>
 </node>
 ...
 <node id="298884272" lat="54.0901447" lon="12.2516513" user="SvenHRO" uid="46882" visible="true" version
 <way id="26659127" user="Masch" uid="55988" visible="true" version="5" changeset="4142606" timestamp="20
  <nd ref="292403538"/>
  <nd ref="298884289"/>
  ...
  <nd ref="261728686"/>
  <tag k="highway" v="unclassified"/>
  <tag k="name" v="Pastower Straße"/>
 </way>
 <relation id="56688" user="kmvar" uid="56190" visible="true" version="28" changeset="6947637" timestamp=
  <member type="node" ref="294942404" role=""/>
  ...
  <member type="node" ref="364933006" role=""/>
  <member type="way" ref="4579143" role=""/>
  ...
  <member type="node" ref="249673494" role=""/>
  <tag k="name" v="Küstenbus Linie 123"/>
  <tag k="network" v="VVW"/>
  <tag k="operator" v="Regionalverkehr Küste"/>
  <tag k="ref" v="123"/>
  <tag k="route" v="bus"/>
  <tag k="type" v="route"/>
 </relation>
 ...
</osm>
```

```
913900" maxlon="12.2524800"/>
"SvenHRO" uid="46882" visible="true" version="1" changeset="676636" timestamp="2008-09-21T21:37:45Z"/>
"PikoWinter" uid="36744" visible="true" version="1" changeset="323878" timestamp="2008-05-03T13:39:23Z"/>
54.0900666" lon="12.2539381" user="lafkor" uid="75625" visible="true" timestamp="2012-07-20T09:43:19Z">



"SvenHRO" uid="46882" visible="true" version="1" changeset="676636" timestamp="2008-09-21T21:37:45Z"/>
ersion="5" changeset="4142606" timestamp="2010-03-16T11:47:08Z">



 version="28" changeset="6947637" timestamp="2011-01-12T14:23:49Z">
```

Data Types Overview

Excellent work in guessing the types present in the data above! If you look closely at the XML element types in the sample above, you should see the three element types which are important to the code you will be writing: Nodes, Ways, and Relations.

Node



A *node* is one of the most basic elements in the OpenStreetMap data model. Each node indicates a single point with an identifier `id`, latitude `lat`, and longitude `lon`. There are other XML attributes in a node element that won't be relevant to this project, such as the `user` id and the `timestamp` when the node was added to the data set. Additionally, a node can have several tags which provide additional information.

Way

A *way* is an ordered list of nodes that represents a feature in the map. This feature could be a road, or a boundary of a park, or some other feature in the map. Each way has at least one `tag` which denotes some information about the way, and each way also belongs to at least one relation, which is described below.

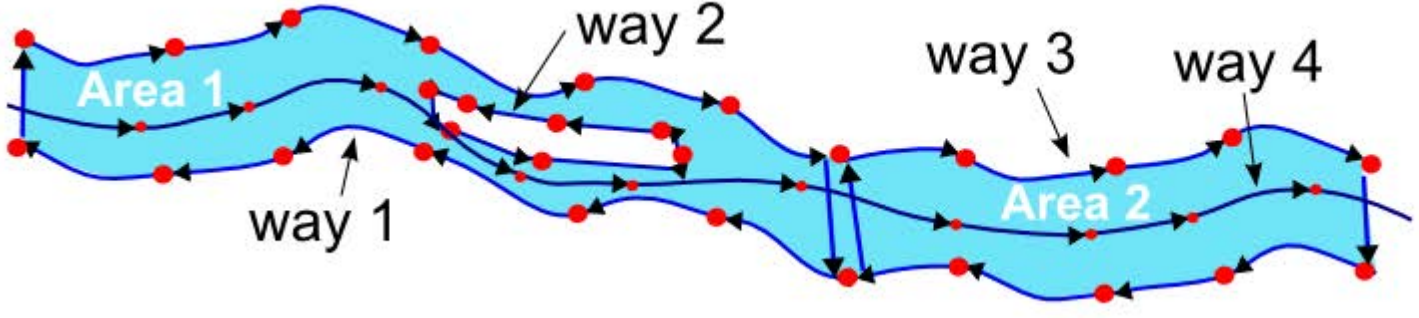
Relation

A *relation* is a data structure which documents a relationship between other data elements. Examples from the OpenStreetMap wiki include:

> - A route relation which lists the ways that form a major highway, cycle route, or bus route.
> - A multipolygon that describes an area with holes, where the outer and inner boundaries of the area are given by two ways.

Example

To help you understand how all these types are used together, consider the following example from the OpenStreetMap wiki of mapping a large river with distinct banks on either side of the river. In the image below, nodes are used to provide the coordinates of points along the banks of the river. Multiple nodes are then connected using ways; there are ways which form closed sections of the river, labeled as "Areas" in the image below. Another way is used to represent the island in the middle of the river. These ways are then grouped together using a relation, which represents the entire river.



QUESTION 2 OF 2

Imagine you are going to create a bus route using OpenStreetMap data. Match the OpenStreetMap data type with the part of the bus route it represents.

Submit to check your answer choices!

BUS ROUTE PART	OSM DATA TYPE
A collection of points that form a road on the bus route	Way
A single point on the bus route	Node
A collection of roads the form the entire route	Relation

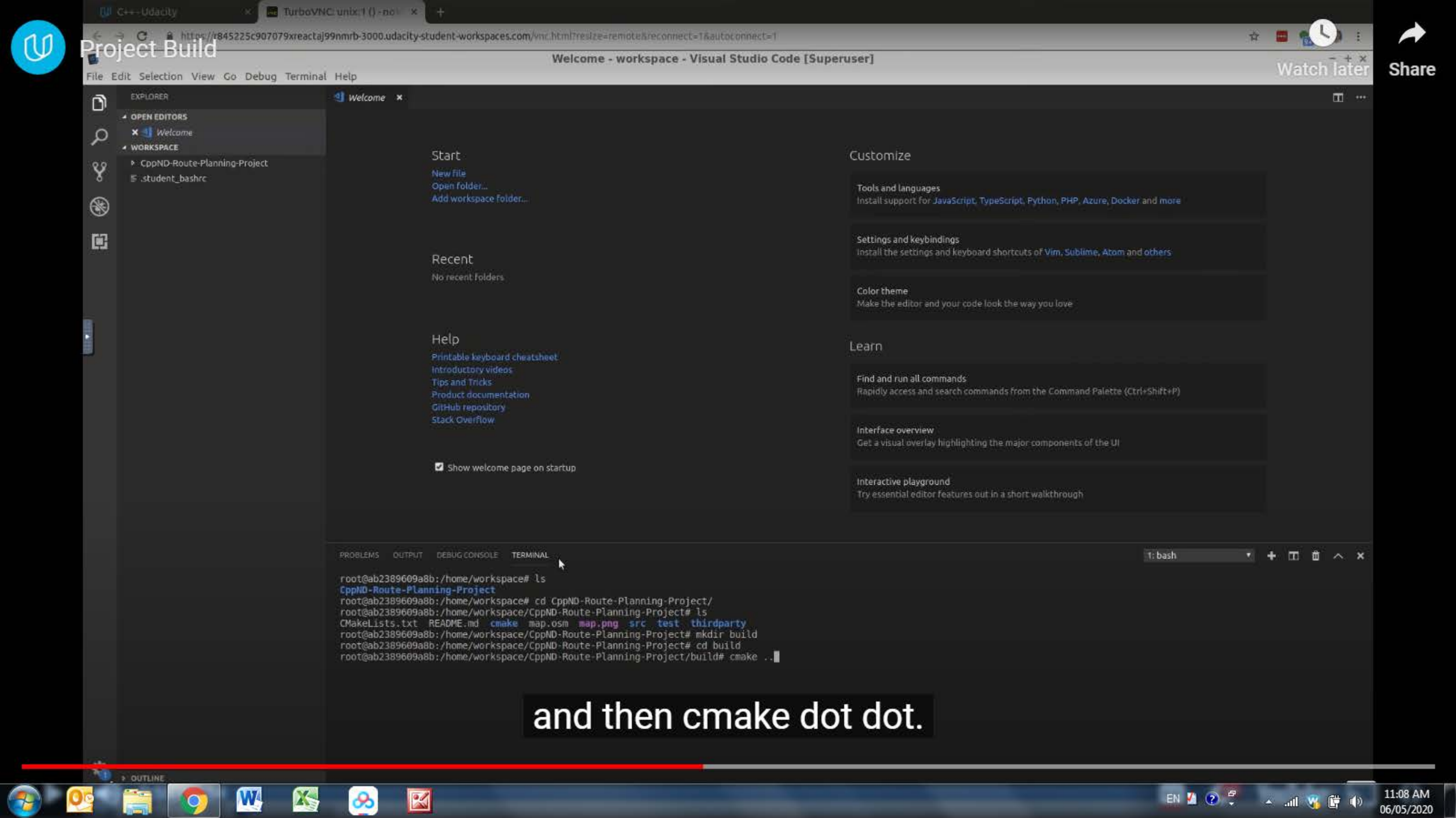
SUBMIT

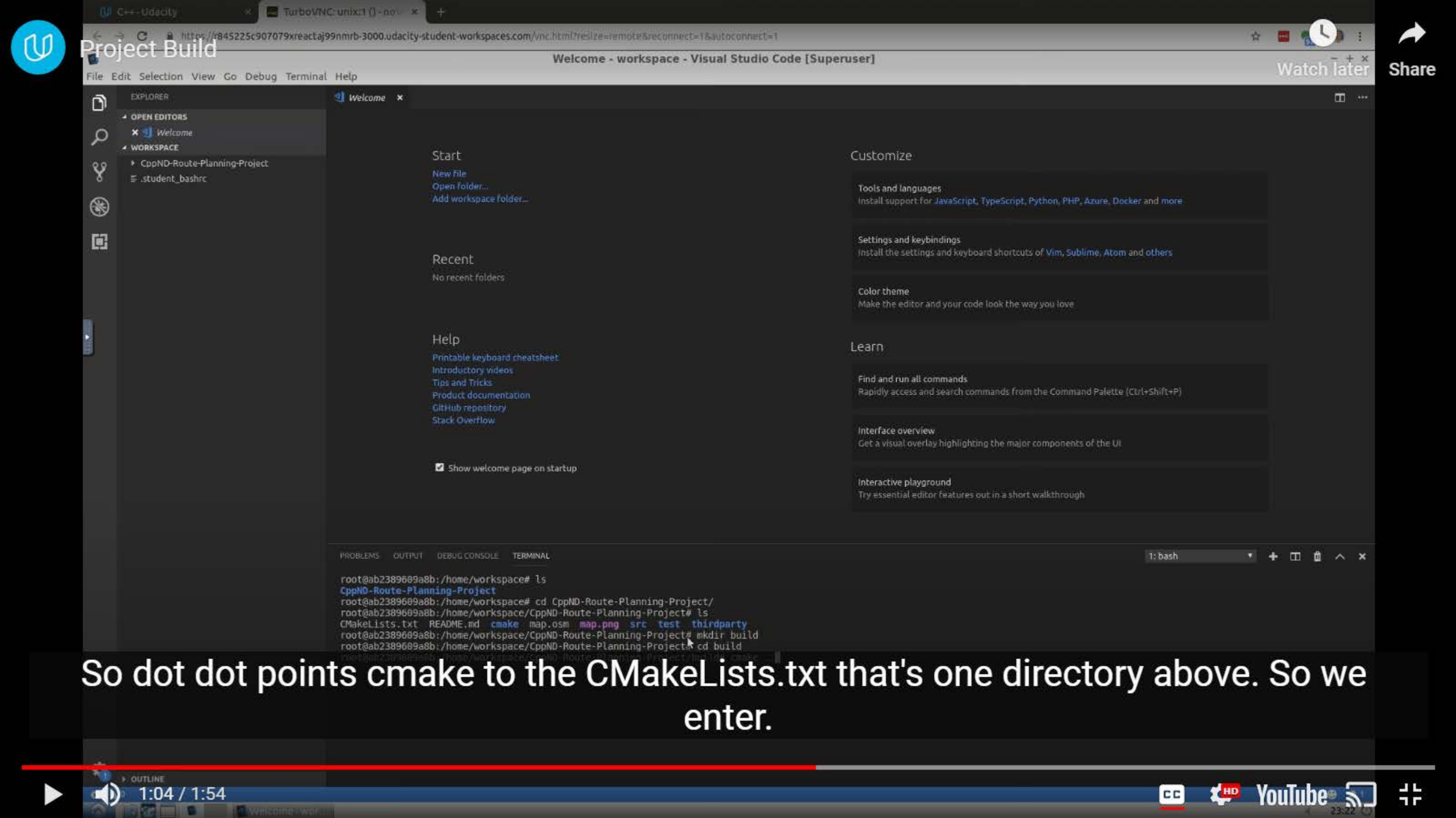
Up Next

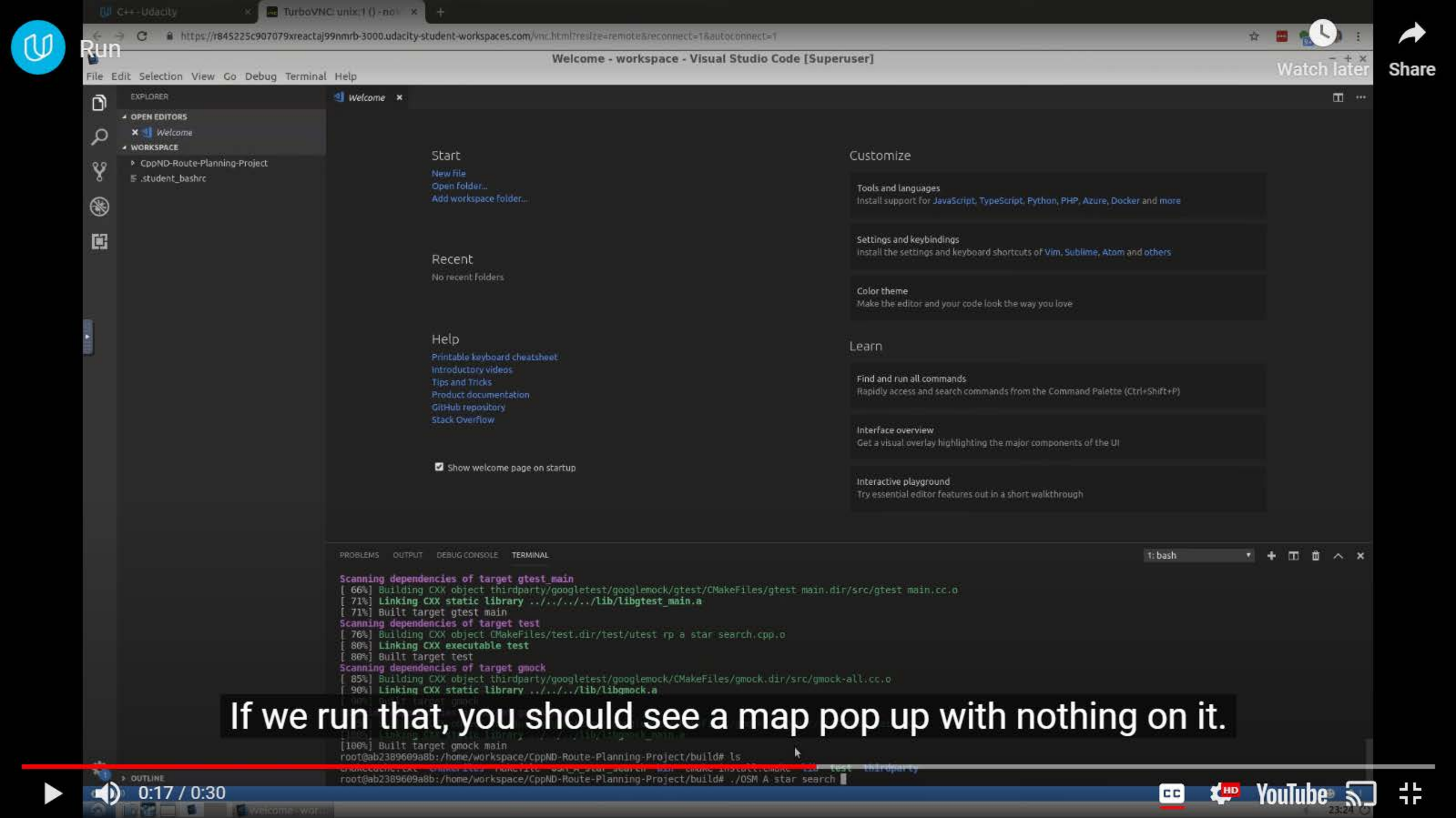
For your project, you will primarily be working with Nodes, although the code that we provide determines which nodes are neighbors by using the ways and relations those nodes belong to.

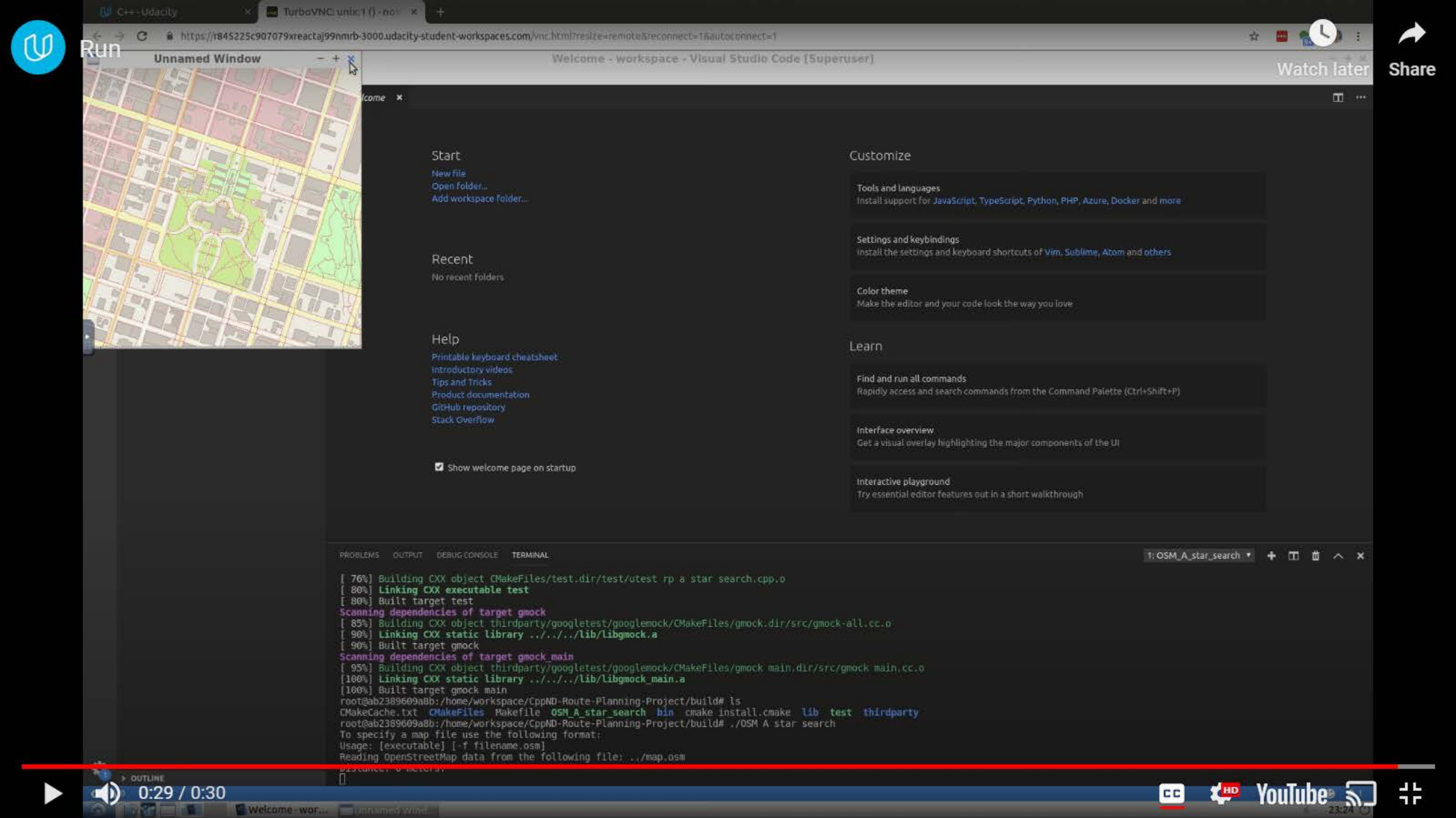
Both the code to parse the OSM data and the data structures which are used to store the data in your program have already been written in the IO2D OpenStreetMap example. In your project, you won't need to recreate any of this code - you task will be writing code that extends the code in order to plot a path between two points.

Up next, you will learn how to build, run, and test the project code. After that, we'll have an in-depth look at the files in the project starter code so you will be prepared to work on your project.


Project Build
Watch later
Share
Welcome - workspace - Visual Studio Code [Superuser]
File Edit Selection View Go Debug Terminal Help
EXPLORER
OPEN EDITORS
Welcome
WORKSPACE
CppND-Route-Planning-Project
.student_bashrc
Start
New file
Open folder...
Add workspace folder...
Recent
No recent folders
Help
Printable keyboard cheatsheet
Introductory videos
Tips and Tricks
Product documentation
GitHub repository
Stack Overflow
Show welcome page on startup
Customize
Tools and languages
Install support for JavaScript, TypeScript, Python, PHP, Azure, Docker and more
Settings and keybindings
Install the settings and keyboard shortcuts of Vim, Sublime, Atom and others
Color theme
Make the editor and your code look the way you love
Learn
Find and run all commands
Rapidly access and search commands from the Command Palette (Ctrl+Shift+P)
Interface overview
Get a visual overlay highlighting the major components of the UI
Interactive playground
Try essential editor features out in a short walkthrough
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
root@ab2389609a8b:/home/workspace# ls
CppND-Route-Planning-Project
root@ab2389609a8b:/home/workspace# cd CppND-Route-Planning-Project/
root@ab2389609a8b:/home/workspace/CppND-Route-Planning-Project# ls
CMakeLists.txt README.md cmake map.osm map.png src test thirdparty
root@ab2389609a8b:/home/workspace/CppND-Route-Planning-Project# mkdir build
root@ab2389609a8b:/home/workspace/CppND-Route-Planning-Project# cd build
root@ab2389609a8b:/home/workspace/CppND-Route-Planning-Project/build# cmake ..
and then cmake dot dot.
OUTLINE
EN
11:08 AM
06/05/2020


Project Build
Watch later
Share
Welcome - workspace - Visual Studio Code [Superuser]
File Edit Selection View Go Debug Terminal Help
EXPLORER
OPEN EDITORS
Welcome
WORKSPACE
CppND-Route-Planning-Project
.student_bashrc
Start
New file
Open folder...
Add workspace folder...
Recent
No recent folders
Help
Printable keyboard cheatsheet
Introductory videos
Tips and Tricks
Product documentation
GitHub repository
Stack Overflow
Show welcome page on startup
Customize
Tools and languages
Install support for JavaScript, TypeScript, Python, PHP, Azure, Docker and more
Settings and keybindings
Install the settings and keyboard shortcuts of Vim, Sublime, Atom and others
Color theme
Make the editor and your code look the way you love
Learn
Find and run all commands
Rapidly access and search commands from the Command Palette (Ctrl+Shift+P)
Interface overview
Get a visual overlay highlighting the major components of the UI
Interactive playground
Try essential editor features out in a short walkthrough
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
root@ab2389609a8b:/home/workspace# ls
CppND-Route-Planning-Project
root@ab2389609a8b:/home/workspace# cd CppND-Route-Planning-Project/
root@ab2389609a8b:/home/workspace/CppND-Route-Planning-Project# ls
CMakeLists.txt README.md cmake map.osm map.png src test thirdparty
root@ab2389609a8b:/home/workspace/CppND-Route-Planning-Project# mkdir build
root@ab2389609a8b:/home/workspace/CppND-Route-Planning-Project# cd build
So dot dot points cmake to the CMakeLists.txt that's one directory above. So we enter.
OUTLINE
1:04 / 1:54
YouTube


Run
Watch later
Share
Welcome - workspace - Visual Studio Code [Superuser]
File Edit Selection View Go Debug Terminal Help
EXPLORER
OPEN EDITORS
Welcome
WORKSPACE
CppND-Route-Planning-Project
.student_bashrc
Start
New file
Open folder...
Add workspace folder...
Recent
No recent folders
Help
Printable keyboard cheatsheet
Introductory videos
Tips and Tricks
Product documentation
GitHub repository
Stack Overflow
Show welcome page on startup
Customize
Tools and languages
Install support for JavaScript, TypeScript, Python, PHP, Azure, Docker and more
Settings and keybindings
Install the settings and keyboard shortcuts of Vim, Sublime, Atom and others
Color theme
Make the editor and your code look the way you love
Learn
Find and run all commands
Rapidly access and search commands from the Command Palette (Ctrl+Shift+P)
Interface overview
Get a visual overlay highlighting the major components of the UI
Interactive playground
Try essential editor features out in a short walkthrough
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
Scanning dependencies of target gtest_main
[66%] Building CXX object thirdparty/googletest/googlemock/gtest/CMakeFiles/gtest_main.dir/src/gtest_main.cc.o
[71%] Linking CXX static library ../../../../lib/libgtest_main.a
[71%] Built target gtest_main
Scanning dependencies of target test
[76%] Building CXX object CMakeFiles/test.dir/test/utest_rp_a_star_search.cpp.o
[80%] Linking CXX executable test
[80%] Built target test
Scanning dependencies of target gmock
[85%] Building CXX object thirdparty/googletest/googlemock/CMakeFiles/gmock.dir/src/gmock-all.cc.o
[90%] Linking CXX static library ../../../lib/libgmock.a
[100%] Built target gmock_main
root@ab2389609a8b:/home/workspace/CppND-Route-Planning-Project/build# ls
root@ab2389609a8b:/home/workspace/CppND-Route-Planning-Project/build# ./OSM_A_star_search
If we run that, you should see a map pop up with nothing on it.
OUTLINE
0:17 / 0:30
CC
HD
YouTube


Run
Share
Watch later
Unnamed Window
Welcome - workspace - Visual Studio Code [Superuser]
Start
New file
Open folder...
Add workspace folder...
Recent
No recent folders
Help
Printable keyboard cheatsheet
Introductory videos
Tips and Tricks
Product documentation
GitHub repository
Stack Overflow
Show welcome page on startup
Customize
Tools and languages
Install support for JavaScript, TypeScript, Python, PHP, Azure, Docker and more
Settings and keybindings
Install the settings and keyboard shortcuts of Vim, Sublime, Atom and others
Color theme
Make the editor and your code look the way you love
Learn
Find and run all commands
Rapidly access and search commands from the Command Palette (Ctrl+Shift+P)
Interface overview
Get a visual overlay highlighting the major components of the UI
Interactive playground
Try essential editor features out in a short walkthrough
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: OSM_A_star_search
[76%] Building CXX object CMakeFiles/test.dir/test/utest_rp_a_star_search.cpp.o
[80%] Linking CXX executable test
[80%] Built target test
Scanning dependencies of target gmock
[85%] Building CXX object thirdparty/googletest/googlemock/CMakeFiles/gmock.dir/src/gmock-all.cc.o
[90%] Linking CXX static library ../../../lib/libgmock.a
[90%] Built target gmock
Scanning dependencies of target gmock_main
[95%] Building CXX object thirdparty/googletest/googlemock/CMakeFiles/gmock_main.dir/src/gmock_main.cc.o
[100%] Linking CXX static library ../../../lib/libgmock_main.a
[100%] Built target gmock_main
root@ab2389609a8b:/home/workspace/CppND-Route-Planning-Project/build# ls
CMakeCache.txt CMakeFiles Makefile OSM_A_star_search bin cmake_install.cmake lib test thirdparty
root@ab2389609a8b:/home/workspace/CppND-Route-Planning-Project/build# ./OSM_A_star_search
To specify a map file use the following format:
Usage: [executable] [-f filename.osm]
Reading OpenStreetMap data from the following file: ../map.osm
OUTLINE
0:29 / 0:30
YouTube

Building and Running

To get started with your project, you can download the code from the GitHub repo here, or you can use the workspace provided below. If you decide to work on this project on your local machine, you will need to install the dependencies outlined in the GitHub README for the project.

In this classroom concept, you will see the instructions for building, running, and testing your project. In the next classroom concept, you will see a detailed overview of the project code, so you'll be well prepared to start work on the project.

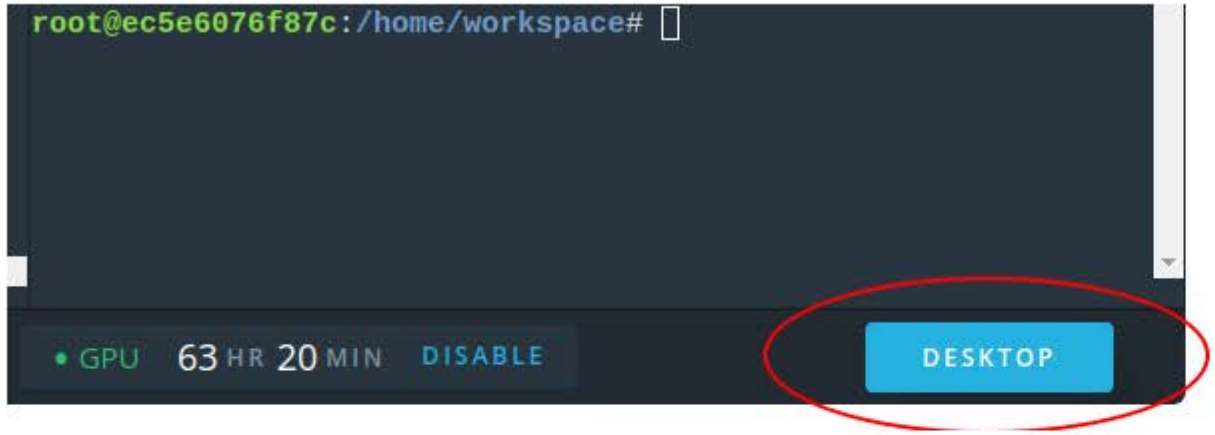
Building

This project will use CMake as the build system. Building the project code can be done from the terminal workspace or from the virtual desktop workspace. It is also possible to build this project locally. To do this, you will need to install the project dependencies in your local environment, including the IO2D library, which can be difficult. The complete list of dependencies can be found in the GitHub README for the project.

Building Using Udacity Workspaces

To build the code from the terminal workspace, GPU must be enabled. If using the virtual desktop, **you will want to use a terminal from within VS Code**, as the correct compiler options have been set up in the VS Code preferences. See the video below for a walkthrough of how to do this.

The video above demonstrates building the project in the virtual desktop. To follow along, be sure to click on the "Desktop" button in your workspace to open a virtual desktop window:



In the workspace desktop, you can open Visual Studio Code and a terminal within Visual Studio. To build the code, make a `build` directory and then `cd` into that directory. From within the `build` directory, you can then run the following commands to compile:

```
cmake ..
make
```

Running

Once the build is complete, the executable binary files will be in the `build` folder of the project, and the project will be ready to run.

Running the project must be done from the workspace desktop in order for you to be able to see the mapped output. In the terminal, navigate to the build folder. From the `build` directory, you can run the compiled executable with map data using the following command:

```
./OSM_A_star_search
```

Or to specify a map file:

```
./OSM_A_star_search -f ../<your_osm_file.osm>
```

When you run the project for the first time, you should see a blank map like the one below:

When running the project code for the first time, you should see a map like the one above.

Your project is supplied with unit tests that you can use to ensure that you are on the right track. When you complete the project, the unit tests should all pass. The tests are located in the `<PROJECT_DIRECTORY>/test/utest_rp_a_star_search.cpp` file, and you can view the tests to help you as you write your code.

After you compile the project, the testing executable is placed in the `build` directory along with the project executable. You can run the tests from the `build` directory as follows:

```
./test
```

The initial output of your tests should look something like the following:

```
[==========] Running 4 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 4 tests from RoutePlannerTest
[ RUN      ] RoutePlannerTest.TestCalculateHValue
/home/stephen/Desktop/CppND-Route-Planning-Project/test/utest_rp_a_star_search.cpp:66: Failure
Expected equality of these values:
  route_planner.CalculateHValue(start_node)
    Which is: 0.031656995
  1.1329799
/home/stephen/Desktop/CppND-Route-Planning-Project/test/utest_rp_a_star_search.cpp:68: Failure
Expected equality of these values:
  route_planner.CalculateHValue(mid_node)
    Which is: 0
  0.58903033
[  FAILED  ] RoutePlannerTest.TestCalculateHValue (109 ms)
[ RUN      ] RoutePlannerTest.TestAddNeighbors
/home/stephen/Desktop/CppND-Route-Planning-Project/test/utest_rp_a_star_search.cpp:82: Failure
Expected equality of these values:
  neighbors.size()
    Which is: 0
  4
[  FAILED  ] RoutePlannerTest.TestAddNeighbors (78 ms)
[ RUN      ] RoutePlannerTest.TestConstructFinalPath
/home/stephen/Desktop/CppND-Route-Planning-Project/test/utest_rp_a_star_search.cpp:102: Failure
Expected equality of these values:
  path.size()
    Which is: 0
  3
Segmentation fault (core dumped)
```

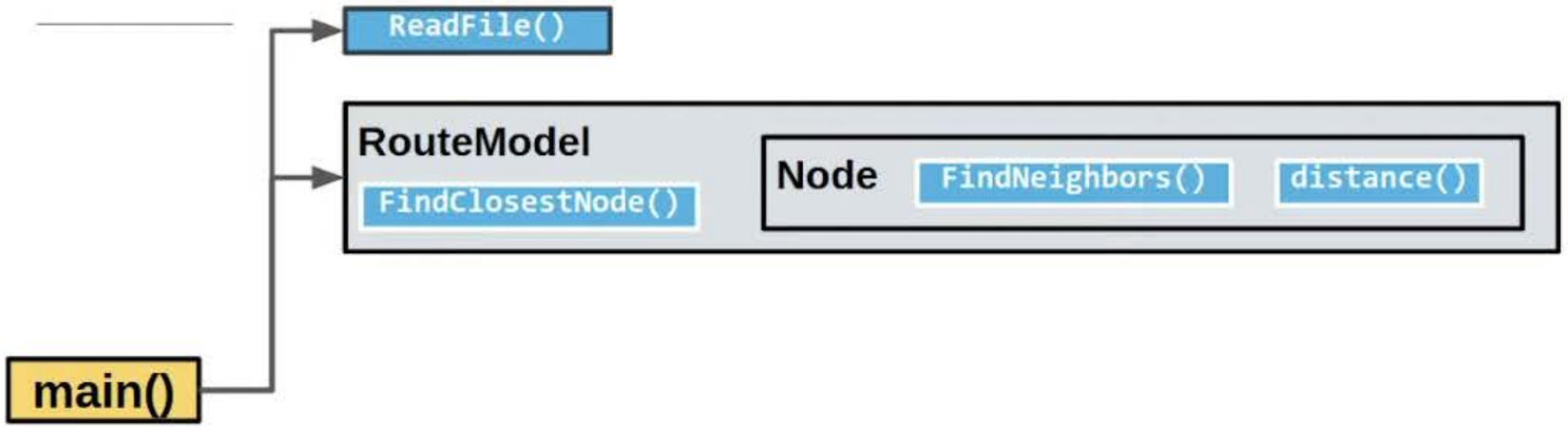
When you have completed the project, the tests should look like the following:

```
[==========] Running 4 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 4 tests from RoutePlannerTest
[ RUN      ] RoutePlannerTest.TestCalculateHValue
[       OK ] RoutePlannerTest.TestCalculateHValue (114 ms)
[ RUN      ] RoutePlannerTest.TestAddNeighbors
[       OK ] RoutePlannerTest.TestAddNeighbors (81 ms)
[ RUN      ] RoutePlannerTest.TestConstructFinalPath
[       OK ] RoutePlannerTest.TestConstructFinalPath (81 ms)
[ RUN      ] RoutePlannerTest.TestAStarSearch
[       OK ] RoutePlannerTest.TestAStarSearch (85 ms)
[----------] 4 tests from RoutePlannerTest (361 ms total)

[----------] Global test environment tear-down
[==========] 4 tests from 1 test case ran. (361 ms total)
[  PASSED  ] 4 tests.
```

IO2D

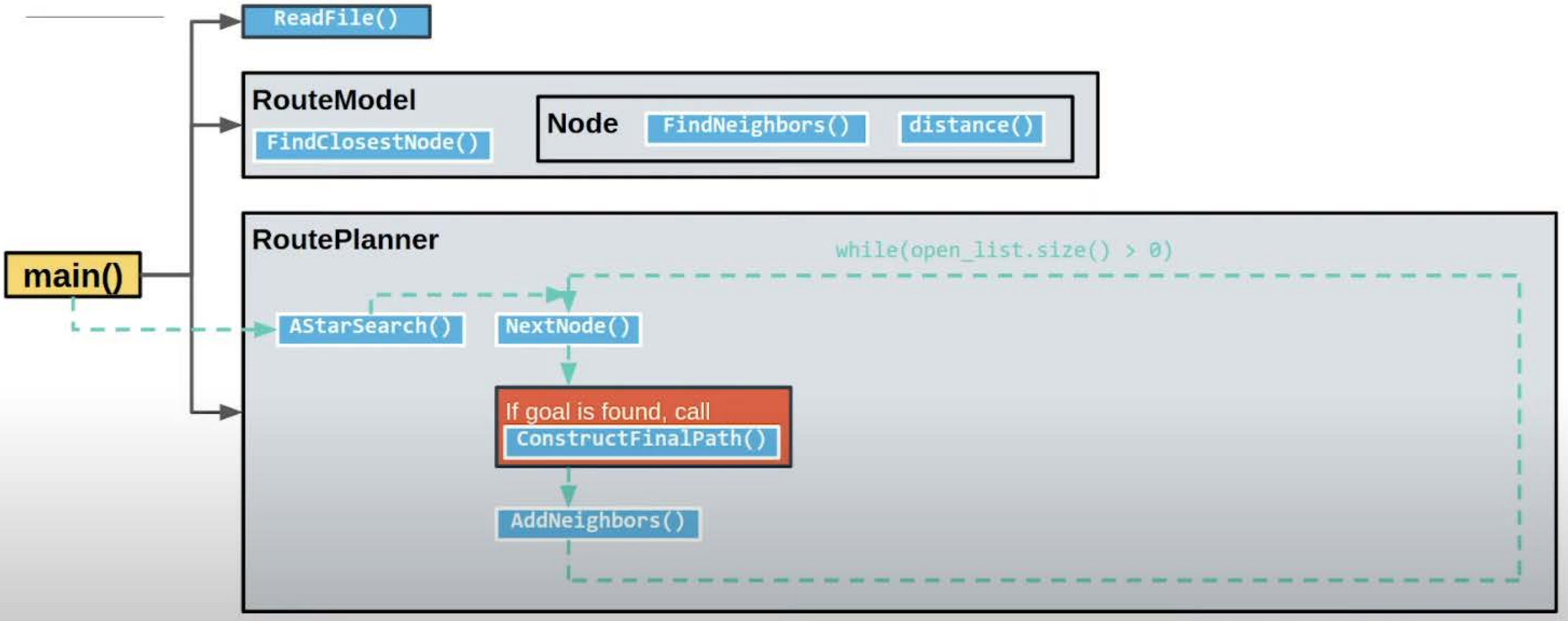
This project extends the IO2D OpenStreetMap demo to render a path between two points.

OSM Route Planner Code Structure:



The route model class has a subclass called node which is used frequently.

OSM Route Planner Code Structure:



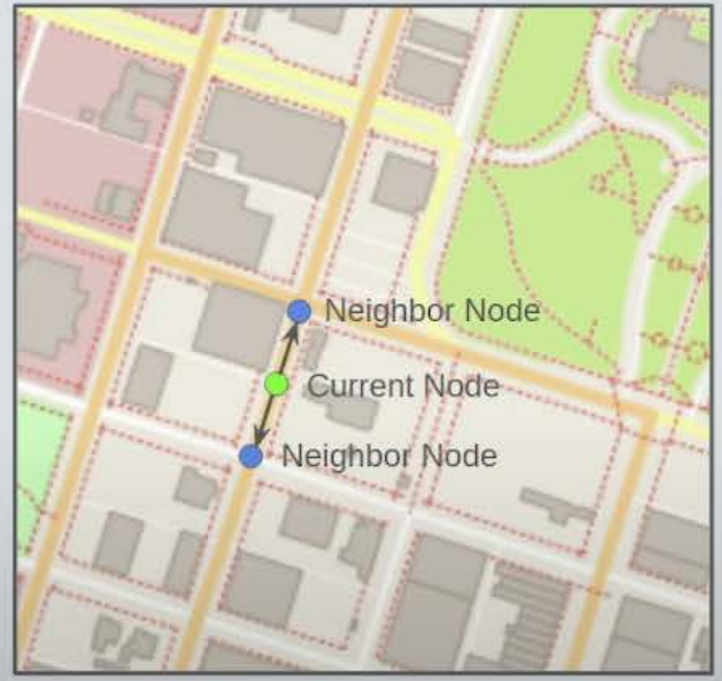
So let's examine them in more detail.

OSM Route Planner AddNeighbors:

RoutePlanner

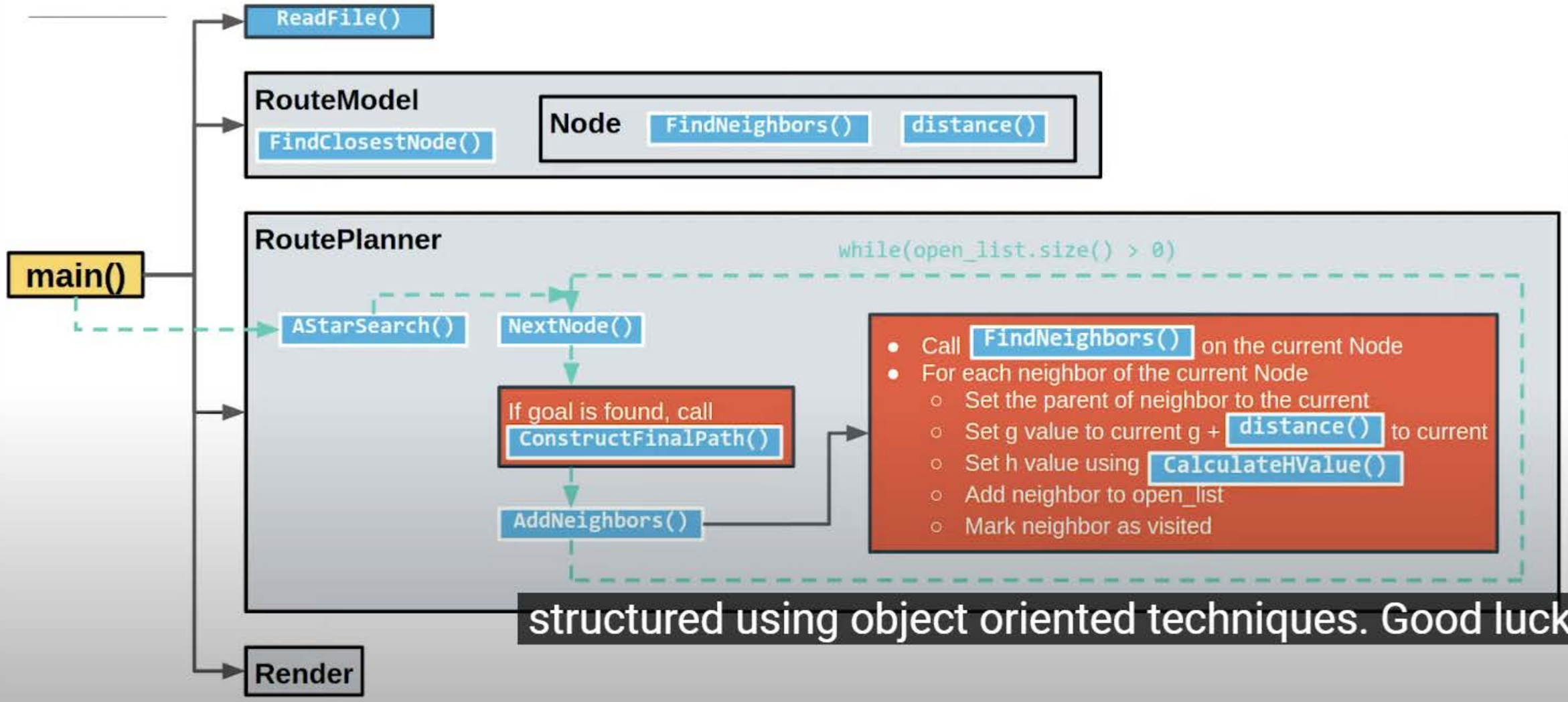
`AddNeighbors()`

- Call `FindNeighbors()` on the current Node
- For each neighbor of the current Node
  - Set the parent of neighbor to the current
  - Set g value to current g + `distance()` to current
  - Set h value using `CalculateHValue()`
  - Add neighbor to open_list
  - Mark neighbor as visited



the open list and then the neighbor is also marked as visited.

OSM Route Planner Code Structure:

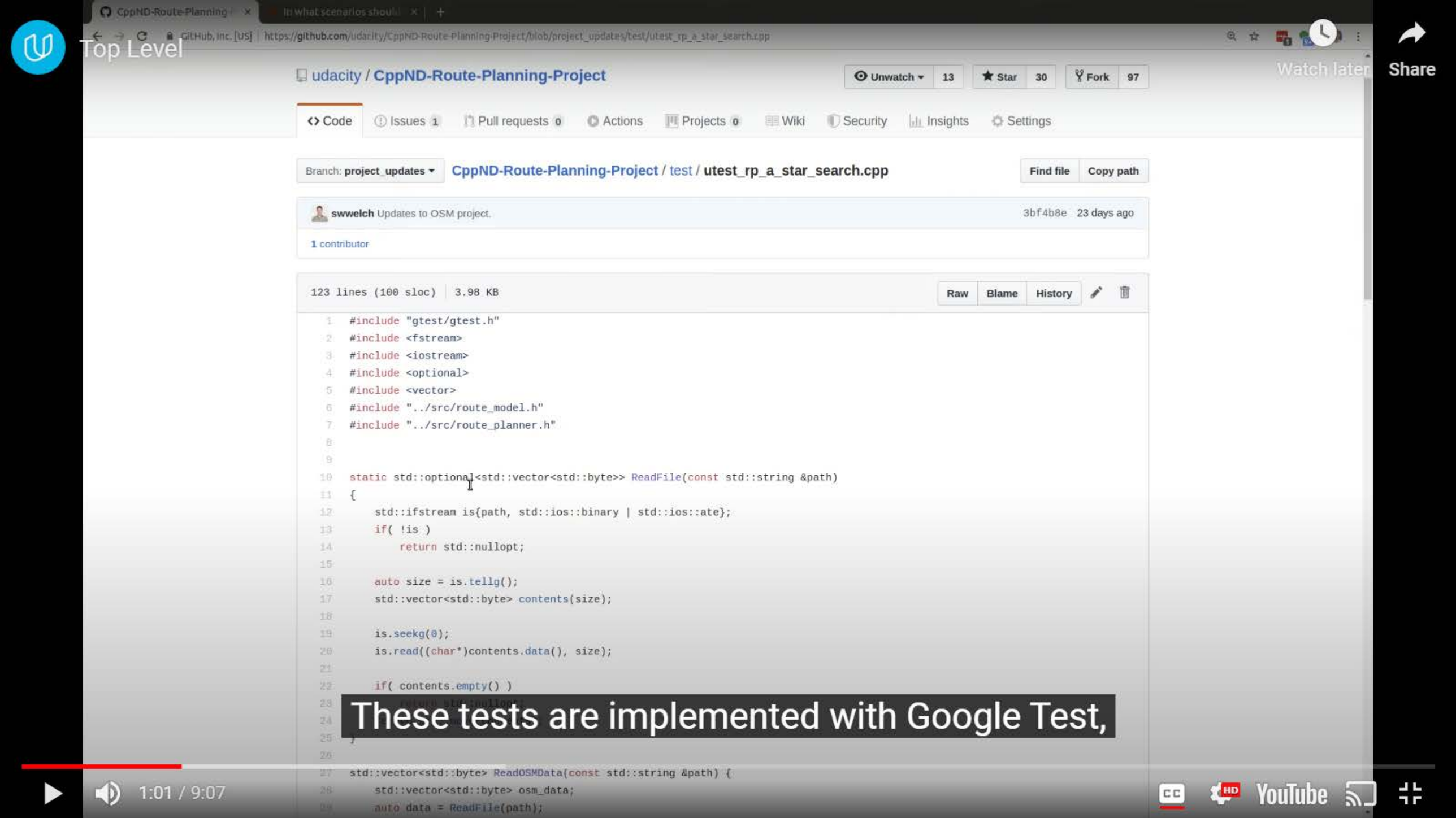


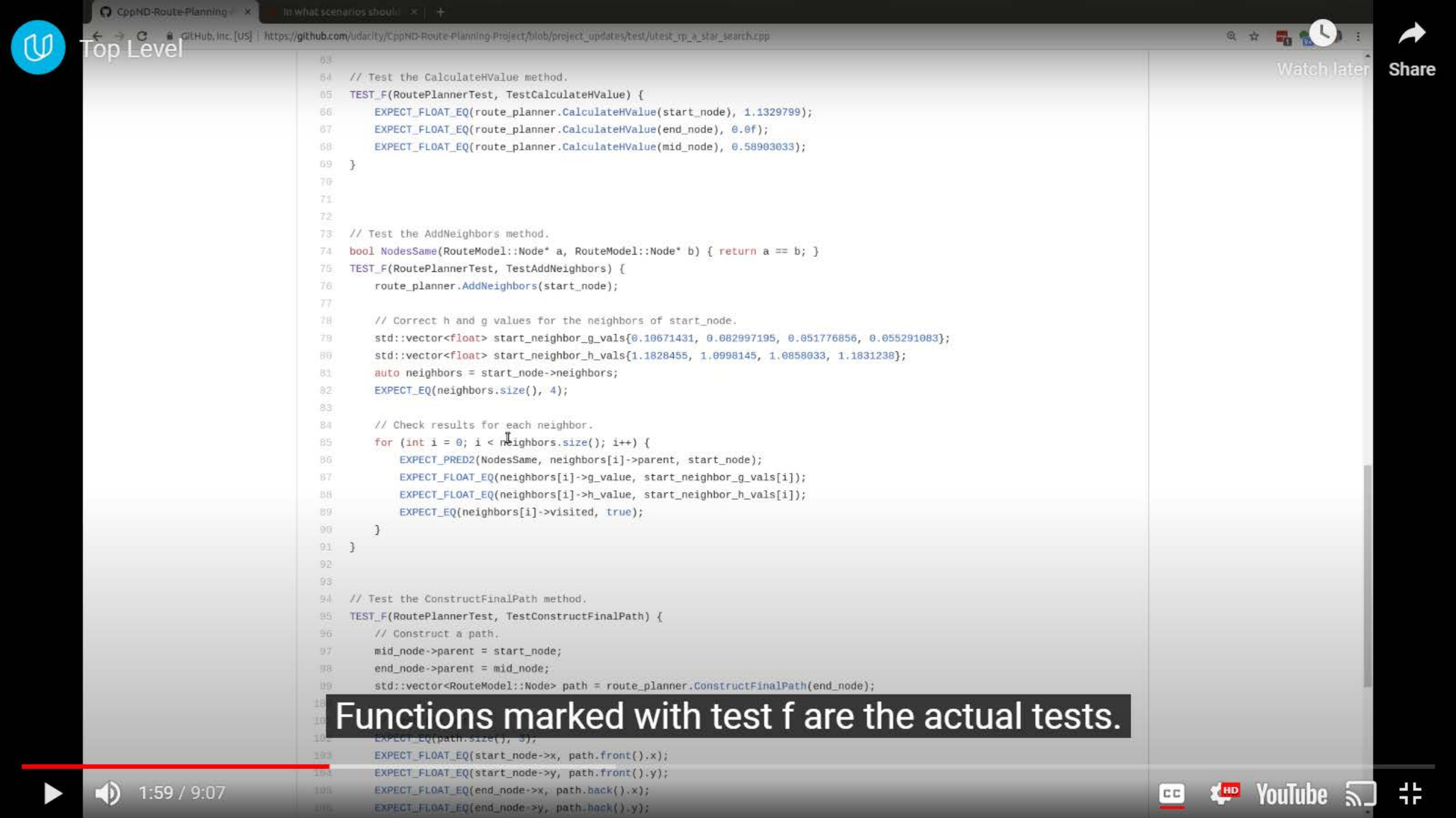
structured using object oriented techniques. Good luck.

Project Code Overview

Now that you are able to build, run, and test the project starter code, you are ready to have a closer look at the code and its structure.

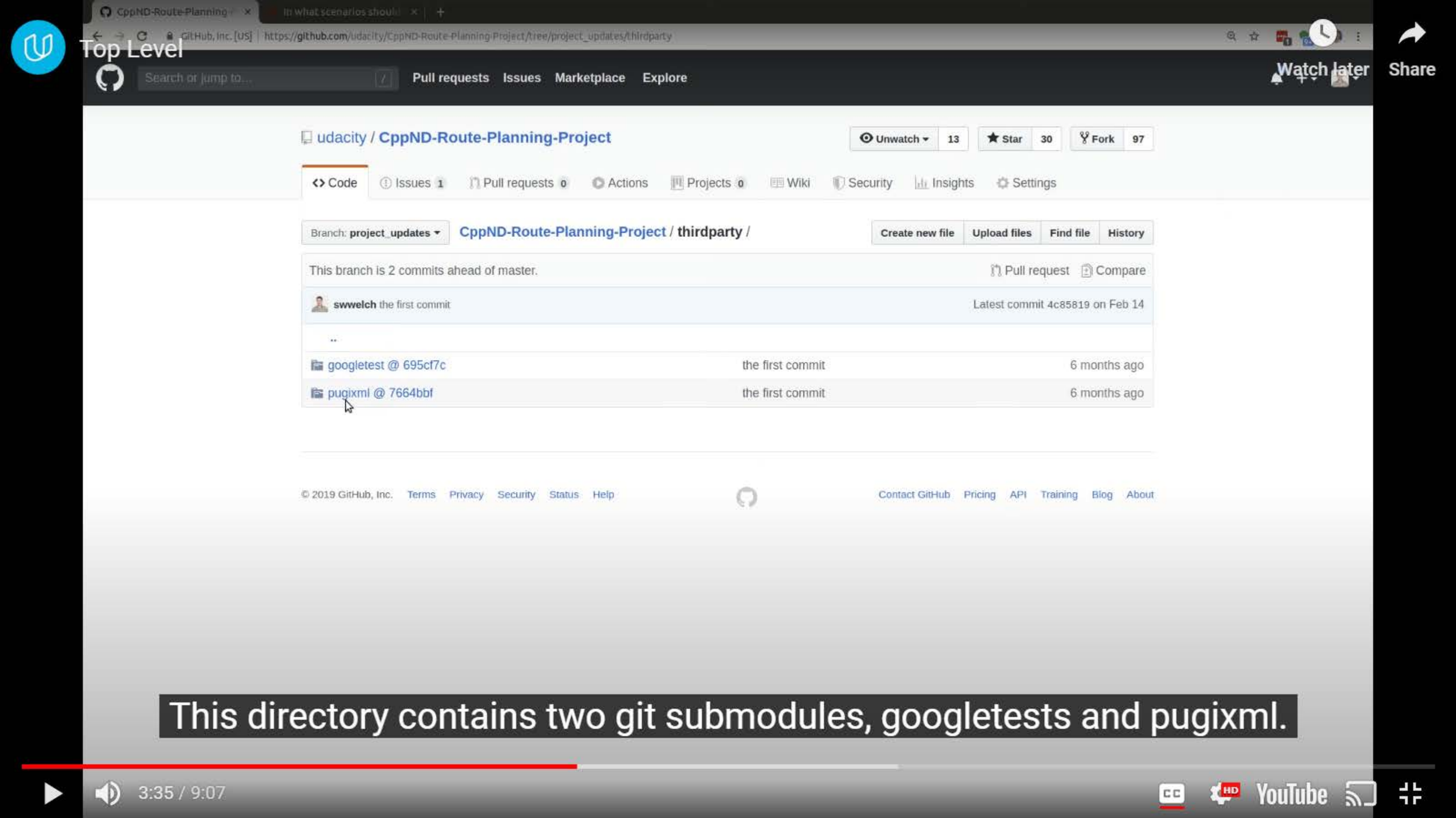
We've modified the IO2D example code slightly to help you get started with the project. In particular, we included some classes that will help you extend the existing code, and we added a few short plotting functions.

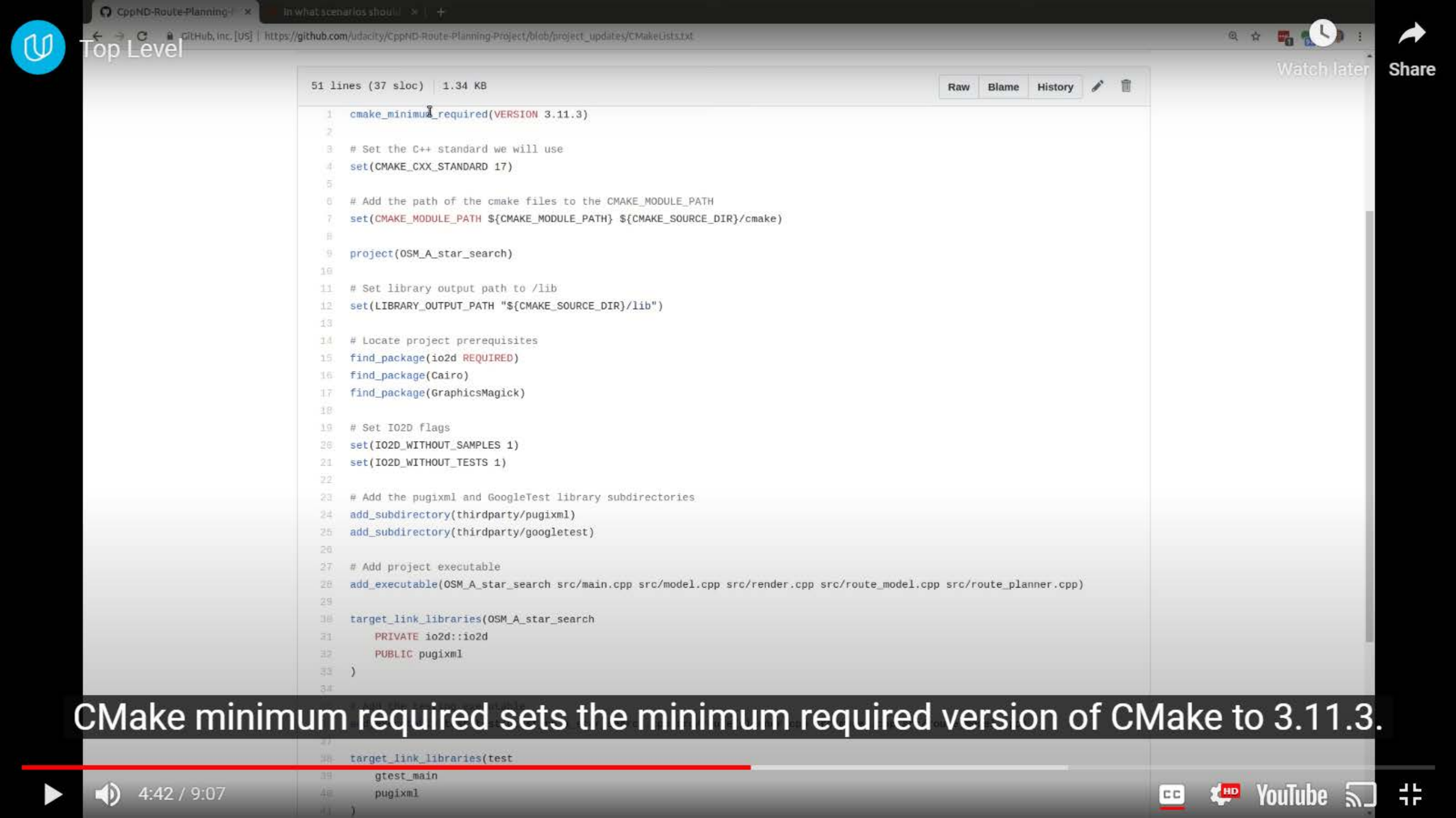

Top Level
udacity / CppND-Route-Planning-Project
Unwatch 13
Star 30
Fork 97
Code
Issues 1
Pull requests 0
Actions
Projects 0
Wiki
Security
Insights
Settings
Branch: project_updates
CppND-Route-Planning-Project / test / utest_rp_a_star_search.cpp
Find file
Copy path
swwelch Updates to OSM project.
3bf4b8e 23 days ago
1 contributor
123 lines (100 sloc) 3.98 KB
Raw
Blame
History
#include "gtest/gtest.h"
#include <fstream>
#include <iostream>
#include <optional>
#include <vector>
#include "../src/route_model.h"
#include "../src/route_planner.h"
static std::optional<std::vector<std::byte>> ReadFile(const std::string &path)
{
std::ifstream is{path, std::ios::binary | std::ios::ate};
if(!is)
return std::nullopt;
auto size = is.tellg();
std::vector<std::byte> contents(size);
is.seekg(0);
is.read((char*)contents.data(), size);
if(contents.empty())
std::vector<std::byte> ReadOSMData(const std::string &path) {
std::vector<std::byte> osm_data;
auto data = ReadFile(path);
These tests are implemented with Google Test,
1:01 / 9:07
Watch later
Share


Top Level
CppND-Route-Planning
GitHub, Inc. [US] https://github.com/udacity/CppND-Route-Planning-Project/blob/project_updates/test/utest_rp_a_star_search.cpp
Watch later
Share
// Test the CalculateHValue method.
TEST_F(RoutePlannerTest, TestCalculateHValue) {
EXPECT_FLOAT_EQ(route_planner.CalculateHValue(start_node), 1.1329799);
EXPECT_FLOAT_EQ(route_planner.CalculateHValue(end_node), 0.0f);
EXPECT_FLOAT_EQ(route_planner.CalculateHValue(mid_node), 0.58903033);
}
// Test the AddNeighbors method.
bool NodesSame(RouteModel::Node* a, RouteModel::Node* b) { return a == b; }
TEST_F(RoutePlannerTest, TestAddNeighbors) {
route_planner.AddNeighbors(start_node);
// Correct h and g values for the neighbors of start_node.
std::vector<float> start_neighbor_g_vals{0.10671431, 0.082997195, 0.051776856, 0.055291083};
std::vector<float> start_neighbor_h_vals{1.1828455, 1.0998145, 1.0858033, 1.1831238};
auto neighbors = start_node->neighbors;
EXPECT_EQ(neighbors.size(), 4);
// Check results for each neighbor.
for (int i = 0; i < neighbors.size(); i++) {
EXPECT_PRED2(NodesSame, neighbors[i]->parent, start_node);
EXPECT_FLOAT_EQ(neighbors[i]->g_value, start_neighbor_g_vals[i]);
EXPECT_FLOAT_EQ(neighbors[i]->h_value, start_neighbor_h_vals[i]);
EXPECT_EQ(neighbors[i]->visited, true);
}
}
// Test the ConstructFinalPath method.
TEST_F(RoutePlannerTest, TestConstructFinalPath) {
// Construct a path.
mid_node->parent = start_node;
end_node->parent = mid_node;
std::vector<RouteModel::Node> path = route_planner.ConstructFinalPath(end_node);
Functions marked with test f are the actual tests.
EXPECT_FLOAT_EQ(start_node->x, path.front().x);
EXPECT_FLOAT_EQ(start_node->y, path.front().y);
EXPECT_FLOAT_EQ(end_node->x, path.back().x);
EXPECT_FLOAT_EQ(end_node->y, path.back().y);
1:59 / 9:07
YouTube

```
TEST_F(RoutePlannerTest, TestCalculateHValue) {
    EXPECT_FLOAT_EQ(route_planner.CalculateHValue(start_node), 1.1329799);
    EXPECT_FLOAT_EQ(route_planner.CalculateHValue(end_node), 0.0f);
    EXPECT_FLOAT_EQ(route_planner.CalculateHValue(mid_node), 0.58903033);
}



// Test the AddNeighbors method.
bool NodesSame(RouteModel::Node* a, RouteModel::Node* b) { return a == b; }
TEST_F(RoutePlannerTest, TestAddNeighbors) {
    route_planner.AddNeighbors(start_node);

    // Correct h and g values for the neighbors of start_node.
    std::vector<float> start_neighbor_g_vals{0.10671431, 0.082997195, 0.051776856, 0.055291083};
    std::vector<float> start_neighbor_h_vals{1.1828455, 1.0998145, 1.0858033, 1.1831238};
    auto neighbors = start_node->neighbors;
    EXPECT_EQ(neighbors.size(), 4);

    // Check results for each neighbor.
    for (int i = 0; i < neighbors.size(); i++) {
        EXPECT_PRED2(NodesSame, neighbors[i]->parent, start_node);
        EXPECT_FLOAT_EQ(neighbors[i]->g_value, start_neighbor_g_vals[i]);
        EXPECT_FLOAT_EQ(neighbors[i]->h_value, start_neighbor_h_vals[i]);
        EXPECT_EQ(neighbors[i]->visited, true);
    }
}


// Test the ConstructFinalPath method.
TEST_F(RoutePlannerTest, TestConstructFinalPath) {
    // Construct a path.
    mid_node->parent = start_node;
    end_node->parent = mid_node;
    std::vector<RouteModel::Node> path = route_planner.ConstructFinalPath(end_node);

    // Test the path.
    EXPECT_EQ(path.size(), 3);
    EXPECT_FLOAT_EQ(start_node->x, path.front().x);
    EXPECT_FLOAT_EQ(start_node->y, path.front().y);
    EXPECT_FLOAT_EQ(end_node->x, path.back().x);
    EXPECT_FLOAT_EQ(end_node->y, path.back().y);
}
```

These tests all use the route planner test fixture which is defined above,


Top Level
Watch later
Share
udacity / CppND-Route-Planning-Project
Unwatch 13
Star 30
Fork 97
Code
Issues 1
Pull requests 0
Actions
Projects 0
Wiki
Security
Insights
Settings
Branch: project_updates
CppND-Route-Planning-Project / thirdparty /
Create new file
Upload files
Find file
History
This branch is 2 commits ahead of master.
Pull request
Compare
swwelch the first commit
Latest commit 4c85819 on Feb 14
..
googletest @ 695cf7c
the first commit
6 months ago
pugixml @ 7664bbf
the first commit
6 months ago
This directory contains two git submodules, googletests and pugixml.
3:35 / 9:07


Top Level
CppND-Route-Planning
In what scenarios shoul
GitHub, Inc. [US] | https://github.com/udacity/CppND-Route-Planning-Project/blob/project_updates/CMakeLists.txt
Watch later
Share
51 lines (37 sloc) | 1.34 KB
Raw
Blame
History
cmake_minimum_required(VERSION 3.11.3)
# Set the C++ standard we will use
set(CMAKE_CXX_STANDARD 17)
# Add the path of the cmake files to the CMAKE_MODULE_PATH
set(CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH} ${CMAKE_SOURCE_DIR}/cmake)
project(OSM_A_star_search)
# Set library output path to /lib
set(LIBRARY_OUTPUT_PATH "${CMAKE_SOURCE_DIR}/lib")
# Locate project prerequisites
find_package(io2d REQUIRED)
find_package(Cairo)
find_package(GraphicsMagick)
# Set IO2D flags
set(IO2D_WITHOUT_SAMPLES 1)
set(IO2D_WITHOUT_TESTS 1)
# Add the pugixml and GoogleTest library subdirectories
add_subdirectory(thirdparty/pugixml)
add_subdirectory(thirdparty/googletest)
# Add project executable
add_executable(OSM_A_star_search src/main.cpp src/model.cpp src/render.cpp src/route_model.cpp src/route_planner.cpp)
target_link_libraries(OSM_A_star_search
PRIVATE io2d::io2d
PUBLIC pugixml
)
target_link_libraries(test
gtest_main
pugixml
)
CMake minimum required sets the minimum required version of CMake to 3.11.3.
4:42 / 9:07
YouTube

The starter code for the project can be found here. This code is already loaded into a workspace, and you are not required to download or run the code locally to do the project. However, it may be a good idea to open the repository in a different tab or download the code so you can follow along as you watch the video above or read through the descriptions below. In the repo, you should see the following four directories:

- `cmake`

  This directory contains some `.cmake` files that are needed for the project to find necessary libraries. You will not need to work with this directory for this project.

- `src`

  The source code for the project is contained here, and this is where you will be doing all of the project work. See the next classroom concepts more information about the contents of this directory.

- `test`

  This directory contains unit tests for various exercises, implemented using the Google Test framework. As you are developing your code, *it may be helpful to look at the relevant tests in this directory to see the expected results and corresponding code*. The code written here can be used to understand how different classes and objects in your work. If your code fails a test, the console message will indicate which is the failing test.

- `thirdparty`

  This directory contains third party libraries that have been included with this project. You will not need to work directly with this code.

CppND-Route-Planning

GitHub, Inc. [US] | https://github.com/udacity/CppND-Route-Planning-Project/blob/project_updates/src/main.cpp

swwelch Updates to OSM project. 3bf4b8e 23 days ago

1 contributor

80 lines (65 sloc) 2.41 KB

Raw Blame History

```cpp
#include <optional>
#include <fstream>
#include <iostream>
#include <vector>
#include <string>
#include <io2d.h>
#include "route_model.h"
#include "render.h"
#include "route_planner.h"

using namespace std::experimental;

static std::optional<std::vector<std::byte>> ReadFile(const std::string &path)
{
    std::ifstream is{path, std::ios::binary | std::ios::ate};
    if( !is )
        return std::nullopt;

    auto size = is.tellg();
    std::vector<std::byte> contents(size);

    is.seekg(0);
    is.read((char*)contents.data(), size);

    if( contents.empty() )
        return std::nullopt;
    return std::move(contents);
}

int main(int argc, const char **argv)
{
    std::string osm_data_file = "";
    if( argc > 1 ) {
        for( int i = 1; i < argc; ++i )
            if( std::string_view{argv[i]} == "-f" && ++i < argc )
                osm_data_file = argv[i];
    }
    else {
```

An ate stands for at the end,

0:33 / 3:13

Watch later Share

n/Udacity/CppND-Route-Planning-Project/blob/project_updates/src/main.cpp

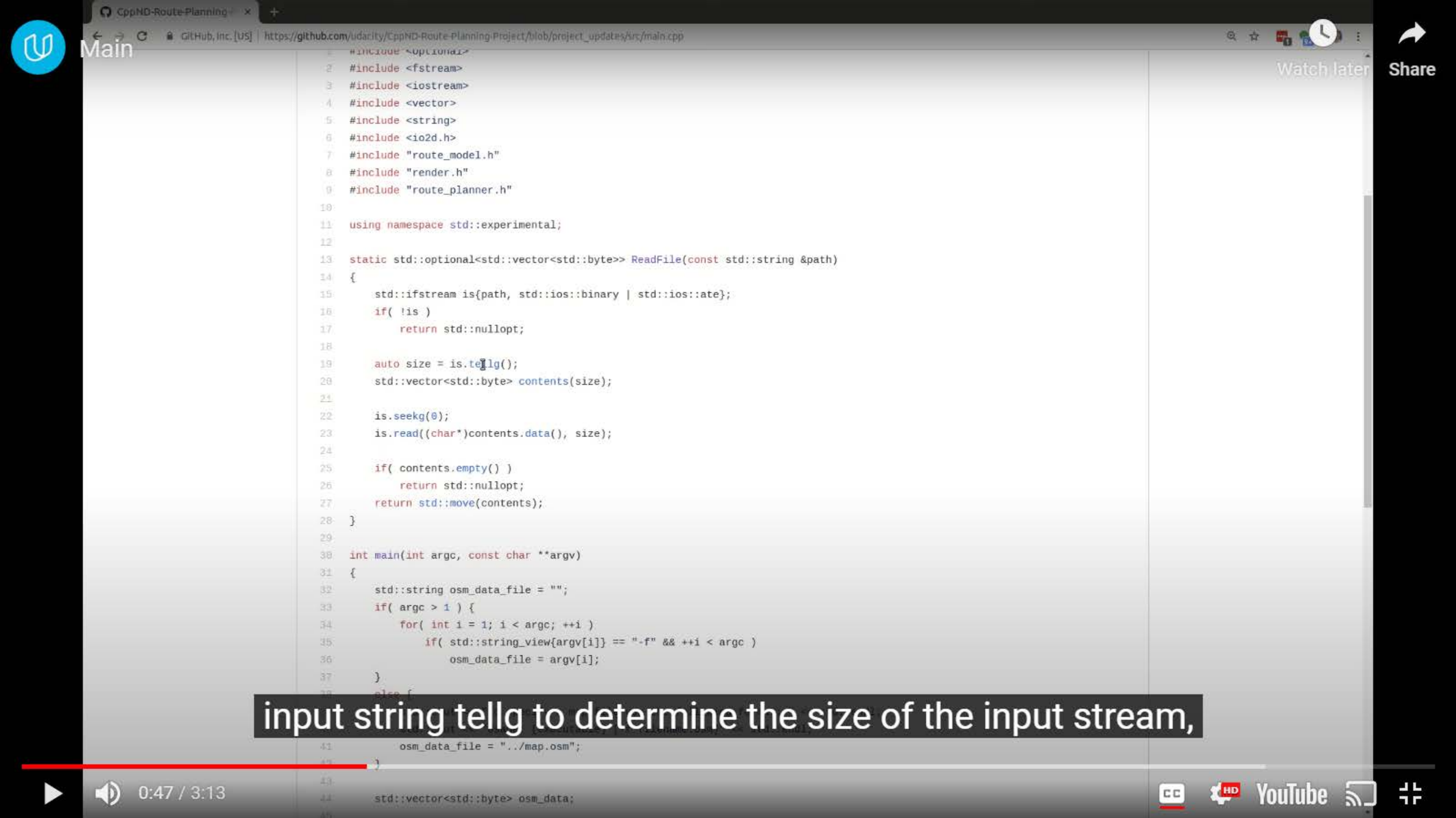
rwelch Updates to OSM project. 3bf4b8e 23 days ago

1 contributor

80 lines (65 sloc) 2.41 KB Raw Blame History

```cpp
#include <optional>
#include <fstream>
#include <iostream>
#include <vector>
#include <string>
#include <io2d.h>
#include "route_model.h"
#include "render.h"
#include "route_planner.h"

using namespace std::experimental;

static std::optional<std::vector<std::byte>> ReadFile(const std::string &path)
{
    std::ifstream is{path, std::ios::binary | std::ios::ate};
    if( !is )
        return std::nullopt;

    auto size = is.tellg();
    std::vector<std::byte> contents(size);

    is.seekg(0);
    is.read((char*)contents.data(), size);

    if( contents.empty() )
        return std::nullopt;
    return std::move(contents);
}

int main(int argc, const char **argv)
{
    std::string osm_data_file = "";
    for( int i = 1; i < argc; ++i )
        if( std::string_view{argv[i]} == "-f" && ++i < argc )
            osm_data_file = argv[i];
    }
    else {
```

which means that it will immediately seek to the end of the InputStream.


Main
CppND-Route-Planning
GitHub, Inc. [US] | https://github.com/udacity/CppND-Route-Planning-Project/blob/project_updates/src/main.cpp
Watch later
Share
#include <optional>
#include <fstream>
#include <iostream>
#include <vector>
#include <string>
#include <io2d.h>
#include "route_model.h"
#include "render.h"
#include "route_planner.h"
using namespace std::experimental;
static std::optional<std::vector<std::byte>> ReadFile(const std::string &path)
{
std::ifstream is{path, std::ios::binary | std::ios::ate};
if(!is)
return std::nullopt;
auto size = is.tellg();
std::vector<std::byte> contents(size);
is.seekg(0);
is.read((char*)contents.data(), size);
if(contents.empty())
return std::nullopt;
return std::move(contents);
}
int main(int argc, const char **argv)
{
std::string osm_data_file = "";
if(argc > 1) {
for(int i = 1; i < argc; ++i)
if(std::string_view{argv[i]} == "-f" && ++i < argc)
osm_data_file = argv[i];
}
else {
osm_data_file = "../map.osm";
}
std::vector<std::byte> osm_data;
input string tellg to determine the size of the input stream,
0:47 / 3:13
YouTube

The `src` Directory

The `src` directory contains the following files:

- `main.cpp`
- `model.h` and `model.cpp`
- `render.h` and `render.cpp`
- `route_model.h` and `route_model.cpp
- `route_planner.h` and `route_planner.cpp

You will be writing all of your code in `main.cpp` and `route_planner.cpp`, so we will cover those files first.

The `main.cpp` controls the flow of the program, accomplishing four primary tasks:

- The OSM data is read into the program.
- A `RouteModel` object is created to store the OSM data in usable data structures.
- A `RoutePlanner` object is created using the `RouteModel`. This planner will eventually carry out the A* search on the model data and store the search results in the `RouteModel`.
- The `RouteModel` data is rendered using the IO2D library.

Your first task is to complete **TODO** in `main.cpp`.

These files define the `RoutePlanner` class and methods for the A* search . Your task is to implement the A* search by completing all of the **TODOs** in `route_planner.cpp`.

The src Directory Additional Files

You will not need to write code in the following files:

- `model.h` and `model.cpp`
- `render.h` and `render.cpp`
- `route_model.h` and `route_model.cpp`

The videos below are included to give you a better understanding of the code in those files. We will not cover `model.cpp` or `render.cpp`, as these are beyond the scope of the first course in this Nanodegree. However, we will walk through the header files for these two files, which will provide an overview of the corresponding classes and methods.

The `model.h` and `model.cpp` files come from the IO2D example code. We provide an overview of only the `model.h` file, as the method implementations in `model.cpp` file are beyond the scope of the course.

These files are used to define the data structures and methods that read in and store OSM data. OSM data is stored in a `Model` class which contains nested structs for Nodes, Ways, Roads, and other OSM objects. Have a look at the video above for an overview of the code in the header file.

These files contain classes which are used to extend the `Model` and `Node` data structures from `model.h` and `model.cpp` using class inheritance. Remember that inheritance in this case will allow you to use all of the *public* methods and attributes of the `Model` class and `Node` struct in the derived `RouteModel` and `RouteModel::Node` classes.

The reason for extending the existing `Model` class and `Node` struct is to include additional methods and variables which are useful for A* search. In particular, the new `RouteModel::Node` class now allows nodes to store the following:

- the h-value,
- the g-value,
- a "visited" flag,
- a vector of pointers to neighboring nodes.

In addition, there are now methods for

- finding neighboring Node objects of a Node,
- getting the distance to other nodes,
- finding the closest node to a given (x, y) coordinate pair.

The `render.h` and `render.cpp` files come from the IO2D example code. These take map data that is stored in a `Model` object and render that data as a map. We have modified these files slightly to include three extra methods which render the start point, end point, and path from the A* search. You won't need to work with these files directly, but you can watch the video above for a brief overview of the header file code.

Steps to Complete the Project

Completing the Project

To complete this project you will need to:

1. Complete the **TODO** in `main.cpp`
2. Complete the **TODO**s in `route_planner.cpp`
3. Check that all tests are passed when you run the `test` executable in your `build` directory.
4. Check that you have satisfied all of the criteria in the project rubric. If you feel like all the criteria from the rubric are met, you are ready to submit!

Submitting the Project

You can submit your project using the "SUBMIT PROJECT" button in the terminal in the next classroom concept.

This project can only be submitted through the workspace in the next concept. This is done to ensure that you are able to test your code in an environment similar to what reviewers will use.

If you have developed your project locally, you can upload the project directory to the workspace for submission. Similarly, if you have your code in Github, you can clone your project into the workspace for submission.

PROJECT SPECIFICATION

Build an OpenStreetMap Route Planner

Compiling and Testing

CRITERIA	MEETS SPECIFICATIONS
The submission must compile.	The project code must compile without errors using `cmake` and `make`.
The submission must pass the final set of unit tests.	Code must pass tests that are built with the `./test` executable from the `build` directory of the project. See the project submission instructions for more details on how to run the tests.

User Input

CRITERIA	MEETS SPECIFICATIONS
The user should be able to provide inputs to the search.	A user running the project should be able to input values between 0 and 100 for the start x, start y, end x, and end y coordinates of the search, and the project should find a path between the points.
The user inputs should correspond with areas on the map.	The coordinate (0, 0) should roughly correspond with the lower left corner of the map, and (100, 100) with the upper right. Note that for some inputs, the nodes might be slightly off the edges of the map, and this is fine.

Code Efficiency

CRITERIA	MEETS SPECIFICATIONS
The methods in the code should avoid unnecessary calculations.	Your code does not need to sacrifice comprehension, stability, or robustness for speed. However, you should maintain good and efficient coding practices when writing your functions. Here are some things to avoid. This is not a complete list, but there are a few examples of inefficiencies. • Running the exact same calculation repeatedly when you can run it once, store the value and then reuse the value later. • Loops that run too many times. • Creating unnecessarily complex data structures when simpler structures work equivalently. • Unnecessary control flow checks.