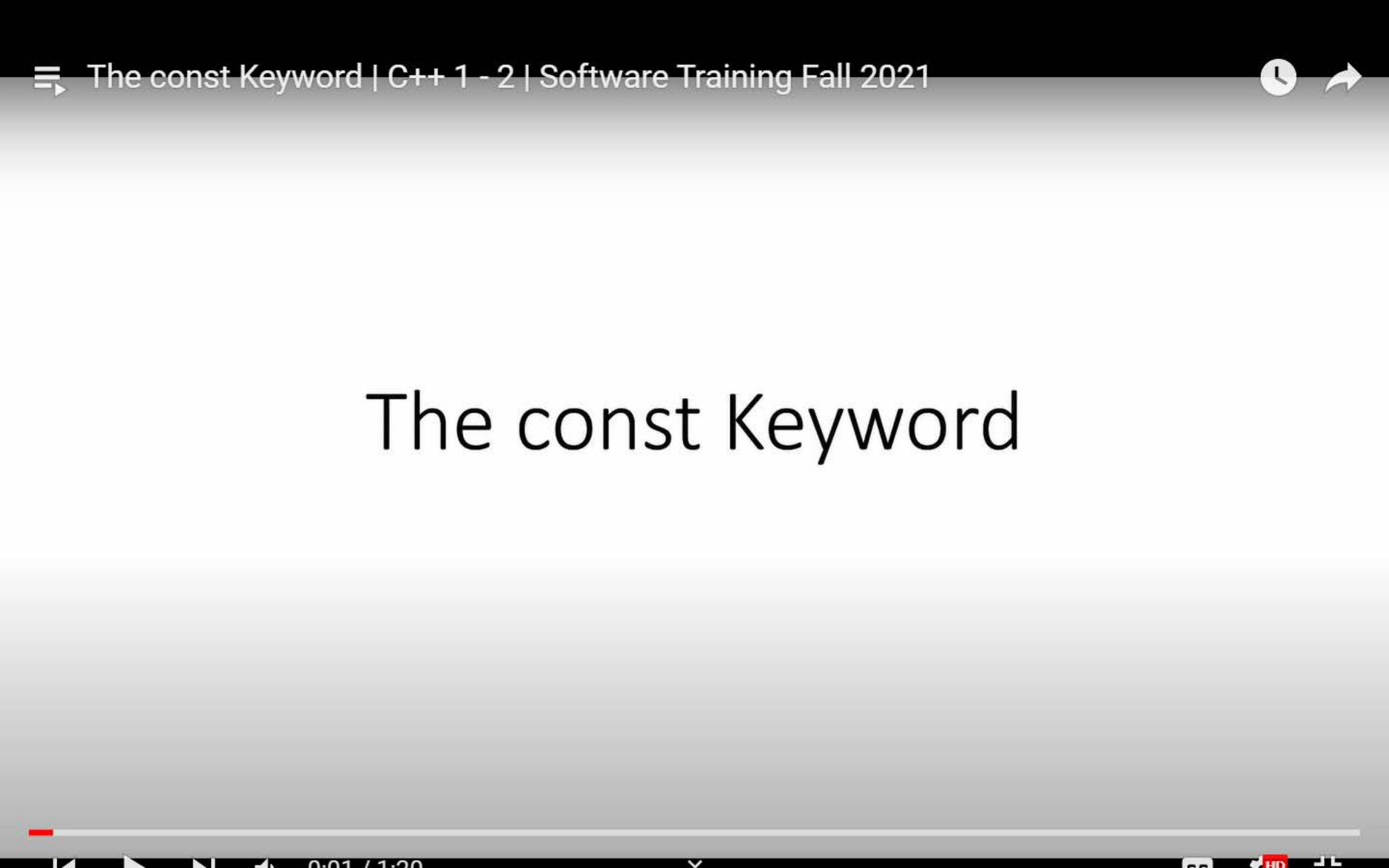
C++ Introduction | C++ 1 - 1 | Software Training Fall 2021

godbolt.org

COMPILER EXPLORER

0:57 / 4:28


The const Keyword | C++ 1 - 2 | Software Training Fall 2021
The const Keyword

Why use constants?

- Let our compiler optimize our code better
- Communicates to other developers that a value can't change

The const Keyword

```
int my_variable = 10;
const int my_constant = 42;

my_variable += 5;  // works fine

my_constant += 5;  // won't compile
```

constexpr

- "Constant Expression"
- Marks variables and functions that can be evaluated at compile time

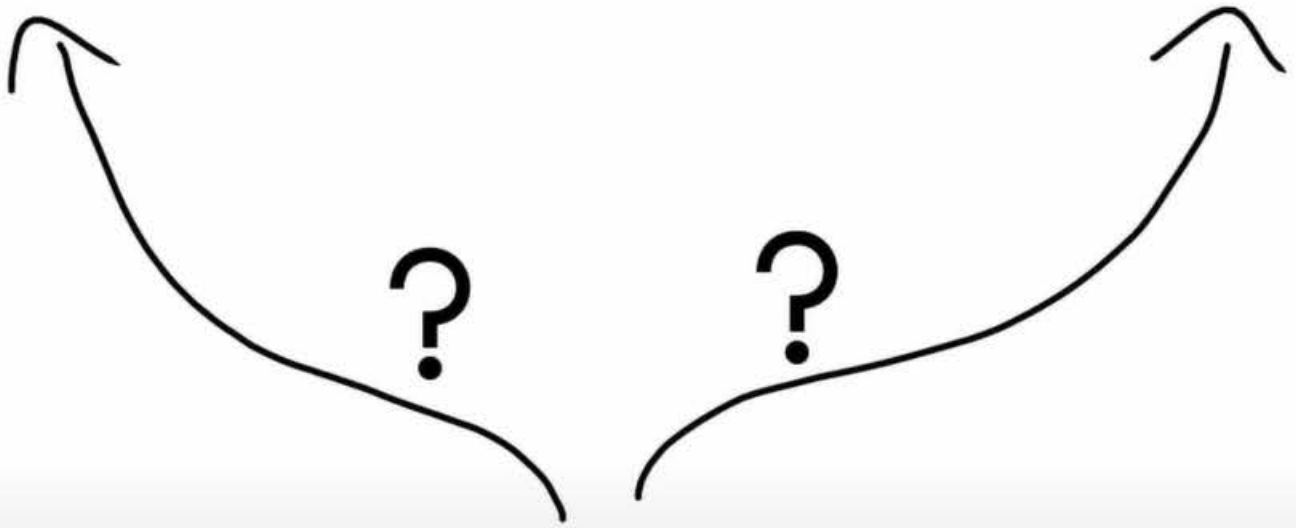

Namespaces | C++ 1 - 3 | Software Training Fall 2021
Namespaces
0:00 / 2:21
HD

Namespaces

- Group named symbols together
- Important for avoiding name collisions

```
void SaveLogs(string filepath)
{
   …
}
```

```
void SaveLogs(string filepath)
{
   …
}
```

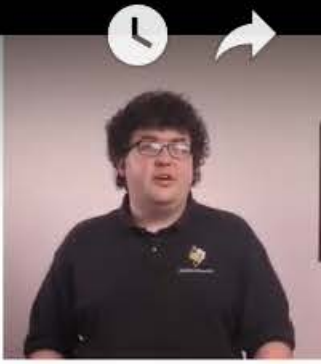



```
SaveLogs("/my/logs/file.txt");
```

```
namespace LibraryA {
void SaveLogs(string filepath)
{
   …
}
}
```

```
namespace LibraryB {
void SaveLogs(string filepath)
{
   …
}
}
```

```
LibraryA::SaveLogs("/my/logs/file.txt");
```



Common Namespaces (for us)

- std
  - The C++ standard library namespace
- rclcpp
  - The ROS C++ client library namespace

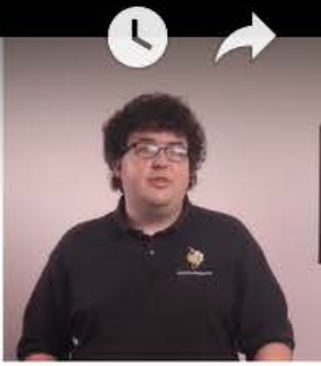
Nested Namespaces

```
cv::aruco::drawDetectedMarkers(...);
```

"Using" Namespaces

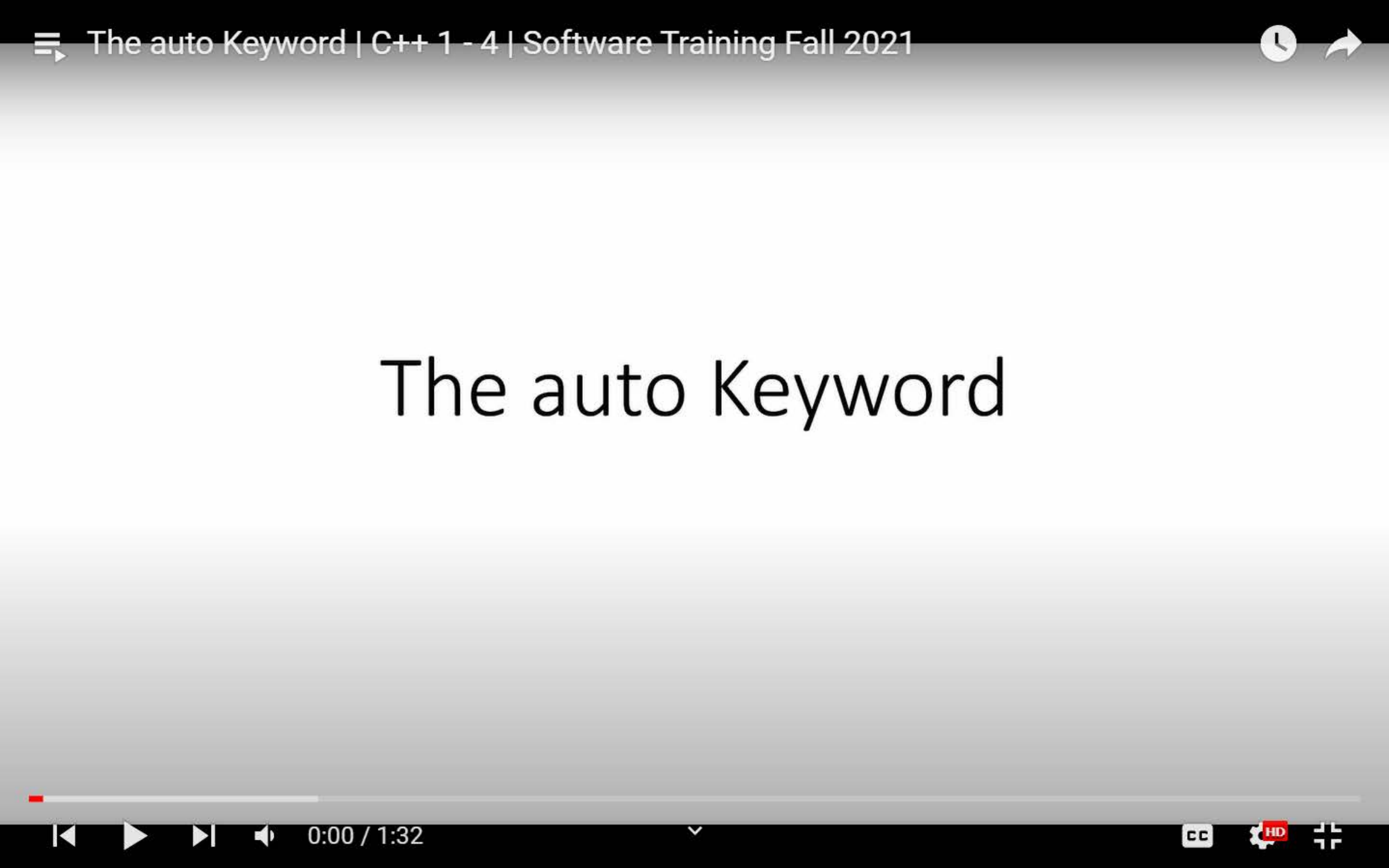
```
using namespace std;

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

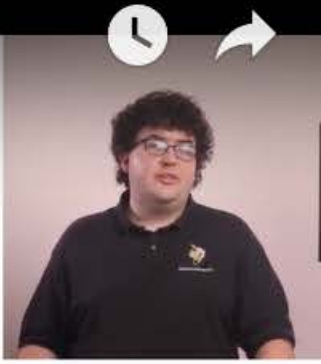
No "std::"



Aliasing Namespaces

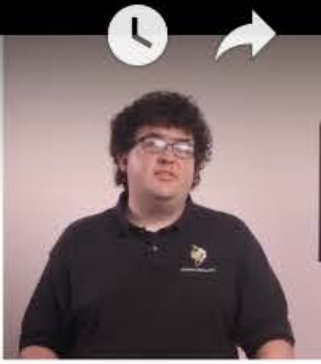
```
namespace rju = robojackets::utils;
```


The auto Keyword | C++ 1 - 4 | Software Training Fall 2021
The auto Keyword
0:00 / 1:32
HD



Variables Have Types

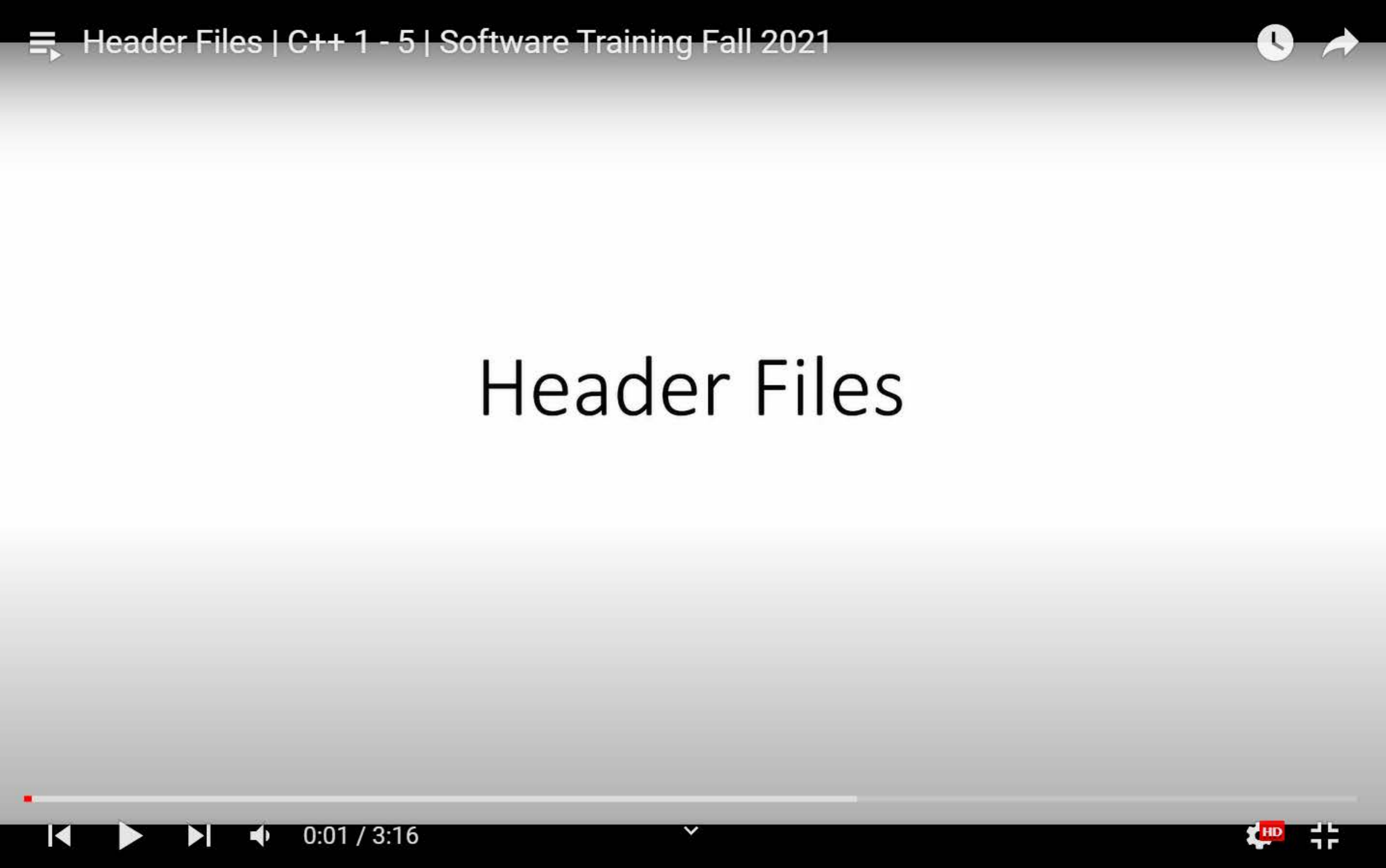
```
int a = 10;
a = "Hello!\n";  // won't compile. Wrong type!
```



auto

- Used instead of type names to tell compiler to deduce the type for us.

```
auto i = 10;  // "int" deduced from initializer

i = "Hello!";  // still doesn't compile
```


Header Files | C++ 1 - 5 | Software Training Fall 2021
Header Files
0:01 / 3:16
HD

Two types of files in C++

Header Files

- *.h or *.hpp
- Declare symbols
  - Classes / functions / global variables

```
void Func();
```

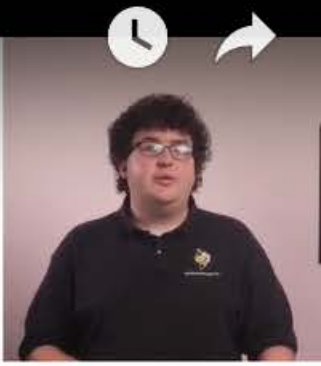
Implementation Files

- *.cpp
- Define symbols
  - Function defniitions
  - Global variable initializers

```
void Func() {
  // Func implementation
}
```

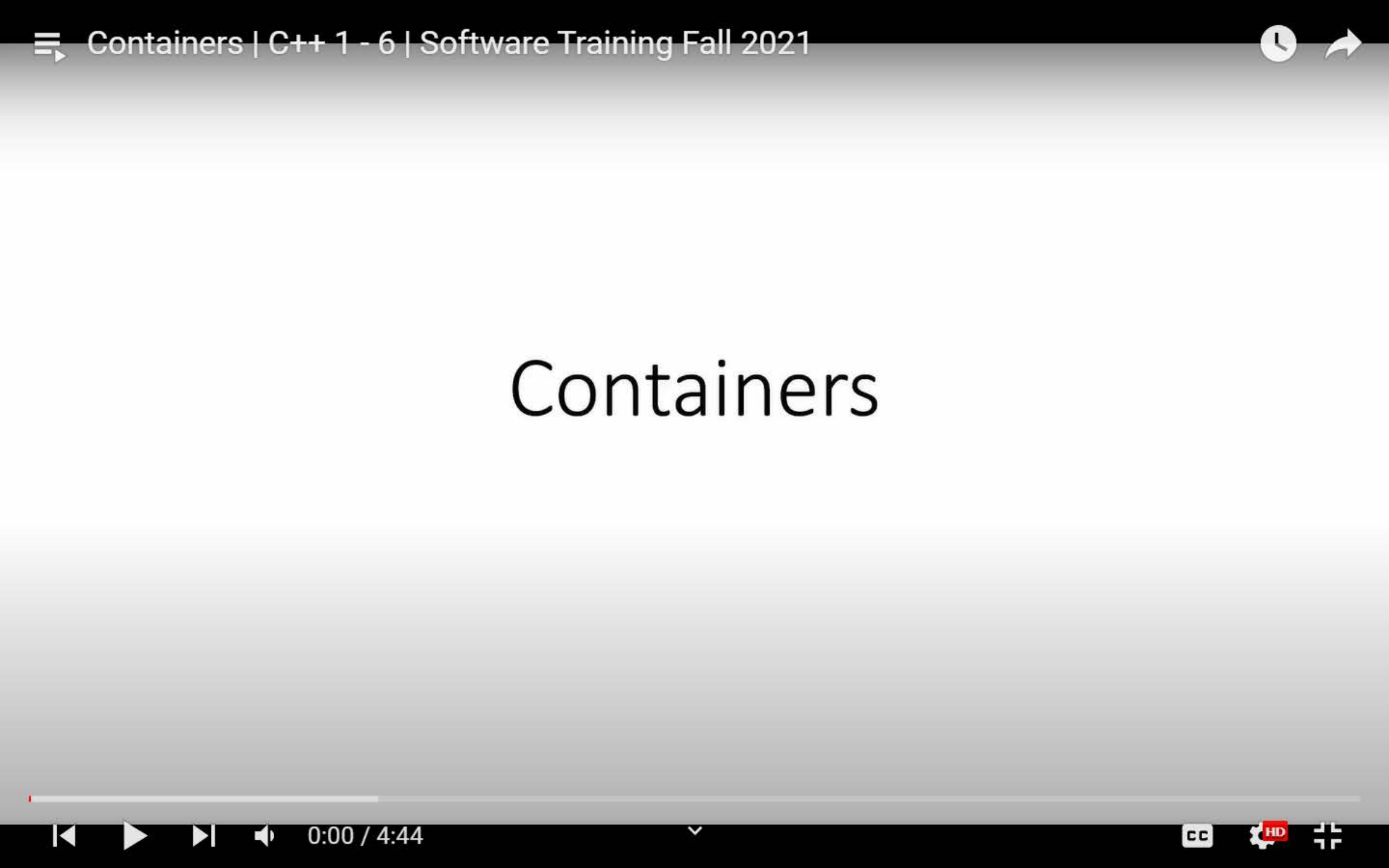
Including Headers

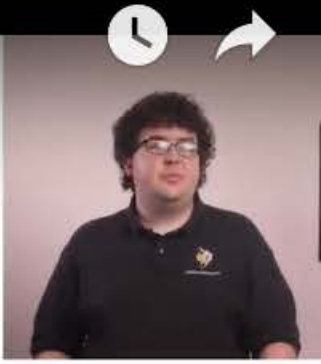
```
#include <rclcpp/rclcpp.hpp>

#include "MyHeader.hpp"
```



Header Guards

```
#ifndef MY_HEADER_HPP
#define MY_HEADER_HPP

// All code here

#endif
```


Containers | C++ 1 - 6 | Software Training Fall 2021
Containers
0:00 / 4:44
HD



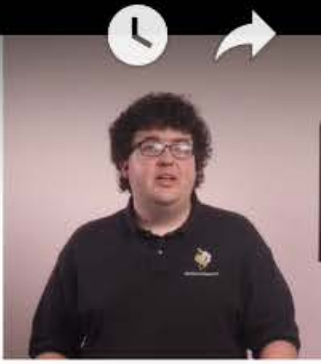
C-Style Arrays

```
int arr[3] = {9,10,11};

std::cout << arr[1];  // prints "10"
```

std::array

```
std::array<int, 3> arr = {9, 10, 11};
```

Template Arguments



std::array

```
std::array<int, 3> arr = {9, 10, 11};

std::cout << arr[1];      // prints "10"

std::cout << arr.at(1);   // prints "10"

std::cout << arr.at(4);   // throws exception

std::cout << arr.size(); // prints "3"
```

std::vector

```
std::vector<int> vec = {3,4,5};

std::cout << vec[0];              // prints "3"

std::cout << vec.at(1);           // prints "4"

std::cout << vec.size();          // prints "3"

std::cout << vec.push_back(6); // appends 6
```

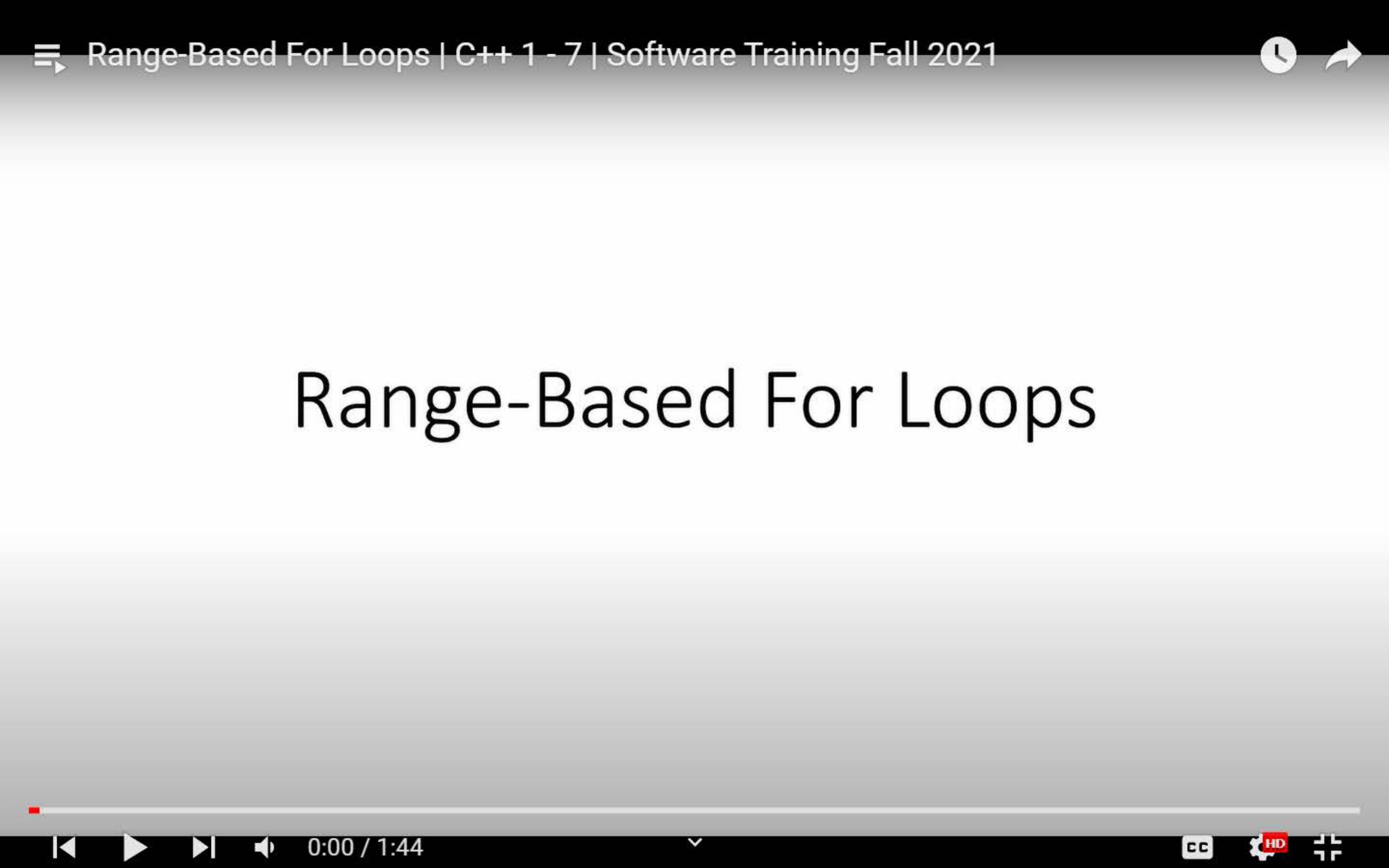
Iterators

- An iterator represents a specific position inside a specific container
- `*.begin()` gives us the iterator for the first element in a container
- `*.end()` gives us the iterator for one position past the last element

```
vec.erase(vec.begin() + 3);  // erases third element
```

Other Containers

std::forward_list	Linked list
std::list	Doubly linked list
std::set	Collection of unique, sorted values
std::unordered_set	Collection of unique, hased values
std::map	Key-value pairs, with sorted & unique keys
std::unordered_map	Key-value pairs, with hashed & unique keys
...	


Range-Based For Loops | C++ 1 - 7 | Software Training Fall 2021
Range-Based For Loops
0:00 / 1:44

Classic for loop

```
std::vector<doubles> v = {…};

for(int i = 0; i < v.size(); ++i)
{
   std::cout << v[i] << "\n";
}
```

Range-based for loop

```
std::vector<doubles> v = {…};

for(double elem : v)
{
  std::cout << elem << "\n";
}
```

Any object that provides `begin()` and `end()`

Range-based for loop

```
std::vector<doubles> v = {…};

for(const auto elem : v)
{
  std::cout << elem << "\n";
}
```


Strings | C++ 1 - 8 | Software Training Fall 2021
Strings
0:00 / 2:43
HD

Handling Text

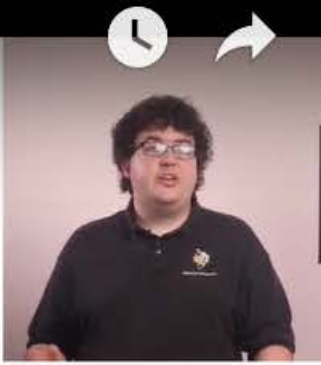
```
// Chars hold one character
char var = 'H';

// Raw containers aren't convenient for text
std::vector<char> v = {'H','e','l','l','o'};

// std::string is purpose-built for text
std::string s =  "Hello";
```

std::string

```
std::string s = "Hello";
s.empty();  // false
s.size();   // 5
s.at(0);    // 'H'
s[1];       // 'e'
s.erase(3); // erases third character
s.clear();  // erases all characters
```

std::string

```
std::string a = "apple";
std::string b = "banana";
a < b; // true
```



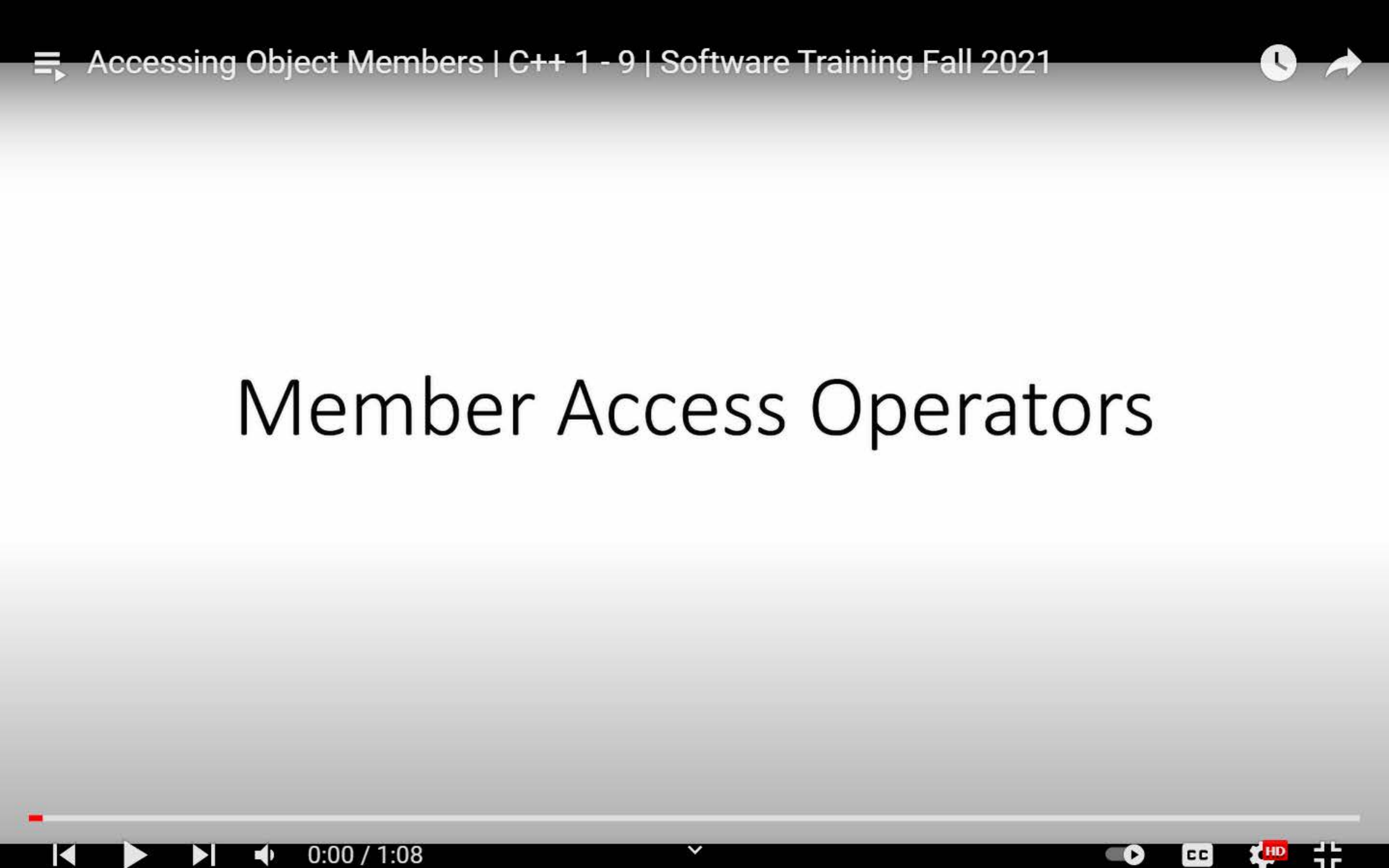
Converting to/from strings

```
std::to_string(1.23); // "1.23"
std::stoi("100"); // 100 as int
std::stod("2.5"); // 2.5 as double
```

std::string::npos

- Special constant in std::string
- Represents an invalid index
- Sometimes means "the end of the string"

```
// substring from the 5th character to the end
s.substr(5,std::string::npos);

// returns std::string::npos if not found
s.find("Hi");
```


Accessing Object Members | C++ 1 - 9 | Software Training Fall 2021
Member Access Operators
0:00 / 1:08
HD

Accessing Object Members

```
obj.member();  // dot syntax, for plain objects

ptr->member(); // arrow syntax, for pointers
```