

CS106L: Lecture 3

Initialization & References

Preston Seay & Emily Ai

Recap

auto



"Hey compiler, figure out this type."

- Use at your discretion
- Helpful when type is **annoying**

Recap

```
1 #include <iostream>
2 #include <string>
3 #include <map>
4 #include <unordered_map>
5 #include <vector>
6
7 int main()
8 {
9     std::map<std::string, std::vector<std::pair<int, std::unordered_map<char, double>>>>
10     complexType;
11     /// confusing iterator type (We'll find out what this is in the iterators lecture!)
12     std::map<std::string, std::vector<std::pair<int, std::unordered_map<char, double>>>>::iterator
13     it = complexType.begin();
14     // clear(er) iterator type!
15     auto it = complexType.begin();
16     return 0;
17 }
```

Recap

auto



"Hey compiler, figure out this type."

struct



"Group these variables together, in one type, please 🙏."

Plan

1. Initialization
2. References
3. L-values vs R-values
4. `const`
5. Compiling C++ programs

Activities and Honor Code!

106l.com Activities

- Work with friends, ask AI for clarification.
- Meant to be done in lecture.
- Don't share the attendance code.

Assignments

- Do assignments on your own.
- Never have AI submit answers or write your code.
- Feel free to ask AI questions to learn!

Go to 106l.com and fill out the first page of today's activity!



[https://www.106l.com/live/initialization-references/welcome?
key=gWHBA5tXa2qJuhcNSfpdLHth-LXjMg2vdsVFWbCXIJ4](https://www.106l.com/live/initialization-references/welcome?key=gWHBA5tXa2qJuhcNSfpdLHth-LXjMg2vdsVFWbCXIJ4)



Initialization

What it is...

"Provides **initial** values at the time of construction"



<https://en.cppreference.com/w/cpp/language/initialization.html> ➤

Initialization

What it is...

"Provides **initial** values at the time of construction"

How to...

1. Direct initialization
2. Uniform initialization
3. Structured Binding

(1) Direct Initialization

Code Console

CPP

Run

```
1 #include <iostream>
2
3 int main() {
4     int numOne = 12.0;
5     int numTwo(12.0);
6     std::cout << "numOne is: " << numOne << std::endl;
7     std::cout << "numTwo is: " << numTwo << std::endl;
8     return 0;
9 }
```

12.0 not an int...
does this work?

YES

(1) Direct Initialization

Code

Console

CPP

Run

```
1 #include <iostream>
2
3 void checkCool(float temperature) {
4     if (temperature > 100.0) {
5         std::cout << "Emergency cooling activated!" << std::endl;
6     } else {
7         std::cout << "Temperature normal. No emergency cooling required.";
8     }
9 }
10
11 int main() {
12     float temperatureReading(100.8);
13     int temperature = temperatureReading;
14     checkCool(temperature);
15     return 0;
16 }
```

Critical? Yes

(1) Direct Initialization

Code Console

CPP Run

```
1 #include <iostream>
2
3 void checkCool(float temperature) {
4     if (temperature > 100.0) {
5         std::cout << "Emergency cooling activated!" << std::endl;
6     } else {
7         std::cout << "Temperature normal. No emergency cooling required.";
8     }
9 }
10
11 int main() {
12     float temperatureReading(100.8);
13     int temperature = temperatureReading;
14     checkCool(temperature);
15     return 0;
16 }
```

C++ does not care
"You want 100.8 to be an integer? Okay 😊" - compiler
Narrowing Conversion

Initialization

What it is...

"Provides **initial** values at the time of construction"

How to...

1. Direct initialization
2. **Uniform initialization**
3. Structured Binding

(2) Uniform Initialization

Notice the curly brackets!

```
1 #include <iostream>
2
3 int main() {
4     int numOne = {12.0};
5     int numTwo{12.0};
6     std::cout << "numOne is: " << numOne << std::endl;
7     std::cout << "numTwo is: " << numTwo << std::endl;
8     return 0;
9 }
```

12.0 not an int...
does this work?

NO

(2) Uniform Initialization

Benefits:

1 . Safe

No narrowing conversion.

2 . Ubiquitous

Can use with vectors, maps, custom classes, etc.

(2) Uniform Initialization

```
Code Console CPP Run
1 #include <iostream>
2 #include <map>
3
4 int main() {
5     // Uniform initialization of a map.
6     std::map<std::string, int> ages{
7         {"Alice", 25},
8         {"Bob", 30},
9         {"Charlie", 35}
10    };
11    // Accessing map elements.
12    std::cout << "Alice's age: " << ages["Alice"] << std::endl;
13    std::cout << "Bob's age: " << ages.at("Bob") << std::endl;
14    return 0;
15 }
```

(2) Uniform Initialization

```
Code Console CPP Run
1 #include <iostream>
2 #include <vector>
3 int main() {
4     // Uniform initialization of a vector.
5     std::vector<int> numbers{1, 2, 3, 4, 5};
6     // Accessing vector elements.
7     for (int num : numbers) {
8         std::cout << num << " ";
9     }
10    std::cout << std::endl;
11    return 0;
12 }
```

Recall

```
1 StanfordID issueID() {  
2     StanfordID id;  
3     id.name = "THE Stanford Tree";  
4     id.sunet = "theTREE";  
5     id.idNumber = 0000002;  
6     return id;  
7 }
```



```
1 StanfordID issueID() {  
2     StanfordID id = {"THE Stanford Tree", "theTREE", 0000002};  
3     return id;  
4 }
```

Initialization

What it is...

"Provides **initial** values at the time of construction"

How to...

1. Direct initialization
2. Uniform initialization
3. **Structured Binding**

(3) Structured Initialization

Initializes multiple variables from fixed-size data structures.

Access multiple values returned by a function.

(3) Structured Binding

Code Console

CPP

Run

```
1 #include <iostream>
2 #include <tuple>
3 #include <string>
4
5 std::tuple<std::string, std::string, std::string> getClassInfo() {
6     std::string className = "CS106L";
7     std::string buildingName = "Thornton 110";
8     std::string language = "C++";
9     return {className, buildingName, language};
10 }
11
12 int main() {
13     auto [className, buildingName, language] = getClassInfo();
14     std::cout << "Come to " << buildingName << " and join us for " << className
15     << " to learn " << language << "!" << std::endl;
16     return 0;
17 }
```

What do we call this?
Uniform initialization

What do we call this?
Structured Binding

(3) Structured Binding



```
1 auto classInfo = getClassInfo();  
2 std::string className = std::get<0>(classInfo);  
3 std::string buildingName = std::get<1>(classInfo);  
4 std::string language = std::get<2>(classInfo);
```



```
1 auto [className, buildingName, language] = getClassInfo();
```

(3) Structured Binding

Initializes multiple variables from fixed-size data structures.

Access multiple values returned by a function.

Size must be known at compile time.

Go to 106l.com and fill out slides 2 and 3 of today's activity!

Enter a URL to generate a QR code.

Plan

1. Initialization
2. **References**
3. L-values vs R-values
4. `const`
5. Compiling C++ programs

References

What they are...

"An alias to an already-existing object or function."



<https://en.cppreference.com/w/cpp/language/reference.html> ↗

References

What they are...

How to...

"An alias to an already-existing object or function."

&

References Example

Code

Console

CPP

Run

```
1 #include <iostream>
2
3 int main() {
4     int miToMoon = 238855; // That's how far the moon is.
5     std::cout << "Moon is " << miToMoon << "mi away." << std::endl;
6
7     int& ISS = miToMoon;
8     ISS -= 254; // That's how high the ISS is.
9
10    std::cout << "ISS is " << ISS << "mi to moon." << std::endl;
11    std::cout << "Moon is " << miToMoon << "mi away." << std::endl;
12    return 0;
13 }
```

References Example

```
➔ 1 int miToMoon = 238855;  
   2 int& ISS = miToMoon;  
   3 ISS -= 254;
```

Memory

miToMoon

238855



References Example

```
1 int miToMoon = 238855;  
2 int& ISS = miToMoon;  
3 ISS -= 254;
```

Memory

miToMoon

238855

~~ISS~~

~~238855~~

References Example

Memory

```
1 int miToMoon = 238855;  
2 int& ISS = miToMoon;  
3 ISS -= 254;
```

ISS / miToMoon

238855



References Example

Memory

```
1 int miToMoon = 238855;  
2 int& ISS = miToMoon;  
→ 3 ISS -= 254;
```

→ ISS / miToMoon

238601

Pass by Reference

Note the ampersand!

```
1 #include <iostream>
2 #include <math.h>
3
4 void squareN(int& n) {
5     n = pow(n, 2);
6 }
7
8 int main() {
9     int num = 5;
10    squareN(num);
11    std::cout << num << std::endl;
12    return 0;
13 }
```

n is a reference to num.

So num gets updated to 25.

Recall: Pass by Value

```
Code Console CPP Run
1 #include <iostream>
2 #include <math.h>
3
4 void squareN(int n) {
5     n = pow(n, 2);
6 }
7
8 int main() {
9     int num = 5;
10    squareN(num);
11    std::cout << num << std::endl;
12    return 0;
13 }
```

n is a *copy* of num.

So num does not get updated.

Recall: Pass by Value

Code Console

CPP Run

```
1 #include <iostream>
2 #include <math.h>
3
4 void squareN(int n) {
5     n = pow(n, 2);
6 }
7
8 int main() {
9     int num = 5;
10    squareN(num);
11    std::cout << num << std::endl;
12    return 0;
13 }
```

Memory

num

5

n

~~5~~

25

Value

vs

Reference

```
1 void squareN(int n) {  
2   n = pow(n, 2);  
3 }
```

Copies the variable!
(Can be expensive.)

```
1 void squareN(int& n) {  
2   n = pow(n, 2);  
3 }
```

Uses the same variable
& memory!
(Can modify it!)

A Classic Reference Copy Bug

```
1 #include <iostream>
2 #include <math.h>
3 #include <vector>
4 void shift(std::vector<std::pair<int, int>> &nums) {
5     for (auto [num1, num2] : nums)
6         num1++;
7         num2++;
8     }
9 }
```

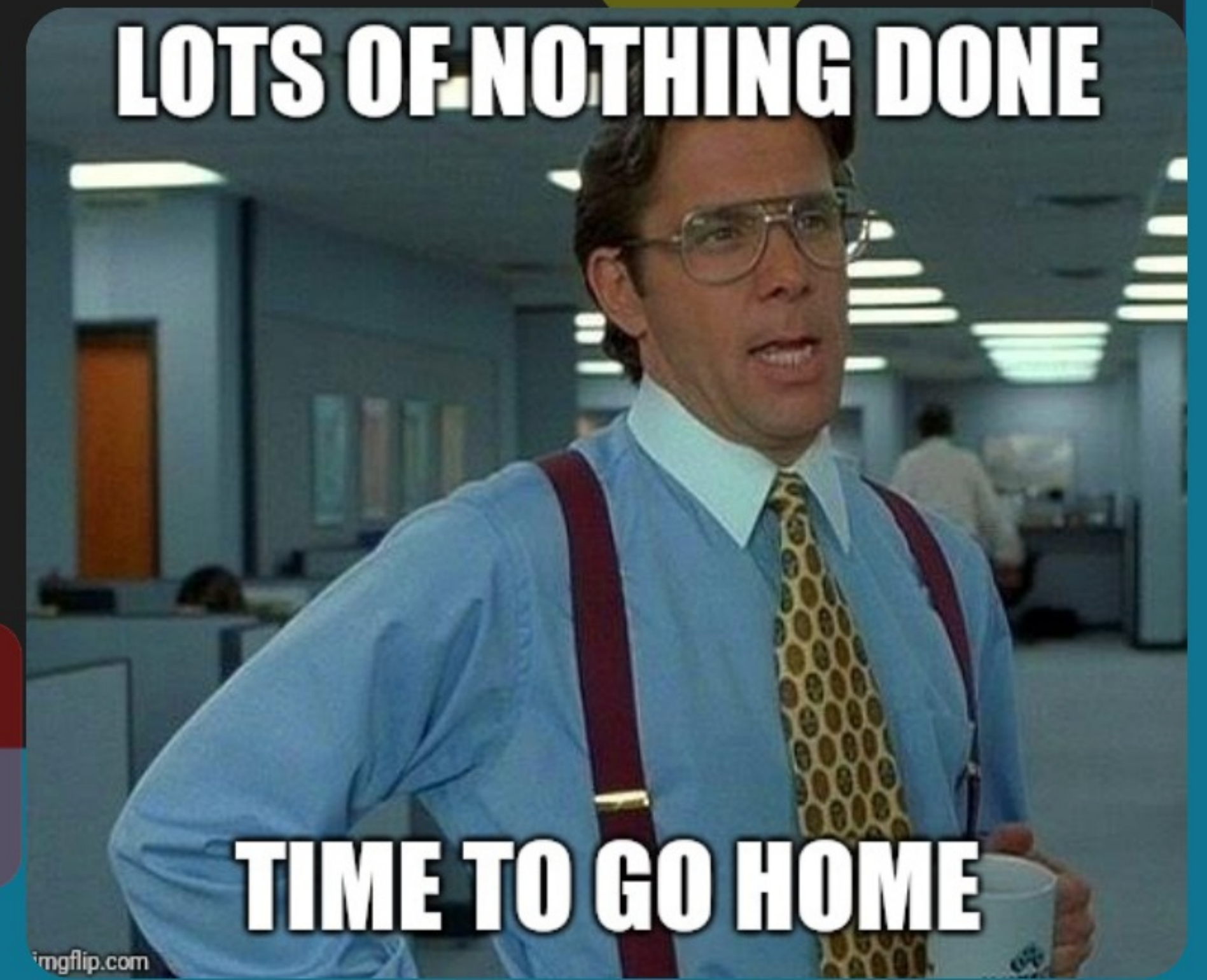
Pass by Reference! 👍

Structured Binding! 👍

Does not modify nums

LOTS OF NOTHING DONE

TIME TO GO HOME



A Classic Reference Copy Bug

```
1 #include <iostream>
2 #include <math.h>
3 #include <vector>
4 void shift(std::vector<std::pair<int, int>> &nums) {
5     for (auto [num1, num2] ← nums) {
6         num1++;
7         num2++;
8     }
9 }
```

Each pair gets copied!

A Classic Reference Copy Bug

```
1 #include <iostream>
2 #include <math.h>
3 #include <vector>
4 void shift(std::vector<std::pair<int, int>> &nums) {
5     for (auto& [num1, num2] : nums) {
6         num1++;
7         num2++;
8     }
9 }
```

Fixed!

Go to 106l.com and fill out slide 4 of today's activity!

Enter a URL to generate a QR code.

Plan

1. Initialization
2. References
3. **L-values vs R-values**
4. **const**
5. Compiling C++ programs

Yay or nay?



```
1 int x = 5;
```

"Variables" can appear on left or right.



```
1 int 5 = x;
```

"Values" can appear on the right.



```
1 int y = x;
```

"Values" cannot appear on left.

L-values & R-values

	L-value	r-value
Full Name	Locator Value	Read Value
Where with respect to equal sign?	left or right	right
Memory	Has a memory address	Temporary value (No memory address)
Example	<pre>int x = 10; int y = x;</pre>	<pre>int x = 10; int y = x;</pre>

L-values & R-values

```
1 #include <iostream>
2 #include <math.h>
3 // note the ampersand!
4 void squareN(int& n) {
5     // calculates n to the power of 2
6     n = pow(n, 2);
7 }
8 int main() {
9     int num = 5;
10    squareN(num);
11    std::cout << num << std::endl;
12    return 0;
13 }
```

L or R?

& / reference

- Means it must be a non-temporary value (an L-value).

So, we must
pass an L-value.

L-values & R-values

Code

Console

CPP

Run

```
1 #include <iostream>
2 #include <math.h>
3 // note the ampersand:
4 void squareN(int& n) {
5     // calculates n to the power of 2
6     n = pow(n, 2);
7 }
8 int main() {
9     squareN(5);
10    return 0;
11 }
```

L

R



Go to 106l.com and fill out slide 5 of today's activity!

Enter a URL to generate a QR code.

Plan

1. Initialization
2. References
3. L-values vs R-values
4. **const**
5. Compiling C++ programs

Const

What it is...

"Such object cannot be modified."



<https://en.cppreference.com/w/cpp/language/cv.html> ↗

Const & Reference

```
1 #include <iostream>
2
3 int main() {
4     std::vector<int> vec{ 1, 2, 3 };
5     const std::vector<int> const_vec{ 1, 2, 3 };
6     std::vector<int>& ref_vec{ vec };
7     const std::vector<int>& const_ref{ vec };
8
9     vec.push_back(3);
10    const_vec.push_back(3);
11    ref_vec.push_back(3);
12    const_ref.push_back(3);
13    return 0;
14 }
```



Const & Reference

Code

Console

CPP

Run

```
1 #include <iostream>
2
3 int main() {
4     const int a = 5;
5
6     int& b = a;
7     b++;
8
9     std::cout << a << std::endl;
10    return 0;
11 }
```

Const & Reference

`const int a`

5

`int& b`

const

(compiler)

**THAT IS NOT
ALLOWED**



Const & Reference

Code

Console

CPP

Run

```
1 #include <iostream>
2
3 int main() {
4     const int a = 5;
5
6     const int& b = a;
7     //b++;
8
9     std::cout << a << std::endl;
10    return 0;
11 }
```

Go to 106l.com and fill out slide 6 of today's activity!

Enter a URL to generate a QR code.

Plan

1. Initialization
2. References
3. L-values vs R-values
4. `const`
5. **Compiling C++ programs**

Compiling

source.cpp

```
1 #include <iostream>
2
3 int main() {
4     std::cout << "hello";
5     return 0;
6 }
```

Compiling

machine.exe

```
00110001
00110000
00110110
01101100
```

Compiling

- C++ is compiled.
 - It cannot be "interpreted" or run as it is, like Python.
- C++ needs a compiler.
 - A program that converts it from source code.
 - clang and g++ are popular.
- How to compile:

```
• g++ -std=c++23 main.cpp -o main
```

g++ is our compiler program.

We specify we want c++ version 23.

We specify our input file(s) are main.cpp

Our output file should be named main.

What you need to know...

main.cpp

```
1 #include <iostream>
2
3 int main() {
4     std::cout << "hello";
5     return 0;
6 }
```



main

```
00110001
00110000
00110110
```

Compile:

```
$ g++ --std=c++23 main.cpp -o main
```

Run:

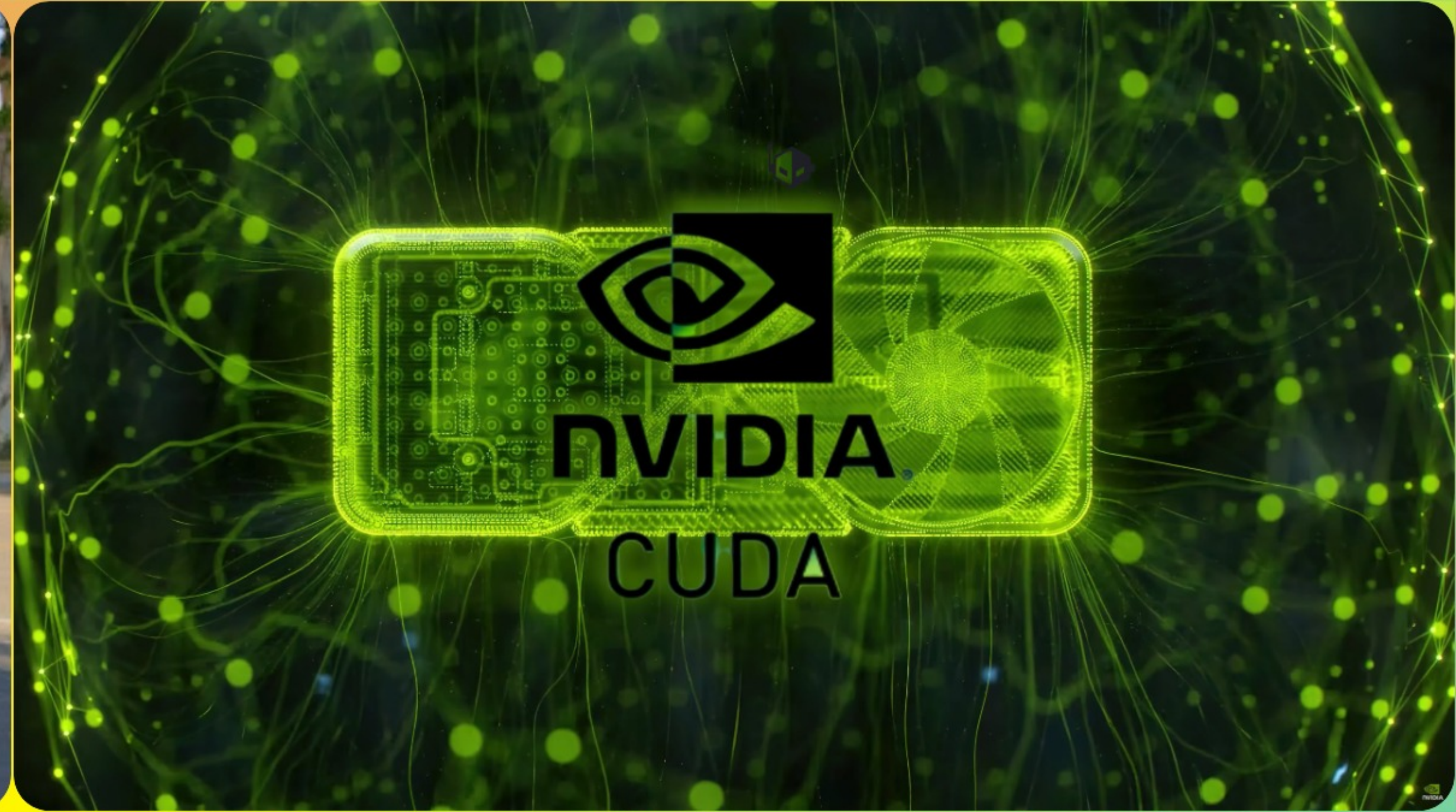
```
$ ./main
```

or on Windows:

```
$ .\main.exe
```

Go to 106l.com and complete today's activity!

Enter a URL to generate a QR code.





TensorFlow

python 3.10 | 3.11 | 3.12 | 3.13 pypi package 2.21.0 DOI 10.5281/zenodo.4724125 openssf best practices passing
openssf scorecard 7.2 oss-fuzz build failing oss-fuzz build failing custom badge inaccessible Contributor Covenant v1.4 adopted

Documentation

api reference

[TensorFlow](#) is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of [tools](#), [libraries](#), and [community](#) resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML-powered applications.

The TensorFlow Core is written largely in C++ and it is composed of 2,000+ source files.

Recap

1. Use uniform initialization!
2. References can alias variables
3. You can only reference L-values
4. `const` ensures you can't modify a variable