CS106L: Lecture 3
Initialization & References

Preston Seay, Rachel Fernandez

Recap

`auto` → "Hey compiler, figure out this type."

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

Recap

```cpp
#include <iostream>
#include <string>
#include <map>
#include <unordered_map>
#include <vector>

int main()
{
    std::map<std::string, std::vector<std::pair<int, std::unordered_map<char, double>>>>
    complexType;
    /// confusing iterator type (We'll find out what this is in the iterators lecture!)
    std::map<std::string,std::vector<std::pair<int,std::unordered_map<char,double>>>>::iterator
    it = complexType.begin();
    // clear(er) iterator type!
    auto it = complexType.begin();
    return 0;
}
```

Recap

`auto` → "Hey compiler, figure out this type."

`struct` → "Group these variables together, in one type, please 🥺."

Life & Logistics

OH Preferences

https://forms.gle/JZ3kKbBKjS3ejq4N8

Plan

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

Initialization

What it is...

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

C++ Reference Definition

Open in new tab

C++ Reference Definition

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

Scan the code or open the URL in a browser to view the live page.

Initialization

What it is...

How to...

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

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

(1) Direct Initialization

Code Console CPP Run

```cpp
#include <iostream>

int main() {
    int numOne = 12.0;
    int numTwo(12.0);
    std::cout << "numOne is: " << numOne << std::endl;
    std::cout << "numTwo is: " << numTwo << std::endl;
    return 0;
}
```

12.0 not an int... does this work?

YES

(1) Direct Initialization

Code Console CPP Run

```cpp
#include <iostream>

void checkCool(float temperature) {
    if (temperature > 100.0) {
        std::cout << "Emergency cooling activated!" << std::endl;
    } else {
        std::cout << "Temperature normal. No emergency cooling required.";
    }
}

int main() {
    float temperatureReading(100.8);
    int temperature = temperatureReading;
    checkCool(temperature);
    return 0;
}
```

Critical? Yes

(1) Direct Initialization

Code Console CPP Run

```cpp
#include <iostream>

void checkCool(float temperature) {
    if (temperature > 100.0) {
        std::cout << "Emergency cooling activated!" << std::endl;
    } else {
        std::cout << "Temperature normal. No emergency cooling required.";
    }
}

int main() {
    float temperatureReading(100.8);
    int temperature = temperatureReading;
    checkCool(temperature);
    return 0;
}
```

C++ does not care

"You want 100.8 to be an integer? Okay 😃" - compiler

Narrowing Conversion

Initialization

What it is...

How to...

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

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

(2) Uniform Initialization

Code Console

CPP Run

Notice the curly brackets!

```cpp
#include <iostream>

int main() {
    int numOne = {12.0};
    int numTwo{12.0};
    std::cout << "numOne is: " << numOne << std::endl;
    std::cout << "numTwo is: " << numTwo << std::endl;
    return 0;
}
```

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

```cpp
#include <iostream>
#include <map>

int main() {
    // Uniform initialization of a map.
    std::map<std::string, int> ages{
        {"Alice", 25},
        {"Bob", 30},
        {"Charlie", 35}
    };
    // Accessing map elements.
    std::cout << "Alice's age: " << ages["Alice"] << std::endl;
    std::cout << "Bob's age: " << ages.at("Bob") << std::endl;
    return 0;
}
```

(2) Uniform Initialization

Code Console CPP Run

```cpp
#include <iostream>
#include <vector>
int main() {
    // Uniform initialization of a vector.
    std::vector<int> numbers{1, 2, 3, 4, 5};
    // Accessing vector elements.
    for (int num : numbers) {
        std::cout << num << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

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...

How to...

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

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

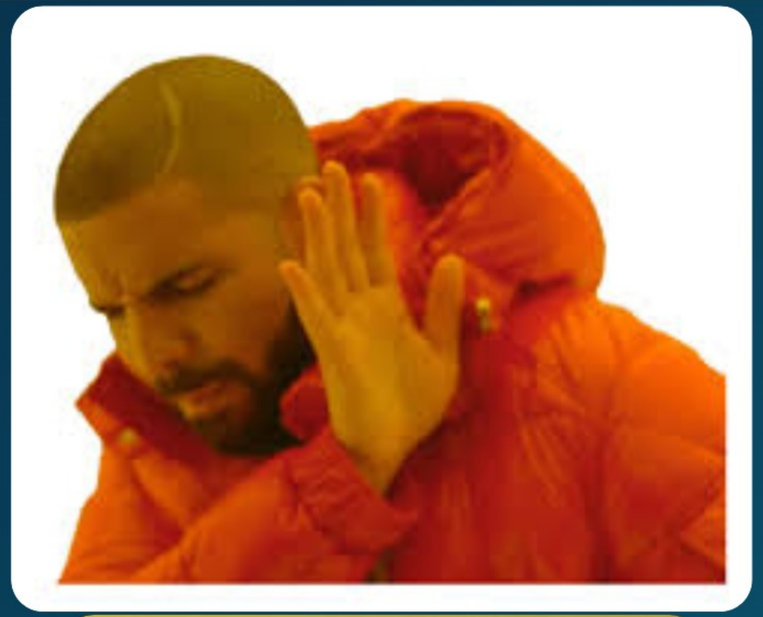
Code Console CPP Run

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

std::tuple<std::string, std::string, std::string> getClassInfo() {
    std::string className = "CS106L";
    std::string buildingName = "Thornton 110";
    std::string language = "C++";
    return {className, buildingName, language};
}

int main() {
    auto [className, buildingName, language] = getClassInfo();
    std::cout << "Come to " << buildingName << " and join us for " << className
              << " to learn " << language << "!" << std::endl;
    return 0;
}
```

What do we call this?
Uniform initialization

What do we call this?
Structured Binding

(3) Structured Binding



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

```cpp
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.

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."

C++ Reference Definition

Open in new tab

C++ Reference Definition

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

Scan the code or open the URL in a browser to view the live page.

References

What they are...

How to...

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

&

References Example

Code Console CPP Run

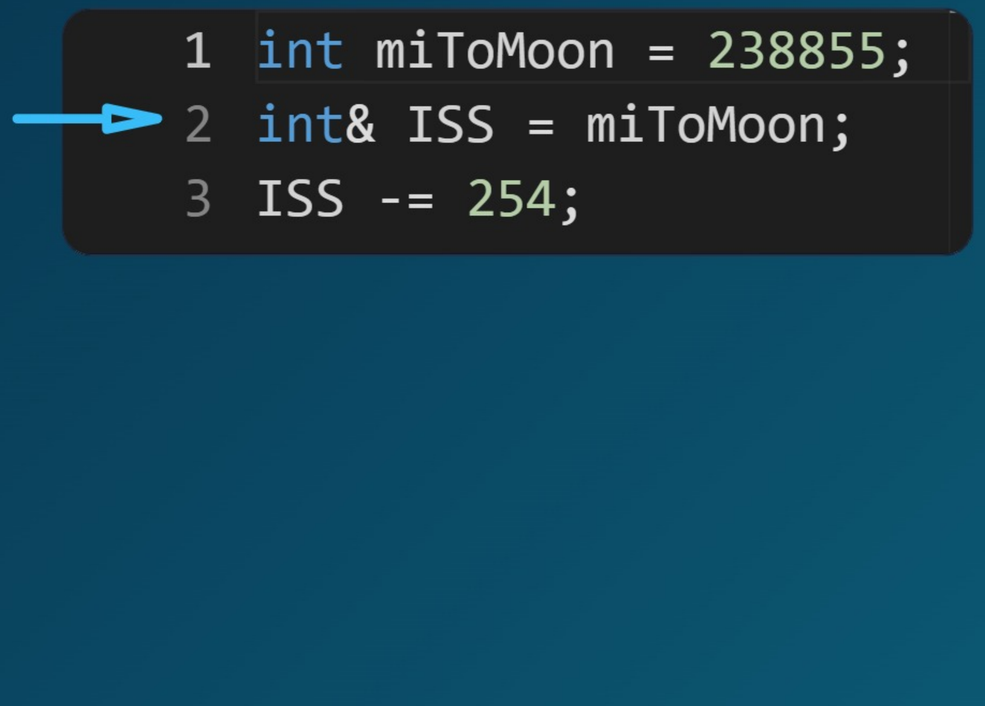
```cpp
#include <iostream>

int main() {
    int miToMoon = 238855; // That's how far the moon is.
    std::cout << "Moon is " << miToMoon << "mi away." << std::endl;

    int& ISS = miToMoon;
    ISS -= 254; // That's how high the ISS is.

    std::cout << "ISS is " << ISS << "mi to moon." << std::endl;
    std::cout << "Moon is " << miToMoon << "mi away." << std::endl;
    return 0;
}
```

References Example

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

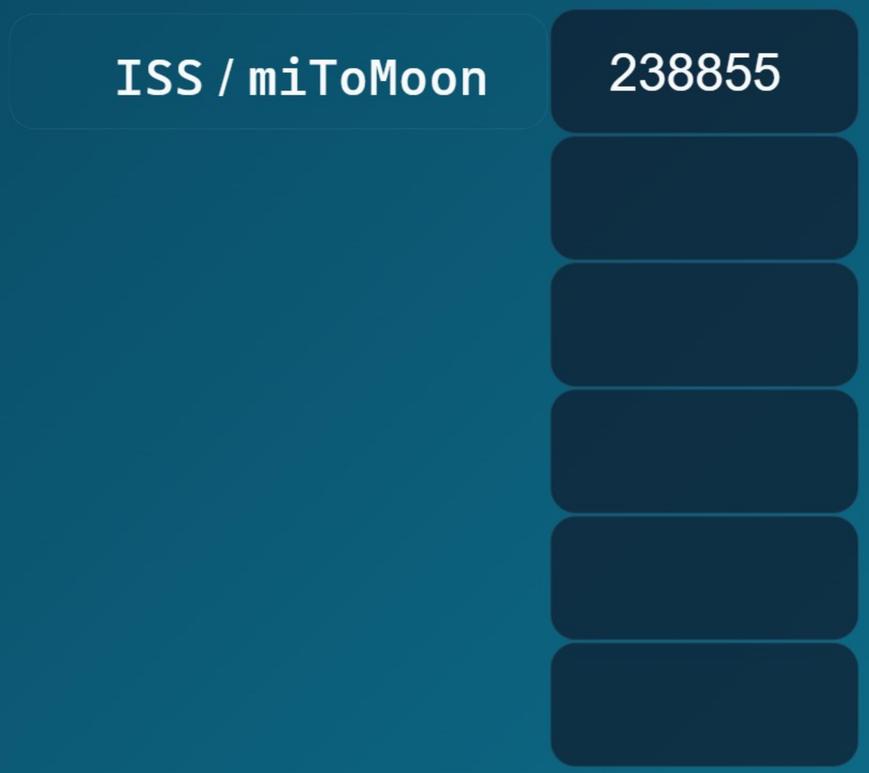
References Example

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



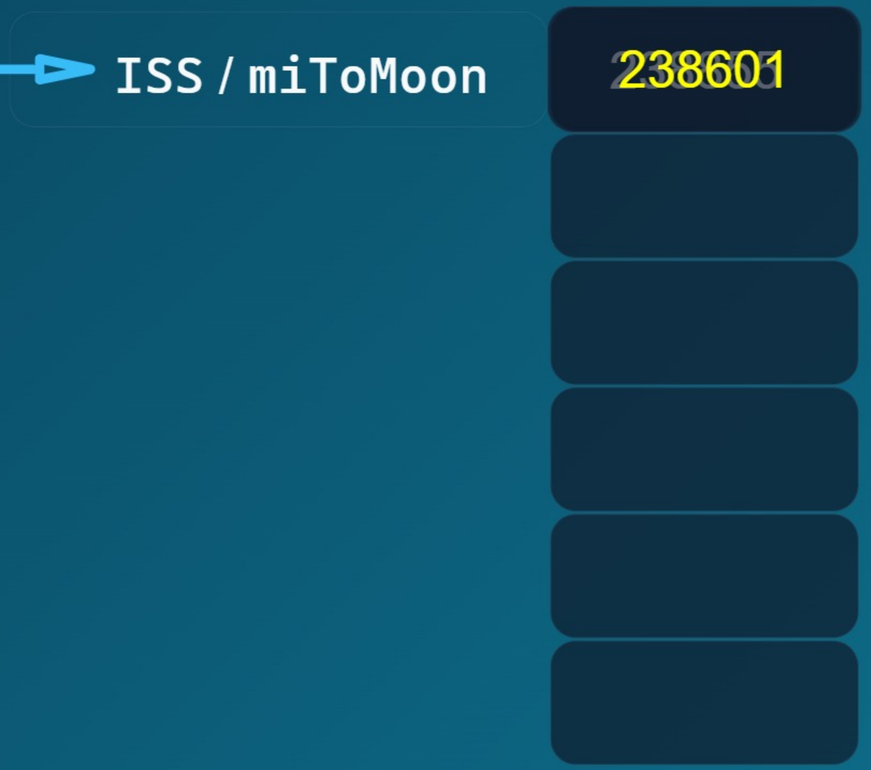
References Example

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



References Example

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



Memory

Pass by Reference

Code Console CPP Run

```cpp
#include <iostream>
#include <math.h>

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

int main() {
    int num = 5;
    squareN(num);
    std::cout << num << std::endl;
    return 0;
}
```

Note the ampersand!

n is a reference to num.

So num gets updated to 25.

Recall: Pass by Value

Code Console CPP Run

```cpp
#include <iostream>
#include <math.h>

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

int main() {
    int num = 5;
    squareN(num);
    std::cout << num << std::endl;
    return 0;
}
```

n is a ***copy*** of num.

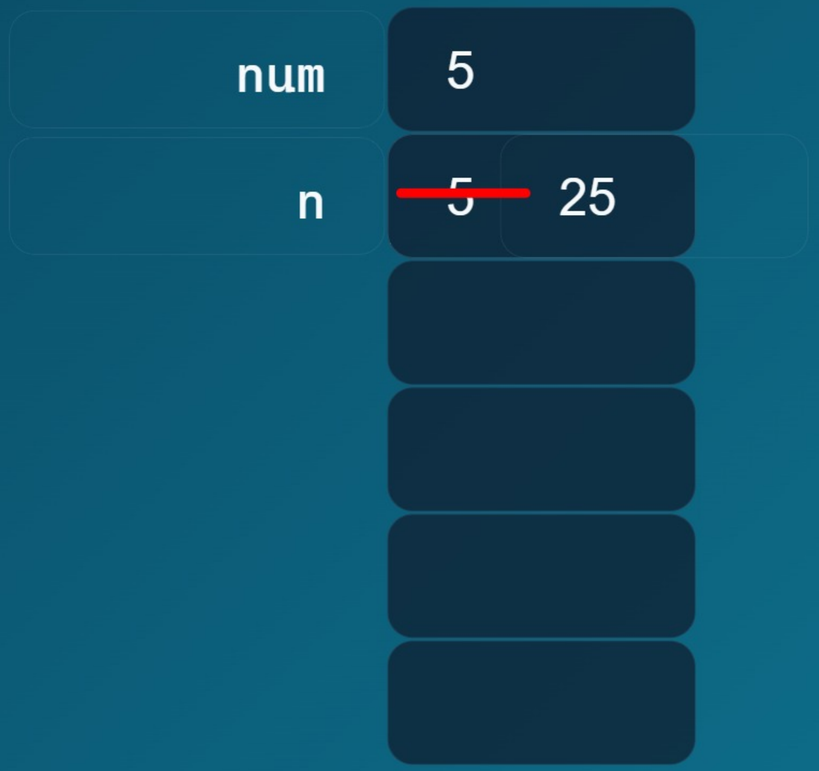
So num does not get updated.

Recall: Pass by Value

Code Console CPP Run

```cpp
#include <iostream>
#include <math.h>

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

int main() {
    int num = 5;
    squareN(num);
    std::cout << num << std::endl;
    return 0;
}
```

Memory



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!)

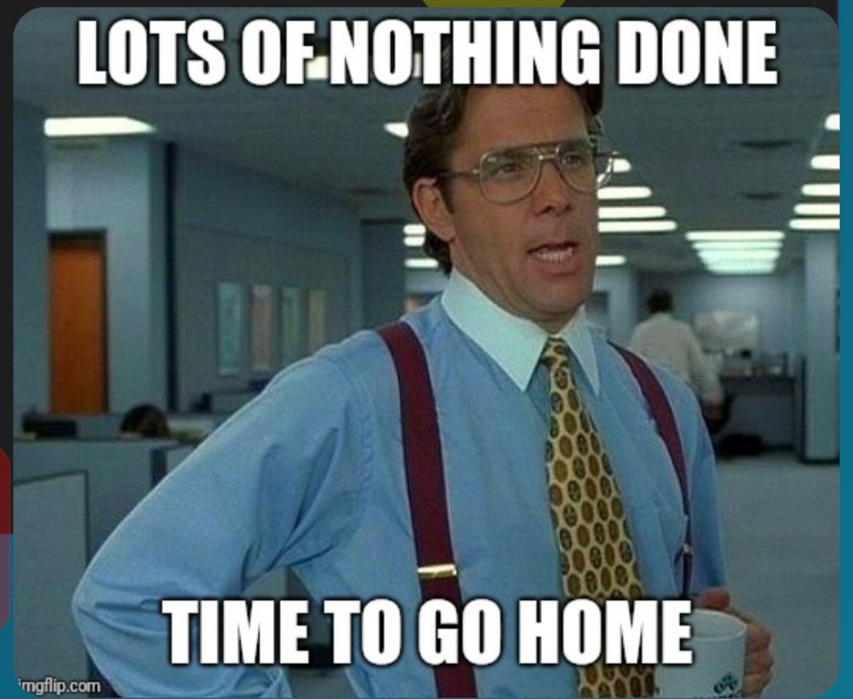
A Classic Reference Copy Bug

```cpp
#include <iostream>
#include <math.h>
#include <vector>
void shift(std::vector<std::pair<int, int>> &nums) {
    for (auto [num1, num2] : nums)
        num1++;
        num2++;
    }
}
```

Pass by Reference! 👍

Structured Binding! 👍

Does not modify nums



A Classic Reference Copy Bug

```cpp
#include <iostream>
#include <math.h>
#include <vector>
void shift(std::vector<std::pair<int, int>> &nums) {
    for (auto [num1, num2] : nums) {
        num1++;
        num2++;
    }
}
```

Each pair gets copied!

A Classic Reference Copy Bug

```cpp
#include <iostream>
#include <math.h>
#include <vector>
void shift(std::vector<std::pair<int, int>> &nums) {
    for (auto& [num1, num2] : nums) {
        num1++;
        num2++;
    }
}
```

Fixed!

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** apear 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	`int x = 10;` `int y = x;`	`int x = 10;` `int y = x;`

L-values & R-values

Code Console CPP Run

```cpp
#include <iostream>
#include <math.h>
// note the ampersand!
void squareN(int& n) {
    // calculates n to the power of 2
    n = pow(n, 2);
}
int main() {
    int num = 5;
    squareN(num);
    std::cout << num << std::endl;
    return 0;
}
```

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

```cpp
#include <iostream>
#include <math.h>
// note the ampersand
void squareN(int& n) {
    // calculates n to the power of 2
    n = pow(n, 2);
}
int main() {
    squareN(5);
    return 0;
}
```

L

R

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."

C++ Reference Definition

Open in new tab

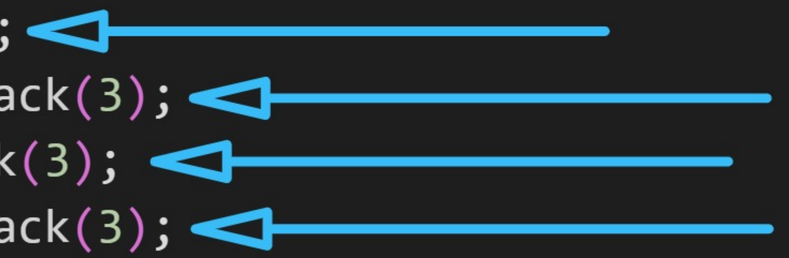
C++ Reference Definition

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

Scan the code or open the URL in a browser to view the live page.

Const & Reference

```cpp
#include <iostream>

int main() {
    std::vector<int> vec{ 1, 2, 3 };
    const std::vector<int> const_vec{ 1, 2, 3 };
    std::vector<int>& ref_vec{ vec };
    const std::vector<int>& const_ref{ vec };

    vec.push_back(3);
    const_vec.push_back(3);
    ref_vec.push_back(3);
    const_ref.push_back(3);
    return 0;
}
```



Const & Reference

Code Console CPP Run

```cpp
#include <iostream>

int main() {
    const int a = 5;

    int& b = a;
    b++;

    std::cout << a << std::endl;
    return 0;
}
```

Const & Reference

```
const int a
```

```
int& b
```

Const & Reference

Code Console CPP Run

```cpp
#include <iostream>

int main() {
    const int a = 5;

    const int& b = a;
    //b++;

    std::cout << a << std::endl;
    return 0;
}
```

Plan

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

Compiling

source.cpp

```cpp
#include <iostream>

int main() {
    std::cout << "hello";
    return 0;
}
```

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++ 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

```cpp
#include <iostream>

int main() {
    std::cout << "hello";
    return 0;
}
```

main

```
00110001
00110000
00110110
```

Compile:

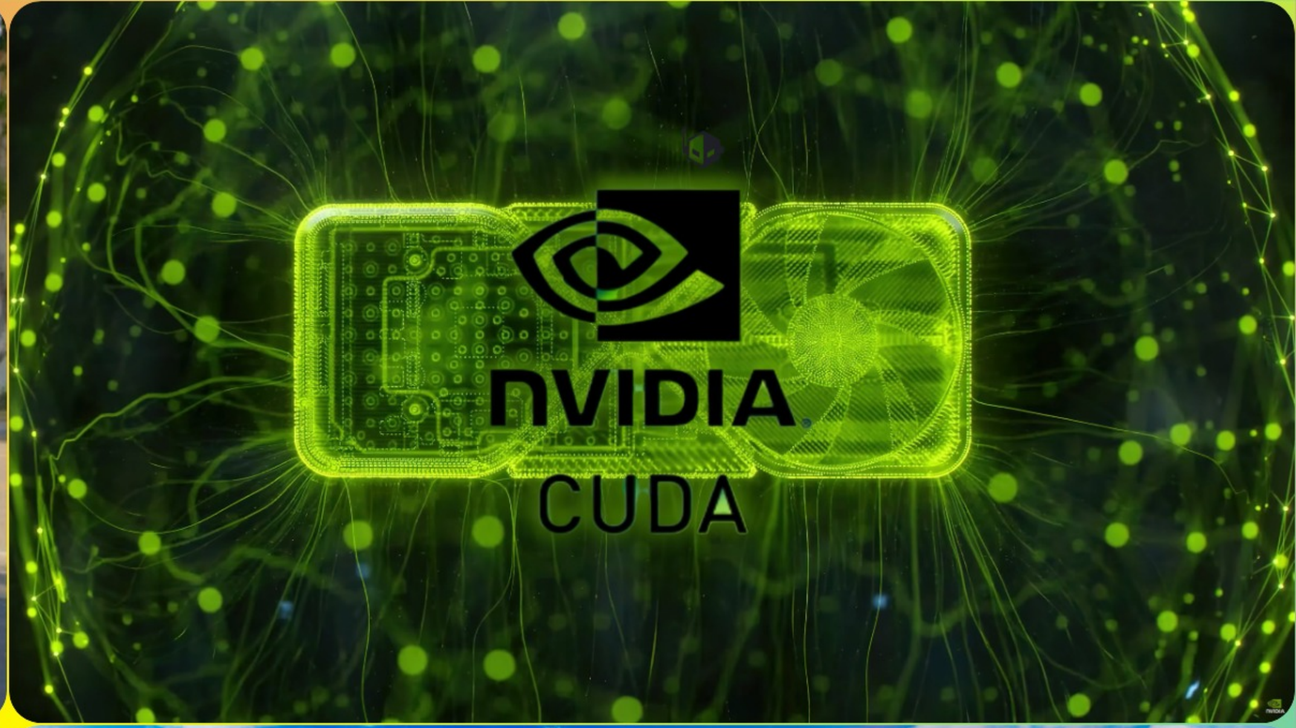
```
$ g++ --std=c++23 main.cpp -o main
```

Run:

```
$ ./main
```

or on Windows:

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


NVIDIA
CUDA


AQR
D E Shaw & Co
CITADEL
2σ TWO SIGMA
Point 72

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