

Lecture 2: Types and Structs

Stanford CS106L, Winter 2025

Last Time

- Introductions!
- Why you should take 106L?
- Course Logistics

cs106l.stanford.edu

What's one thing you remember from last lecture?

Pair up and discuss!

Today's Agenda

We'll cover a LOT of material in this class

What questions do you have?



bjarne_about_to_raise_hand

What is C++?

Structure of a C++ Program

```
// Include other libraries, similar to Python's "import"
#include <iostream>
#include <utility>
#include <cmath>

// Main logic of your program goes here
int main() {
    std::cout << "Hello World" << std::endl;
    std::cout << "Welcome to " << std::endl;
    for (char ch : "CS106L")
    {
        std::cout << ch << std::endl;
    }
}
```

Python

```
print("Hello World")
print("Welcome to ")
for ch in "CS106L":
    print(ch)
```

C++

```
std::cout << "Hello World" << std::endl;
std::cout << "Welcome to " << std::endl;
for (char ch : "CS106L")
{
    std::cout << ch << std::endl;
}
```

Okay... but what *is* C++?

How do we run code?

Source Code

```
print("Hello World")
print("Welcome to ")
for ch in "CS106L":
    print(ch)
```

How do we run code?

Source Code

```
print("Hello World")
print("Welcome to ")
for ch in "CS106L":
    print(ch)
```

CPU



How do we run code?

Source Code

```
print("Hello World")
print("Welcome to ")
for ch in "CS106L":
    print(ch)
```

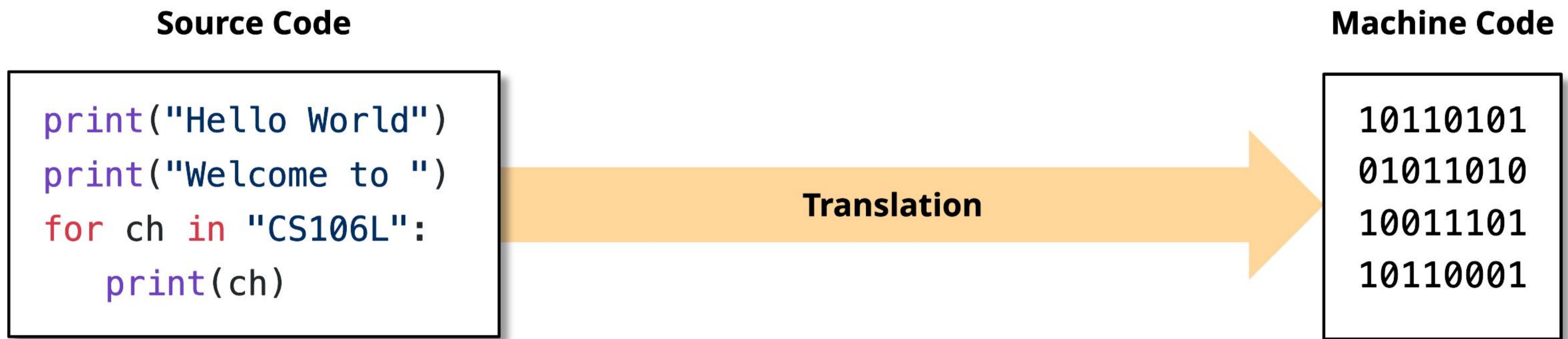
Machine Code

```
10110101
01011010
10011101
10110001
```

CPU



Taking a closer look...



Interpreted Languages

Source Code

```
print("Hello World")
print("Welcome to ")
for ch in "CS106L":
    print(ch)
```

Machine Code

```
10110101
01011010
10011101
10110001
```

Interpreted Languages

Source Code

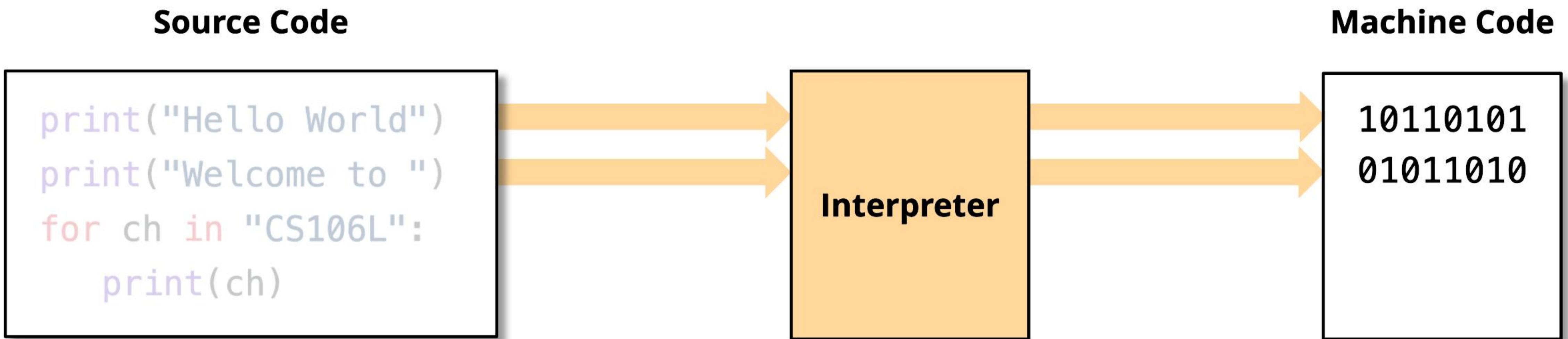
```
print("Hello World")
print("Welcome to ")
for ch in "CS106L":
    print(ch)
```

Interpreter

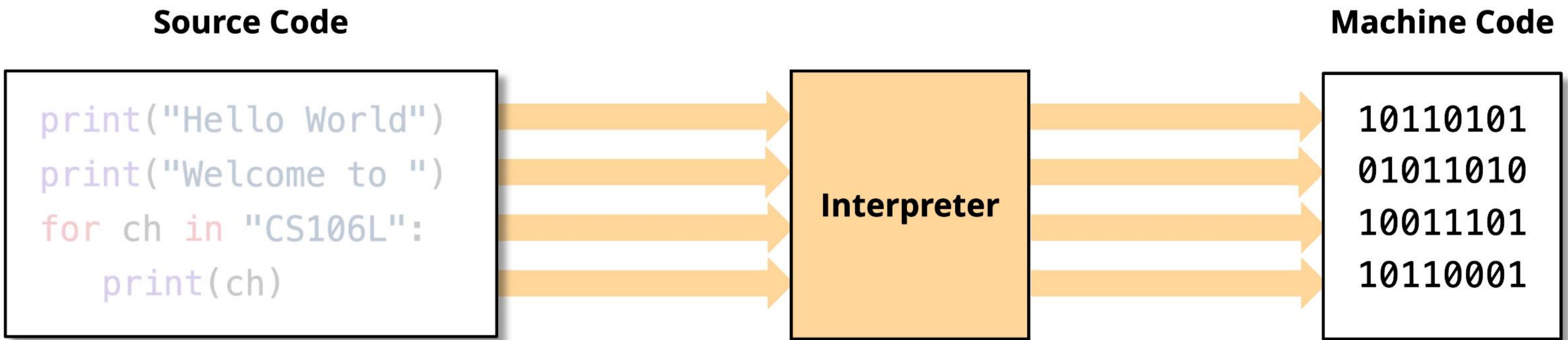
Machine Code

```
10110101
01011010
10011101
10110001
```

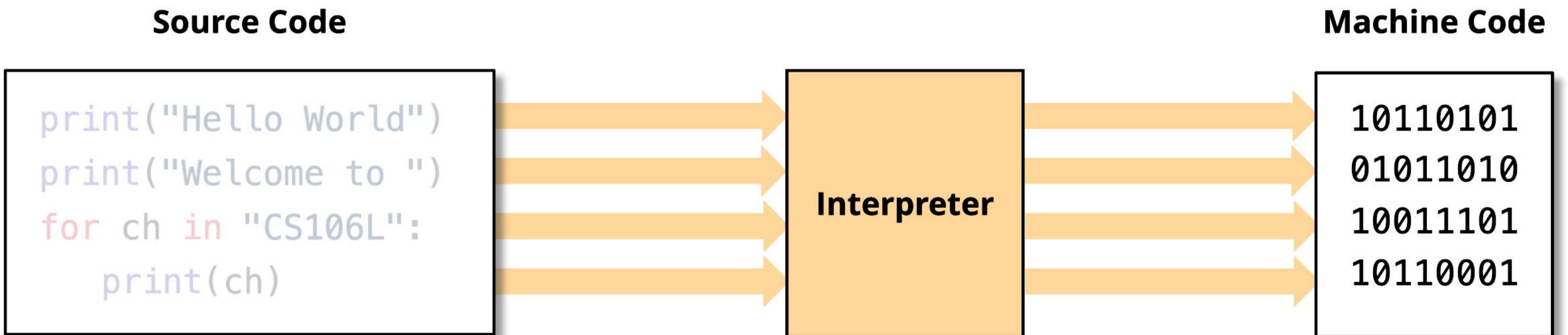
Interpreted Languages



Interpreted Languages



Interpreted Languages



```
$ python3 main.py # python3 is the interpreter
```

Compiled Languages

Source Code

```
std::cout << "Hello World" << std::endl;
std::cout << "Welcome to " << std::endl;
for (char ch : "CS106L")
{
    std::cout << ch << std::endl;
}
```

Machine Code

```
10110101
01011010
10011101
10110001
```

Compiled Languages

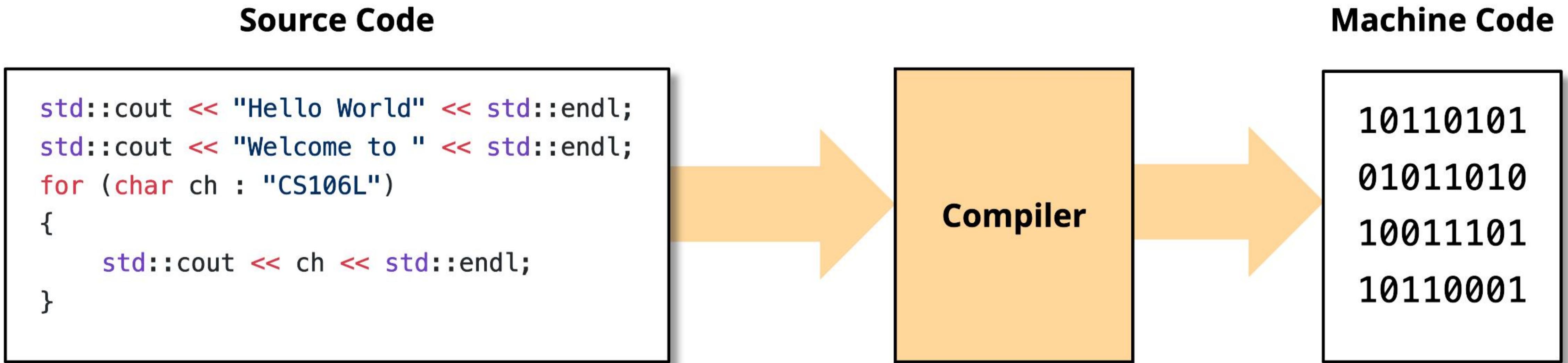
Source Code

```
std::cout << "Hello World" << std::endl;
std::cout << "Welcome to " << std::endl;
for (char ch : "CS106L")
{
    std::cout << ch << std::endl;
}
```

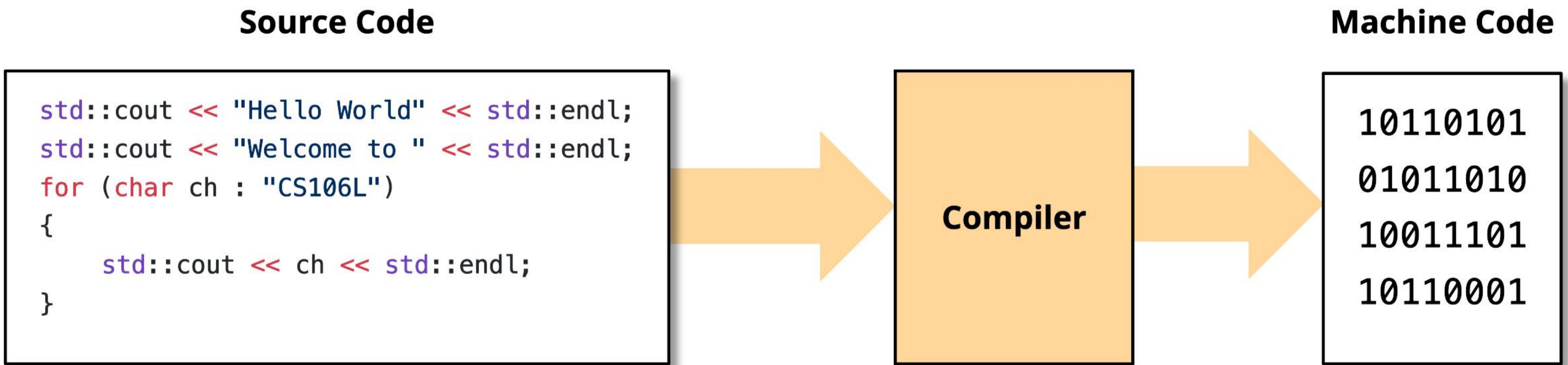
Machine Code

Compiler

Compiled Languages



Compiled Languages



```
$ g++ main.cpp -o main      # g++ is the compiler, outputs binary to main
```

Why compile over interpret?

Why compile over interpret?

- It allows us to generate more efficient machine code!
 - Interpreters only see one small part of code at a time
 - Compilers see everything
- However, compilation takes time!

C++ is a compiled language

What questions do you have?



bjarne_about_to_raise_hand

Compile time vs. runtime

```
std::cout << "Hello World" << std::endl;
std::cout << "Welcome to " << std::endl;
for (char ch : "CS106L")
{
    std::cout << ch << std::endl;
}
```

Compile
Time

```
10110101
01011010
10011101
10110001
```

Runtime



Aside: “compiled language” is a misleading term

Q1: How do we run code?

Q2: How do we verify code?

Python

```
print("Running...")
hello = "Hello ";
world = "World!";
print(hello * world)
```

Python

```
print("Running...")  
hello = "Hello ";  
world = "World!";  
print(hello * world)
```

```
$ python3 program.py
```

```
Running...  
TypeError: can't multiply sequence by  
non-int of type 'str'
```

C++

```
int main() {  
    std::cout << "Running..." << std::endl;  
    std::string hello = "Hello ";  
    std::string world = "World!";  
    std::cout << hello * world << std::endl;  
    return 0;  
}
```

C++ compilers can be noisy... why?

```
rtmap.cpp: In function `int main()':
rtmap.cpp:19: invalid conversion from `int' to `
std::__Rb_tree_node<std::pair<const int, double>*>
rtmap.cpp:19: initializing argument 1 of `std::__Rb_tree_iterator<_Val, _Ref,
_Ptr>::__Rb_tree_iterator(std::__Rb_tree_node<_Val>*)' [with _Val =
std::pair<const int, double>, _Ref = std::pair<const int, double>&, _Ptr =
std::pair<const int, double>*]'
rtmap.cpp:20: invalid conversion from `int' to `
std::__Rb_tree_node<std::pair<const int, double>*>
rtmap.cpp:20: initializing argument 1 of `std::__Rb_tree_iterator<_Val, _Ref,
_Ptr>::__Rb_tree_iterator(std::__Rb_tree_node<_Val>*)' [with _Val =
std::pair<const int, double>, _Ref = std::pair<const int, double>&, _Ptr =
std::pair<const int, double>*']
```

Types

Types

- A **type** refers to the “category” of a variable
- C++ comes with built-in types

- **int** 106
- **double** 71.4
- **string** “Welcome to CS106L!”
- **bool** true false

C++ is a statically typed language

Static Typing

- Every variable must declare a type
- Once declared, the type cannot change

Static Typing

- Every variable must declare a type
- Once declared, the type cannot change

Python (Dynamic Typing)

```
a = 3  
b = "test"
```

C++ (Static Typing)

```
int a = 3;  
string b = "test";
```

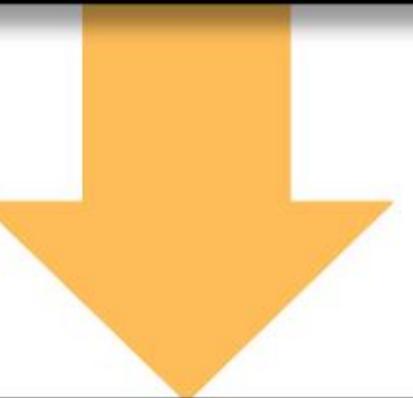
Why static typing?

Better error checking

```
def add_3(x):  
    return x + 3  
  
add_3("CS106L") # Oops, that's a string. Runtime error!
```

Better error checking

```
def add_3(x):  
    return x + 3  
  
add_3("CS106L") # Oops, that's a string. Runtime error!
```



```
int add_3(int x) {  
    return x + 3;  
}  
  
add_3("CS106L"); // Can't pass a string when int expected. Compile time error!
```

What questions do you have?



bjarne_about_to_raise_hand

Your turn

- (int) x **casts** x to an int by dropping decimals
 - E.g. (int) 5.3 = 5

```
a = "test";
_____
b = 3.2 * 5 - 1;
_____
c = 5 / 2;
_____
d(int foo) { return foo / 2; }
_____
e(double foo) { return foo / 2; }
_____
f(double foo) { return (int)(foo + 0.5); }
_____
g(double c) { std::cout << c << std::endl; }
```

Aside: Function Overloading

Defining two functions with the same name but different signatures

```
double func(int x) {           // (1)
    return (double) x + 3;     // typecast: int → double
}

double func(double x) {        // (2)
    return x * 3;
}

func(2);          // uses version ___, returns _____
func(2.0);        // uses version ___, returns _____
```

C++ is a compiled, statically typed language

What questions do you have?



bjarne_about_to_raise_hand

Structs

Keeping track of students

- Every student ID has a few properties



A fundamental problem

```
_____ issueNewID( ) {  
}  
}
```

A fundamental problem

```
_____ issueNewID() {  
    // How can we return all three things?  
    // What should our return type be?  
  
    // Python:  
    // return "Jacob Roberts-Baca", "jtrb", 6504417  
}
```

Introducing... structs!

Structs bundle data together

```
struct StanfordID {  
};
```

Structs bundle data together

```
struct StanfordID {  
    string name;           // These are called fields  
    string sunet;          // Each has a name and type  
    int idNumber;  
};  
  
StanfordID id;           // Initialize struct
```

Returning multiple values

```
StanfordID issueNewID() {  
    StanfordID id;  
    id.name = "Jacob Roberts-Baca";  
    id.sunet = "jtrb";  
    id.idNumber = 6504417;  
    return id;  
}
```

List Initialization

```
StanfordID id;  
id.name = "Jacob Roberts-Baca";  
id.sunet = "jtrb";  
id.idNumber = 6504417;
```

List Initialization

```
StanfordID id;  
id.name = "Jacob Roberts-Baca";  
id.sunet = "jtrb";  
id.idNumber = 6504417;
```



```
// Order depends on field order in struct. '=' is optional  
StanfordID jrb = { "Jacob Roberts-Baca", "jtrb", 6504417 };  
StanfordID fi { "Fabio Ibanez", "fibanez", 6504418 };
```

List Initialization

```
StanfordID id;  
id.name = "Jacob Roberts-Baca";  
id.sunet = "jtrb";  
id.idNumber = 6504417;
```



We'll learn more
about this next time!



```
// Order depends on field order in struct. '=' is optional  
StanfordID jrb = { "Jacob Roberts-Baca", "jtrb", 6504417 };  
StanfordID fi { "Fabio Ibanez", "fibanez", 6504418 };
```

Using list initialization

```
StanfordID issueNewID() {  
    StanfordID id = { "Jacob Roberts-Baca", "jtrb", 6504417 };  
    return id;  
}
```

Using list initialization

```
StanfordID issueNewID() {  
    StanfordID id = { "Jacob Roberts-Baca", "jtrb", 6504417 };  
    return id;  
}
```

```
StanfordID issueNewID() {  
    return { "Jacob Roberts-Baca", "jtrb", 6504417 };  
}
```

A **struct** bundles **named variables** into a new type

What questions do you have?



bjarne_about_to_raise_hand

Many Possible Structs

Many Possible Structs

```
struct Name {  
    string first;  
    string last;  
};
```

```
Name jrb = { "Jacob", "R.B." };
```

```
struct Order {  
    string item;  
    int quantity;  
};
```

```
Order dozen = { "Eggs", 12 };
```

Many Possible Structs

```
struct Name {  
    string first;  
    string last;  
};
```

```
Name jrb = { "Jacob", "R.B." };
```

```
struct Point {  
    double x;  
    double y;  
};
```

```
Point origin { 0.0, 0.0 };
```

```
struct Order {  
    string item;  
    int quantity;  
};
```

```
Order dozen = { "Eggs", 12 };
```

```
struct Circle {  
    Point center;  
    double radius;  
};
```

```
Circle circle { {0, 0} , 1.0 };
```

Many Possible Structs

```
struct Name {  
    string first;  
    string last;  
};
```

```
Name jrb = { "Jacob", "R.B." };
```

```
struct Order {  
    string item;  
    int quantity;  
};
```

```
Order dozen = { "Eggs", 12 };
```

Notice anything?

```
struct Point {  
    double x;  
    double y;  
};
```

```
Point origin { 0.0, 0.0 };
```

```
struct Circle {  
    Point center;  
    double radius;  
};
```

```
Circle circle { {0, 0} , 1.0 };
```

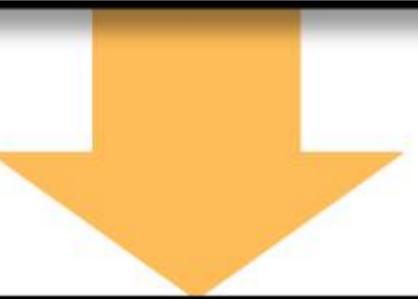
We can use `std::pair`!

std::pair

```
struct Order {  
    std::string item;  
    int quantity;  
};  
  
Order dozen = { "Eggs", 12 };
```

std::pair

```
struct Order {  
    std::string item;  
    int quantity;  
};  
  
Order dozen = { "Eggs", 12 };
```



```
std::pair<std::string, int> dozen { "Eggs", 12 };
```

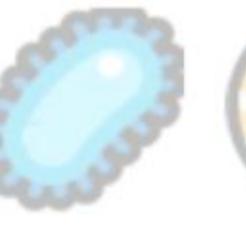
std::pair is a template

(We'll learn more about this later)

```
template <typename T1, typename T2>
struct pair {
    T1 first;
    T2 second;
};

std::pair<std::string, int>
```

There's something we need to discuss...

What is an std !!!?  

What is std !!!?

std — The C++ Standard Library

- Built-in types, functions, and more provided by C++

std — The C++ Standard Library

- Built-in types, functions, and more provided by C++
- You need to **#include** the relevant file
 - **#include <string>** → **std::string**
 - **#include <utility>** → **std::pair**
 - **#include <iostream>** → **std::cout, std::endl**
- We prefix standard library names with **std::**

std — The C++ Standard Library

- See the official standard at [cppreference.com!](https://cppreference.com)

To use `std::pair`, you must `#include` it

`std::pair` is defined in a header file called `utility`

```
#include <utility>
```

```
// Now we can use `std::pair` in our code.
```

```
std::pair<double, double> point { 1.0, 2.0 };
```

What does #include do?

```
#include <utility>

std::pair<double, double> p { 1.0, 2.0 };
```

What does #include do?

```
#include <utility>
```

```
std::pair<double, double> p { 1.0, 2.0 };
```

utility

```
namespace std {  
  
    template  
    <typename T1, typename T2>  
    struct pair {  
        T1 first;  
        T2 second;  
    };  
  
    // Other utility code...  
}
```

What does #include do?

```
namespace std {  
  
    template <typename T1, typename T2>  
    struct pair {  
        T1 first;  
        T2 second;  
    };  
  
    // Other utility code...  
}  
  
std::pair<double, double> p { 1.0, 2.0 };
```

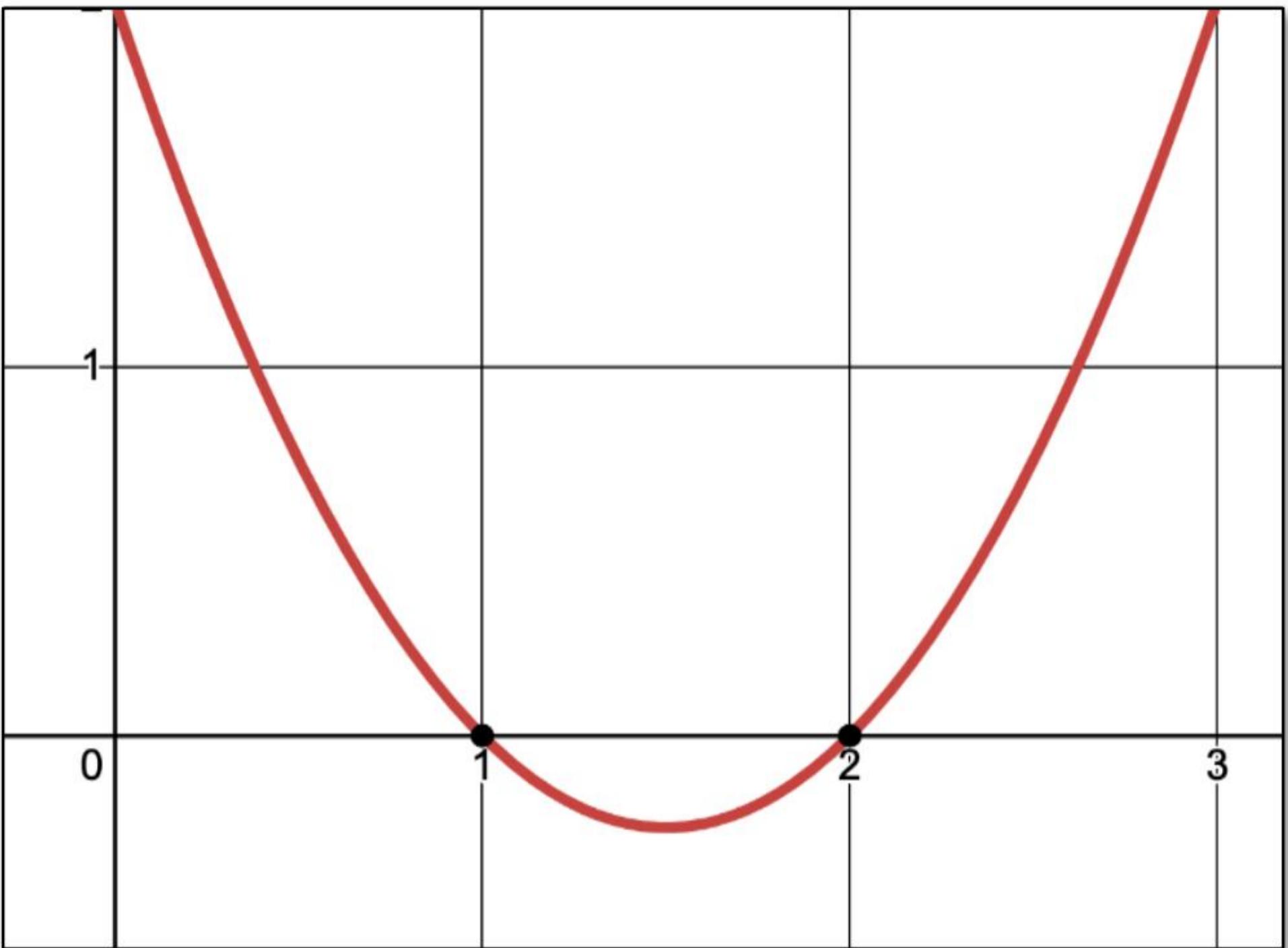
What questions do you have?



bjarne_about_to_raise_hand

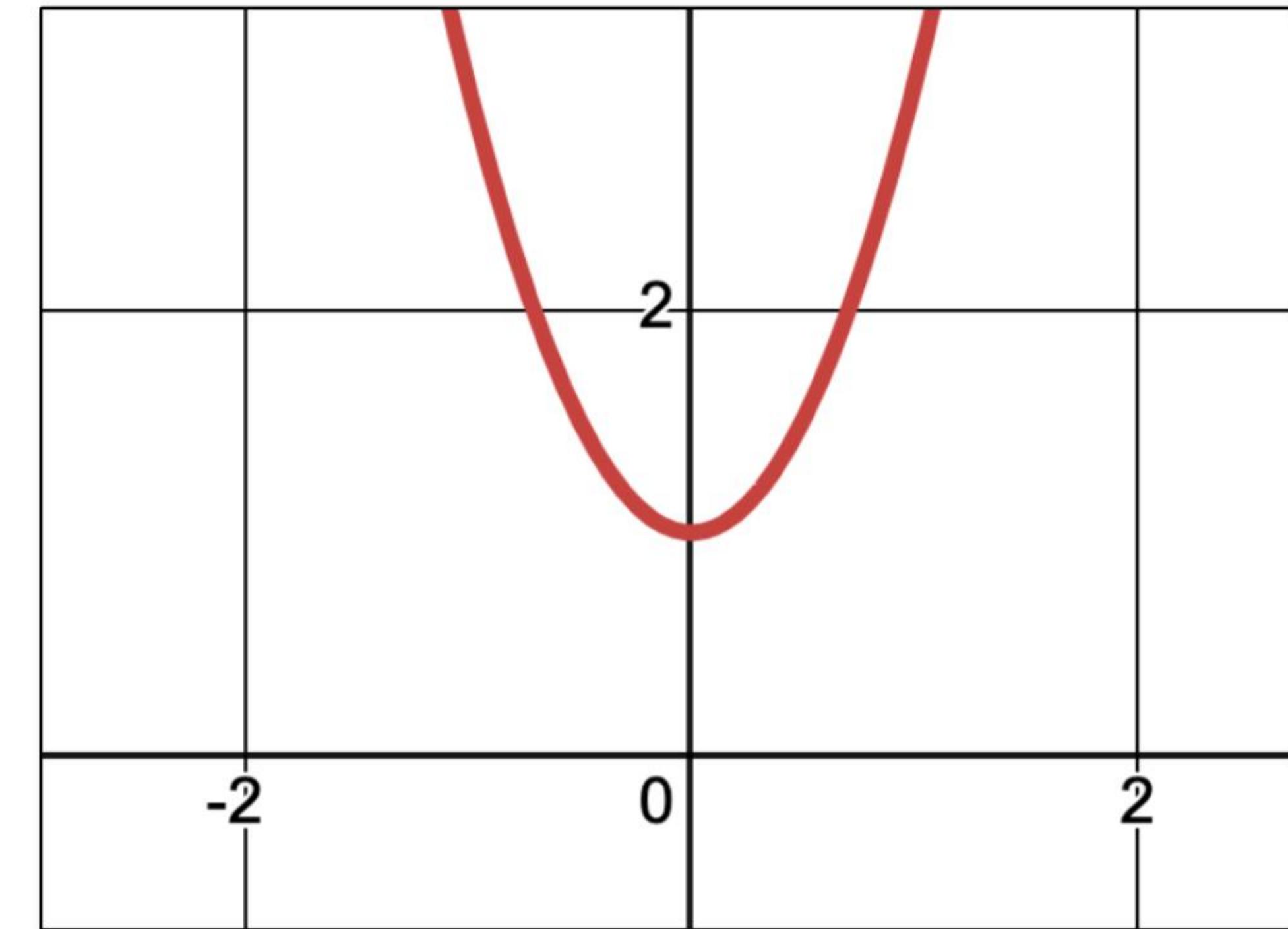
Code Demo

Solving a Quadratic Equation



$$x^2 - 3x + 2 = 0$$

$$x = 1, x = 2$$



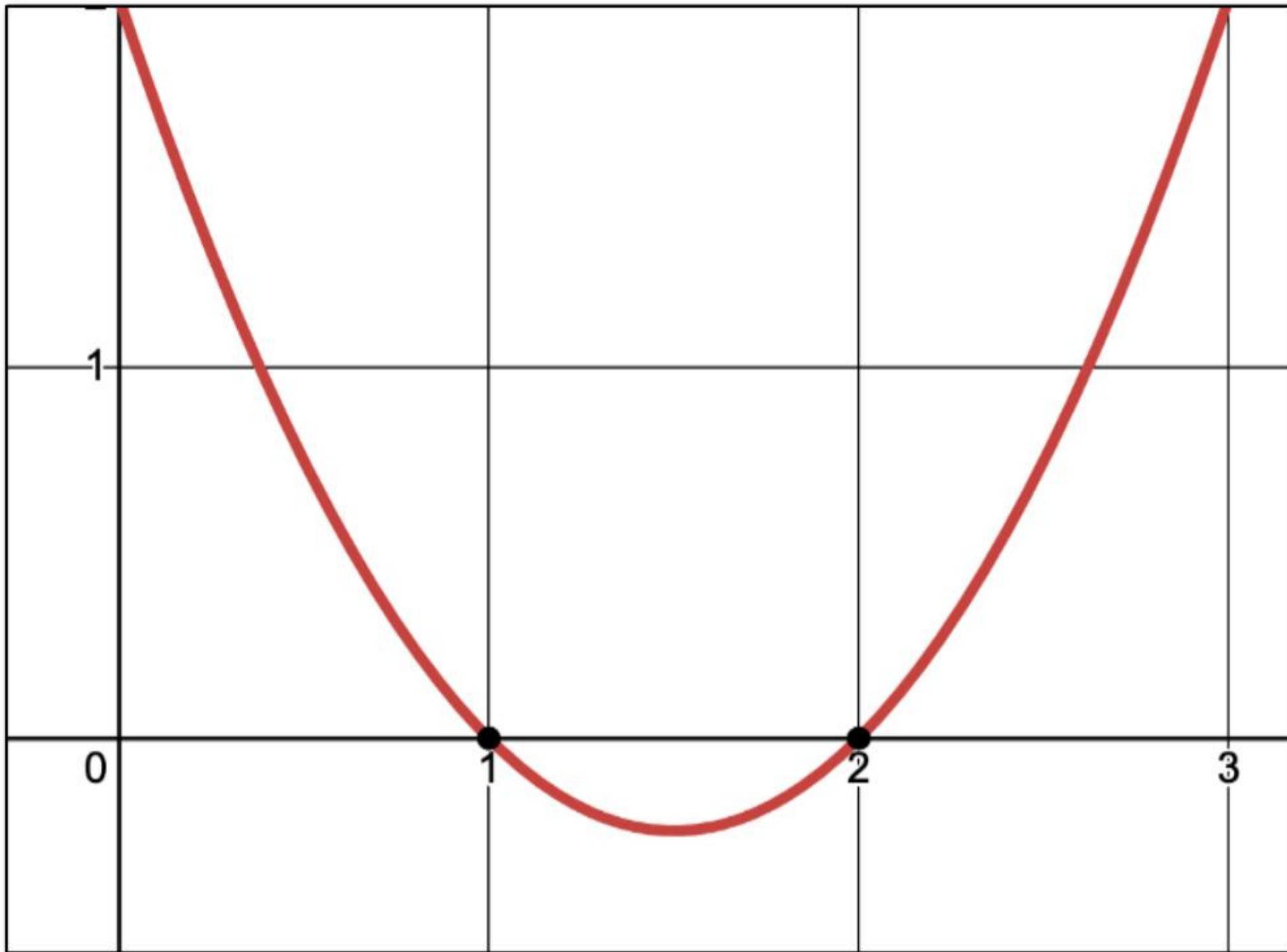
$$2x^2 + 1 = 0$$

no solution

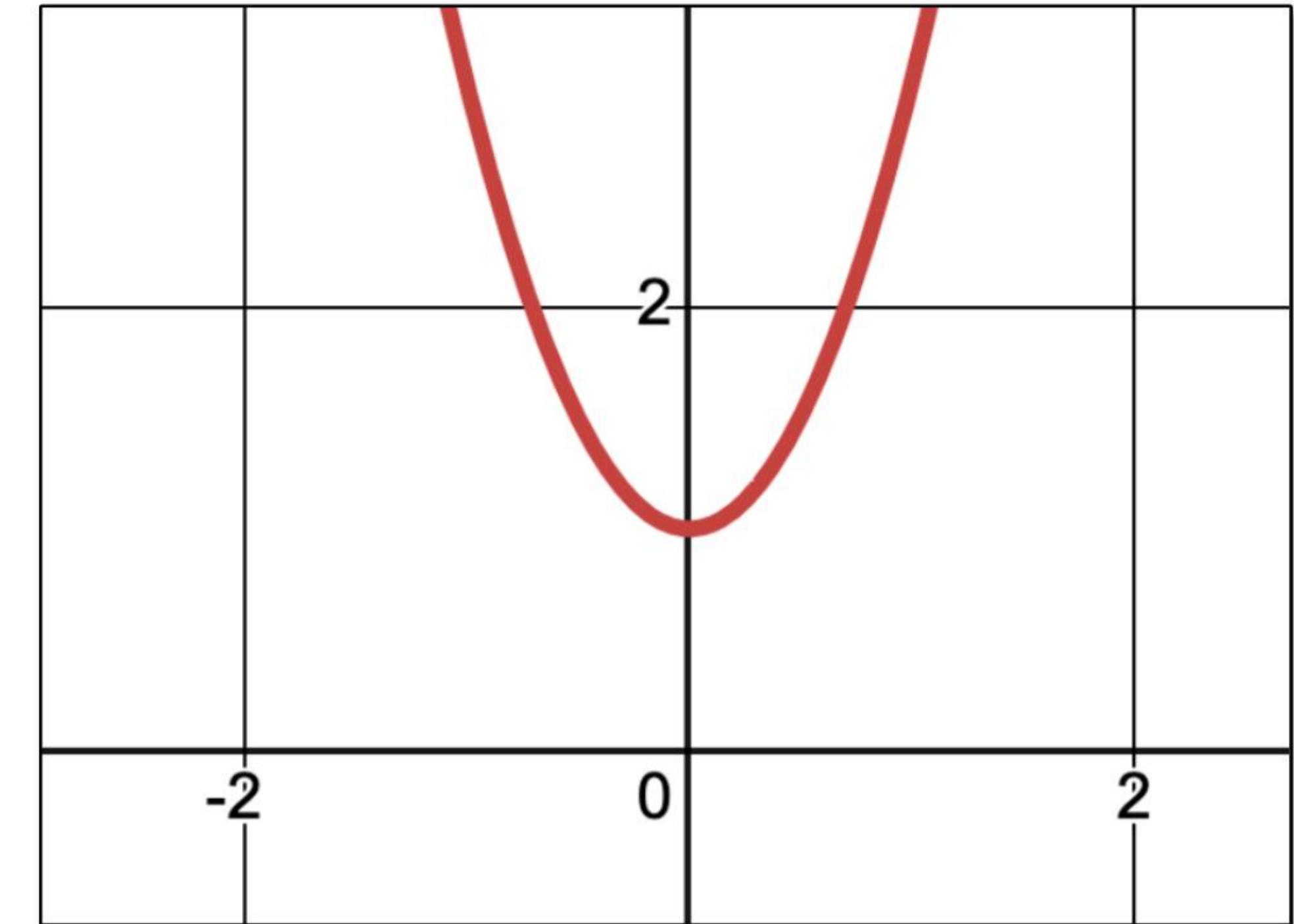
Solving a Quadratic Equation

```
std::pair<bool, std::pair<double, double>> solveQuadratic(double a, double b, double c);
```

`std::pair<bool, std::pair<double, double>>`



{ true, { 1.0, 2.0 } }



{ false, doesnt_matter }

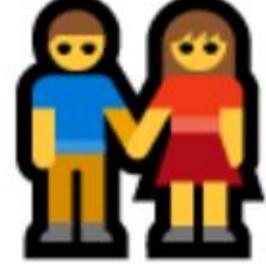
Solving a Quadratic Equation

- If we have $ax^2 + bx + c = 0$
- Solutions are $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
- If $b^2 - 4ac$ is negative, there are no solutions
- **Your task:** Write a function to solve a quadratic equation:

```
std::pair<bool, std::pair<double, double>> solveQuadratic(double a, double b, double c);
```



The `sqrt` function from the `<cmath>` header can calculate the square root

Let's code this together 

`106l.vercel.app/quadratic`

Improving Our Code

The **using** keyword

The **using** keyword

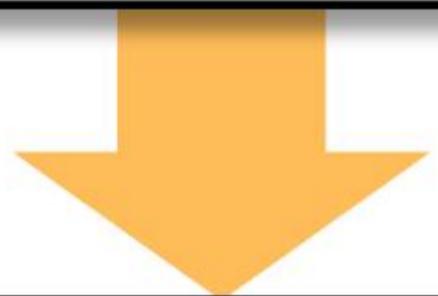
- Typing out long type names gets tiring
- We can create **type aliases** with the **using** keyword

```
std::pair<bool, std::pair<double, double>> solveQuadratic(double a, double b, double c);
```

The **using** keyword

- Typing out long type names gets tiring
- We can create **type aliases** with the **using** keyword

```
std::pair<bool, std::pair<double, double>> solveQuadratic(double a, double b, double c);
```



```
using Zeros = std::pair<double, double>;
using Solution = std::pair<bool, Zeros>;
Solution solveQuadratic(double a, double b, double c);
```

using is kind of like a variable for types!

The **auto** keyword

- The **auto** keyword tells the compiler to infer the type

```
std::pair<bool, std::pair<double, double>> result = solveQuadratic(a, b, c);
```

The `auto` keyword

- The `auto` keyword tells the compiler to infer the type

```
std::pair<bool, std::pair<double, double>> result = solveQuadratic(a, b, c);
```



```
auto result = solveQuadratic(a, b, c);
```

// This is exactly the same as the above!

// `result` still has type `std::pair<bool, std::pair<double, double>>`

// We just told the compiler to figure this out for us!

auto is still statically typed!

```
auto i = 1;    // int inferred
```

Which one is clearer?

```
std::pair<bool, std::pair<double, double>> result = ...;  
auto result = ...;
```

Which one is clearer?

```
auto i = 1;  
int i = 1;
```

What questions do you have?



bjarne_about_to_raise_hand

Recap

Recap

- C++ is a compiled, statically typed language
- Structs bundle data together into a single object

Recap

- C++ is a compiled, statically typed language
- Structs bundle data together into a single object
- **std::pair** is a general purpose struct with two fields
- #include from the C++ Standard Library to use built-in types
 - And use the std:: prefix too!

End of slide show, click to exit.