# Learn C++ by Example

Frances Buontempo

# A workshop in 3 parts

- First, some basics
  - Context
  - Output
- Second, some standard stuff
  - Input
  - Validation and exceptions
  - Options
- Finally, a game
  - Rock, paper, scissors
  - Randomness
  - Enums and arrays

# Not covering

- Using external libraries
- Build systems
- OOP
- ALL The Things
  - But we will cover some of the things

# I’m Fran

- https://mastodon.social/@fbuontempo
- https://twitter.com/fbuontempo
- https://www.linkedin.com/in/francesbuontempo/
- https://accu.org/journals/nonmembers/overload_cover_members/

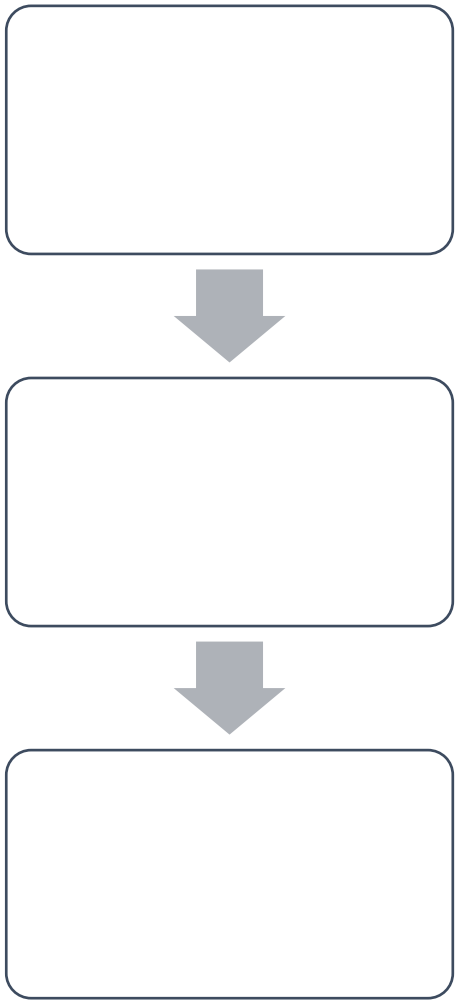


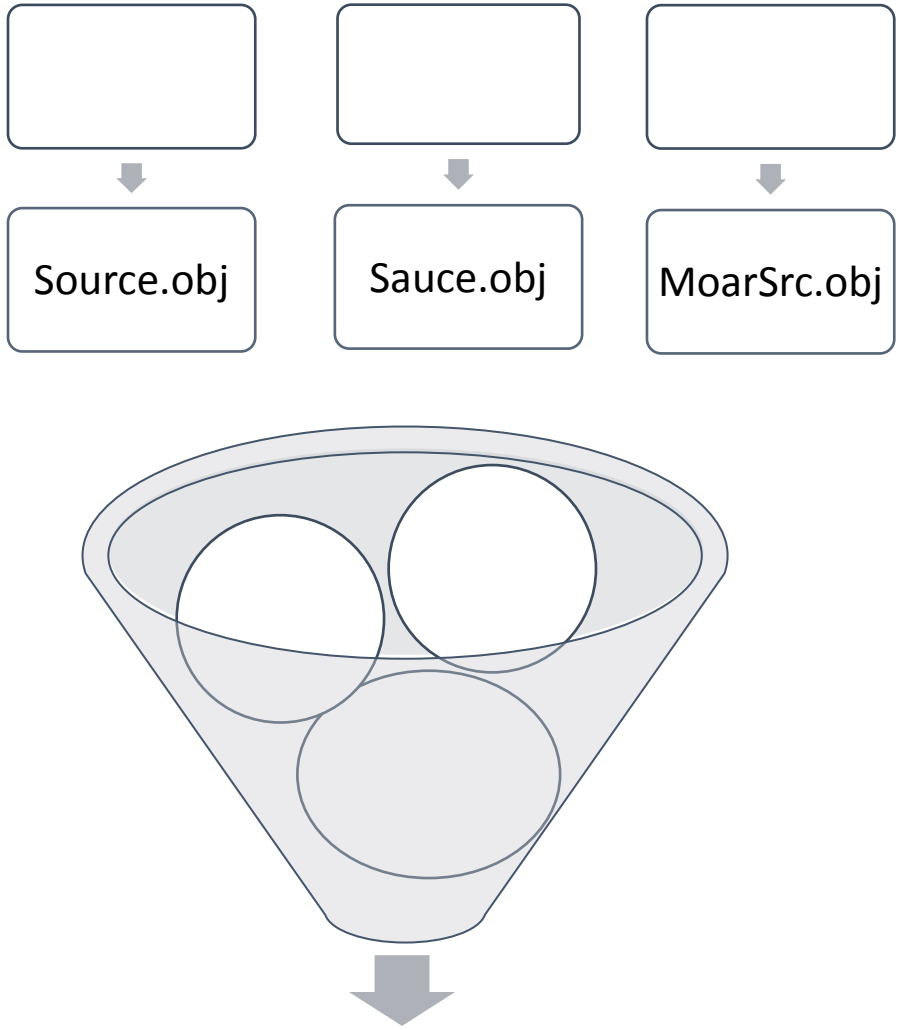
# Part 1

- First, some basics
  - Context
  - Output, input
  - Quick recap

# What is C++?

- Compiled (and linked)
  - To a specific target (no JVM, IL, …)
- Old
  - But changing every 3 years
  - C++11, C++14, C++17, C++20, C++23…
    - But not every compiler does everything
- https://isocpp.org/get-started




Source.obj
Sauce.obj
MoarSrc.obj
program

# Let’s write some code

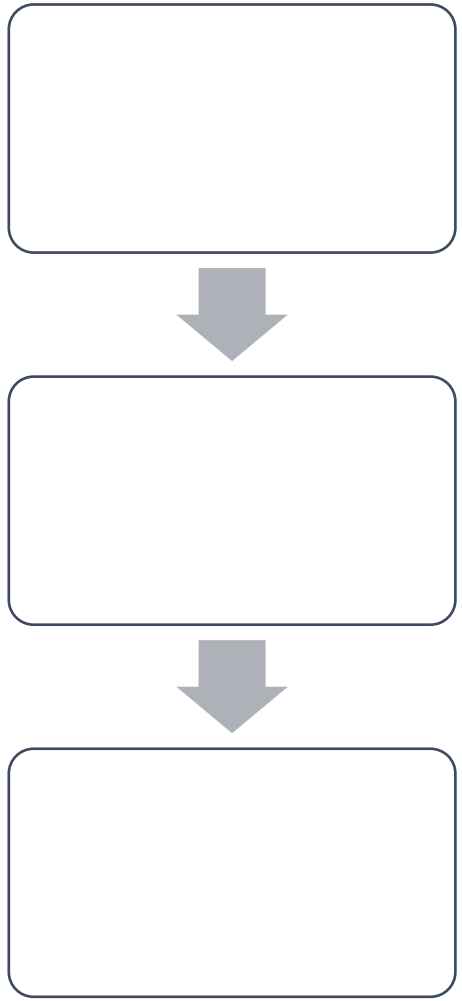
- Git repo: https://github.com/doctorlove/Socrates2024LearnCpp.git
- Godbolt: https://godbolt.org/z/qanE4doMo
- Lots of possible approaches to any problem
- C++23 introduced `std::println("Hello, world!");`
  - Using `#include <print>`
  - Or `import std;`

# Today

```
// This is a comment
// Put this in a file called hello.cpp and save it
// Pair/mob/stay on your own as desired

#include <iostream>

int main()
{
    std::cout << "Hello, world!\n";
}
//(See https://godbolt.org/z/KbWrKP65h )
```

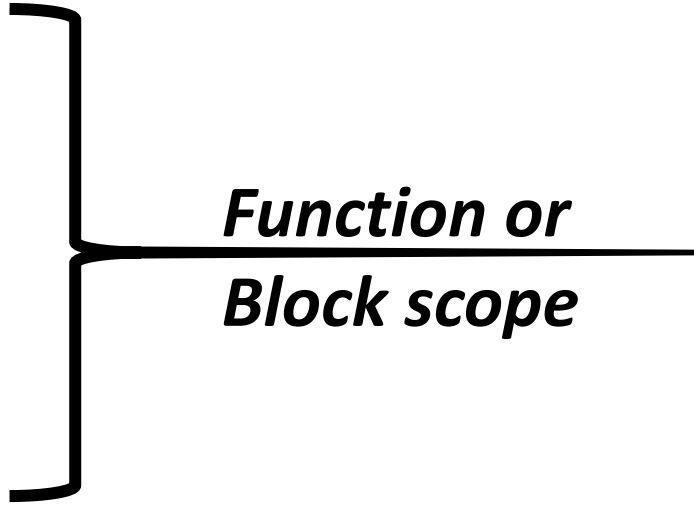
# Compiling and linking, and running

- **Either**
  - **g++ -Wall -std=c++23 hello.cpp -o hello.exe**
- **or**
  - **clang++ -Wall -std=c++2b hello.cpp -o hello.exe**
- **Then**
  - **./hello.exe**
- **Or cl /Wall /std:c++latest /EHsc hello.cpp**
  - **From a developer prompt**
  - **Then hello.exe**
- (Or build then run in your IDE)



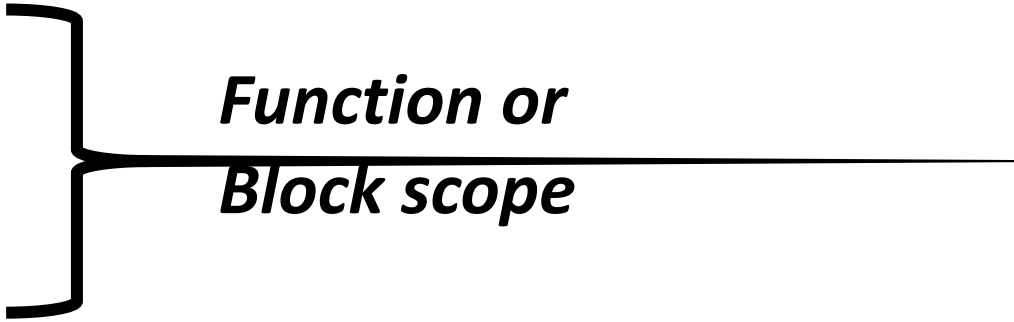
# Details

```
#include <iostream> //<-standard library

int main() //<- main  entry point, returns 0 by default
{
    std::cout << "Hello, world!\n";
    //^-  namespace scope (name then two colons)
    // '\n' is the new line character
}
```



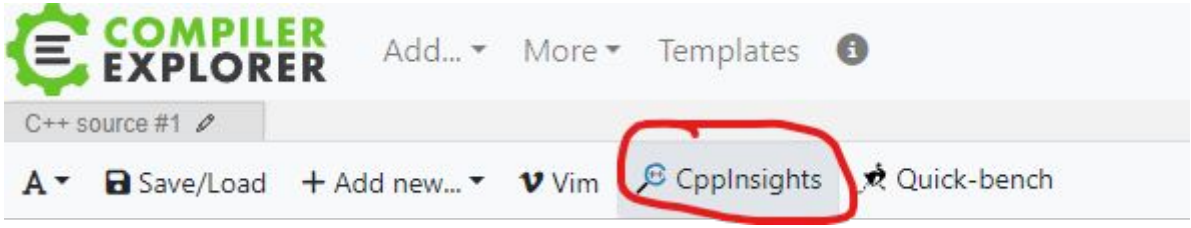
# Functions

```
//declaration
return_type function_name(params);

//definition
return_type function_name(params) //<- head
{
    // statements;
}
```



# What is << and how does it work?

- CppInsights is your friend
  - https://cppinsights.io/s/e1b67fed




```
std::cout << "Hello, world!\n";
->
std::operator<<(std::cout, "Hello, world!\n");
```

# Define `std::operator<<`

- https://en.cppreference.com/w/cpp/string/basic_string/operator_ltltgtgt
- **`stream&  operator<<(stream& os, const string& str);`**
- Streams: `std::basic_ostream<CharT, Traits>`
  - `e.g. std::cout`
- **`whatever<Type>`** indicates a template
  - Like generics but more powerful

# And the other punctuation?

- `const int x = 0;`
  - Promises we won't change the variable
- `T function_name(T & t);`
  - **By reference**
  - **To something a t exists already**
- **`T function_name(T t);`**
  - **By value**
  - **Makes a copy of t**

# We have a program BUT globals are evil

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!\n";
}
```

Streams: `std::basic_ostream<CharT, Traits>`
- e.g. `std::cout`
- `std::ostringstream`

# Testing, testish

```
#include <cassert> // for assert
#include <sstream> // for ostringstream
#include <string> // for string
void show(std::ostream & os, const std::string & words) // no copy!
{
    os << words;
}
void tests()
{
    std::ostringstream oss;
    show(oss, "Hello, world!");
    assert(oss.str() == "Hello, world!");
}
```

- Copy into a cpp file
- call from main
- Save, build and run

```
#include <iostream>

//... Show and tests

int main()
{
    tests();
    show(std::cout, "Hello, world!\n");
}
```

# Recap

- Include headers (or Import modules – still a work in progress)
- Functions:
  - return_type function_name(params)
  - Body in a block in {}
  - Giving block scope
- Namespace scope
- Operator<< for std::cout and other output streams

# Part 2

- Second, some standard stuff, including templates
  - Input
  - Validation and exceptions
  - Options

# Input

- << goes out, so what comes in?
- >>
- std::cout went out. What do we get stuff in from?
- **std::cin** (also in <iostream>)

# Variables

- const int x = 0;
- int y{};
- **type name;**
  - **“unsafe” – not initialized and explodes if you try to read it**
- **type name{};**
  - **safe**

# Try some input

```
int x{};
std::cin >> x;
```

```
#include <string>
…
std::string input{};
std::cin >> input;
```

- Copy into a cpp file
- Experiment in main
- Save, build and run

# Try some more

Write a function

```
int input(std::istream& is);
```

which makes this test pass:

```
std::istringstream iss{ "42" };
int value = input(iss);
assert(value == 42);
```

```
int input(std::istream& is)
{
    int number{};
    is >> number;
    return number;
}
```

# Mission: get numeric input

What should

```
std::istringstream iss_invalid{ "Not a number" };
```

do?

# Is it broken?

```
int choice{};
std::cin >> choice;
if (std::cin)
{
  // All good
}
```

# Exceptions

```
#include <stdexcept>
int input(std::istream& is)
{
    int number{};
    is >> number;
    if (!is)
    {
        is.clear(); // clear the fail flag (should mop up too)
        throw std::exception(); //or something else
    }
    return number;
}
```

# Catch – add to your tests

```
std::istringstream iss_invalid{ "Not a number" };
try
{
  input(iss);
  assert(false);
}
catch (const std::exception & )
{

}
```

# A function maybe returning the input

Lives in #include <optional>

```
template< class T >
    class optional;
```

# Maybe

```
std::optional<int> input{}; // or {42}
bool ok = input.has_value();
auto i = input.value();
// or
if(input) {}
```

```
#include <iostream>
#include <optional>

std::optional<int> user_choice(std::istream & is)
{
    int number{};
  // Try to add details
}
```

```
#include <limits>

std::optional<int> user_choice(std::istream & is)
{
    int number{};
    is >> number;
    if (is)
        return number;
    is.clear();
    is.ignore(
        std::numeric_limits<std::streamsize>::max(),
        '\n'); // mop up too
    return {};
}
```

# Pause

- Variables
- Input
- Functions
- Templates
- Clearing input streams
- Throwing and catching exceptions
- Optional

# Part 3

- Third, a game
  - Rock, paper, scissors
  - Randomness
  - Enums and arrays

# Restrict input to 0, 1, or 2

```
std::optional<int> zero_one_or_two(std::istream& is)
```

- Write a function with this signature
- And maybe some tests

```
#include <limits>
std::optional<int> zero_one_or_two(std::istream& is)
{
    int number{};
    is >> number;
    if (is && 0<= number && number<3)
        return number;
    is.clear();
    is.ignore(std::numeric_limits<std::streamsize>::max(),
        '\n');
    return {};
}
```

# How do you make random numbers in C++

- #include <random>
- Engine (maybe seeded)
  - std::default_random_engine gen{seed};
- (Usually) a good seed is
  - std::random_device{}() ;
- Distribution
  - std::uniform_int_distribution dist{ 0,2 };
- Distribution(engine) gives a number
  - dist(gen);

```
#include <random>
void game()
{
    std::default_random_engine gen{ std::random_device{}()};
    std::uniform_int_distribution dist{ 0, 2 };

    while (auto choice = zero_one_or_two(std::cin))
    {
        auto computer_choice = dist(gen);

        // Rock, paper or scissors?
        // Who won?
    }
}
//See https://godbolt.org/z/Gb48b3G65
```

- Copy into a cpp file
- call from main
- We'll add the two details

# Numeric choice to Rock/Paper/Scissors

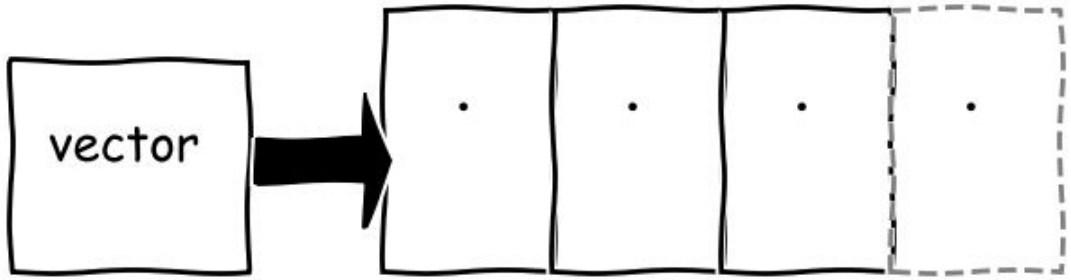
Strongly typed enums

```
enum class Choice
{
    Rock,
    Paper,
    Scissors,
};
```

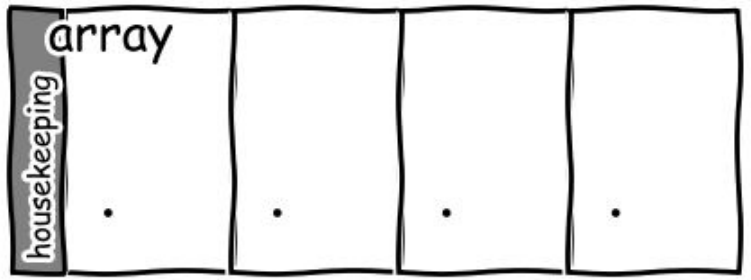
- Copy into you cpp file
- Show the computer choice

# Vectors (can resize) and arrays (fixed size)

**std::vector v{1};**
// or
**std::vector<int> v;**
**v[0] = 1;**
**auto thing = v[0]** //1st element




**std::array a{1};**
// or
**std::array<int, 1> a;**
**a[0] = 1;**
// then
**auto thing = a[0]** // 1st element



# No reflection

```
#include <array>
std::ostream& operator<<(std::ostream & os,
    Choice choice)
{
    std::array choice_str{"Rock", "Paper",
            "Scissors"};
    os << choice_str[std::to_underlying(choice)];
    return os;
}
```

# Show choices

```
void game()
{
    std::default_random_engine gen{ std::random_device{}()};
    std::uniform_int_distribution dist{ 0, 2 };

    while (auto choice = zero_one_or_two(std::cin))
    {
        auto computer_choice = dist(gen);
        auto human_choice = static_cast<Choice>(choice.value());

        // Show choices
        // Rock, paper or scissors? //  -- Add this
    }
}
```

# Who won (of two players)

```
enum class Result {
    Draw,
    FirstWins,
    SecondWins,
};

Result outcome(Choice first, Choice second);
```

# Testing, testing

```
assert(outcome(Choice::Rock, Choice::Rock) ==
        Result::Draw);

// etc…
```

```
Result outcome(Choice first, Choice second)
{
    if (first == second)
        return Result::Draw;
    else if ((first==Choice::Rock && second==Choice::Scissors)
        || (first==Choice::Paper && second==Choice::Rock)
        || (first==Choice::Scissors && second==Choice::Paper))
        return Result::FirstWins;
    return Result::SecondWins;
}
```

# Play the game

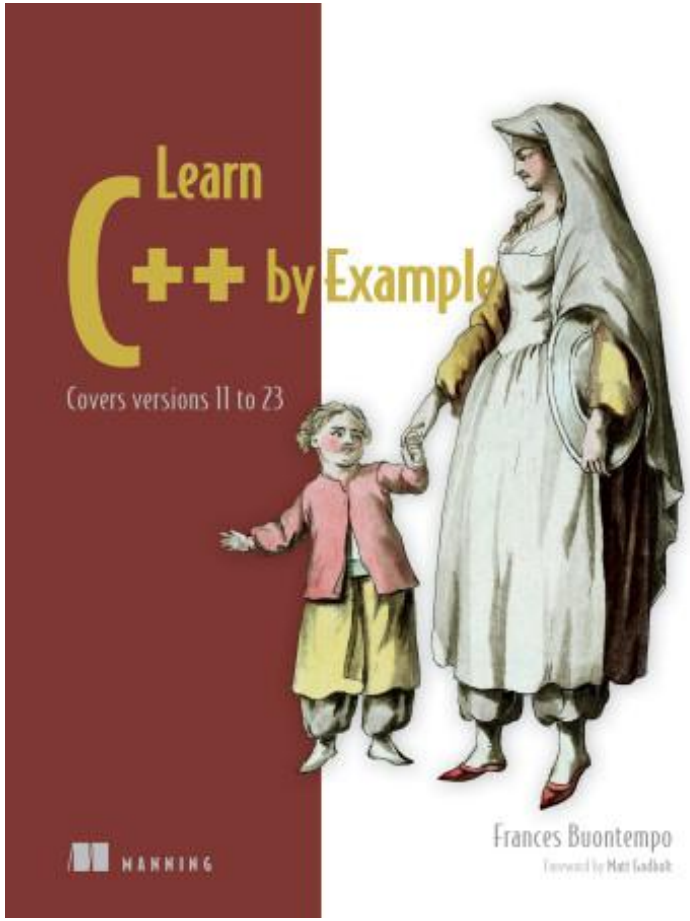
```
void game();

// call from main
int main()
{
    std::cout <<
        "Rock (0), paper (1) or scissors (2)?\n";
    game();
}
```

```
void game() {
    std::default_random_engine gen{ std::random_device{}() };
    std::uniform_int_distribution dist{ 0, 2 };

    while (auto input = zero_one_or_two(std::cin))     {
        auto human_choice = static_cast<Choice>(input.value());
        auto computer_choice = static_cast<Choice>(dist(gen));
        std::cout << human_choice << " v. " << computer_choice << ": ";

        auto result = outcome(computer_choice, human_choice);
        if (result == Result::FirstWins)
            std::cout << "computer wins\n";
        else if (result == Result::SecondWins)
            std::cout << "human wins\n";
        else
            std::cout << "Draw\n";
    }
}
```

# Pause/relax

- Arrays, fixed size
- Enum class
- Wrote our own std::ostream& operator<<
- Used random numbers
  - Two parts
  - (Maybe) seeded engine and a distribution
- Keep a tally in an array<int, 3> of human choices
  - Computer picks most likely
- How about a sliding window for the most likely?

# Time for questions/discussion




http://mng.bz/2KXw
45% off all Manning

Code: **buontemposct24**
(expires 15th Oct 2024)