Live Coding

1) Pointers
2) C-style arrays
3) New and delete
4) Vector

Sequence Containers

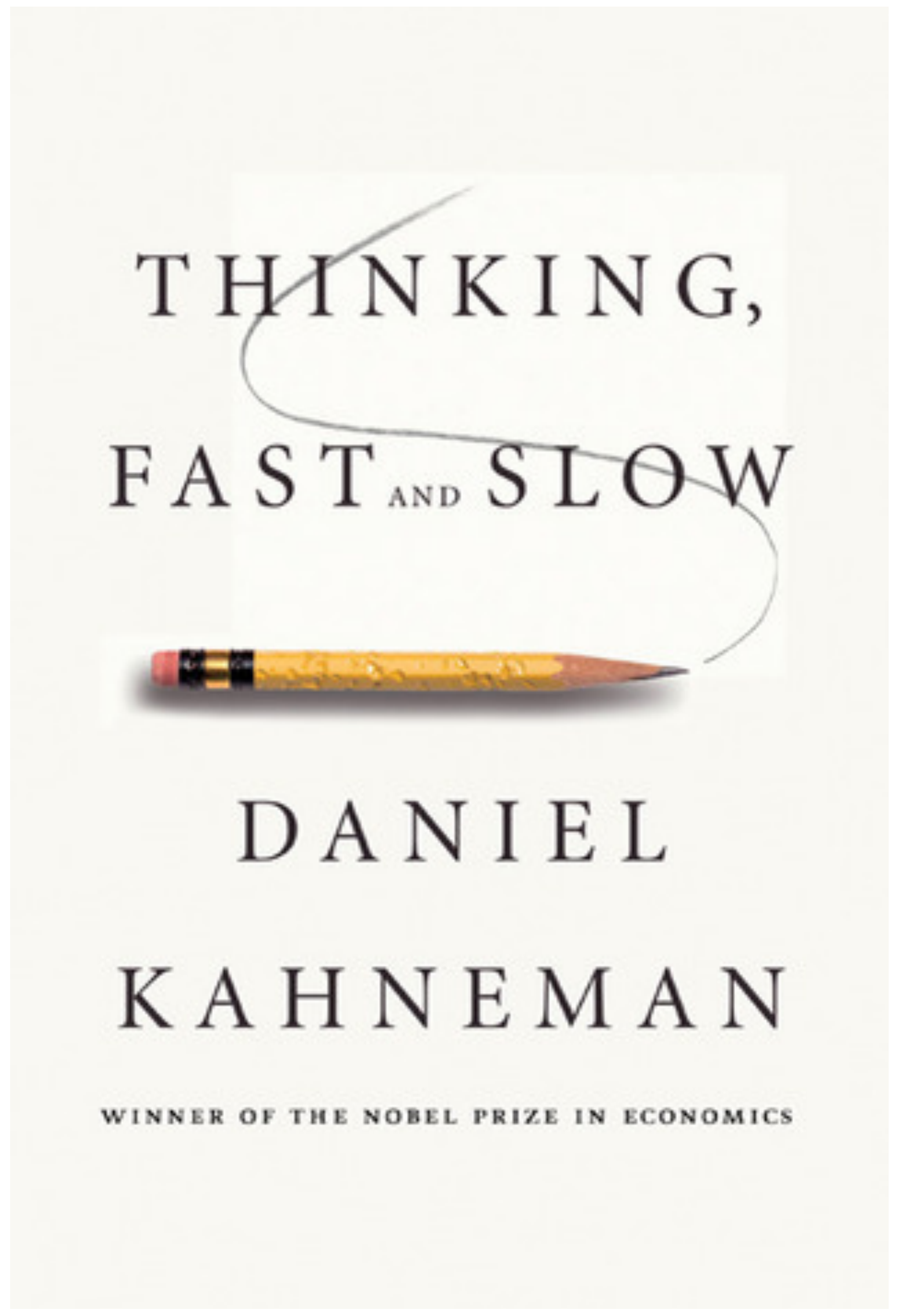
Sequence Containers

A sequence container holds an ordered collection of values of the same type.

In the recorded lectures we introduced several sequence containers:

Abstract Data Type	**C++ data structure**
Fixed size array	C-style array and std::array
Resizable array	std::vector
Linked list	std::list (doubly linked) and std::forward_list (singly linked)

Fast and Slow



Before big Oh, there is fast and slow.

An operation is fast if it takes constant time.

An operation is slow if it can take time proportional to the number of elements in the container.

Bonus: An operation is fast* if it takes amortized constant time.

Doing the operation k times takes time at most a constant times k.

Sometimes the operation is slow, but the average time per operation is fast.

Jeopardy

std::vector	std::list
$100	$100
$200	$200
$300	$300
$400	$400

Fast or Slow

	std::vector	std::list	std::deque
push_back	fast*	fast	fast*
push_front	slow	fast	fast*
pop_back	fast	fast	fast*
pop_front	slow	fast	fast*
insert in middle	slow	fast	slow
erase from middle	slow	fast	slow
get/set [i]	fast	slow	fast

Practical Summary

Abstract Data Type	**C++ data structure**	**Troy's comments**
Fixed size array	C-style array and std::array	use std::vector instead
Resizable array	std::vector	start here
Linked list	std::list (doubly linked) and std::forward_list (singly linked)	limited use cases: Bjarne's talk
Deque	std::deque	alternative to std::vector when need to push_front

Classes

Student Class

Godbolt

```cpp
class Student {
    std::string name {};    // default access is private
 private:                   // we can explicitly use private
    int ID {};
 public:
    // constructors
    Student() {}            // default constructor
    Student(std::string inputName) {
        name = inputName;
    }
    // we can have many public and private sections
 private:
    std::vector<int> scores;
 public:
    // getter
    std::string getName() {
        return name;
    }
};
```

Header Files

Header File

```cpp
#ifndef STUDENT_HPP
#define STUDENT_HPP

#include <string>

class Student {
 private:
  std::string name;
  int ID {};

 public:
  // constructors
  Student();
  Student(std::string, int = 0);
  // getters
  std::string getName();
  int getID();
};

#endif    // STUDENT_HPP
```

student.hpp

In large projects code is typically split into header (.hpp) files and implementation (.cpp) files.

A header file contains the declaration of member functions—the types of the parameters and return value.

Usually the definition (actual implementation) goes into a corresponding .cpp file.

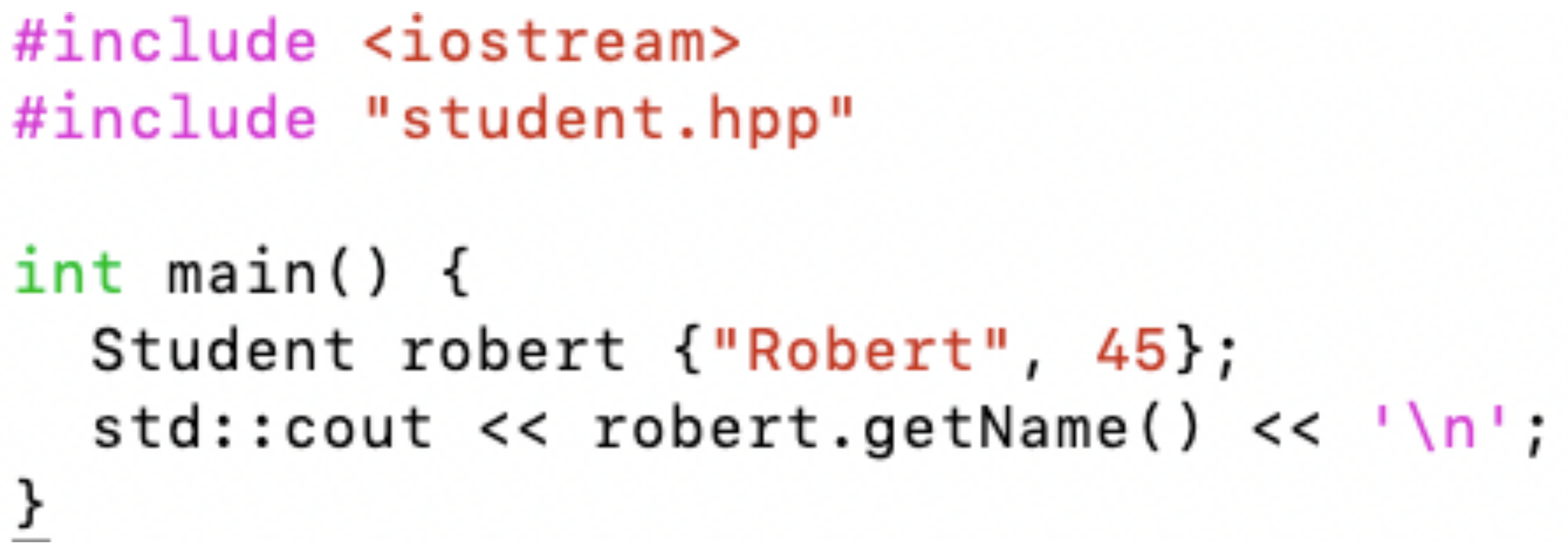
Implementation

include header →

```
#include <vector>
#include <string>
#include "student.hpp"

// Constructors
Student::Student() {}
Student::Student(std::string inputName, int inputID) :
    name {inputName}, ID {inputID} {}

// Getters
std::string Student::getName() {
      return name;
  }

int Student::getID() {
      return ID;
  }
```

student.cpp

User of Student Class



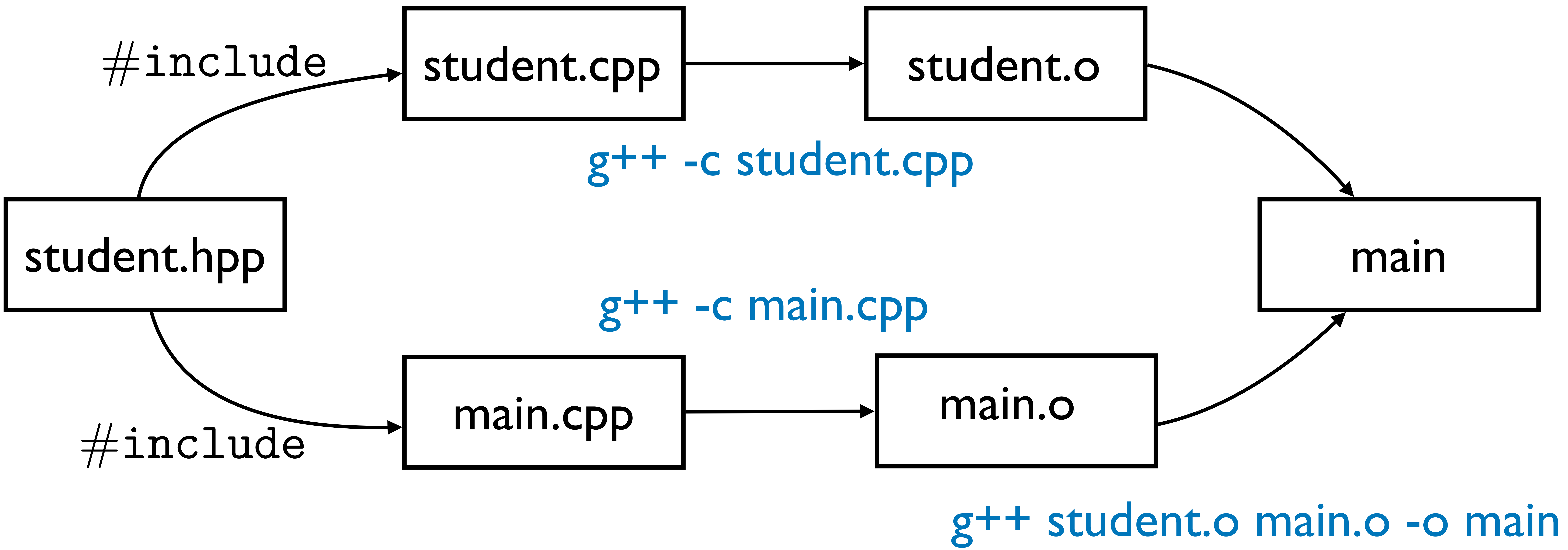
```cpp
#include <iostream>
#include "student.hpp"

int main() {
  Student robert {"Robert", 45};
  std::cout << robert.getName() << '\n';
}
```

main.cpp

With the student header file the compiler can check if this code makes sense.

This allows separate compilation—we can separately compile main.cpp and student.cpp and only later link them together.

Compilation



To compile main.cpp we just need student.hpp and the object file student.o

Header Guards

```
#include "student.hpp"
```

roster.hpp

```
#include "student.hpp"
#include "roster.hpp"
```

main.cpp

Now the student.hpp is (indirectly) included twice in main.cpp.

These results in the Student class being defined twice, an error.

We prevent this with header guards.

```
#include "student.hpp"
```

roster.hpp

```
#include "student.hpp"
#include "roster.hpp"
```

main.cpp

```
#ifndef STUDENT_HPP
#define STUDENT_HPP
```

The first time we encounter student.hpp the name STUDENT_HPP has not been defined. The second line then defines it.

The next time we encounter student.hpp, STUDENT_HPP has already been defined. The "if not defined" is false, so we skip including student.hpp again.