What Makes Good C++ Programmers:
a Continuous Search for C++ Teaching Recipes

Amir Kirsh

About me

Lecturer
Academic College of Tel-Aviv-Yaffo
and Tel-Aviv University

Developer Advocate at

INCREDIBUILD

Co-Organizer of the **CoreCpp** conference and meetup group

Member of the Israeli ISO C++ NB

Goal

- Discuss the difficulties in teaching C++
- Discuss teaching recipes

Chapters

1. Why teach C++
2. The difficulties in teaching C++
3. What should we teach
4. Class lab exercise
5. Homework exercise
6. The exam
7. Conclusions and recommendations

At the end of each chapter we would have Q&A + a short discussion

Ch.1 - Why teach C++

JOEL ON SOFTWARE

YOUR HOST

DECEMBER 29, 2005 *by* JOEL SPOLSKY

The Perils of JavaSchools

TOP 10, NEW DEVELOPER, ARTICLES

Lazy kids.

Whatever happened to hard work?

A sure sign of my descent into senility is bitchin' and moanin' about "kids these days," and how they won't or can't do anything hard any more.

When *I* was a kid, I learned to program on punched cards. If you made a mistake, you didn't have any of these modern features like a backspace key to correct it. You threw away the card and started over.

When I started interviewing programmers in 1991, I would generally let them use any

> "You were lucky. We lived for three months in a brown paper bag in a septic tank. We had to get up at six in the morning, clean the bag, eat a crust of stale bread, go to work

https://www.joelonsoftware.com/2005/12/29/the-perils-of-javaschools-2

An alternative curriculum

- First language: python
- System language: C (or Rust??)
- OOP language: Java (or C#?)

- C++ as a non-mandatory course

Why C++ should be a mandatory course?

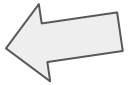
Why C++ should be a mandatory course?

- Because it is being widely used in the industry
- It would give you an advantage in the job market
  (over those students / institutions who do not do C++)
- It is harder to complete on your own

And yes, you would probably understand better how things work under the hood, but that’s a weaker argument

Why C++ should be a mandatory course?

- Because it is being widely used in the industry
- It would give you an advantage in the job market
  (over those students / institutions who do not do C++)
- It is harder to complete on your own



I raise these arguments to the students, in our first lesson

And yes, you would probably understand better how things work under the hood, but that’s a weaker argument

End of Ch.1 - Why teach C++

Q&A and Discussion

But please…

- Not about “C++ challenges” (would be discussed in Ch.2)
- Not about “what should we teach” (would be discussed in Ch.3)

Ch.2 - The difficulties in teaching C++

Ch.2 - The ~~difficulties~~ challenges in teaching C++

The first answer to a student

The first answer to a student

What is the first answer to a student asking:

“Can I do *this with that*?”

The first answer to a student

What is the first answer to a student asking:

“Can I do *this with that*?”

(i.e. “Can I access an item in a vector with square brackets?”)

The first answer to a student

Just try it!

The first answer to a student

Just try it!

Well… that’s not the best answer in C++

Trying it is not enough

Juniors tend to feel secure when code works.

Experienced C++ developers know that “working” is not a guarantee.

```
int main() {
    std::vector<int> vec = {1, 2};
    vec.push_back(3);
    vec[3] = 42; // index out of bounds, but it works!
    std::cout << vec[3] << '\n';
    std::cout << vec.capacity() << '\n';
}
```

https://ub.godbolt.org/z/EbrTb3qMb

Trying it is not enough

Juniors tend to feel secure when code works.

Experienced C++ developers know that “working” is not a guarantee.

```
int main() {
    std::vector<int> vec = {1, 2};
    vec.push_back(3);
    vec[3] = 42; // index out of bounds, but it works!
    std::cout << vec[3] << '\n';
    std::cout << vec.capacity() << '\n';
}
```

this is true for any SW language, not only for C++, but it’s much more relevant in C++ where “it’s UB” is the answer for 7.1% of questions

https://ub.godbolt.org/z/EbrTb3qMb

Trying it is not enough

Juniors tend to feel secure when code works.

Experienced C++ developers know that “working” is not a guarantee.

```
int main() {
    std::vector<int> vec = {1, 2};
    vec.push_back(3);
    vec[3] = 42; // index out of bounds, but it works!
    std::cout << vec[3] << '\n';
    std::cout << vec.capacity() << '\n';
}
```

this is true for any SW language, not only for C++, but it’s much more relevant in C++ where “it’s UB” is the answer for 7.1% of questions

don’t take this number too seriously, I just made it up

https://ub.godbolt.org/z/EbrTb3qMb

So, should or shouldn’t we say:

Just try it!

I do say:

Just try it!

I do say:

Just try it!

- if it fails, analyze why
- ~~if it works, you need to make sure it's not UB~~
- if it works, you may assume it's OK, unless it's not

I do say:

Just try it!

- if it fails, analyze why
- ~~if it works, you need to make sure it's not UB~~
- if it works, you may assume it's OK, unless it's not

there is no teacher guarantee for anything,
unless I actually see your entire code

We are still in:
Ch.2 - The challenges in teaching C++

Things change…

- New ways of doing things
- Old correct answers become obsolete or just wrong

Things change…

Even in 

you get occasionally (Good!) answers like:

- “it depends” (on the C++ version, on your compiler)
- “well I’m not sure”
- “I don’t know”

So what can we do as teachers?

What are the rules for copy elision?

Well it changed

```
class A {
public:
    A(int) {}
    A(const A&) = delete;
};

int main() {
    A a = 5; // fails in C++14, compiles in C++17
}
```

What are the rules for implicit move on return?

Well it changed

(See for example Arthur O'dwyer's talk at C++Now 2021).

What makes an aggregate type?

Well it changed - a few times…

An aggregate is one of the following types:

- array type
- class type (typically, `struct` or `union`), that has
  - no private or protected direct (since C++17) non-static data members

• no user-declared constructors	(until C++11)
• no user-provided constructors (explicitly defaulted or deleted constructors are allowed)	(since C++11) (until C++17)
• no user-provided, inherited, or explicit constructors (explicitly defaulted or deleted constructors are allowed)	(since C++17) (until C++20)
• no user-declared or inherited constructors	(since C++20)

  - no virtual, private, or protected (since C++17) base classes
  - no virtual member functions

• no default member initializers	(since C++11) (until C++14)

(From cppreference)

Things change…

Things change… - my advice

Don’t get into each and every bit.

Keep things simple.

Things change… - my advice

Don’t get into each and every bit.
Keep things simple.

Remember that “Teaching” and “Cheating” share the same letters.
Cheat a little, occasionally.

(And saying only part of the truth is even not exactly cheating).

Things change… - my advice

You don’t have to be “exact” in each and every answer or explanation.

It might be that you are not up-to-date with a specific detail, that’s fine.

Things change… - my advice

You don’t have to be “exact” in each and every answer or explanation.
It might be that you are not up-to-date with a specific detail, that’s fine.

If you are not sure, throw a little disclaimer, e.g.:

- “for any practical aspect, this should be the answer”
- “there is something more to it, but we will not dive to it”

You don’t have to be up-to-date to C++20

On the other hand!

You can’t stay too much behind.

Teaching C++98 today is not acceptable.

We are still in:
Ch.2 - The challenges in teaching C++

C++ language rules are difficult

C++ language rules are difficult - my advice

Students (most of us actually) learn much better from examples.

Every rule and methodology that I teach has at least one code example and the attention in class is given to the example.

C++ language rules are difficult - my advice

I try to personalize the process, e.g. when I show this piece of code:

```
void foo(const std::string& s);

int main() {
    foo("hello");
}
```

I’m telling that the compiler is tempted to issue compilation error, but then it recalls that if a legal casting is available it should use it.

We are still in:
Ch.2 - The challenges in teaching C++

The logic behind C++ rules may be arbitrary

The logic behind C++ rules may be arbitrary

C++98 didn’t allow direct member initialization.

There were “good” logical explanations for that.
None of them actually stands.

Arbitrary logic behind C++ rules - my advice

You may have a logical explanation, or not.
But sometimes the best explanation is: *that's what the spec says.*

Use it also in order to cut too long discussions.

We are still in:
Ch.2 - The challenges in teaching C++

Too many questions in class

Too many questions in class

Many details and high pace generates many questions that divert you from the planned materials.

Too many questions in class - my advice

Start with direct questions on presented pieces of code.

Students should point at the exact line number they ask about.

Postpone extrapolation question (“what if”) for after answering understanding questions (“what happens at line 12”, “why is there a need to do this thing at line 16”).

Take some questions offline (to the break, to a digital Q&A forum).

Summarizing:
The challenges in teaching C++ - my advice

Be careful with too many details (spare it, cheat a bit).

Stick to good code examples.

Make everyone aligned before getting to extrapolation questions.

Use modern C++!

And keep updating your materials.

End of Ch.2 - The challenges in teaching C++

Q&A and Discussion

Specifically:

- Comments on the challenges raised
- Any additional challenge that you would add
- Other ways to mitigate the challenges discussed

Ch.3 - What should we teach

My C++ Courses

Course 1 (The Academic College of Tel-Aviv-Yaffo)

- CS undergraduates, 1st semester of their 2nd year, mandatory course
- After C (used in “Intro to programming” and “Advanced Programming”)
- Course includes:
  - ➔ the basic C++ syntax
  - ➔ OOP understanding and practice
  - ➔ rvalue and move semantics - as an optional bonus
  - ➔ templates
  - ➔ lambda expressions
  - ➔ std containers and algorithms

Course 2 (Tel-Aviv University)

- CS undergraduates, 3rd year, non-compulsory course
- After Java (as their OOP lang) and C (as their “System programming lang”)
- Course includes:
  - ➔ the basic C++ syntax
  - ➔ rvalue and move semantics
  - ➔ templates
  - ➔ lambda expressions
  - ➔ std containers and algorithms
  - ➔ smart pointers
  - ➔ concurrency and multithreading

Goal: students should be able to

- Implement a complicated project in C++
  - autonomous vacuum cleaner
  - ship stowage management and algorithm
  - games: tetris, pac-man, thunderbird
- Read complicated C++ code
  - an implementation similar to std::shared_ptr
  - an implementation similar to std::any
  - an implementation similar to vector<bool>

Stop teaching C (when you teach C++)

Kate Gregory, CppCon 2015

Stop teaching C (when you teach C++)

- Well, they already started with C in my case
- I want them to understand how a string class is being implemented

- At the end I mix and match *top-to-bottom* and *bottom-up*
  - They implement their own string *before* using std::string
  - They use std containers before they learn templates

End of Ch.3 - What should we teach

- An open discussion
- Comments
- Questions

Ch.4 - Class lab exercise

Live coding with the students!

Live coding with the students!

- Handle actual compilation errors and runtime bugs
- Yes, the teacher also has bugs and compilation errors!
- Practice with the students:
  - actually reading the compilation error message
  - going to the web if needed
  - debugging

Live coding with the students!

A recent small example:

Initial version (with an embarrassing bug):

https://replit.com/@AmirKirsh/VetShop0

Fixed:

https://replit.com/@AmirKirsh/VetShop2

The smart pointers version:

https://replit.com/@AmirKirsh/VetShop3

End of Ch.4 - Class lab exercise

Any questions?

Comments?

Ch.5 - Homework exercise

An example from this semester (link to the exercise spec):

They get comments on their code

Examples:

- Code Review Comments
- Run Comments and Final Grades

For most comments we reduce points (even minor, e.g. bad names).

- Comments without point reduction tend to be ignored.

An important note that I emphasize

An important note that I emphasize

A working, though imperfect (e.g. smelly code) exercise

is better than

A perfect (great code) non-working exercise

End of Ch.5 - Homework exercise

Questions? Comments?

Ch.6 - The exam

Test - In the past

Two parts exam:

- Writing code
- Reading code

3.5 hours for both parts

Test - Writing Code

Examples:

- Implement the backend of a monopoly game
- Implement the backend of a 9*9 spreadsheet engine

Test - Reading Code

~100 lines of code program with multiple choice questions

(Questions and answers at this part are scrambled, each students gets a unique instance)

An example

Important notes:

- They get before the exam ~6 complicated programs, out of which 2 are selected for their exam, with some changes. They don’t get the questions ahead.
- Code in the exam may contain intentional bugs.

Test - In the past - Grades

- 40% failures
- Average: ~70/100

- Many complaints about the implementation part, being too hard

Things that I tried

- ➔ Dividing the two parts of the exam into separate time slots
  - 2.5 hours for the implementation part
  - 1.5 hours for the reading part
- ➔ Allowing writing code in a text editor (in a closed environment at the lab)
- ➔ Lowering my demands on the implementation part

Nothing helped in making the exam more reasonable

Today

One part exam:

- Only reading code

(They practice writing code in their exercise…)

Today - Grades

- 10% failures
- Average: ~80/100

Exam - bonus part

The exam has ~15 points bonus, on non-mandatory material.

They do quite well on the bonus part!

End of Ch.6 - The exam

Questions? Comments?

Ch.7 - Conclusions and recommendations

Most important part when teaching

Most important part when teaching

- ➔ Build their motivation
- ➔ Create sense of ability, make them succeed
- ➔ Let them enjoy the beauty of C++

Remember

➔ They are just students, adapt your expectations

- On the other hand, don’t aim too low!
- Let them work hard, but help them

➔ The C++ syntax is the tool, but you want also to establish good software engineering practices. Try to hold both

Any questions before we conclude?

2 final things before we depart…

Core C++ 2022: https://corecpp.org

Tel-Aviv, 5th to 7th of September

Registration is Open


CPP Game Show
from Incredibuild
25:00
C++
CPP GAME SHOW
Amir
Daisy
Dima
Vittorio
Kilian

Thank you!