C++ONLINE

AMIR KIRSH

TALK:

A CONSTEXPR VIRTUAL CRTP COMPARISON

2025

About me

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

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

Trainer and Advisor
(C++, but not only)

CLARITUNE
Making Things Happen

Goals

Goals

- Play with C++ Comparison
- Get into some hidden corners
- Have fun

Let the fun begin

Let the fun begin

```
CppOnline<2025> talk("A constexpr virtual CRTP comparison");
auto itr = talk.begin();
```

What should happen in the following cases?

What ~~should~~ happen in the following cases?

would

What would happen? (1)

What would happen? (1)

```
// Assume:
// Student inherits publicly from Person
// without adding any additional data member
constexpr Person p(124); // setting id
constexpr Student s(124); // setting id
static_assert(p == s); // #1
static_assert(s == p); // #2
```

What would happen? (1)

```
// Assume:
// Student inherits publicly from Person
// without adding any additional data member
constexpr Person p(124); // setting id
constexpr Student s(124); // setting id
static_assert(p == s); // #1
static_assert(s == p); // #2
```

What would happen? (1)

```
// Assume:
// Student inherits publicly from Person
// without adding any additional data member
constexpr Person p(124); // setting id
constexpr Student s(124); // setting id
static_assert(p == s); // #1
static_assert(s == p); // #2
```

A Both pass

B #1 passes
#2 fails

C Both fail

D #1 passes
#2 doesn’t compile

What would happen? (1)

```
// Assume:
// Student inherits publicly from Person
// without adding any additional data member
constexpr Person p(124); // setting id
constexpr Student s(124); // setting id
static_assert(p == s); // #1
static_assert(s == p); // #2
```

[✓ **A** Both pass]

B #1 passes
#2 fails

C Both fail

D #1 passes
#2 doesn’t compile

With C++20 spaceship operator: https://compiler-explorer.com/z/YvPv14ozG

With C++17 / C++20 user defined operator== in the base: https://compiler-explorer.com/z/7qo4GMh79

What would happen? (2)

What would happen? (2)

```
// Assume:
// Student inherits publicly from Person
// with an additional data member
constexpr Person p(124); // id
constexpr Student s(124, 9); // id, student_id
static_assert(p == s); // #1
static_assert(s == p); // #2
```

A Both pass

B #1 passes
#2 fails

C Both fail

D #1 passes
#2 doesn’t compile

What would happen? (2)

```
// Assume:
// Student inherits publicly from Person
// with an additional data member
constexpr Person p(124); // id
constexpr Student s(124, 9); // id, student_id
static_assert(p == s); // #1
static_assert(s == p); // #2
```

A Both pass C++20

B #1 passes
#2 fails

C Both fail

D #1 passes C++17
#2 doesn't compile

With C++20 spaceship operator: https://compiler-explorer.com/z/Wvr9q8dxG

With C++17 / C++20 user defined operator==: https://compiler-explorer.com/z/8a6YxK6W6

What would happen? (3)

What would happen? (3)

```
// Assume:
// Student inherits publicly from Person
// with an additional data member
Person p(124); // id
Student s(124, 9); // id, student_id
Person* pPerson = &s;
assert(*pPerson == p); // #1
assert(p == *pPerson); // #2
```

A `Both pass`

B **#1 passes**
#2 fails

C `Both fail`

D **#1 passes**
#2 doesn't compile

What would happen? (3)

Both in C++17 and C++20

```
// Assume:
// Student inherits publicly from Person
// with an additional data member
Person p(124); // id
Student s(124, 9); // id, student_id
Person* pPerson = &s;
assert(*pPerson == p); // #1
assert(p == *pPerson); // #2
```

[✓ **A** **Both pass**]

B **#1 passes**
#2 fails

C **Both fail**

D **#1 passes**
#2 doesn’t compile

With C++20 spaceship operator: https://compiler-explorer.com/z/aacfsvqWd

With C++17 / C++20 user defined operator==: https://compiler-explorer.com/z/odTTxbnTa

What would happen? (4)

What would happen? (4)

```
// Assume:
// Student inherits publicly from Person
// with an additional data member
Student s1(124, 101);
Student s2(124, 102);
Person* ptrP = &s2;
assert(*ptrP == s1); // #1
assert(s1 == *ptrP); // #2
```

A Both pass

B #1 passes
#2 fails

C Both fail

D #1 passes
#2 doesn’t compile

What would happen? (4)

```
// Assume:
// Student inherits publicly from Person
// with an additional data member
Student s1(124, 101);
Student s2(124, 102);
Person* ptrP = &s2;
assert(*ptrP == s1); // #1
assert(s1 == *ptrP); // #2
```

A Both pass C++20

B #1 passes
#2 fails

C Both fail

D #1 passes C++17
#2 doesn't compile

With C++20 spaceship operator: https://compiler-explorer.com/z/Y8cezWYnb

With C++17 / C++20 user defined operator==: https://compiler-explorer.com/z/EorfKK6bM

Comparison between Polymorphic Types in C++ (1)

`Student == Person`
`Person == Student`

Comparison between Polymorphic Types in C++ (1)

	Should
`Student == Person`	Not compile If compiles should return ***false***
`Person == Student`	return ***false***

Comparison between Polymorphic Types in C++ (1)

	Should	Actual, C++17	Actual, C++20
`Student == Person`	Not compile If compiles should return ***false***	Doesn’t compile ✅	Compiles ❌ Calls Person::== May return ***true***
`Person == Student`	return ***false***	Compiles ❌ Calls Person::== May return ***true***	Compiles ❌ Calls Person::== May return ***true***

Comparison between Polymorphic Types in C++ (2)

pPerson -> Person Student == *pPerson
pPerson -> Person *pPerson == Student

Comparison between Polymorphic Types in C++ (2)

	Should

`pPerson -> Person`
`Student == *pPerson`
	Not compile
	If compiles should return ***false***

`pPerson -> Person`
`*pPerson == Student`
	return ***false***

Comparison between Polymorphic Types in C++ (2)

	Should	Actual, C++17	Actual, C++20

`pPerson -> Person`
`Student == *pPerson`
	Not compile
	If compiles should return ***false***
		Doesn’t compile ✅	Compiles ❌
			Calls Person::==
			May return ***true***

`pPerson -> Person`
`*pPerson == Student`
	return ***false***	Compiles ❌
		Calls Person::==
		May return ***true***
			Compiles ❌
			Calls Person::==
			May return ***true***

* quite similar to (1) above - same exact issue

Comparison between Polymorphic Types in C++ (3)

pPerson -> Student Student == *pPerson
pPerson -> Student *pPerson == Student

Comparison between Polymorphic Types in C++ (3)

	Should

`pPerson -> Student`
`Student == *pPerson`
	Call Student::==

`pPerson -> Student`
`*pPerson == Student`
	Call Student::==

Comparison between Polymorphic Types in C++ (3)

	Should	**Actual, C++17**	**Actual, C++20**

`pPerson -> Student`
`Student == *pPerson`
	Call Student::==	Doesn’t compile ❌	Calls Person::== ❌
			May return ***true*** *when false*

`pPerson -> Student`
`*pPerson == Student`
	Call Student::==	Calls Person::==
		May return ***true*** *when false* ❌
			Calls Person::==
			May return ***true*** *when false* ❌

Comparison between Polymorphic Types in C++ is utterly broken :(

Comparison between Polymorphic Types in C++ is utterly broken :(

An important note – the problem appears even if you’re keeping the rule of abstract base class (non-leaf classes should be abstract) – the problem will pop **when comparing siblings**:

```
constexpr BAStudent s1(124, 101);
constexpr MAStudent s2(124, 102);
const Student* ptrS = &s1;
assert(*ptrS == s2); // passes and should fail
assert(s2 == *ptrS); // passes and should fail
```

https://compiler-explorer.com/z/GoKEbThMY

Comparison between Polymorphic Types in C++ is utterly broken :(

- Non symmetric (before C++20)
- Two different types may be considered equal
- Default comparison is non polymorphic: pointer to base will call base `operator==` even when pointing to derived

Comparison between Polymorphic Types in C++ is utterly broken :(

What should we do?

Comparison between Polymorphic Types in C++ is utterly broken :(

What should we do?

avoid comparing different types!
i.e. do not use == on objects of different types
beware when using == with references and pointers
(can it be enforced? how can we catch bugs?)

Comparison between Polymorphic Types in C++ is utterly broken :(

Can we implement something better?

Comparison between Polymorphic Types in C++ is utterly broken :(

Can we implement something better?

maybe…

Comparison between Polymorphic Types in C++ is utterly broken :(

Can we implement something better?

maybe…

But hey! It’s not only for…

- It’s not only a C++ problem, same thing may happen in (choose your lang)
  e.g.: Java Example

Not exactly the same as in C++
Yes, it’s easy to accidentally make it wrong,
but this example actually works fine.
(Or use lombok’s @EqualsAndHashCode).

- It’s not only related to comparison.
  Similar issue also with assignment:
  https://compiler-explorer.com/z/3Es474eYe

  * and in fact it may happen:

  - with any ***non-virtual*** function call on a pointer to base which actually points to derived
  - with any function taking an argument a pointer or ref to base, and assuming it’s a base object

Back to comparison

Implementing a virtual comparison

Start from here:

- ➔ https://compiler-explorer.com/z/3Y5vEEd9v

Implementing a virtual comparison (1)

Start from here:

- ➔ https://compiler-explorer.com/z/3Y5vEEd9v

Let’s try to make the operator itself virtual…

things do not go that well: https://compiler-explorer.com/z/7hz6e9oj9

Implementing a virtual comparison (2)

Start from here:

➔ https://compiler-explorer.com/z/3Y5vEEd9v

Let’s try it with a virtual helper function:

1st attempt: https://compiler-explorer.com/z/9W1hafhGn - still some failures

2nd attempt: https://compiler-explorer.com/z/xo5EY6d7P - works!

And, with CRTP: https://compiler-explorer.com/z/4jn36cxTz

C++20 version, typeid is still not constexpr, implementing our own RTTI:

https://compiler-explorer.com/z/TPxhehhTs

It may go further

Supporting virtual comparison with multiple bases, including multiple *virtual* bases:

https://coliru.stacked-crooked.com/a/9c63b53b1cfb28f3

With bugs opened on gcc and clang … :

https://gcc.gnu.org/bugzilla/show_bug.cgi?id=117317

https://github.com/llvm/llvm-project/issues/113801#issue-2616516632

There must be another way…

(https://www.youtube.com/watch?v=bBTQFOkFZw8)

There must be another way (1)

A global templated function (e.g. equal) that checks type and delegates to operator== (or to a member function equal)

Still needs virtual operator== (or, a virtual equal member function)
(think why…)

Option a: https://compiler-explorer.com/z/c8csxbjKM
Option b: https://compiler-explorer.com/z/oqM4aq8rc

There must be another way (2)

Make non-leaf classes non-comparable, e.g. with protected ==
expose a public == only in leaf classes

https://compiler-explorer.com/z/Kr4rK5G71

There must be another way (3)

Do something with C++26 reflection???

Summarizing

Comparison between Polymorphic Types in C++ is utterly broken

Summary (1)

➔ **If you avoid using comparison between different types, you are fine!**

- but, are you sure you avoid it?
- (comparison may be called by infra code, that is not yours)

➔ **There is an option for polymorphic comparison, but it is not available out-of-the-box. You have to implement it yourself.**

(We played with that in this talk. It’s not necessarily the best solution).

➔ **If you want to make it constexpr, you can do so starting with C++20,** even though it depends on virtual functions and RTTI.
You can use custom RTTI, or if you're in C++23, typeid is constexpr.

➔ **CRTP can assist in removing some code duplications.**

Summary (2)

Other Options (Better, Maybe):

- ➔ A global templated function (e.g. equal) that checks type and delegates to a virtual member equal
- ➔ Make non-leaf classes non-comparable, e.g. with protected == expose a public == only in leaf classes

Additional Links

What's the right way to overload operator== for a class hierarchy? - StackOverflow

Implementing operator== when using inheritance - StackOverflow

Any questions before we conclude?

Photo by Howie R on Unsplash

Amir Kirsh
kirshamir@gmail.com

Appendix - Solutions for Assignment

The problem: https://coliru.stacked-crooked.com/a/bc05e6aa062a8cad

virtual default operator= doesn't work:
https://coliru.stacked-crooked.com/a/43d869c5c5aa08f6

Solution 1 - block it:
https://coliru.stacked-crooked.com/a/152b040c12249ec0

Solution 2 - virtual implementation:
https://coliru.stacked-crooked.com/a/a929d1f54158e385