Six Ways for Implementing Math Expressions Calculator

AMIR KIRSH

Cppcon
The C++ Conference

2023

October 01 - 06

About me

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

Developer Advocate at
INCREDIBUILD

Member of the Israeli ISO C++ NB

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

INCREDIBUILD

Suffering from slow builds?

It’s not just waste of time

It affects your dev cycles
and productivity

Incredibuild Happy Customers (partial list)

Cerence

Minitab

THE COALITION

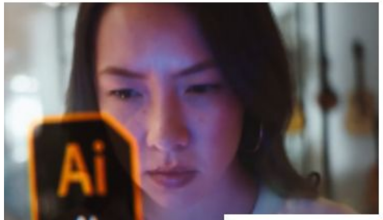
The Coalition Transforms Azure VMs into a 700-Core Incredibuild "Virtual Supercomputer", Releases 2 AAA...

VR Group

Accelerate building of terrain tiles from elevation and imagery data for large terrain surfaces rendered for use in...

Proletariat

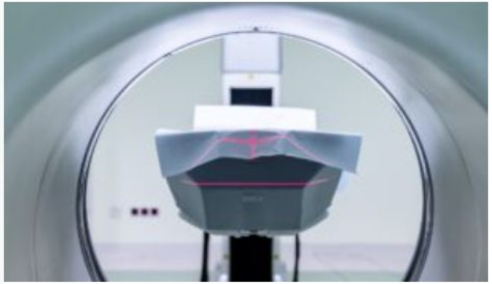
How the award-winning indie game studio managed to cut PS4 full cook by half while working on AWS Cloud wit...



Adobe

Vizendo

Vizendo's 4 developers prove they can make a change by turning 15 daily hours into 45 minutes with Incredibuil...



ALGOTEC

AlgoTec implements continuous integration and expands automated testing for medical imaging technolog...

Movavi

Movavi revolutionized its build and testing times from 80 to 20 minutes with just 12 workstations and 2 build...

GeoTeric

Retalix, an NCR Company

Retalix speeds up thousands of unit and integration tests accelerating continuous integration cycle times by...

Riverblade

Accelerating PC-lint C++ code analysis to complete the static analysis of a Visual Studio solution in a fraction of...

id Software

Cellebrite

Cellebrite dramatically accelerates build time and packaging processes, reducing over-all build process by 70%

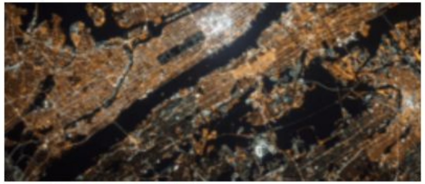
Epic Games

Accelerating the build process for Unreal Engine the driving technology behind many of today's leading video...

CompuGroup Medical

Sarine Technologies

Embedding Incredibuild in Advisor diamond analysis software to achieve superior results and offer enhanced...



LOGIBALL

LOGIBALL uses Incredibuild to significantly reduce Android build time

We also accelerate Yocto builds!

Our recent talks at Yocto Project Summit:

https://bit.ly/YPS-2022_IB_bitbake

https://bit.ly/YPS-2022_IB_Cache

Incredibuild + Yocto:

https://www.incredibuild.com/blog/announcing-incredibuild-support-for-yocto
https://www.incredibuild.com/lp/yocto

Incredibuild for Automotive

Relevant Sub-Sectors:

- Infotainment
- Instrument cluster
- Heads-up-display (HUD)
- Telematics/connected car
- Advanced driver assistance systems (ADAS)
- Functional safety and autonomous driving

Jaguar Land Rover, Nissan, Toyota, DENSO Corporation, Fujitsu, HARMAN, NVIDIA, Renesas, Samsung

AUTOMOTIVE
GRADE LINUX

146 members

Relevant Linux OS's / distribution collaborations:

Yocto, QNX, AOSP, Bazel, AGL

Cppcon
The C++ Conference
2023
October 01-06
Aurora, Colorado, USA

Six ways for Implementing Math Expressions Calculator

A walk through polymorphism, smart pointers, templates, concepts and more

Amir Kirsh

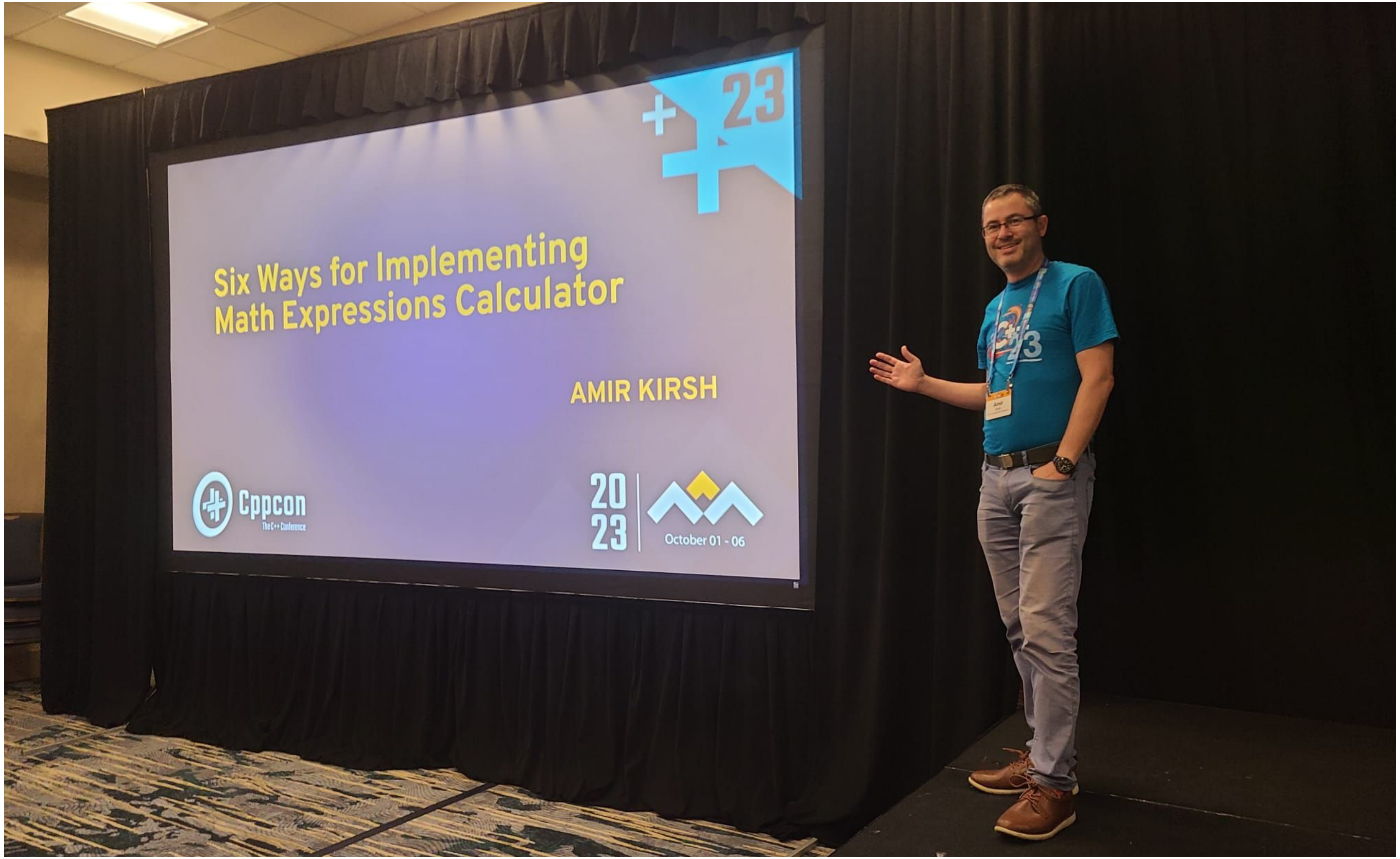
NOTE:
It is NOT

2022
Six Ways for Implementing Max:
A Walk Through Api Design, Dangling
References and C++20 Constraints
Amir Kirsh
I C++

Picture by @ciura_victor : https://ciura.ro/blog/cpponsea2022/sessions.html

Youtube link: https://www.youtube.com/watch?v=yRnYIBJq6sM

It’s this one


23
Six Ways for Implementing Math Expressions Calculator
AMIR KIRSH
Cppcon
The C++ Conference
20
23
October 01 - 06

The initial challenge

The initial challenge

```
// we want the following code:
auto e = new Sum (
        new Exp(new Number(3), new Number(2)),
        new Number(-1)
    );
cout << *e << " = " << e->eval() << endl;
delete e;

// to print something like:
// ((3 ^ 2) + (-1)) = 8
```

A quick polymorphism exercise

Let’s start here:

http://coliru.stacked-crooked.com/a/192d90699cd08eb5

(Or just skip to this:)

http://coliru.stacked-crooked.com/a/0387ba22e796fc7a

But..., do you like it?

```
// what is bothering you with the code below
auto e = new Sum (
        new Exp(new Number(3), new Number(2)),
        new Number(-1)
    );
cout << *e << " = " << e->eval() << endl;
delete e;
```

But..., do you like it?

```
// what is bothering you with the code below
auto e = new Sum (
        new Exp(new Number(3), new Number(2)),
        new Number(-1)
    );
cout << *e << " = " << e->eval() << endl;
delete e;
```

Non-symmetric calls to new and delete

But..., do you like it?

```
// what is bothering you with the code below
auto e = new Sum (
        new Exp(new Number(3), new Number(2)),
        new Number(-1)
    );
cout << *e << " = " << e->eval() << endl;
delete e;
```

Non-symmetric calls to new and delete

Why do we use new and delete to begin with?
Shouldn’t we use smart pointers? 🤔

First improvement attempt: unique_ptr

Let’s try it together…

Live Coding!

Let’s try it together…

Live Coding!

Starting from here:

http://coliru.stacked-crooked.com/a/0387ba22e796fc7a

(Or just skip to this:)

http://coliru.stacked-crooked.com/a/8f16e80c8a3351ca

But..., do you like it?

```
// what is bothering you with the code below
auto e = make_unique<Sum>(
            make_unique<Exp>(
                    make_unique<Number>(3),
                    make_unique<Number>(2)
            ),
            make_unique<Number>(-1)
        );
cout << *e << " = " << e->eval() << endl;
```

Let’s try to hide the unique_ptr

We aim for something like this

```
auto e = Sum(Exp(Number(3), Number(2)), Number(-1));
cout << e << " = " << e.eval() << endl;
```

Why is it better?

```
auto e = Sum(Exp(Number(3), Number(2)), Number(-1));
cout << e << " = " << e.eval() << endl;

// what makes the code above better? compared to:

auto e = make_unique<Sum>(
    make_unique<Exp>(make_unique<Number>(3), make_unique<Number>(2)),
    make_unique<Number>(-1)
);
cout << *e << " = " << e->eval() << endl;
```

Hiding your implementation details!

The user doesn’t have to know we use unique_ptr

We may want to change it later

It’s a “private implementation detail”

And it’s noisy

Hiding your implementation details:
Let’s try it together…

Live Coding!

Starting from here:

http://coliru.stacked-crooked.com/a/270aab96c2972490

(Or just skip to this:)

http://coliru.stacked-crooked.com/a/35f49a5a014224f8

unique_ptr or shared_ptr?

Can we support this with unique_ptr?

```cpp
int main() {
    auto e = Sum(Exp(Number(3), Number(2)), Number(-1));
    cout << e << " = " << e.eval() << endl;
    // passing rvalues
    auto num1 = Number(-1);
    auto e2 = Sum(std::move(e), std::move(num1));
    cout << e2 << " = " << e2.eval() << endl;
    // passing lvalues
    auto num2 = Number(-1);
    auto e3 = Sum(e2, num2);
    cout << e3 << " = " << e3.eval() << endl;
}
```

Can we support this with unique_ptr?

```
int main() {
    auto e = Sum(Exp(Number(3), Number(2)), Number(-1));
    cout << e << " = " << e.eval() << endl;
    // passing rvalues
    auto num1 = Number(-1);
    auto e2 = Sum(std::move(e), std::move(num1));
    cout << e2 << " = " << e2.eval() << endl;
    // passing lvalues
    auto num2 = Number(-1);
    auto e3 = Sum(e2, num2);
    cout << e3 << " = " << e3.eval() << endl;
}
```

The problem is here…

See code

Can we support this ^ with unique_ptr?

Yes! By implementing a clone operation (with CRTP!):

http://coliru.stacked-crooked.com/a/390b7ac654e4ccd5

Is there any difference if we use shared_ptr?

Compare the behavior with shared_ptr (without clone):

http://coliru.stacked-crooked.com/a/01c8a1c64831b962

Getting rid of some additional noise…

Getting rid of some additional noise…

Can we simplify the below?

```cpp
auto e = Sum(Exp(Number(3), Number(2)), Number(-1));
```

Getting rid of some additional noise…

Can we simplify the below?

```
auto e = Sum(Exp(Number(3), Number(2)), Number(-1));
```

Sure! Why not just:

```
auto e = Sum(Exp(3, 2), -1);
```

Removing the need to explicitly create Numbers!

Attempt #1:

http://coliru.stacked-crooked.com/a/6ae72598edb5cc8c

What’s wrong?

How can we fix it?

Removing the need to explicitly create Numbers!

Attempt #2:

http://coliru.stacked-crooked.com/a/4ed4be79cea4300d

Narrowing a greedy constructor by restricting its template arguments!

Using C++20 concepts!

Removing the need to explicitly create Numbers!

Attempt #3:

http://coliru.stacked-crooked.com/a/79f7ced32e51fe98

No need to handle all different combinations of Expression and number!

Removing the need to explicitly create Numbers!

Attempt #3:

http://coliru.stacked-crooked.com/a/79f7ced32e51fe98

No need to handle all different combinations of Expression and number!

🎯 **Design Pros:**
Simple usage, short and concise, hiding implementation details, polymorphism with value semantics

Do we need derived classes for Sum and Exp?

What do you say about something like:

```
template<typename Op>
class BinaryExpression: public Expression {
    unique_ptr<Expression> e1, e2;
public:
    // ...constructors...
    void print(ostream& out) const override {
        Op::print(out, *e1, *e2);
    }
    double eval() const override {
        return Op::eval(*e1, *e2);
    }
};
```

Why is it better?

Why is it better?

Reduce coupling (*“inheritance is the base class of evil”*)

A step towards eliminating the need for polymorphism!

Let’s see the code…

Implementation without deriving Sum and Exp

http://coliru.stacked-crooked.com/a/3527deb15496d43d

Implementation without deriving Sum and Exp

http://coliru.stacked-crooked.com/a/3527deb15496d43d

Design Pros:
Simple usage, short and concise, hiding implementation details, polymorphism with value semantics, **narrowing the inheritance to the minimum required**

Can we implement it without polymorphism?

What do you say about something like:

```cpp
template<typename Op, typename Expression1, typename Expression2>
class BinaryExpression {
    Expression1 e1;
    Expression2 e2;
public:
    // ...
};

int main() {
    auto e1 = Sum(Exp(3, 2), -1);
    cout << e1 << " = " << e1.eval() << endl;
}
```

Why is it better?

Why is it better?

No need for virtual functions => static polymorphism

(Is it actually better? not necessarily…)

Shorter (and nicer?) code.

No need for allocations!

And… we may even evaluate expressions at compile time!

Let’s see the code…

http://coliru.stacked-crooked.com/a/5060e5f189135362

What else can we add??

Variadic Templates!

```
constexpr auto e1 = Sum(4.5, Exp(Sum(1, 2), 2), -1);
cout << e1 << " = " << e1.eval() << endl;
```

Variadic Templates Version

http://coliru.stacked-crooked.com/a/65cfadef28d39607

Variadic Templates Version

http://coliru.stacked-crooked.com/a/65cfadef28d39607

Design Pros:
Simple usage, short and concise,
hiding implementation details,
static polymorphism
no inheritance required

Summary

Summary (1)

C++ is a multi-paradigm programming language!

Summary (2)

Pointers / References are not mandatory for Polymorphism!

- Prefer using smart pointers for internal members
- Try to hide your smart pointers, exposing **value type semantics** (Treat the use of smart pointers as an implementation detail!)

Summary (3)

When using unique_ptr you can still copy!

- Implementing clone for your types can allow copying

Summary (4)

Using shared_ptr to hold immutable data is an easy way to achieve Copy-on-Write

- Avoiding the need to copy, without worrying about data race or modifications done while the object is being in use
- It can be viewed as an implementation of the flyweight design pattern.

Summary (5)

Static Polymorphism may replace Dynamic Polymorphism

- Consider this option when relevant
- Use it with care, generic programming may require safety nets to avoid abuse or obscure compilation errors (use static_assert and SFINAE or concepts to restrict usage)
- Don’t avoid using templates because of longer compilation time, there are solutions for that :-)

A Credit Note

Thanks

… to Arthur O’Dwyer, for sharing valuable comments on a previous version of this presentation!

… to CppBayArea meetup group and to Haifa::C++ meetup group participants, for a fruitful discussion of the code snippets.

Any questions before we conclude?

Thank you!

```
void conclude(auto&& greetings) {
    while(still_time() && have_questions()) {
        ask();
    }
    greetings();
}

conclude([]{ std::cout << "Thank you!"; });


// Comments, feedback: kirshamir@gmail.com
// let me help you accelerate you builds: amir.kirsh@incredibuild.com
```