Six ways for Implementing Math Expressions Calculator

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

Amir Kirsh

About me

Lecturer
Academic College of Tel-Aviv-Yaffo
Visiting lecturer Stony Brook 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

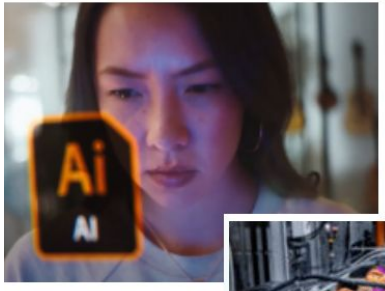
The Coalition Transforms Azure VMs into a 700-Core Incredibuild "Virtual Supercomputer", Releases 2 AAA...

VR Group

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

Proletariat

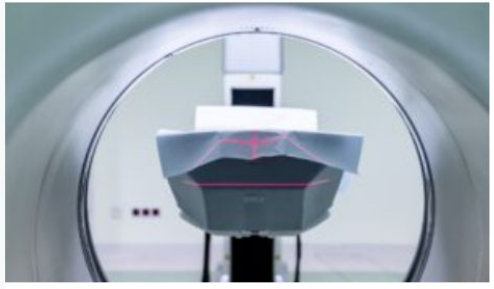
How the award-winning indie game studio managed to cut PS4 full cook by half while working on AWS Cloud wit...



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...

Cellebrite

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

CompuGroup Medical

Epic Games

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

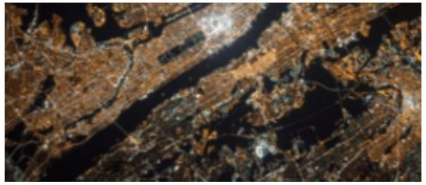
Sarine Technologies

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

Riverblade

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

id Software



LOGIBALL

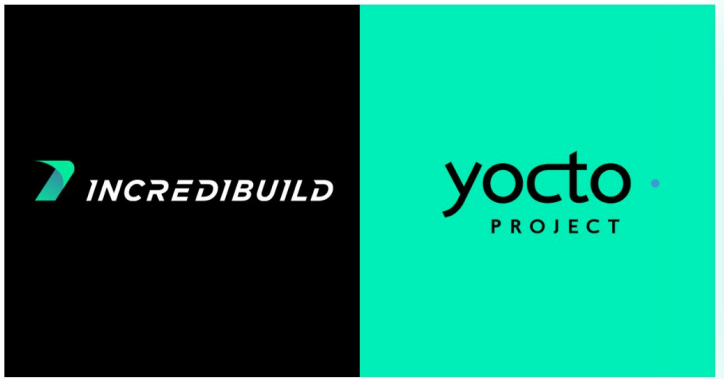
LOGIBALL uses Incredibuild to significantly reduce Android build time

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

146 members

Relevant Linux OS's / distribution collaborations:

Yocto, QNX, AOSP, Bazel, AGL

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;
```

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

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;
```

Let’s try it together…

Live Coding!

Starting from here:
http://coliru.stacked-crooked.com/a/1f45731ffa822752

(Or just skip to this:)
http://coliru.stacked-crooked.com/a/0ea552dabe10f81f

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?

Reduces coupling (?)

A step towards eliminating the need for polymorphism!

Let’s see the code…

http://coliru.stacked-crooked.com/a/e548a26330177341

unique_ptr or shared_ptr?

Can we support this with unique_ptr?

```
int main() {
    auto e = sum(expo(3, 2), -1);
    cout << e << " = " << e.eval() << endl;
    auto e2 = sum(e, 2);
    cout << e2 << " = " << e2.eval() << endl;
    auto e3 = sum(std::move(e), 2);
    cout << e3 << " = " << e3.eval() << endl;
}
```

Can we support this ^ with unique_ptr?

Yes! By implementing a clone operation:

http://coliru.stacked-crooked.com/a/d1a7a91ce3236070

Is there any difference if we use shared_ptr?

Yes! Compare the behavior with shared_ptr (without clone):

http://coliru.stacked-crooked.com/a/7d2e9a955a173d20

Can we implement it without polymorphism?

What do you say about something like:

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

int main() {
    auto e1 = sum(expo(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…)

Let’s see the code…

http://coliru.stacked-crooked.com/a/a8481a79149bf147

Adding back class Number for fancy printing

Version 1: with class number and type deduction guides:
http://coliru.stacked-crooked.com/a/852ecbded966d154

Version 2: with class number and ‘to_expression’ converter:
http://coliru.stacked-crooked.com/a/33953f1fd72d6238

What else can we add??

Variadic Templates!

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

Variadic Templates Version

http://coliru.stacked-crooked.com/a/46f436cdf85f6003

Summary

Summary (1)

C++ is a multi-paradigm programming language

Summary (2)

Pointers / References are not mandatory for Polymorphism

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
```