C++ Meetup @ OCI

OCI | WE ARE SOFTWARE ENGINEERS.

# New People?


PRESENTERS
WANTED

# 2018 C++ Conferences

- C++Now is.... NOW!!!!!
  - Aspen, Colorado - May 6 - 11– cppnow.org
- CPPCon
  - Bellevue, Washington - Sept 23 - 29 – cppcon.org
- Pacific++
  - Sydney, Australia - Oct 18 - 19 – pacificplusplus.com
- Meeting C++
  - Berlin, Germany - Nov 15 - 17 – meetingcpp.com

# ISO C++ Committee Meetings

- Upcoming Committee Meetings:
  - June 4-9 in Rapperswil, Switzerland
  - November 5-10 in San Diego, CA

# Template Metaprogramming

- Wikipedia: *"Template metaprogramming (TMP) is a metaprogramming technique in which templates are used by a compiler to generate temporary source code, which is merged by the compiler with the rest of the source code and then compiled. The output of these templates include compile-time constants, data structures, and complete functions. <u>The use of templates can be thought of as compile-time execution.</u> "*

# Factorial

The classic example...

# Factorial at Runtime

```
#include <iostream>

unsigned int factorial(unsigned int n)
{
    return n == 0 ? 1 : n * factorial(n - 1);
}

int main()
{
    std::cout << "Factorial 5 is " << factorial(5) << std::endl;
}
```

# Factorial Calculated at Compile Time

```
#include <iostream>

template <unsigned int n>
struct factorial
{
    enum { value = n * factorial<n - 1>::value };
};

template <>
struct factorial<0>
{
    enum { value = 1 };
};

int main()
{
    std::cout << "Factorial 5 is " << factorial<5>::value << std::endl;
}
```

# Non-Enum Version

```cpp
#include <iostream>

template <unsigned int n>
struct factorial
{
    static const int value = n * factorial<n - 1>::value ;
};

template <>
struct factorial<0>
{
    static const int value = 1 ;
};

int main()
{
    std::cout << "Factorial 5 is " << factorial<5>::value << std::endl;
}
```

# Array Example

```
#include <iostream>

void PrintArray(int (&array)[5])
{
    for(auto i : array)
    {
        std::cout << i << std::endl;
    }
}

int main()
{
    int myIntArray[] = {1, 2, 3, 4, 5};
    PrintArray(myIntArray);
}
```

# Array Size at Compile Time

```
#include <iostream>

template<typename T, std::size_t N>
std::size_t ArraySize(T (&)[N])
{
    return N;
}

int main()
{
    int myIntArray[] = {1, 2, 3, 4, 5};
    std::cout << "myIntArray size: " << ArraySize(myIntArray) << std::endl;
}
```

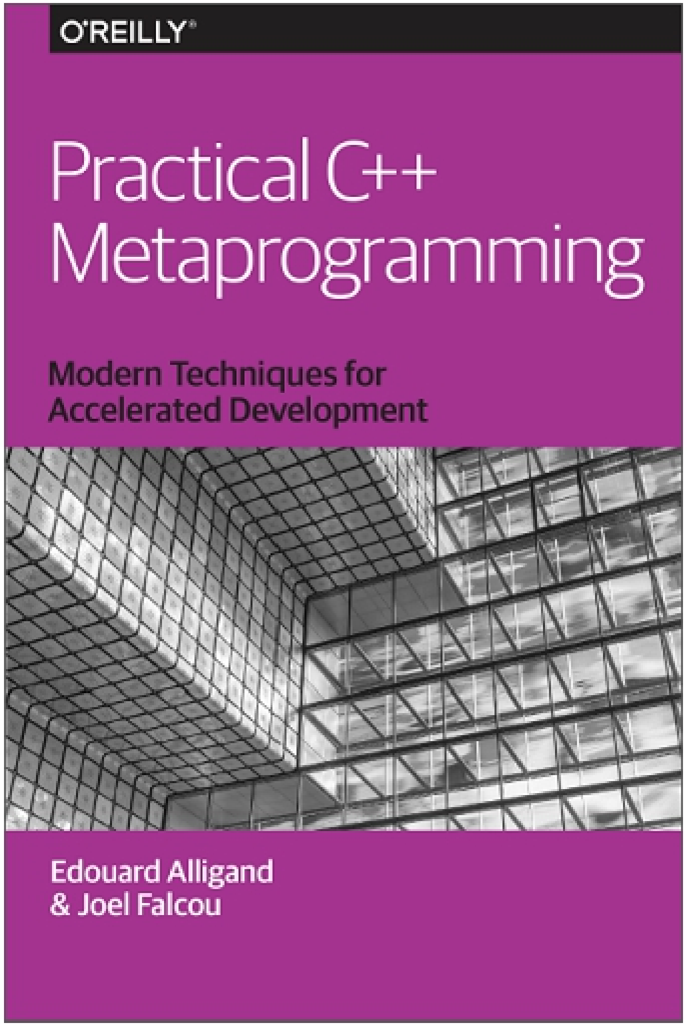
# Prime Number Check at Runtime

```cpp
#include <iostream>

bool DoPrimeCheck(int number, int divisor)
{
    if(divisor == 2)
    {
        return number % 2;
    }
    return DoPrimeCheck(number, divisor / 2);
}

bool IsPrime(int number)
{
    return DoPrimeCheck(number, number / 2);
}

int main()
{
    if(IsPrime(13))
    {
        std::cout << "Is prime." << std::endl;
    }
    else
    {
        std::cout << "Is not prime." << std::endl;
    }
}
```

# Prime Number Check at Compile Time

```
#include <iostream>

template<unsigned p, unsigned d>
struct DoIsPrime
{
    static const bool value = (p % d != 0) && DoIsPrime<p, d - 1>::value;
};

template<unsigned p>
struct DoIsPrime<p, 2>
{
    static const bool value = (p % 2 != 0);
};

template<unsigned p>
struct IsPrime
{
    static const bool value = DoIsPrime<p, p / 2>::value;
};

int main()
{
    if(IsPrime<13>::value)
    {
        std::cout << "Is prime." << std::endl;
    }
    else
    {
        std::cout << "Is not prime." << std::endl;
    }
}
```

# More Practical Application?





https://www.oreilly.com/programming/free/practical-c-plus-plus-metaprogramming.csp

# Chapter 2 Begins...

*"Let's imagine that you are responsible for the construction—from the ground up—of a brand new module in a big weather prediction system. Your task is to take care of the distribution of complex computations on a large computing grid, while another team has the responsibility for the actual computation algorithms (in a library created two decades previously)".*

# When Codebases Collide

Example of a public function from the existing (two decade old) weather simulation C API:

```
// alpha and beta are parameters to the mathematical
// model underlying the weather simulation algorithms
void adjust_values(double* alpha1, double* beta1, double* alpha2, double* beta2);
```

Example of a class from our distrubted system:

```
// A class from our distributed computation system that
// accesses the alpha and beta values.
class reading
{
public:
    double alpha_value(location l, time t) const;
    double beta_value(location l, time t) const;
};
```

# Mapping Between Class and API

```
void adjust_values(double* alpha1, double* beta1, double* alpha2, double* beta2);


class reading

{
public:
    double alpha_value(location l, time t) const;
    double beta_value(location l, time t) const;
};
```

For a given location *l*, and time values *t1* and *t2*, we want to use:

- reading::alpha_value(*l, t1*) for the alpha1 param
- reading::beta_value(*l, t1*) for the beta1 param
- reading::alpha_value(*l, t2*) for the alpha2 param
- reading::beta_value(*l, t2*) for the beta2 param

# First Attempt

One way to interface our class with the weather simulator:

```
std::tuple<double, double, double, double>
get_adjusted_values(const reading& r, location l, time t1, time t2)
{
    double alpha1 = r.alpha_value(l, t1);
    double beta1 = r.beta_value(l, t1);
    double alpha2 = r.alpha_value(l, t2);
    double beta2 = r.beta_value(l, t2);
    adjust_values(&alpha1, &beta1, &alpha2, &beta2);
    return std::make_tuple(alpha1, beta1, alpha2, beta2);
}
```

# Problem

What if the C API has dozens of adjust_value functions we need to interace with?

Do we really want to write a get_adjusted_values intermediary function for each one???

Probably not.

# A Solution

The solution the book ultimately comes to...