# The Power of C++

Joseph Mansfield
josephmansfield.uk
@sftrabbit

Thanks to our sponsors


# What makes C++ so badass?

# What makes C++ so badass?

- Is it because it's close to the metal?

# What makes C++ so badass?

- Is it because it's close to the metal?
- Is it the high-performance?

# What makes C++ so badass?

- Is it because it's close to the metal?
- Is it the high-performance?
- Is it the complete control?

# What makes C++ so badass?

- Is it because it's close to the metal?
- Is it the high-performance?
- Is it the complete control?
- What about the standard library?

# What makes C++ so badass?

- Is it because it's close to the metal?
- Is it the high-performance?
- Is it the complete control?
- What about the standard library?
- I know! It must be the declaration syntax...

```
int (*(*foo[5])(char(*)()))[3]
```

C++

# C with Classes

making better use of the compiler.

```
class Foo
{
  public:
    Foo() : x(10)
    { }

    int get_double() const
    {
      return x * 2;
    }
  private:
    int x;
};
```

# C with Classes

making better use of the compiler.

```
class Foo
{
  public:
    Foo() : x(10)
    { }

    int get_double() const
    {
      return x * 2;
    }
  private:
    int x;
};
```

```
typedef struct Foo
{
  int x;
} Foo;

void init(Foo* this)
{
  this->x = 10;
}

int get_double(const Foo* this)
{
  return this->x * 2;
}
```

# Templates

give us compile-time generated code.

```
template <typename T>
T add(T x, T y)
{
  return x + y;
}
```

# Templates

give us compile-time generated code.

```
template <typename T>
T add(T x, T y)
{
  return x + y;
}
```

```
int add_i(int x, int y)
{
  return x + y;
}

char add_c(char x, char y)
{
  return x + y;
}

std::complex add_c(std::complex x,
                   std::complex y)
{
  return x + y;
}

// ...
```

# Constant expressions

are evaluated at compile-time.

```
constexpr int factorial(int n)
{
    return n <= 1 ?
           1 : (n * factorial(n-1));
}

int main()
{
  std::cout << factorial(10)
            << std::endl;
}
```

# Constant expressions

are evaluated at compile-time.

```
constexpr int factorial(int n)
{
    return n <= 1 ?
           1 : (n * factorial(n-1));
}

int main()
{
  std::cout << factorial(10)
            << std::endl;
}
```

```
int main()
{
  std::cout << 3628800
            << std::endl;
}
```

# Coming soon: concepts!

compile-time polymorphism.

```
template <class InputIt, class OutputIt>
OutputIt copy(InputIt first,
              InputIt last,
              OutputIt d_first);
```

# Coming soon: concepts!

compile-time polymorphism.

```
OutputIterator copy(InputIterator first,
                    InputIterator last,
                    OutputIterator d_first);
```

```
template <class InputIt, class OutputIt>
OutputIt copy(InputIt first,
              InputIt last,
              OutputIt d_first);
```

High-performance

Close to the metal

Complete control

Standard library

High-performance

Close to the metal

# Zero-cost abstractions

are what make C++ badass.

Complete control

Standard library

# Compile-time Snake!

Hey, why not?

`github.com/mattbierner/STT-C-Compile-Time-Snake`

Questions?