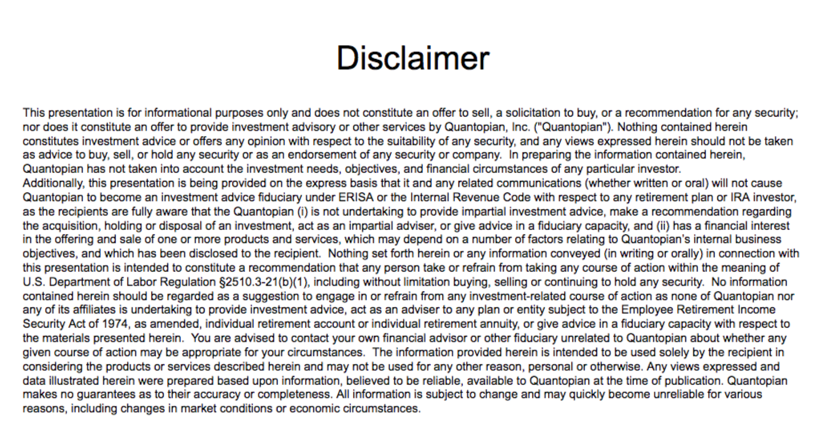
Template Metaprogramming and `<type_traits>`

Joe Jevnik

March 31, 2018

Boston C++



Disclaimer

This presentation is for informational purposes only and does not constitute an offer to sell, a solicitation to buy, or a recommendation for any security; nor does it constitute an offer to provide investment advisory or other services by Quantopian, Inc. ("Quantopian"). Nothing contained herein constitutes investment advice or offers any opinion with respect to the suitability of any security, and any views expressed herein should not be taken as advice to buy, sell, or hold any security or as an endorsement of any security or company. In preparing the information contained herein, Quantopian has not taken into account the investment needs, objectives, and financial circumstances of any particular investor.

Additionally, this presentation is being provided on the express basis that it and any related communications (whether written or oral) will not cause Quantopian to become an investment advice fiduciary under ERISA or the Internal Revenue Code with respect to any retirement plan or IRA investor, as the recipients are fully aware that the Quantopian (i) is not undertaking to provide impartial investment advice, make a recommendation regarding the acquisition, holding or disposal of an investment, act as an impartial adviser, or give advice in a fiduciary capacity, and (ii) has a financial interest in the offering and sale of one or more products and services, which may depend on a number of factors relating to Quantopian's internal business objectives, and which has been disclosed to the recipient. Nothing set forth herein or any information conveyed (in writing or orally) in connection with this presentation is intended to constitute a recommendation that any person take or refrain from taking any course of action within the meaning of U.S. Department of Labor Regulation §2510.3-21(b)(1), including without limitation buying, selling or continuing to hold any security. No information contained herein should be regarded as a suggestion to engage in or refrain from any investment-related course of action as none of Quantopian nor any of its affiliates is undertaking to provide investment advice, act as an adviser to any plan or entity subject to the Employee Retirement Income Security Act of 1974, as amended, individual retirement account or individual retirement annuity, or give advice in a fiduciary capacity with respect to the materials presented herein. You are advised to contact your own financial advisor or other fiduciary unrelated to Quantopian about whether any given course of action may be appropriate for your circumstances. The information provided herein is intended to be used solely by the recipient in considering the products or services described herein and may not be used for any other reason, personal or otherwise. Any views expressed and data illustrated herein were prepared based upon information, believed to be reliable, available to Quantopian at the time of publication. Quantopian makes no guarantees as to their accuracy or completeness. All information is subject to change and may quickly become unreliable for various reasons, including changes in market conditions or economic circumstances.

Outline

1. Template types
2. Template functions
3. Advanced template concepts
4. Computation model of templates
5. Overview of `<type_traits>`
6. Selected examples

Template Types

Template Types

```
class vec2d_int {
private:
    const int m_x;
    const int m_y;

public:
    vec2d_int(int, int);
    int x() const;
    int y() const;
    int dot(const vec2d_int&) const;
};
```

Template Types

```
class vec2d_float {
private:
    const float m_x;
    const float m_y;

public:
    vec2d_float(float, float);
    float x() const;
    float y() const;
    float dot(const vec2d_float&) const;
};
```

Template Types

```
class vec2d_double {
private:
    const double m_x;
    const double m_y;

public:
    vec2d_double(double, double);
    double x() const;
    double y() const;
    double dot(const vec2d_double&) const;
};
```

Template Types

```
int vec2d_int::dot() {
    return m_x * other.m_x + m_y * other.m_y;
}
```

Template Types

```
int vec2d_int::dot() {
    return m_x * other.m_x + m_y * other.m_y;
}

float vec2d_float::dot() {
    return m_x * other.m_x + m_y * other.m_y;
}
```

Template Types

```
int vec2d_int::dot() {
    return m_x * other.m_x + m_y * other.m_y;
}

float vec2d_float::dot() {
    return m_x * other.m_x + m_y * other.m_y;
}

double vec2d_double::dot() {
    return m_x * other.m_x + m_y * other.m_y;
}
```

Template Types

```
template<typename T>
class vec2d {
private:
    const T m_x;
    const T m_y;

public:
    vec2d(const T&, const T&);
    const T& x() const;
    const T& y() const;
    T dot(const vec2d&) const;
};
```

Template Types

```cpp
template<typename T>
T vec2d<T>::dot(const vec2d<T>& other) const {
    return m_x * other.m_x + m_y * other.m_y;
}
```

Template Types

```
vec2d<int> int_vector(2, 4);
vec2d<float> float_vector(2.5, 4.5);
vec2d<double> double_vector(2.5, 4.5);
```

Template Types

```
struct not_a_numeric_type {};

vec2d<not_a_numeric_type> v(not_a_numeric_type{},
                            not_a_numeric_type{});
not_a_numeric_type dotted = v.dot(v);
```

Template Types

```
scratch.cc: In instantiation of
    T vec2d<T>::dot(const vec2d<T>&) const
    [with T = not_a_numeric_type]:
scratch.cc:31:40:   required from here
scratch.cc:21:20: error: no match for operator*
    (operand types are
     const not_a_numeric_type and
     const not_a_numeric_type)
         return m_x * other.m_x + m_y * other.m_y;
                ~~~~~~~~~~~~~~~
```

Template Types

```
class z5 {
private:
    std::uint8_t m_value;

public:
    z5(int);
    z5 operator+(const z5&) const;
    z5 operator*(const z5&) const;
};

vec2d<z5> v(1, 2);
z5 dotted = v.dot(v);
```

Value Parameters

```
template<typename T>
class vec3d {
private:
    const T m_x;
    const T m_y;
    const T m_z;

public:
    vec3d(const T&, const T&, const T&);
    const T& x() const;
    const T& y() const;
    const T& z() const;
    vec3d dot(const vec3d& other) const;
};
```

Value Parameters

```
template<typename T>
class vec4d {
private:
    const T m_axis_0;
    const T m_axis_1;
    const T m_axis_2;
    const T m_axis_3;

public:
    vec4d(const T&, const T&, const T&, const T&);
    const T& axis_0() const;
    // ...
    vec4d dot(const vec4d& other) const;
};
```

Value Parameters

```cpp
#include <array>

template<typename T, std::size_t size>
class vecnd {
private:
    std::array<T, size> m_data;

public:
    vecnd(const std::array<T, size>&>);
    const T& operator[](std::size_t) const;
    T dot(const vecnd&) const;
};
```

Value Parameters

```
template<typename T, std::size_t size>
T vecnd<T, size>::dot(const vecnd<T, size>& other) const {
    T sum = 0;
    for (std::size_t ix = 0; ix < size; ++ix) {
        sum += (*this)[ix] * other[ix];
    }
    return sum;
}
```

Value Parameters

```
vecnd<int, 4> v({1, 2, 3, 4});
```

Thank You

Questions?

JoeJev
Performance:5,757pp
Accuracy:98.38%
Lv100
#5892

`github.com/llllllllll` **(10 lowercase L’s)**

- `/lain` (model implementation)
- `/slider` (tools for working with osu! data and API)
- `/combine` (irc server running lain-as-a-service)