

Enhanced typename

Tian Liao

November 25, 2023

Document number:
Date: November 25, 2023
Audience:
Authors: Tian Liao, Mingxin Wang
Reply-to: Tian Liao <tilia@microsoft.com>

1 Introduction

2 Proposal

A quick glance:

```
// declare some materials.
struct Color { void apply() {} };
struct Texture { void apply() {} };
struct Glass { void apply() {} };

// define a type alias.
typename Material { void apply(); };

void foo() {
    {
        // use Material as a pointer.
        Color color;
        Material* material = color;
        material->apply();
    }
    {
        // use Material as a reference.
        Texture texture;
        Material& material = texture;
        material.apply();
    }
    {
        // host a Material in unique ptr.
        std::unique_ptr<Material> material{new Glass()};
        material->apply();
    }
}

// defining a type alias is not allowed.
void Material::apply() {} // compile error.

// instantiate a type alias is also not allowed.
Material some_material; // compile error.
```

Type alias can also combine with other type aliases to form a new type alias.

```
typename Gettable{ void get(); };
typename Settable{ void set(); };

typename GetSet : Gettable, Settable {};
typename GetSetEquivalent {
    void get();
    void set();
}
```

```
};
static_assert(std::is_same_v<GetSet, GetSetEquivalent>);
```

Type alias can also have specific constraints to control its copiability, relocatability, etc.

```
typename NoCopyNoMove {
    NoCopyNoMove() = default;
    NoCopyNoMove(const NoCopyNoMove&) = delete;
    NoCopyNoMove(NoCopyNoMove&&) = delete;
};

void foo(NoCopyNoMove& a, NoCopyNoMove& b) {
    a = b; // compile error.
    a = std::move(b); // compile error.
}
```

Function overloads.

```
typename Addition {
    void operator>()() const;
    int operator()(int , int) const;
    float operator()(float, float) const;
};

void foo(const Addition& add) {
    add();
    add(1, 2);
    add(0.1f, 0.2f);
}
```

Type alias can also have templates.

```
template <typename T, std::size_t I>
typename GenericMaterial {
    using type = T;
    static constexpr std::size_t index = I;
    void apply(const T& target);
};
```

3 Motivation