Язык C++

Мещерин Илья

Лекция №21

Compile-time вычисления

1) Compile-time *if*, *for*

```
void do_smth(){}

template<bool b>
void f(){}

template<>
void f<true>(){ do_smth(); }

int main(){
        const int x = 5;
        f<x * x == 25>();
}
```

Таким трюком еще на стадии компиляции определится какая из веток должна быть скомпилирована и запущена.

```
template<int n>
struct Fibonacci{
        static const int value = Fibonacci<n - 1>::value + Fibonacci<n - 2>::value;
};

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

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

int main(){
        cout << Fibonacci<10>::value;
}
```

Таким способом можно сделать имитацию цикла на этапе компиляции.

2) *Constexpr*

```
constexpr int x = 5;
```

Значение переменной типа *constexpr* будет вычислено еще на этапе компиляции. Т.е. она является константой уже на этапе компиляции.

В присваивании выражение справа также должно быть типа *constexpr*.

```
constexpr int fib(int n){
    return n <= 1 ? 1 : fib(n - 1) + fin(n - 2);
}

template <typename T>
auto get_value(T t) {
    if constexpr (std::is_pointer_v<T>)
        return *t; // deduces return type to int for T = int*
    else
        return t;  // deduces return type to int for T = int
}
```

3) *Type_traits*

```
if (std::is_const<decltype(x)>::value) {}
```

Таким способом можем проверить, является ли переменная константой.

```
template<class U, class V>
struct is_same{
    static const bool value = false;
};

template<class T>
struct is_same{
    static const bool value = true;
};

if (std::is_same<decltype(x), decltype(y)>::value) {}
```

Так можно проверять правда ли, что переменные одного типа.

```
struct true_type{
    static const bool value = true;
}

template<class U, class V>
struct is_same : public std::false_type {};

template<class T>
struct is_same : public std::true_type {};
```

Или то же самое через наследование.

```
template<class U, class V>
struct conjunction{
    static const bool value = U::value && V::value;
};
```

Можно написать для переменного числа параметров.

```
template<bool b, class U, class V>
struct conditional{
    typedef V type;
};

template<class U, class V>
struct conditional<false, U, V>{
    typedef U type;
};
```

Можно ставить условия на выбор типа.

```
template<class T>
struct rank{
    static const size_t value = 0;
};

template<class T>
struct rank<T[]>{
    static const size_t value = rank<T>::value + 1;
};
```

Так можно узнать размерность массива.