Программирование на языке C++

Вводный курс

Александр Морозов
gelu.speculum@gmail.com

ИТМО, весенний семестр 2020

Содержание

Шаблоны и параметры шаблонов

`template` < *список шаблонных параметров* > *объявление*

Параметры шаблонов:

- типы `class Id` или `typename Id`
- шаблоны `template <...> class Id` или `template <...> typename Id`
- значения `type Id` или `auto Id`

Шаблонный идентификатор – *имя шаблона* `< params list >`.

Пример шаблонов

```
1    template <class T>
2    class Queue { ... }; // template
3
4    Queue<int> q_int; // template-id
5
6    template <std::size_t Size>
7    class StackString { ... }; // template
8
9    void f(const StackString<10> & str); // template-id
```

Параметры шаблонов со значением по умолчанию

```
1    template <class = int>
2    struct X {};
3
4    template <std::size_t Size = 10>
5    void f();
6
7    template <template <class> class T = X>
8    struct Y {};
```

Переменное число шаблонных параметров

```
 1    template <class... Args, class T = char>
 2    void g(Args &&... args);
 3
 4    template <class... Args>
 5    void f(Args &&... args)
 6    {
 7        g<Args...>(std::forward<Args>(args)...);
 8    }
 9    f();
10    f(1, 0.5);
11
12    template <bool... Flags>
13    struct S {};
14    S<> zero;
15    S<false> one;
16    S<false, true> two;
17
18    template <template <class...> class... Templates>
19    class C {};
```

Раскрытие переменного параметра шаблона

нечто ...

нечто должно включать в себя хотя бы один переменный параметр. Идентификатор переменного параметра должен встречаться там, где допустимо использовать идентификатор.

нечто ... →*нечто1*, *нечто2*, ...

При раскрытии переменный параметр в *нечто* заменяется на очередное значение параметра. Если параметров было указано несколько, то все их наборы должны иметь одинаковую длину.

Примеры раскрытий переменных параметров

```
1     template <class... Args>
2     void f(Args &&... args)
3     {
4         g(std::forward<Args>(args)...);
5     }
6
7     f(); // g();
8     f(1, 2, 3); // g(1, 2, 3);
9
10    template <class... Xs> struct Zip
11    {
12        template <class... Ys>
13        struct With
14        {
15            using type = std::tuple<std::pair<Xs, Ys>...>;
16        };
17    };
18
19    using X = Zip<int, double>::With<char, std::string>::type;
```

Точки возможного раскрытия переменных параметров

- список аргументов функции
- список аргументов при вызове функции
- список аргументов в скобочной инициализации
- список параметров шаблонного идентификатора
- список параметров шаблона
- список наследования и список инициализации конструктора
- специальная форма оператора `sizeof`
- операция свёртки
- `using`-объявление
- список захвата лямбды

Некоторые примеры раскрытия переменных параметров

```
1   template <int... Is>
2   void f()
3   {
4       std::tuple<decltype(Is)...> t(Is...); // direct initialization
5       std::initializer_list<int> i = { // list initialization
6           Is...
7       };
8   }
9
10  template<class... Ts, int... N>
11  void g(Ts (&...arr)[N]); // function parameters
12
13  g("Hello", "world");
14  // g(const char (&)[5], const char (&)[5])
15
16  template <class... T>
17  struct S
18  {
19      std::tuple<T...> data; // template-id
20
21      template <T... tt> // template parameters
22      struct Nested {};
23  };
24
25  template <class... Bases>
26  class Derived : Bases... // inheritance list
27  {
28  public:
29      Derived(const Bases & ...bases)
30        : Bases(bases)... // member initialization list
31      {}
32
33      using Bases::operator()...; // using declaration
34  };
35
36  template <class... T>
37  constexpr std::size_t count = sizeof...(T); // special sizeof
38
39  template <class... T>
40  constexpr std::size_t sum_sizes()
41  {
42      return (sizeof(T) + ...); // fold expression
43  }
```

Шаблонные параметры-типы

Имя параметра становится идентификатором типа внутри шаблона и ведёт себя, как псевдоним типа.

При указании шаблонного параметра-типа это должен быть корректный идентификатор типа, в т.ч. неполного типа.

Шаблонные параметры-значения

Допустимые типы:

- lvalue ссылка
- указатель
- интегральный тип
- `enum`
- `std::nullptr_t`

Значение при подстановке шаблонным параметром должно быть известно на этапе компиляции. Параметр-указатель или параметр-ссылка не могут быть связаны со строковым литералом, подобъектом или временным объектом.

Внутри шаблона такой параметр – константное значение.

Шаблонные параметры-шаблоны

Описание параметра имеет форму описания шаблона.

Внутри шаблона такой параметр является шаблонным идентификатором.

```
 1    template <template <class, bool> class X>
 2    struct S {};
 3
 4    template <template <class K = void> class C>
 5    void f()
 6    {
 7        C<> c;
 8        C<int> cc;
 9    }
10
11    template <class T>
12    struct SS {};
13    f<SS>();
```

Виды шаблонов

- переменные
- функции
- классы
- псевдонимы типов

Член класса может быть шаблонным (статическое поле, метод, вложенный класс или псевдоним типа).

Примеры шаблонов переменных

```
 1    template <class T>
 2    T t_value = T();
 3
 4    struct S
 5    {
 6        template <int I>
 7        static int x;
 8    };
 9
10    template <int I>
11    int S::x = I;
12
13    template <class T>
14    constexpr int t_tag = 11;
15
16    int main()
17    {
18        return t_value<int> + S::x<1> + t_tag<double>;
19    }
```

Примеры шаблонов функций

```
 1    template <class T>
 2    T max(const T & a, const T & b)
 3    {
 4        return a < b ? b : a;
 5    }
 6
 7    template <class R, class T>
 8    R convert(const T & t)
 9    {
10        return t;
11    }
12
13    int main()
14    {
15        char a = '\n', b = '\t';
16        return convert<int>(max(a, b));
17    }
```

Примеры шаблонов классов

```
 1    template <class A, bool B>
 2    struct S
 3    {
 4        A data;
 5        static A s_data;
 6
 7        S();
 8        bool get() const;
 9        void set(const A & data_)
10        { data = data_; }
11    };
12
13    template <class A, bool B>
14    A S<A, B>::s_data = A();
15
16    template <class A, bool B>
17    S<A, B>::S()
18    {}
19
20    template <class A, bool B>
21    bool S<A, B>::get() const
22    { return B; }
23
24    int main()
25    {
26        S<int, false> s;
27        // s.set(11);
28        return s.data + s.s_data;
29    }
```

Примеры шаблонов методов классов

```
 1    struct S
 2    {
 3        template <class T>
 4        operator T ();
 5    };
 6
 7    template <class T>
 8    S::operator T ()
 9    { return {}; }
10
11    template <class A, class B>
12    class C
13    {
14        template <class X>
15        void f();
16    };
17
18    template <class A, class B>
19    template <class X>
20    void C<A, B>::f()
21    { }
22
23    int main()
24    {
25        C<int, char> c;
26        c.f<double>(); // ?
27        S s;
28        return s;
29    }
```

Примеры шаблонов псевдонимов типов

```
1    template <class T>
2    using Map = std::map<T, T>;
3
4    template <class T>
5    using Identity = T;
```

Содержание

Полная специализация шаблонов

```
 1  template <class T>
 2  constexpr int t_tag = 11;
 3
 4  template <>
 5  constexpr int t_tag<int> = 20;
 6  template <>
 7  constexpr int t_tag<char> = 30;
 8  template <>
 9  constexpr int t_tag<double> = 40;
10
11  template <class T>
12  T max(const T & a, const T & b)
13  { return a < b ? b : a; }
14
15  template <>
16  int max(const int & a, const int & b)
17  { return a - b; }
18
19  template <bool>
20  struct S;
21
22  template <>
23  struct S<false> {};
24
25  template <class T>
26  std::size_t check(const T & x)
27  { return sizeof(S<t_tag<T> == 20>); }
28
29  int main()
30  {
31      check('a'); // ?
32      check(1); // ?
33  }
```

Частичная специализация шаблонов

```
1   template <class T>
2   constexpr int t_tag = 11;
3
4   template <template <class...> class L, class... Ts>
5   constexpr int t_tag<L<Ts...>> = 111;
6
7   template <class T, bool = true>
8   struct MakeNonAbstract
9   {
10      template <class... Args>
11      static std::unique_ptr<T> make(Args &&... args)
12      { return {}; }
13  };
14  template <class T>
15  struct MakeNonAbstract<T, false>
16  {
17      template <class... Args>
18      static std::unique_ptr<T> make(Args &&... args)
19      { return std::make_unique<T>(std::forward<Args>(args)...); }
20  };
21
22  template <class T, class... Args>
23  auto make_non_abstract(Args &&... args)
24  {
25      return MakeNonAbstract<T, std::is_abstract_v<T>>::make(std::forward<Args>(args)...);
26  }
27
28  struct A { virtual void f() = 0; };
29
30  int main()
31  {
32      auto p_a = make_non_abstract<A>();
33      auto p_i = make_non_abstract<int>(1);
34      return t_tag<std::tuple<>>;
35  }
```

Содержание

Инстанциация шаблонов классов

```
 1    template <class T>
 2    struct A;
 3
 4    template <class T>
 5    struct B
 6    {
 7        T create()
 8        { return {}; }
 9
10        void use(const T & x)
11        { x.use(); }
12    };
13
14    int main()
15    {
16        A<int> * p_a = nullptr;
17
18        B<int> b;
19        return b.create();
20    }
```

Разрешение неоднозначности зависимых имён

```
 1    template <class T>
 2    struct S
 3    {
 4        using value_type = typename T::value_type;
 5
 6        void f(const T & x)
 7        {
 8            x.template f<int>();
 9        }
10    };
```

CRTP

```
 1    template <class T>
 2    class Base
 3    {
 4    public:
 5        void f()
 6        {
 7            static_cast<T *>(this)->f_impl();
 8        }
 9    };
10
11    class Derived : public Base<Derived>
12    {
13        friend class Base<Derived>;
14
15        void f_impl();
16    };
```