Лекция 10. Введение в шаблоны. Часть 1.

Шаблоны классов и функций

Шаблонный класс

```
1.  template<class T>
2.  struct vector
3.  {
4.      vector(size_t count = 0, T def = T());
5.      vector(vector const& other);
6.      // don't need to write vector<T> inside
7.      vector& operator=(vector const& other);
8.
9.      void resize(size_t size);
10.     void clear ();
11.
12.     T&       operator[](size_t index);
13.     T const& operator[](size_t index) const;
14.     size_t size() const;
15.
16. private:
17.     T*      data_;
18.     size_t  size_;
19. };
```

Особенности шаблонов

- Шаблонны обеспечивают **статический полиморфизм** (обобщенное программирование). Зависят только от тех свойств, что используют.
- **Параметры** шаблонов – **типы и константы** времени выполнения.
- Также **быстры**, как и специальный код для соответствующих типов – статическое связывание не ограничивает оптимизатор.
- **Не требует** связи типов между собой – например, **одной иерархии**.

Шаблонные функции

```
1.  template<class T>
2.  T max(T a, T b)
3.  {
4.      return a < b ? b : a;
5.  }
6.
7.  template<class FwdIt>
8.  void sort(FwdIt begin, FwdIt end)
9.  {
10.     /*...*/
11. }
```

- Все шаблонные **функции** становятся автоматически **inline**

Инстанцирование

- Шаблонные функции **не компилируются до инстанцирования**, поэтому хорошо проверять шаблоны на конкретных типах.
- Сгенерированный шаблон – это **полноценный** конкретный **класс** или **функция**.
- Если вы хотите предоставить шаблон для использование в разных единицах трансляции, его нужно **определить в хидере**.

```
1.  template<class FwdIt, class T>
2.  FwdIt find(FwdIt begin, FwdIt end, T const& value)
3.  {
4.      while (begin != end)
5.          if (*(begin++) == value) break;
6.      return begin;
7.  }
8.
9.  // instantiation
10. auto it = find<vector<int>::const_iterator>(v.begin(), v.end(), 42);
```

Параметры шаблонов

- Параметрами могут быть:
  - типы,
  - константные выражения времени компиляции,
  - указатели на объекты и функции с внешней компоновкой (в т.ч. указатели на члены-функции)

```
 1. template<class T, size_t N>
 2. struct array
 3. {
 4.     typedef T* iterator;
 5.
 6.     array& operator=(array& other);
 7.     T& operator[](size_t index);
 8.
 9.     iterator begin();
10.     iterator end();
11.
12. private:
13.     T data[N];
14. };
```

Эквивалентность типов

```
1.  typedef unsigned int uint;
2.
3.  typedef vector<unsigned int> uint_vec;
4.  // just the same
5.  typedef vector<uint> uint_vec;
6.
7.  ///////////////  *  ////////////////
8.
9.  template<size_t N>
10. struct fib
11. {
12.     enum {value = fib<N - 1>::value + fib<N - 2>::value};
13. };
14.
15. template<>
16. struct fib<0> { enum{value = 1}; };
17.
18. template<>
19. struct fib<1> { enum{value = 1}; };
20.
21. /*...*/
22. cout << fib<10>::value;
```

Выведение типов

```
1.  template<class T>
2.  T max(T a, T b)
3.  {
4.      return a < b ? b : a;
5.  }
6.
7.  /*...*/
8.  max(42, 10); // max<int>
9.  max(2., 5.); // max<double>
10. auto x = max<int>(3, 4.); // max<int>
11. ////////////////////////////
12.
13. template<class T>
14. T any_cast(any const&);
15.
16. /*...*/
17.
18. any any_value(42);
19. int x = any_cast<int>(any_value);
```

Перегрузка функций

```
1.  template<class T> T norm(T) { return sqrt(T * T); }
2.  template<class T> T norm(point<T>);
3.                double norm(double);
4.  /*...*/
5.  norm(-2);                     // norm<int>
6.  norm(point<double>(3, 17)); // norm<point<T>>
7.  norm(3.14);                   // norm(double)
```

- Ищется какие шаблонные функции с этим именем вообще могли бы быть вызваны и с какими аргументами.
- Откидываются менее специализированные функции.
- Подключаются нешаблонные функции. Выведенные параметры не продвигаются.
- Если одинаково хороши шаблонная и обычная функции, выбирается обычная.
- Если алгоритм привел к одной единственной функции – перегрузка успешна.
- **NB.** Шаблонный класс нельзя перегрузить (как функцию), но можно специализировать (как полностью, так и частично).

Выбор алгоритма

```
1.  template<class type, class cmp_t>
2.  void sort(T* begin, T* end, cmp_t cmp)
3.  {
4.      auto mid = (end - begin) / 2;
5.      if (cmp(*begin, *mid)) /*...*/
6.      /*...*/
7.  }
8.
9.  struct istr_cmp
10. {
11.     bool operator()(string const& lhs, string const& rhs) const
12.     {
13.         return stricmp(lhs.c_str(), rhs.c_str()) < 0;
14.     }
15. };
16.
17. /*...*/
18. string names[10] = {/*...*/};
19. sort(names, names + 10, istr_cmp());
```

Аргументы по умолчанию

```
1.  template<class T>
2.  struct less
3.      : public binary_function<T, T, bool>
4.  {   // functor for operator<
5.      bool operator()(const T& lhs, const T& rhs) const
6.      {// apply operator< to operands
7.          return (lhs < rhs);
8.      }
9.  };
10.
11. template<class type, class cmp_t = std::less<T>>
12. void sort(T* begin, T* end, cmp_t cmp = cmp_t)()
13. {
14.     auto mid = (end - begin) / 2;
15.     if (cmp(*begin, *mid)) /*...*/
16.         /*...*/
17. }
18.
19. /**/
20. int a[10] = {/*...*/};
21. sort(a, a + 10);
```

Вопросы?