

Семинар 1.9 «Шаблоны»

В C++ строгая статическая типизация, использующие конкретные типы, но некоторые действия выполняются по другой схеме, например, сортировка - необходим массив, компаратор и возможность сравнивания \Rightarrow можно написать общий алгоритм (шаблон) и при компиляции указать тип.

Альтернативные решения: (см. также варианты на уровне C)

- дублирование
 - base-derived
 - указатели void*
 - препроцессор
- } неудобно, сложно, опасно \Rightarrow шаблоны

Шаблон функций - фактически, задаётся семейство функций

Пример (live) функция max (синтаксис, требования, использование)

Инициализирование шаблона - создание конкретного экземпляра

Двухэтапная трансляция \Rightarrow нельзя разделять объявление / определение

- проверка на общие ошибки
 - проверка с конкретным типом
- } см. Intellisense + компилятор

Пример error.cpp - пример ошибки компиляции (сочинение)

Вывод аргументов шаблона

(?)

int \rightarrow const T& \Rightarrow const int& (см. max) - можно составной

Преобразования (с примерами) - см. std::decay

- по ссылке - ничего, нет даже тривиальных преобразований
- по значению - decay - явное приведение или квалификация

Несколько аргументов шаблона

Пример (live) функция max с типами T_1, T_2 и RT:

- третий тип RT - как выводить? - указывать явно
- использовать common_type_t - аргумент по умолчанию *
- позволить компилятору определять тип через auto

* допустимо для ведущих параметров шаблона

Перегрузка шаблонов функций

Пример (live) предпочтение отдается нешаблонной версии

ШАБЛОНЫ КЛАССОВ - см. контейнеры STL

Пример (live) - реализация стека

Инициализирование выполняется только для тех арифметич-
гленов, которые используются (большая гибкость для типов)

Проблема интерфейса стека (разделение на pop/top + искл-ия)
friend operator<< проще всего реализовывать прямо в классе

Специализация шаблонов класса

- оптимизация реализаций
- корректировка поведения - допускаются любые отмехи

Пример (live) - специализация стека для `std::string`

Пример (live) - специализация стека для указателей

Частичная специализация

```
template <typename T1, typename T2>
class C {...};
```

```
class C < T, T >
```

```
class C < T, int >
```

```
class C < T1* , T2* >
```

Аргументы шаблона класса по умолчанию

```
template <typename T, typename C = std::vector < T > >
```

```
class Stack
```

```
{
```

```
private:
```

```
    C container;
```

```
    ...
```

```
};
```

повторное
использование

Шаблоны псевдонимов

```
template <typename T >
```

```
using DequeStack = Stack < T, std::deque < T > > + использование
```

Суперфикс -t у свойств типов:

```
template <typename T1, typename T2>
```

```
using common_type_t = typename common_type < T1, T2 > :: type;
```

загадка?

Шаблоны переменных

```
template <typename T>
const T pi = T(3.141592); // constexpr

template <typename T>
T area_circle (T r)
{
    return pi<T> * r * r;
}
```

```
template <typename T, typename U>
inline const bool is_same_v = is_same<T,U>::value;
```

Нетиповые параметры шаблона

```
template <typename T, std::size_t size>
class Stack
{
private:
    std::array<T, size> container;
    ...
};
```

МОЖНО ЗАДАТЬ ЗНАЧЕНИЕ ПО УМОЛЧАНИЮ

... auto Size...

using size_type = decltype (size); + ИСПОЛЬЗОВАНИЕ ПСЕВДОИМЕЙ

```
std::array<T, Size> container;
```