Шаблоны классов

13 июня 2017 г.

Повторение

1. Как вы понимаете, что такое шаблон?
2. Зачем нужны шаблоны?
3. Каков синтаксис объявления шаблона функции?
4. Генерируется ли код при объявлении шаблона?
5. Что такое специализация шаблона функции?
6. Приведите пример шаблона функции.
7. Что компилятор выбирает, если есть шаблон, его специализация и нешаблонная функция с подходящими для вызова параметрами?

Шаблоны классов: объявление

```
template <typename T>
class имя_класса {
  // описание класса
};
```

- начинается с ключевого слова template
- далее в < > идут параметры шаблона
- шаблон класса может иметь несколько параметров
- внутри определения шаблона класса можно использовать имя этого класса

Шаблоны классов: объявление (пример)

```
template <typename T>
class Vector {
    T* arr;
    size_t size;
public:
    Vector();
    Vector(const size_t vecSize);
    Vector(const Vector<T>& vec);
    Vector(T* data, const size_t vecSize);
    ~Vector();
    Vector& operator=(const Vector<T>& vec);
    /* ... */
};
```

Параметры шаблона

- обобщенные типы
- стандартные типы (int, enum, указатели, ссылки)

```
template <typename T, int size>
class MyArray {
    T arr[size];
public:
    MyArray(const T& arr);
};

template <typename T, int size> // size - константа
MyArray<T, size>::MyArray(const T& arr) { /* … */ }

MyArray<double, 10> obj;    // для obj и obj1
MyArray<double, 11> obj1; // генерируются разные классы
```

Параметры по умолчанию

- можно указать значения по умолчанию для параметров шаблона
- параметры со значениями по умолчанию должны идти в конце списка параметров

```
template <typename T, typename U = int>
class MyClass { /* … */ };

MyClass<std::string, double> obj1;
MyClass<std::string> obj2;  // генерируется класс с типами
                            // std::string, int
```

Шаблоны классов: использование

- при создании объекта класса после имени класса обязательно явно указывается тип в < >

(имя_класса<тип> имя_экземляра)

```
void f() {
    Vector<int> myVec1;
    Vector<char> myVec2;
    Vector< Vector<int> > myVec3;
}
// сгенерируется код для классов Vector<int> и
Vector<char>, но не, например, Vector<double>
```

Методы

- все методы шаблонного класса тоже шаблоны
  => при определении метода за пределами класса этот метод описывается как шаблон:
  – сначала нужно написать template<typename …>
  – после имени класса указывается имя параметра
- при определении метода inline писать template<typename …> не нужно

```
template <typename T>
Vector<T>::Vector(const size_t vecSize){
    // тело метода
}
```

Организация кода

- реализацию методов шаблона класса нельзя выносить в отдельный .cpp файл (иначе возникнет ошибка)
- обычно шаблон класса пишется полностью в .h файле, который потом подключается там, где используется этот шаблон
- реализация методов может идти вне тела класса в том же .h файле

Генерация кода

- обычно код класса генерируется из шаблона не при объявлении шаблона, а при создании объекта этого класса
- код классов генерируется только для тех типов, которые заданы как параметры при создании экземпляров
- код генерируется только для тех методов, которые вызываются

Специализация шаблона класса

- версия шаблона для конкретного типа
- специализация имеет то же имя, что и шаблон, который она специализирует
- код класса-специализации может отличаться

```
template <typename T>
class Vector {
    // делаем одно
};

template < >
class Vector<bool> {
    // делаем совсем другое
};
```

Виды специализации шаблона

1) полная

- template < >
  class имя_класса<конкретный_тип> {…};

- определяет шаблон для конкретных типов, так что параметров у шаблона не остается
- после слова template пишутся пустые < >
- после имени класса в < > задаются конкретные типы, для которых специализируется шаблон

Виды специализации шаблона

2) частичная

- template <typename T>
  class имя_класса<конкретный_тип> {…};

- шаблон имеет несколько типов, но только некоторые из них заменены на конкретные
- шаблон имеет только один тип,
  но специализация нужна для типов указателя или ссылки

Выбор шаблона

- если при вызове можно использовать и шаблон, и специализацию, то компилятор выбирает специализацию
- если есть несколько подходящих специализаций, то компилятор выбирает наиболее специальный (конкретизированный по параметрам) шаблон

Выбор шаблона (пример)

```
template <typename T1, typename T2>
class MyClass { … };

template <typename T1>
class MyClass<T1, int> { … };

template < >
class MyClass<int, int> { … };


void f() {
    MyClass<int, int> obj1;          // 3
    MyClass<double, int> obj2;       // 2
    MyClass<double, char> obj3;      // 1
}
```

Советы

- помещайте весь шаблон в один .h файл
- если нужны классы, которые могут работать с разными типами, следует сделать такой класс шаблонным
- многие классы-контейнеры следует делать шаблонными (разные классы отличаются только типом объектов, которые содержат)
- сначала напишите код класса для какого-нибудь конкретного типа данных, а уже потом обобщите ее до шаблона

Псевдонимы шаблонов

- если полное имя шаблонного класса громоздкое, можно задать псевдоним с помощью ключевого слова typedef или using

```
typedef MyNewClass<int, 10> MyClassI;
typedef MyNewClass<double, 10> MyClassD;
typedef MyNewClass<char, 10> MyClassC;


template <typename T>
    using classType = MyClass<T, 10>;

classType<int> obj;
```

Вопросы?