Программирование на C++

Лекция 4 Шаблоны классов. Часть 1.

Шаблоны классов

Как и в случае функций, классы также могут быть параметризованы одним или несколькими типами. Типичным примером может служить реализация класса контейнеров, которые используются для хранения и управления элементами определенного типа. Такие контейнеры можно реализовать с помощью шаблонов классов, тип элементов контейнера остается открытым. Всюду ниже мы будем использовать в качестве демонстрации возможностей шаблон класса стек (надеюсь, вы помните, что эта за структура и каковы ее особенности, если нет – обязательно прочитайте, прежде чем приступить к реализации).

Базовая реализация шаблона класса Stack

Как и в случае с шаблонными функциями, мы объявляем и определяем класс Stack<> в заголовочном файле

stack1.hpp* | My_stack.cpp

My_stack | Stack<T>

```cpp
#include<vector>
#include<cassert>
template<typename T>
class Stack {
private:
    std::vector<T> elems; // сами элементы
public:
    void push(T const& elem); // добавление в стек
    void pop(); // снятие со стека
    T const& top() const; // верхний элемент стека
    bool empty() const // пуст ли стек
    {
        return elems.empty(); // используется метод empty для std::vector
    }
};
template<typename T>
void Stack<T>::push(T const& elem) {
    elems.push_back(elem); // добавление копии переданного элемента
}
template<typename T>
void Stack<T>::pop() {
    assert(!elems.empty()); // просто прервем работу, если стек пуст:)
    elems.pop_back(); // удаление последнего элемента
}
template<typename T>
T const& Stack<T>::top() const {
    assert(!elems.empty());
    return elems.back();
}
```

Базовая реализация шаблона класса Stack

Конечно, в данном случае, нас можно уличить в читерстве, ведь мы реализовали наш шаблон класса с использованием шаблона класса vector<> из стандартной библиотеки C++, избежав в этом случае менеджмента с памятью, реализации копирующего конструктора, оператора присваивания и прочих вещей, которые нравятся не всем☺ (кстати, загляните в документацию и посмотрите на реализацию стека в стандартной библиотеке).

Итак, используя такую нехитрую реализацию, мы можем сосредоточиться на интерфейса шаблона класса Stack<>.

Объявление шаблона класса

Объявление шаблона класса очень похоже на объявление шаблона функции: перед объявлением необходимо объявить одни или несколько идентификаторов в качестве параметров типов:

```
Template<typename T>
class Stack { … };
```

Здесь также допустимо использовать ключевое слово class вместо typename.

Внутри шаблона класса имя T может использоваться также, как и любой другой тип, для объявления членов-данных или функций-членов. В приведенном выше примере T используется для объявления типа элементов вектора, в объявлении метода push, как функции, которая получает T в качестве аргумента и для объявления top(), как функции, возвращающей T.

Объявление шаблона класса

Типом этого класса является Stack<T>, где T представляет собой параметр шаблона. Таким образом, мы должны использовать нотацию Stack<T> всякий раз, когда тип этого класса применяется в объявлении, за исключением тех случаев, когда аргументы шаблона могут быть выведены. Однако внутри шаблона класса имя класса, за которым не следуют аргументы шаблона, представляет класс с параметрами шаблона в качестве аргументов (настоятельно рекомендую порефлексировать над тем, что здесь написано ☺)

Если, например, нам нужно объявить свой копирующий конструктор и оператор присваивания, обычно мы делаем это так:

```
template<typename T>
class Stack {
   …
   Stack(Stack const &);
   Stack &operator=(Stack const &);
   …
} ;
```

Что формально эквивалентно следующему коду:

Объявление шаблона класса

Что формально эквивалентно следующему коду:

```
template<typename T>
class Stack {
   …
   Stack(Stack<T> const &);
   Stack<T> &operator=(Stack<T> const &);
   …
} ;
```

Однако, обычно <T> сигнализирует об особой обработке параметров шаблона, поэтому лучше использовать первую форму записи. Однако, вне структуры класса такое указание является необходимым:

```
template<typename T>
bool operator==(Stack<T> const & lhs, Stack<T> const & rhs);
```

Обратите внимание на то, что в местах, где требуется имя, а не тип класса, можно использовать только Stack. Это, в частности, относится к указанию имен конструкторов (но не их аргументов) и деструкторов.

Объявление шаблона класса

Что формально эквивалентно следующему коду:

```
template<typename T>
class Stack {
   …
   Stack(Stack<T> const &);
   Stack<T> &operator=(Stack<T> const &);
   …
} ;
```

Однако, обычно <T> сигнализирует об особой обработке параметров шаблона, поэтому лучше использовать первую форму записи. Однако, вне структуры класса такое указание является необходимым:

```
template<typename T>
bool operator==(Stack<T> const & lhs, Stack<T> const & rhs);
```

Заметим также, что в отличие от нешаблонных классов, шаблонные нельзя определять внутри функций или в области видимости блока

Обратите внимание на то, что в местах, где требуется имя, а не тип класса, можно использовать только Stack. Это, в частности, относится к указанию имен конструкторов (но не их аргументов) и деструкторов.

Реализация функций-членов

Для того, чтобы определить функцию-член шаблона класса, нужно указать. Что это шаблон функции. При этом необходимо использовать полностью квалифицированный тип шаблона класса. Таким образом реализация функции-члена push() типа Stack<T> имеет следующий вид:

```
template<typename T>
void Stack<T>::push(T const& elem) {
    elems.push_back(elem); // добавление копии переданного элемента
}
```

В этом случае вызывается функция push_back() вектора элементов, которая добавляет элемент в конец вектора.

Заметим, что в нашей реализации через функцию вектора pop_back() функция pop() удаляет последний элемент, но не возвращает его, что связано с вопросами безопасности исключений. Реализовать полностью безопасную в плане исключений функцию pop(), возвращающую удаленный элемент невозможно* (впервые этот вопрос был рассмотрен Томом Каргиллом).

*

- Tom Cargill, *Exception Handling: A False Sense of Security*: C++ Report, November-December 1994
- Герб Саттер. *Решение сложных задач на C++*.-М: Издательский дом «Вильямс», 2002

Реализация функций-членов

Однако, если игнорировать небезопасность данной функции в плане исключений, то можно реализовать функцию pop(), возвращающую удаленный элемент. Для этого мы просто используем T для объявления локальной переменной соответствующего типа:

```
template<typename T>
T Stack<T>::new_pop() {
    assert(!elems.empty()); // просто прервем работу, если стек пуст:)
    T elem = elems.back(); // сохраняем копию последнего элемента
    elems.pop_back(); // удаление последнего элемента
    return (elem);
}
```

Поскольку поведение функций вектора back() и pop_back() не определено для случая, когда вектор не содержит ни одного элемента, требуется проверка, не является ли стек пустым. Если он пуст, мы завершаем программу с использованием assert (мы же помним, как он работает, правда?), так как вызов pop() для пустого стека является ошибкой. Аналогично поступаем и в функции top(), которая возвращает, но не удаляет элемент на вершине стека:

```
template<typename T>
T const& Stack<T>::top() const {
    assert(!elems.empty());
    return elems.back();
}
```

Реализация функций-членов

Конечно, точно так же, как и в случае любых других функций-членов, функции-члены шаблона класса можно реализовать, как встраиваемые функции, располагающиеся внутри объявления класса, например:

```
template<typename T>
class Stack {
   …
   void push(T const & elem){
      elems.push_back(elem);
   }
   …
} ;
```

Использование шаблона класса Stack

Для того, чтобы использовать объект шаблона класса, до C++17* требовалось явно указать аргументы шаблона. В приведенном ниже примере показано, как используется шаблон класса Stack<>:

stack1.hpp | My_stack.cpp

My_stack | (Глобальная область)

```
1  #include <iostream>
2  #include<string>
3  #include "stack1.hpp"
4  int main()
5  {
6      Stack<int> intStack;
7      Stack<std::string> strStack;
8
9      // Работа со стеком целых чисел
10     intStack.push(7);
11     std::cout << intStack.top() << '\n';
12
13     // Работа со стеком строк
14     strStack.push("Hello!");
15     std::cout << strStack.top() << '\n';
16     strStack.pop();
17 }
```

В C++ 17 введен вывод аргументов шаблонов классов, который позволяет опустить шаблоны аргумента, если они могут быть выведены из конструктора

Использование шаблона класса Stack

Объявление Stack<int> указывает, что внутри шаблона класса в качестве типа T будет использоваться int. Таким образом объект intStack создается как обертка над вектором с целыми элементами, а для всех вызываемых функций членов инстанцируется код для типа int. Аналогичным образом путем объявления и использования Stack<std::string> создается объект на базе вектора строк и для каждой из вызываемой функции-члена инстанцируется код для этого типа.

Заметим, что инстанцирование происходит только для вызываемых функций(членов) шаблона. Для шаблонов классов функции-члены инстанцируются только при их использовании. Очевидно, что такой подход позволяет экономить время, память и использовать шаблоны классов только частично (об этом поговорим чуть позднее).

В данном примере инстанцируются конструктор по умолчанию, а также функции push() и top() как для значений типа int, так и для std::string. А, например, функция pop() инстанцируется только для строк. Если шаблон класса имеет статические члены, то они инстанцируются однократно для каждого типа, для которого используется шаблон класса.

Использование шаблона класса Stack

Тип инстанцированного шаблона класса можно использовать также, как и любой другой тип. Мы можем квалифицировать его ключевыми словами *const* или *volatile*, а также создавать на его основе массивы и ссылки. Его также можно использовать, как часть определения типа с помощью *typedef* или *using* или в качестве параметра типа при создании другого шаблонного класса, например:

```
void foo(Stack<int> const& st) {
    using IntStack = Stack<int>; // псевдоним для Stack<int>
    Stack<int> istack[10]; // массив из 10 стеков
    IntStack istack2[10]; // такой же массив того же типа
    ..
}
```

Аргументы шаблона могут быть любого типа, такими как указатели на float или даже стеками элементов типа int:

Stack<float*> FloatPtrStack;

Stack<Stack<int>> intStackStack;

Использование шаблона класса Stack

Единственным требованием является то, что любая вызываемая операция должна быть возможна для этого типа.

До принятия C++11 между двумя закрывающими угловыми скобками требовалось помещать пробел:

```
Stack<Stack<int> > intStackStack;
```

Если этого не сделать, получался оператор >>, что приводило к синтаксической ошибке. Этого связано с тем, что в старых версиях C++ при первом проходе компилятором выполнялось разделение исходного кода на лексемы независимо от семантики кода. Однако, поскольку отсутствие пробела стало распространенной ошибкой, было принято решение учитывать семантику кода в любом случае, и в стандарте C++11 требование ставить пробел между двумя закрывающими угловыми скобками в шаблонах было убрано.

Частичное использование шаблонов классов

Шаблон класса обычно выполняет несколько операций над аргументами шаблона, для которых он инстанцируется (в том числе при конструировании и уничтожении объекта). Это может привести к впечатлению, что аргументы шаблона должны предоставлять все операции, необходимые для всех функций-членов шаблона класса. Но это не так: аргументы шаблона должна предоставлять только те операции, которые необходимы (а не те, которые могут потребоваться).

Частичное использование шаблонов классов

Пусть, например, класс Stack<> предоставляет функцию-член printOn() для вывода содержимого стека, являющейся оберткой над оператором operator<< для каждого элемента (поместите этот код внутрь класса Stack):

```
void printOn(std::ostream& strm) const {
    for (T const& elem : elems)
        strm << elem << ' '; // Вызов << для каждого элемента стека
}
```

При этом, мы все равно можем использовать этот класс для элементов, для которых не определен operator<<:

```
int main()
{
    Stack<std::pair<int, int>> ps; // для std::pair<> не определен operator<<
    ps.push({ 7,19 }); // OK
    ps.push({ 3,42 }); // OK
    std::cout << ps.top().first << '\n'; // OK
    std::cout << ps.top().second << '\n'; // OK
```

Частичное использование шаблонов классов

Что же будет при использовании функции *printOn*? Для типов, для которых operator<< определен, все отработает согласно нашему алгоритму:

```
Stack<int> iStack;
iStack.push(3);
iStack.push(7);
iStack.printOn(std::cout);
```

И только если мы вызовем данную функцию для стека, описанного ранее, то получим ошибку времени компиляции, потому что компилятор не сможет инстанцировать вызов operator<< для этого конкретного типа:

```
ps.printOn(std::cout); // not OK!!!
```

В следующей лекции мы поговорим в том числе о том, как нам узнать, какие операции нам необходимы для инстанцирования шаблона.

Шаблоны классов. Конспект

Сделайте краткий письменный конспект лекции (используя материалы предыдущих лекций), отвечая на следующие вопросы:

1. Что такое шаблонный класс
2. Для чего нужны шаблонные классы
3. В каких случая надо использовать Stack, а в каких Stack<T> при работе с шаблонным классом Stack. В чем разница?
4. Как инстанцируются шаблонные функции-члены класса? Что такое частичное использование шаблона. Приведите свои примеры

Шаблоны классов. Задание 1

Реализуйте шаблонный класс Очередь (Queue), предусмотрев все необходимые методы для работы с этой структурой данных. Чтобы не гадать, что именно нужно реализовать – обращайтесь к документации https://en.cppreference.com/w/cpp/container/queue

Спасибо за внимание