Программирование на C++

Лекция 1 Шаблоны функций. Часть 1

Шаблоны функций. Определение шаблона

Шаблоны функций представляют функциональное поведение, которое может быть вызвано для разных типов. Шаблон – семейство функций. Основное отличие от обычных функций в том, что некоторые элементы остаются неопределенными и являются параметризованными.

Пример

```
template<typename T> // данный параметр определяется при вызове функции
T max (T a, T b){
    return (b < a ? a : b);
}
```

Это определение шаблона задает семейство функций, возвращающих большее из двух значений. Эти значения передаются функции как ее параметры a и b. Тип этих параметров остается открытым как параметр шаблона T.

Шаблоны функций. Определение шаблона

Как было в примере, параметры шаблонов задаются с помощью синтаксиса:

template< разделенный запятыми список параметров>

В нашем примере список параметров представляет собой typename T. Ключевое слово typename задает так называемый параметр, являющийся типом, или, для краткости – параметр типа, или типовой параметр. Это пока наиболее распространенный вид параметров шаблонов C++, хотя возможные и другие параметры.

В нашем примере параметр типа обозначен как T. В качестве имени параметра допустимо использовать любой идентификатор, но общепринятым является именно T. Параметр типа представляет произвольный тип, который определяется при вызове функции. Можно использовать любой тип, допускающий операции, использованные в шаблоне.

Шаблоны функций. Определение шаблона

Для определения параметра типа вместо typename можно использовать ключевое слово class:

```
template< class T>
T max(T a, T b) {
    return (b<a? a:b);
}
```

Шаблоны функций. Применение шаблонов

Давайте посмотрим на использование шаблона функции max()

```cpp
#include <iostream>
#include "my_max.hpp"
#include <string>
#include <complex>

int main(int argc, char **argv)
{
    int i = 42;
    std::cout << "max(7, i): " << ::max(7, i) << '\n';

    double e = 2.71;
    double pi = 3.14;
    std:: cout << "max(e, pi): "<< ::max(e,pi) << '\n';

    std::string s1 = "math";
    std::string s2 = "mathematics";
    std:: cout << "max(s1, s2): "<< ::max(s1,s2) << '\n';
```

Обратите внимание на вызов функции max(), он предваряется двумя двоеточиями (::). Это не потому, что max() находится в глобальном пространстве имен, а потому, что в стандартной библиотеке тоже есть шаблон std::max(), который может привести к неоднозначности.

Шаблоны функций. Применение шаблонов

Шаблоны не компилируются в единую сущность, способную обработать любой тип данных. Вместо этого из шаблона генерируются различные объекты для каждого типа, для которого применяется шаблон. Таким образом в нашем примере max() компилируется для каждого из трех типов.

Таким образом, первый вызов с типом int имеет семантику вызова следующего кода:

```
int max(int a, int b) {
   return (b < a? a: b);
}
```

Шаблоны функций. Применение шаблонов

Процесс замены параметра шаблона конкретным типом называется инстанцированием шаблона (instantiation). Его результатом является экземпляр шаблона (instance).

Аналогично, последующие вызовы шаблона max() инстанцируют шаблон max() для типов double и std::string

double max (double, double);

std::string max (std::string, std::string);

Тип void является корректным аргументом шаблона при условии, что получающийся код корректен. Например,

```
template<typename  T>
T foo(T*) {
}
```

```
void * vp = nullptr;
foo(vp);
```

void foo(void *)

Шаблоны функций. Двухэтапная трансляция

Попытка инстанцирования шаблона для типа, который не поддерживает все используемые в нем операции, приведет к ошибке компиляции. Например,

```
std::complex<float> c1(1,2);
std::complex<float> c2(1,3);
::max(c1,c2); // получим ошибку времени компиляции
```

Таким образом, шаблоны «компилируются» в два этапа.

1. Во время определения (definition time) код шаблона проверяется на корректность без инстанцирования, с игнорированием параметров шаблона. Этот процесс, как правило, включает в себя:
   - выявление синтаксических ошибок (например, отсутствие точки с запятой);
   - выявление применения неизвестных имен (имена типов, функций и т.д.), которые не зависят от параметров шаблона;
   - выполнение проверок статических утверждений, не зависящих от параметров шаблона.

Шаблоны функций. Двухэтапная трансляция

2. Во время инстанцирования код шаблона вновь проверяется на корректность. Таким образом, все части, которые зависят от параметров шаблонов, подвергаются двойной проверке. Рассмотрим пример:

```
template <typename T>
void func (T t) {
    static_assert(sizeof (int) > 6, "int is too small");
    static_assert(sizeof (T) > 6, "T is too small");
}
```

Проверка имен, выполняемая дважды, называется двухэтапным поиском (two-phase lookup). Стоит обратить внимание, что не все компиляторы выполняют полные проверки на первом этапе. Поэтому можно не увидеть общих проблем, пока шаблон не будет инстанцирован хотя бы раз.

Шаблоны функций. Вывод аргумента шаблона

При вызове с некоторыми аргументами такого шаблона как функция max(), параметры этого шаблона определяются передаваемыми в функцию аргументами. Если передать два значения int, компилятор делает вывод, что вместо T следует подставить int.

Однако T может быть только частью типа. Например, если мы объявим шаблон функции max(), как использующий константные ссылки:

```
template <typename T>
T max(T const& a, T const& b) {
   return (b < a? a: b);
}
```

и передадим значения типа int, то тип T будет выведен как int, поскольку параметры функции будут соответствовать int const&.

Шаблоны функций. Вывод аргумента шаблона

Стоит обратить внимание, что автоматическое преобразование типов в процессе вывода типа ограничено.

- При объявлении вызова по ссылке при выводе типа не происходит даже тривиальных преобразований. Два аргумента, объявленные с одним и тем же параметром шаблона T, должны точно совпадать.
- При объявлении параметров по значению поддерживаются только тривиальные, низводящие (decay) преобразования: игнорируются квалификаторы const и volatile, ссылки преобразуются в тип, на который они ссылаются, а обычные массивы или функции преобразуются в соответствующий тип указателя. Для двух аргументов, объявленных с одним и тем же параметром T низводящие типы должны совпадать.

Далее рассмотрим примеры

Шаблоны функций. Вывод аргумента шаблона

Например,

```
// Преобразование типов в процесс вывода
int b = 17;
int const c = 42;

std::cout<< max(b,c)<< '\n';
std::cout << max(c,c) << '\n';

int &ir = b; // link
std::cout << max(c, ir) << '\n';

int arr[4];
std::cout << max(&ir, arr) << '\n';

//std::cout << max(4, 7.2) << '\n'; // получим ошибку времени компиляции

std::string s = "Hi";
//max("Hello", s); // получим ошибку времени компиляции
```

Шаблоны функций. Вывод аргумента шаблона

Справиться с этими ошибками можно тремя способами.

1. Выполнить приведение аргументов, чтобы они соответствовали одному типу
   max(<static_cast<double>(4), 7.2); // ОК

2. Явно указать (квалифицировать) тип T, для того, чтобы предупредить выведение типа компилятором
   max<double>(4, 7.2); // ОК

3. Указать, что параметры могут иметь разные типы.

Шаблоны функций. Несколько параметров шаблона

До сих пор мы встречались с двумя различными наборами параметров.

1. *Параметры шаблона*, объявленные в угловых скобках перед именем шаблона функции:
   template<typename T> // T – параметр шаблона
2. *Параметры вызова*, объявленные в круглых скобках после имени шаблона функции:
   T max(T a, T b) // a и b – параметры вызова

Количество задаваемых параметров не ограничено. Например. Можно определить шаблон max() для двух потенциально различных типов данных:

```
template<typename T1, typename T2>
T1 max2(T1 a, T2 b) {
    return (b < a ? a : b);
}
```

При этом, тип первого аргумента определяет возвращаемый тип

Шаблоны функций. Несколько параметров шаблона

Может показаться желательным иметь возможность передавать шаблону параметры различных типов, но в данном примере есть свои недостатки.

Если использовать один из параметров в качестве возвращаемого типа, аргумент другого параметра должен конвертироваться в этот же тип, независимо от намерений программиста. Таким образом получаем, что возвращаемый тип зависит от порядка аргументов вызова. Проверим это:

```
auto m = ::max2(4, 7.2);
std::cout << m << '\n';
auto n = max2(7.2, 4);
std::cout << n << '\n';
```

C++ предоставляет различные способы решения этой проблемы.

- Ввести третий параметр шаблона для возвращаемого типа
- Позволить компилятору самому определить возвращаемый тип
- Объявить возвращаемый тип как «общий тип» двух типов параметров.

Шаблоны функций. Параметр шаблона для возвращаемого значения

Попробуем ввести третий параметр шаблона:

```
template<typename T1, typename T2, typename RT>
RT max (T1 a, T2 b);
…
::max<int, double, double>(4, 7.2); //OK, но очень утомительно
```

Другой подход заключается в явном указании только первых аргументов, позволяя компилятору вывести все остальные

```
template< typename RT, typename T1, typename T2>
RT max (T1 a, T2 b);
…
::max<double>(4, 7.2); //OK, возвращаемый тип – double, типы T1 и T2 выведены.
```

Но, не получили особого преимущества по сравнению с однопараметрическим шаблоном.

Шаблоны функций. Вывод возвращаемого типа

Если тип возвращаемого значения зависит от параметров шаблона, для вывода возвращаемого типа проще и лучше позволить компилятору самостоятельно его вывести.

Начиная с C++ 14, это возможно – просто нужно не указывать никакой тип возвращаемого значения (но все равно придется объявить тип возвращаемого значения как *auto*)

```
template <typename T1, typename T2>
auto max(T1 a, T2 b) {
    return (b < a ? a : b);
}
```

Фактически применение auto для возвращаемого типа без соответствующего завершающего возвращаемого типа (который вводится с помощью -> в конце объявления функции) указывает, что возвращаемый тип должен быть выведен из операторов return в теле функции. Конечно, такой вывод должен быть возможным. Таким образом код функции должен быть доступен компилятору, а все операторы должны возвращать значения одного и того же типа.

Шаблоны функций. Вывод возвращаемого типа

В стандарте C++ 11 мы могли объявить, что возвращаемый тип является производным от того, что дает тернарный оператор (?:)

```
template <typename T1, typename T2>
auto max(T1 a, T2 b) -> decltype(b < a ? a : b) {
    return (b < a ? a : b);
}
```

Шаблоны функций. Практическое задание 1

Реализуйте следующие шаблоны функций:

- swap: меняет местами значения двух заданных аргументов. Ничего не возвращает.
- min: сравнивает два значения, переданные в его аргументах, и возвращает наименьшее из них. Если два из них равны, то возвращается второй.
- max: сравнивает два значения, переданные в его аргументах, и возвращает наибольшее из них. Если два из них равны, то возвращается второй.

Перечисленные функции можно вызывать с любым типом аргумента. Единственное требование состоит в том, что два аргумента должны иметь один и тот же тип и должны поддерживать все операторы сравнения.

Шаблоны должны быть определены в заголовочном файле

Шаблоны функций. Практическое задание 1

```cpp
int     main( void ) {

    int a = 2;
    int b = 3;

    ::swap( a, b );
    std::cout << "a = " << a << ", b = " << b << std::endl;
    std::cout << "min( a, b ) = " << ::min( a, b ) << std::endl;
    std::cout << "max( a, b ) = " << ::max( a, b ) << std::endl;

    std::string c = "chaine1";
    std::string d = "chaine2";

    ::swap(c, d);
    std::cout << "c = " << c << ", d = " << d << std::endl;
    std::cout << "min( c, d ) = " << ::min( c, d ) << std::endl;
    std::cout << "max( c, d ) = " << ::max( c, d ) << std::endl;

    return 0;
}
```

Результат работы программы

```
a = 3, b = 2
min(a, b) = 2
max(a, b) = 3
c = chaine2, d = chaine1
min(c, d) = chaine1
max(c, d) = chaine2
```

Шаблоны функций. Практическое задание 2

Реализуйте шаблон функции iter, который принимает три параметра и ничего не возвращает:

- Первый параметр – это адрес массива.
- Второй – длина массива.
- Третий – это функция, которая будет вызываться для каждого элемента массива.

Включите в задание файл main.cpp, который содержит ваши собственные тесты. Позаботьтесь о том, чтобы покрытие тестами было максимальным. Ваш шаблон функции iter должен работать с любыми массивами. Третий параметр может быть инстанцированным шаблоном функции.

Шаблоны функций. Практическое задание 3

Реализуйте программу, решающую квадратное уравнение ax^2 + bx + c = 0 в пространстве C (комплексных чисел). Три коэффициента передаются в качестве аргументов программы (даже если некоторые или все из них нули). Программа должна выводить ответ в виде:

x1 = 2.5
x2 = 0.5

или

x1 = x2 = 0.5

или

x1 = 2 + i

x2 = 2 - i

или

корней нет

или

уравнение имеет бесконечное множество решений

Шаблоны функций. Практическое задание 3

Требования:

- По необходимости используйте шаблонные функции;
- Нельзя использовать библиотечные математические функции (это означает, что функции вида sqrt должны быть реализованы вами самостоятельно);
- Реализовать всевозможные проверки (некорректный ввод, деление на ноль), чтобы программа «не падала»;
- Следите за структурой проекта, .hpp, .cpp, main.cpp, не надо все реализовывать в одном файле.

Спасибо за внимание