Программирование на C++

Лекция 2 Шаблоны функций. Часть 2. Специализация и перегрузка шаблонов

Специализация шаблона

До сих пор мы видели, что шаблоны функций позволяют объявить «обобщенные» функции, которые можно конкретизировать разными типами. Пока все такие функции отличались только типами, а код для них генерировался одинаковый. Но это не всегда желательно, иногда разные типы следует обрабатывать по-разному.

Специализация шаблона позволяет генерировать другой код для некоторых типов, то есть при подстановке различных типов генерируется не одинаковый, а совершенно разный код. В C++ есть два вида специализации шаблона: **явная или полная**, и **частичная**. Начнем с первой.

Явная специализация

В случае явной специализации определяется специальная версия шаблона для определенного набора типов. При этом все обобщенные типы заменяются конкретными. Поскольку явная специализация не является обобщенной функцией, то она и не должна конкретизироваться впоследствии. По этой причине ее иногда называют **полной специализацией**. Если произведена подстановка всех обобщенных типов, то ничего обобщенного не остается.

Явную специализацию не стоит путать с явной конкретизацией шаблона – хотя в обоих случаях создается конкретизация шаблона заданным набором аргументов-типов, механизм этих операций отличается.

Явная специализация

При явной конкретизации создается конкретный экземпляр обобщенного кода, в котором вместо обобщенных типов поставлены конкретные.

Явная специализация, в отличие от явной конкретизации создает экземпляр функции с тем же именем, но подменяет реализацию, так что результирующий код может оказаться совершенно другим. Рассмотрим это на примере.

Специализируем явно шаблонную функцию. В отличии от явно конкретизации, мы должны написать тело функции, которое можем реализовать как угодно:

```
template<typename T>
T do_something(T x) {
    return (++x);
}
template<>
double do_something<double>(double x) {
    return(x / 2);
}
```

```
do_something(3); //4
do_something(3.0); // 1.5
```

Явная специализация

Явная специализация должна быть указана раньше первого использования шаблона, которое вызвало бы неявную конкретизацию обобщенного шаблона теми же типами. Очевидно, что в результате неявной конкретизации была бы создана функция с таким же именем и типами, что при явной специализации. Таким образом, мы бы получили в программе два варианта одной и той же функции, а это нарушает правило одного определения и делает программу некорректной.

Явная специализация полезна, когда имеется один или несколько типов, для которых шаблон должен вести себя совершенно по-другому.

Перегрузка шаблонных функций

Частичная специализация шаблона в C++ предусмотрена только для классов и о ней мы поговорим при рассмотрении шаблонов классов. То, что иногда по ошибке называют частичной специализацией шаблона функции, на самом деле является **перегрузкой шаблонных функций**, поэтому к ней мы и перейдем далее.

Перегрузка шаблонных функций

Мы привыкли к перегрузке обычный функций и методов классов, когда имеется несколько функций с одинаковым именем, но разными типами параметров. Вызывается тот вариант функции, для которого типы параметров лучше всего соответствуют фактическим аргументам, как показано в примере:

```
void whatami(int x) {
    std::cout << x << " типа int" << std::endl;
}
void whatami(long x) {
    std::cout << x << " типа long" << std::endl;
}
```

```
whatami(5);
// whatami(5.0); //ошибка компиляции
```

Перегрузка шаблонных функций

Если аргументы точно соответствуют параметрам одного из перегруженных вариантов функции, то этот вариант и вызывается. В противном случае компилятор рассматривает возможность преобразовать типы параметров существующих функций. Если для какой-то функции имеется *наилучшее* преобразование, то она и вызывается. Иначе вызов считается неоднозначным, как в последней строке предыдущего примера.

Перегрузка шаблонных функций

Точное определение того, что считается *наилучшим* преобразованием можно найти в стандарте. В общем случае самыми *дешевыми* считаются такие преобразования, как добавление const или удаление ссылки; далее идут преобразования между встроенными типами, преобразование указателя на производный класс в указатель на базовый класс и т.д.

Если у функции несколько аргументов, то для каждого аргумента должно существовать наилучшее преобразование. Если из трех аргументов функции два точно соответствуют первому перегруженному варианту, а третий точно соответствует второму варианту, то даже если оставшиеся типы можно неявно преобразовать в типы соответственных параметров, то вызов все равно считается неоднозначным.

Перегрузка шаблонных функций

Наличие шаблонов значительно усложняет процесс разрешения перегрузки. Помимо нешаблонных функций может быть определено несколько шаблонов функций с тем же именем, и, возможно, с таким же количеством аргументов. Все они являются кандидатами на разрешение вызова перегруженной функции, но шаблоны функций могут порождать функции с разными типами параметров. И как в этом случае решить, каково фактическое множество перегруженных функций?

Правила еще сложнее, чем для нешаблонных функций☺

Перегрузка шаблонных функций

Основная идея такова: если существует нешаблонная функция, которая почти идеально соответствует фактическим аргументам, то она и выбирается. Если такой функции нет – компилятор пытается конкретизировать все шаблоны функций с тем же именем, что у вызываемой, стремясь получить почти идеальное соответствие, для чего применяет выведение аргументов шаблона. Если был конкретизирован ровно один шаблон, то вызывается функция, получающаяся в результате этой конкретизации.

Это очень упрощенная схема весьма сложного процесса☺

Следует помнить два важных момента:

- Если шаблонная и нешаблонная функции одинаково хорошо соответствуют вызову, то выбирается нешаблонная;
- Компилятор не пытается конкретизировать шаблоны функции в нечто такое, что можно было бы преобразовать в нужные нам типы. После выведения типов аргументов шаблонная функция должна точно соответствовать вызову, иначе она не рассматривается.

Перегрузка шаблонных функций

Добавим к предыдущему примеру шаблон:

```
template <typename T>
void whatami(T* x) {
    std::cout << x << " указатель" << std::endl;
}
```

```
int i = 5;
whatami(i); // 5 типа int
whatami(&i); // 0x.. указатель
```

Перед нами обычный шаблон, а не частичная специализация какого-то общего шаблона (как могло бы показаться, поскольку пример реализован для указателя). Параметр-тип выводится из тех же аргументов, но немного по другим правилам. Тип шаблона можно вывести, если аргумент является указателем на что-нибудь. Сюда входит и указатель на const, поэтому, если мы вызовем whatami(ptr), где ptr имеет тип const int*, то первый перегруженный вариант шаблона является точным совпадением, в котором T – это const int. Если выведение успешно, то функция, порожденная шаблоном добавляется во множество перегруженных вариантов.

Перегрузка шаблонных функций

Для варианта int* это единственный пригодный вариант, поэтому он и вызывается. Но что произойдет, если вызову соответствует два (или более) шаблона функций и обе конкретизации – допустимые варианты перегрузки?

Для примера добавим еще один шаблон:

```
template <typename T>
void whatami(T&& x) {
    std::cout << "что-то на непонятном" << std::endl;
}
class C {}; //какой-то обычный класс
```

```
C c;
whatami(c); // что-то непонятное
whatami(&c); // 0x.. указатель
```

Этот шаблон функции принимает аргумент по *универсальной ссылке* (**прочитать, что это такое**), поэтому он может быть конкретизирован для любого вызова whatami() с одним аргументом. С первым вызовом, whatami(c), все просто – подходит только последний вариант с параметром типа &&T. Не существует преобразования из с в указатель или целое число.

Перегрузка шаблонных функций

Со вторым вызовом ситуация сложнее – мы имеем не одну, а две конкретизации шаблона, точно соответствующие вызову, без каких-либо преобразований. Тогда почему эта перегрузка не считается неоднозначной?

Потому что правила разрешения перегрузки шаблонов функций отличаются от правил для нешаблонных функций. Лучшим соответствием считают более специфичный шаблон, что характерно для частичной специализации шаблонов классов (об этом мы поговорим позднее).

В нашем случае более специфичен первый шаблон – он может принимать любые указатели, но только указатели. Второй шаблон готов принять вообще любой аргумент. Итак, если более специфичный шаблон можно использовать для порождения функции, являющейся допустимым перегруженным вариантом, то он и будет выбран. В противном случае мы вынуждены обратиться к более общему шаблону.

Очень общие шаблонные функции во множестве перегруженных вариантов иногда приводят к неожиданным результатам

Перегрузка шаблонных функций

Очень общие шаблонные функции во множестве перегруженных вариантов иногда приводят к неожиданным результатам. Пусть у нас имеются три такие перегрузки для int, double и произвольного типа:

```
void whatami(int x) {
    std::cout << x << " типа int" << std::endl;
}
void whatami(double x) {
    std::cout << x << " типа double" << std::endl;
}
template <typename T>
void whatami(T&& x) {
    std::cout << "что-то на непонятном" << std::endl;
}
```

```
int i = 5;
float x = 4.2;
whatami(i); // i типа int
whatami(x); // что-то непонятное
```

Перегрузка шаблонных функций

В первом вызове аргумент имеет тип int, так что вариант whatami(int) является точным совпадением.

Для второго вызова был бы выбран вариант whatami(double), если бы не было перегруженного шаблона.

Дело в том, что преобразование float в double неявное (как и преобразование float в int, но преобразование в double предпочтительнее), однако это все же преобразование, поэтому когда шаблон функции конкретизируется в точное совпадение whatami(double &&), оно оказывается наилучшим соответствием и выбирается в качестве результата разрешения.

Перегрузка шаблонных функций

Наконец, существует еще один вид функций, играющий особую роль в разрешении перегрузки, - функции с переменным числом аргументов.

В объявлении такой функции вместо аргументов указывается многоточие, и ее можно вызвать с любым количеством аргументов любых типов (примером является С-функция printf). Такая функция – последняя надежда разрешить перегрузку, она вызывается, лишь если никаких других вариантов нет:

```
void whatami(...) {
    std::cout << "что угодно" << std::endl;
}
```

Так как у нас имеется перегруженный вариант whatami (T&& x), функция с переменным числом аргументов никогда не будет предпочтительным вариантом, по крайней мере не для вызова whatami() с одним аргументом. Но если такого шаблона не было бы, то whatami(…) вызывается для любого аргумента, не являющегося ни числом, ни указателем. Функции с переменным числом аргументов существовали еще в языке программирования С, но не стоит их путать с шаблонами с переменным числом аргументов, которые появились в 11 стандарте. О них пойдет речь в следующей лекции.

Специализация и перегрузка шаблонов. Конспект

Сделайте краткий письменный конспект лекции, отвечая на следующие вопросы:

1. Что такое специализация шаблона функции?
2. Для чего нужна специализация шаблона?
3. Чем конкретизация шаблона отличается от специализации, приведите пример
4. Что такое перегрузка нешаблонных функций
5. Как определяется множество перегруженных функций с учетом шаблонных?
6. Что означает понятие «более специфичный шаблон», как это работает при выборе перегрузки шаблонной функции?
7. В чем отличие разрешения перегруженных шаблонов от нешаблонных функций?

Специализация шаблонов функций. Задание 1

Реализуйте шаблонную функцию – модуль (для действительных чисел – это абсолютное значение). Для объектов комплексное число и матрица напишите полные специализации, для комплексного числа используйте определение модуля комплексного числа, а для матрицы – определитель. Конечно, сперва нужно объявить классы комплексных чисел и матриц (все методы/поля для работы с этими объектами описывать не нужно, только то, что требуется для выполнения задания).

Перегрузка шаблонов функций. Задание 2

Приведите собственный пример реализации перегрузок: то есть может быть несколько нешаблонных функций и несколько шаблонных, все они должны иметь одно имя, и вызов функции с этим именем должен демонстрировать перегрузку для различных типов. Крайне желательно, чтобы это был не слишком синтетический пример, а пример, который реально можно использовать (подумайте об этом, прежде чем определять такие функции и шаблоны). В данном случае должен быть файл с шаблонными и нешаблонными функциями и main, который демонстрирует перегрузки с комментариями.

Спасибо за внимание