Программирование на C++

Лекция 3 Шаблоны функций. Часть 3. Еще немного о перегрузке. Примеры.

Еще немного примеров с перегрузкой шаблонов функций

В прошлый раз мы обсуждали особенности ~~частичной специализации~~ перегрузки шаблонных функций. Давайте рассмотрим еще несколько полезных примеров.

Первым интересным примером является перегрузка шаблона, в котором можно явно задать только тип возвращаемого значения:

```
template<typename T1, typename T2>
auto max(T1 a, T2 b) {
    std::cout << "The first template is used" << std::endl;
    return (b < a ? a : b);
}
template<typename RT, typename T1, typename T2>
RT max(T1 a, T2 b) {
    std::cout << "The second template is used" << std::endl;
    return (b < a ? a : b);
}
```

Теперь можно вызвать max(), например, следующим образом:

```
auto a = ::max(4, 7.2); // используется первый шаблон
auto b = ::max<long double> (7.2, 4); // используется второй шаблон
```

Еще немного примеров с перегрузкой шаблонов функций

Однако при таком вызове

```
auto c = ::max<int>(4, 7.2); // ошибка, соответствуют оба шаблона
```

ему соответствуют оба шаблона, что приводит к тому, что процесс перегрузки не в состоянии выбрать ни один из них, и мы получаем ошибку неоднозначности. Таким образом, при перегрузке шаблонов функций следует убедиться, что любому вызову соответствует только один из них.

Еще немного примеров с перегрузкой шаблонов функций

Полезным примером может быть перегрузка шаблона максимума для указателей и обычных C-style строк (определили все перегрузки):

```cpp
#include <string>
#include <cstring>

// Максимальное из двух значений любого типа
template<typename T>
T max(T a, T b) {
    std::cout << "1" << std::endl;
    return (b < a ? a : b);
}
// Максимальный из двух указателей определяется значениями,
// на которые указывают указатели, а не значениями указателей
template<typename T>
T* max(T* a, T* b) {
    std::cout << "2" << std::endl;
    return (*b < *a ? a : b);
}
// Максимальная из двух C-style строк
char const* max(char const * a, char const * b) {
    std::cout << "3" << std::endl;
    return std::strcmp(b, a) < 0 ? a : b;
}
```

Еще немного примеров с перегрузкой шаблонов функций

Полезным примером может быть перегрузка шаблона максимума для указателей и обычных C-style строк (вызываем функцию max()):

```
int main(int argc, char **argv)
{
    int a = 7;
    int b = 42;
    auto m1 = ::max(a, b); // max() для двух значений типа int

    std::string s1 = "hey";
    std::string s2 = "you";
    auto m2 = ::max(s1, s2); // max() для двух значений типа std::string

    int *p1 = &b;
    int *p2 = &a;
    auto m3 = ::max(p1, p2); // max() для двух указателей

    char const *str1 = "hello";
    char const *str2 = "world";
    auto m4 = ::max(str1, str2); // max() для двух C-style строк
```

Еще немного примеров с перегрузкой шаблонов функций

Обратите внимание на то, что во всех перегрузках max() мы передавали аргументы по значению. В целом, это хорошая идея – при перегрузке шаблонов функций не изменять больше необходимого. Мы должны стараться ограничивать свои изменения количество параметров или явным указанием параметров шаблона. В противном случае, мы можем получить неожиданные эффекты.

Например, реализуем шаблон функции max() для передачи аргументов по ссылке и перегрузим его для двух C-style строк, передаваемых по ссылке:

```
#include <cstring>

// максимальное из двух значений (передача по ссылке)
template<typename t>
t const & max(t const & a, t const & b) {
    return (b < a ? a : b);
}
// максимальная из двух строк (передача по значению)
char const* max(char const * a, char const * b) {
    return std::strcmp(b, a) < 0 ? a : b;
}
// максимальное из трех значений (передача по ссылке)
template<typename t>
t const & max(t const & a, t const & b, t const &  c) {
    return (max(max(a, b), c));
}
```

Еще немного примеров с перегрузкой шаблонов функций

Например, реализуем шаблон функции max() для передачи аргументов по ссылке и перегрузим его для двух C-style строк, передаваемых по ссылке:

```
int main(int argc, char **argv)
{
    auto m1 = ::max(7, 42, 68);
    char const* s1 = "ivan";
    char const* s2 = "petr";
    char const* s3 = "anton";
    auto m2 = ::max(s1, s2, s3);
```

Еще немного примеров с перегрузкой шаблонов функций

Проблема в том, что если мы вызовем max() для трех C-style строк, инструкция

return max(max(a, b),c)

приведет к ошибке времени выполнения, потому что для C-style строк max(a,b) создает новое, временное локальное значение, которое возвращается по ссылке, но это временное значение становится недействительным, как только завершается оператор return, оставляя main с висячей ссылкой. Это довольно тонкая ошибка, которая проявляется не во всех случаях. Более того, компиляторы, действующие в соответствии со стандартом даже не могут отклонить этот код при компиляции.

Заметим, что первый вызов max() в main этим не страдает. В нем создаются временные переменные для аргументов (7,42,68), но эти переменные создаются непосредственно в main и хранятся до тех пора, пока инструкция не будет полностью выполнена.

Это только один пример кода, который может работать не так, как ожидается, из-за точного применения правил разрешения перегрузки.

Еще немного примеров с перегрузкой шаблонов функций

Кроме этого, нужно всегда следить за тем, чтобы все перегрузки были объявлены до вызова. Тот факт, что не все перегруженные версии видимы в момент вызова, может иметь важное значение:

```
// Максимальное из двух значений любого типа:
template<typename T>
T max (T a, T b) {
    std::cout << "max<T> () \n";
    return (b < a ? a : b);
}
// Максимальное из трех значений любого типа:
template<typename T>
T max (T a, T b, T c) {
    return max(max(a, b), c); // Использует шаблонную версию даже
                              // для int, так как следующее объявление
                              // встречается слишком поздно
}
// Максимальное из двух значений типа int:
int max(int a, int b) {
    std::cout << "max(int, int) \n";
    return (b < a ? a : b);
}

int main(int argc, char **argv)
{
    ::max(47, 11, 13);
```

Еще немного примеров с перегрузкой шаблонов функций

В данном случае определение версии функции max() с тремя аргументами, которому не видно объявление специализированной двухаргументной версии функции max() для int приводит к тому, что в версии с тремя аргументами используется двухаргументный шаблон.

Таким образом, даже эти простые примеры порождают у разработчиков еще больше вопросов. Далее обозначим основные из них.

Передача по значению или по ссылке?

Если вы обратили внимание на примеры, рассмотренные ранее, то мы в основном объявляли функции с передачей аргументов по значению вместо использования ссылок. В общем случае передача по ссылке, как известно, рекомендуется для типов, отличных от «дешевых» простых типов, потому что при этом не создаются ненужные копии. Однако по ряду причин передача по значению часто оказывается лучшим решением:

- Простой синтаксис
- Лучшая оптимизация кода компилятором
- Семантика перемещения часто делает копирование дешевой операцией
- Иногда копирование или перемещение не выполняется вовсе

Для шаблонов кроме всего прочего важны следующие аспекты:

- Шаблон может использоваться как для простых, так и для сложных типов, так что выбор подхода для сложных типов может оказаться контрпродуктивным для простых типов
- Как программист, отвечающий за вызов кода, вы все равно можете принять решение о передаче аргументов по ссылке, используя std::ref() и std::cref() (обязательно прочтите об этом механизме стандарта C++11)
- Хотя передача строковых литералов или простых массивов всегда оказывается проблемой ☺, их передача по ссылке часто становится еще большой проблемой ☺ ☺

Использование inline

В общем случае шаблоны функций не должны быть объявлены с использованием inline (вспомните или прочитайте, что означает inline). В отличие от обычных невстраиваемых (те, что объявлены без inline) функций мы можем определить шаблоны функций, не являющихся inline, в заголовочном файле и включать этот заголовочный файл в несколько единиц трансляции.

Единственным исключением из этого правила являются полные специализации шаблонов для конкретных типов (мы помним, что в этом случае код более не является обобщенным, в нем определены все параметры шаблона).

С точки зрения строгого определения языка inline означает только то, что определение функции может встречаться в программе несколько раз. Однако, это также означает подсказку компилятору, что вызовы этой функции должны быть встраиваемыми (впрочем компилятор может проигнорировать это указание, так как в некоторых случаях это может сделать код менее эффективным). Современные компиляторы способны решать такие вопросы без дополнительных подсказок в виде слова inline, однако, в настоящее время по-прежнему, компиляторы учитывают наличие данного ключевого слова.

Использование constexpr

Иногда нам требуется, чтобы некоторые значения были вычислены уже на этапе компиляции. Начиная, с C++11 для этих целей можно использовать ключевое слово *constexpr*. Это имеет смысл для множества шаблонов.

Например, чтобы иметь возможность использовать функцию получения максимума во время компиляции, необходимо объявить ее следующим образом:

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

При этом шаблон функции можно использовать во время компиляции в соответствующих контекстах, таких, например, как объявление размера массива:

```
int a[::max(sizeof(char), 1000u)];
```

Обратите внимание на то, что мы передаем 1000 как значение типа unsigned int, чтобы избежать предупреждения о сравнении в шаблоне знаковых и беззнаковых значений.

Еще немного о перегрузке. Конспект

Сделайте краткий письменный конспект лекции (используя материалы предыдущих лекций), отвечая на следующий вопрос:

1. Опишите очевидные (и не очень) проблемы, приводящие к ошибкам, неочевидным или непредсказуемым результатам, связанные с перегрузками шаблонных функций со своими небольшим примерами. Описываете проблему, приводите код с результатами выполнения.

Специализация шаблонов функций. Задание 1

Реализуйте шаблонную функцию – сложение (для действительных чисел – это обычное сложение). Для объектов комплексное число и матрица напишите полные специализации, используя свои классы из предыдущего задания. Кроме этого определите соответствующие перегрузки для C-style строк и объектов std::string (сложение в данном случае работает, как конкатенация), а также для указателей, где складываются соответствующие значения, на которые указывают указатели. Продемонстрируйте достаточным количеством примеров работу ваших реализаций.

Спасибо за внимание