Шаблоны функций

12 июня 2017 г.

Задание

- Напишите функцию, которая получает два числа типа int и возвращает наибольшее из них.
- Напишите функцию, которая получает два числа типа long и возвращает наибольшее из них.
- Напишите функцию, которая получает два числа типа double и возвращает наибольшее из них.

и т. д.

Проблема

```
int getMax(const int first, const int second){
    return (first > second ? first : second);
}
long getMax(const long first, const long second){
    return (first > second ? first : second);
}
double getMax(const double first, const double second){
    return (first > second ? first : second);
}
```

Много функций, которые отличаются только типом параметров и возвращаемого значения.

Решение

- обобщенное программирование с помощью шаблонов

```
template<typename T>
T getMax(const T first, const T second){
    return (first > second ? first : second);
}

std::cout  << getMax<int>(10, 13) << std::endl;
std::cout  << getMax<double>(10.89, 13.16) << std::endl;
std::cout  << getMax<int>(10.89, 13.16) << std::endl;
```

Шаблоны в C++

позволяют использовать одну функцию или класс для разных типов данных

- шаблоны функций
- шаблоны классов

Шаблоны функций: объявление

```
template <typename T>
тип имя_функции(параметры) {
  // тело_функции
}
```

- начинается с ключевого слова template
- после слова template в угловых скобках идут параметры шаблона
- обобщенный тип параметра задается с помощью ключевого слова typename или class (здесь typename и class – одно и то же)

Параметры шаблона функции

- если среди параметров шаблона есть обобщенный тип (T), то он должен быть и среди аргументов функции
- шаблон может иметь несколько параметров
- параметрами шаблона могут быть основные типы (int, enum, указатель, ссылка)

```
template<typename T, typename U>
T convert(T to, U from);

template<int bufSize>
char* read();
```

Шаблоны функций: использование

- инстанцирование шаблона – это процесс создания экземпляра функции из шаблона
- при вызове функции нужно указать конкретные типы вместо параметров шаблона
- неявное преобразование типов для параметров не происходит
- в угловых скобках < > после имени функции можно явно указать типы, для которых нужно сгенерировать эту функцию

Инстанцирование шаблона (пример)

```
int a = getMax(10, 13);
double b = getMax(10.1, 13.9);
int c = getMax(10, 13.89);    // Error
int d = getMax<int>(10, 13.89);    // OK
```

Компилятор генерирует следующие функции:

```
int getMax(const int first, const int second){
        return (first > second ? first : second);
}
```

```
double getMax(const double first, const double second){
        return (first > second ? first : second);
}
```

Перегрузка шаблона функций

- шаблоны можно перегружать
- действуют обычные правила перегрузки (должна быть разная сигнатура функций)

```
template<typename T>
T getMax(const T first, const T second){
    return (first > second ? first : second);
}

template<typename T>
T getMax(const T first, const T second, const T third){
    …
}
```

Шаблон и нешаблонная функция

В одном коде могут быть шаблон и нешаблонная версия некоторой функции

```
template<typename T>
T getMax(const T first, const T second);

int getMax(int first, int second);

std::cout << getMax(101, 1001) << std::endl;      // 2
std::cout << getMax(10.11, 10.78) << std::endl;   // 1
```

Если есть подходящая нешаблонная функция, компилятор выберет ее

Специализация шаблона

Специализация – это переопределение шаблона для некоторого конкретного типа данных

- нужны <>
- конкретный тип задается с помощью аргументов или явно указывается в <> после имени функции

```
template<typename T>
T getMax(const T first, const T second) { … };

class myClass;

template< >
myClass  getMax(const MyClass first, const MyClass second){
        // новая реализация
}
```

Порядок выбора функции

(1) нешаблонная функция

(2) специализация шаблона

(3) шаблон (генерация из шаблона)

Всегда выбирается самая специализированная версия

Порядок выбора функции (пример)

```
template<typename T>
T getMax(const T first, const T second) { … };

template< >
complex getMax(const complex first, const complex second){ … };

int getMax(int first, int second);

void f(const complex& cmp1, const complex& cmp2){
    std::cout << getMax(101, 1001) << std::endl;      // 3
    std::cout << getMax(cmp1, cmp2) << std::endl;     // 2
    std::cout << getMax(10.11, 10.78) << std::endl;   // 1
}
```

Ключевое слово decltype

- возвращает точно тот же тип, что и у аргумента

```
int x = 51;
decltype(x) y // int y
```

- позволяет указать возвращаемый для шаблона функции тип, если он включает параметры (хвостовой возвращаемый тип)

```
template<typename T, typename U>
auto f(const T x, const U y) -> decltype(x + y) {
        return x + y;
}
```

Советы

- если нужны функции, которые делают строго одно и то же для разных типов данных, то лучше написать шаблон
- сначала напишите функцию для какого-нибудь конкретного типа данных, а уже потом обобщите ее до шаблона

Вопросы?