Шаблоны с переменным числом аргументов

14 июня 2017 г.

Задание

- Напишите шаблон функции, которая получает **два числа** и возвращает их сумму. Учтите, что типы чисел могут быть разными.
- Напишите шаблон функции, которая получает **три числа** и возвращает их сумму. Учтите, что типы чисел могут быть разными.
- Напишите шаблон функции, которая получает **четыре числа** и возвращает их сумму. Учтите, что типы чисел могут быть разными.

и т. д.

Проблема

```
template<class T1, class T2>
auto sum(T1 a, T2 b) -> decltype(a + b)
{
    return a + b;
}

template<class T1, class T2, class T3>
auto sum(T1 a, T2 b, T3 c) -> decltype(a + b + c)
{
    return a + b + c;
}
```

Много шаблонов функций, которые отличаются только количеством параметров.

Решение

- шаблоны с переменным числом аргументов (variadic templates)

```
template<typename T, typename ... Args>
T sum(T a, Args ... values)
{
    // реализация
}

template<typename T, typename ... Args>
auto sum(T a, Args ... values)        // начиная с C++14
{
    // реализация
}
```

Определение

Шаблон с переменным числом аргументов – это шаблон класса или функции, поддерживающий произвольное число аргументов, типы которых могут быть разными

Объявление

- начинается с ключевого слова template
- в < > идут параметры шаблона
- последний среди параметров – пакет параметров (typename … ИмяПакета)

```
// Шаблон класса
template <typename T,  typename … MyTypes>
class имя_класса { … };
```

```
// Шаблон функции
template <class … MyTypes>
void имя_функции(int a, MyTypes … args){ … }
```

Пакет параметров шаблона

- у пакета параметров может быть любое имя
- в объявлении перед именем пакета параметров должно быть многоточие (…)
- вокруг многоточия можно ставить пробелы
- пакет параметров может быть как обобщенного, так и стандартного типа

```
template <typename T,  typename … Types>
class MyClass { … };

template <int… IntArgs>
class MyIntClass { … };
```

Пакет параметров функции

- пакет параметров функции – это переданный набор аргументов, типы которых соответствуют типам из пакета параметров
- указывается в списке аргументов функции
- перед именем пакета ставится многоточие (...)
- имя пакета может быть любым

```
template <typename ... Types>
void myFunction1(Types ... args1) { … }

template <typename ... Types>
void myFunction2(Types& ... args2) { … }
```

Распаковка пакета

- использование пакета – это распаковка пакета или раскрытие пакета
- после имени пакета ставится многоточие (…)

```
// распаковка в список базовых классов
template <typename … Types>
class MyClass : public Types… { … };

// распаковка в список аргументов функции
template <typename … Types>
void myFunction(const int head, Types … tail) {
        f1(tail…);
}
```

Модификация аргументов

- при распаковке пакета параметров можно использовать модификаторы const, *, & и &&
- эти модификаторы будут применены к каждому объекту в пакете параметров

```
template <typename ... Types>
void myFunction1(Types& ... args) { … }

template <typename ... Types>
void myFunction2(Types* ... args) { … }

template <typename ... Types>
void myFunction2(const Types& ... args) { … }
```

Модификация аргументов (пример)

- разные модификаторы можно указывать и при распаковке пакета параметров функции

```
f(&args...);    // f(&E1, &E2, &E3)
f1(n, ++args...);    // f1(n, ++E1, ++E2, ++E3)
f2(++args..., n);    // f2(++E1, ++E2, ++E3, n);
```

Использование пакетов

- к отдельным аргументам из пакета параметров нельзя обращаться по индексу
- узнать количество аргументов можно с помощью оператора sizeof...(ИмяПакета)

```
template <typename ... Types>
void myFunction(Types& ... args)
{
        std::cout << args[0] << std::endl;   // Error
        std::cout << sizeof...(args) << std::endl;   // OK
}
```

Обработать аргументы из пакета можно рекурсивно

Рекурсия и шаблоны функций

- в функции задается обязательный первый аргумент обобщенного типа T, а уже затем указывается пакет параметров функции
- при рекурсивной обработке пакета на каждой итерации от пакета «отщипывается» первое значение, которое подставляется на место первого аргумента с типом T, а остальные значения остаются в пакете
- для выхода из рекурсии пишется нешаблонная функция или специализация шаблона

Обработка аргументов (пример)

```
template <typename T, typename … Types>
auto sum(const T head, const Types … tail) {
    return head + sum(tail…);
    // при передаче пакета в функцию
    // после имени пакета нужно многоточие (…)
}

template <typename T>
T sum(const T value) {
    return value;
}

void f() {
    std::cout << sum(10, 9.99, 0.01, 10) << std::endl;
}
```

Вопросы?