**Глава 11. Функции с переменным числом параметров. Перегруженные функции. Прототипы и шаблоны функций**

**11.1 Функции с переменным числом параметров**

Если список формальных параметров функции заканчивается многоточием, это означает, что при вызове функции на этом месте можно указать еще несколько параметров. Функции с переменным числом параметров позволяют передавать произвольное количество аргументов при вызове функции. Структура объявления функции с переменным числом параметров представлена на рисунке 11.1.

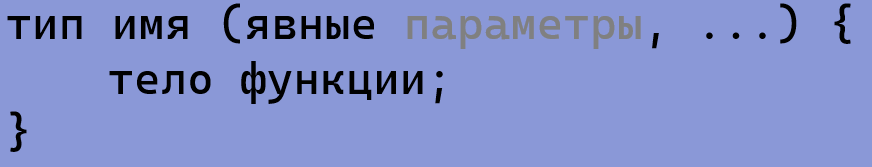


Рисунок 11.1 – Структура объявления и определения функции с переменным числом параметров

Для доступа к необязательным параметрам внутри функции используются макросы библиотеки, находящиеся в заголовочном файле *<сstdarg>*или *<stdarg.h>*.

Для работы с переменными параметрами используется тип *va\_list*, который предоставляет механизм для доступа к аргументам переменной длины.

1. Макрос *va\_start* используется для инициализации *va\_list* перед началом чтения аргументов.
2. Макрос *va\_arg* используется для чтения аргументов из *va\_list*. Тип аргумента определяется через тип, переданный вторым аргументом.
3. Макрос *va\_end* завершает работу с *va\_list* после окончания чтения аргументов.

Пример использования функции с переменным числом параметров представлен на рисунке 11.2.

Главной частью кода является определение функции *average*, которая принимает первым аргументом количество чисел, за которыми следует произвольное количество целочисленных аргументов. Внутри функции используется *механизм переменных аргументов* (va\_list, va\_start, va\_arg, va\_end) для обработки произвольного количества аргументов и вычисления их суммы [12]. После получения суммы всех аргументов, вычисляется и возвращается среднее значение.

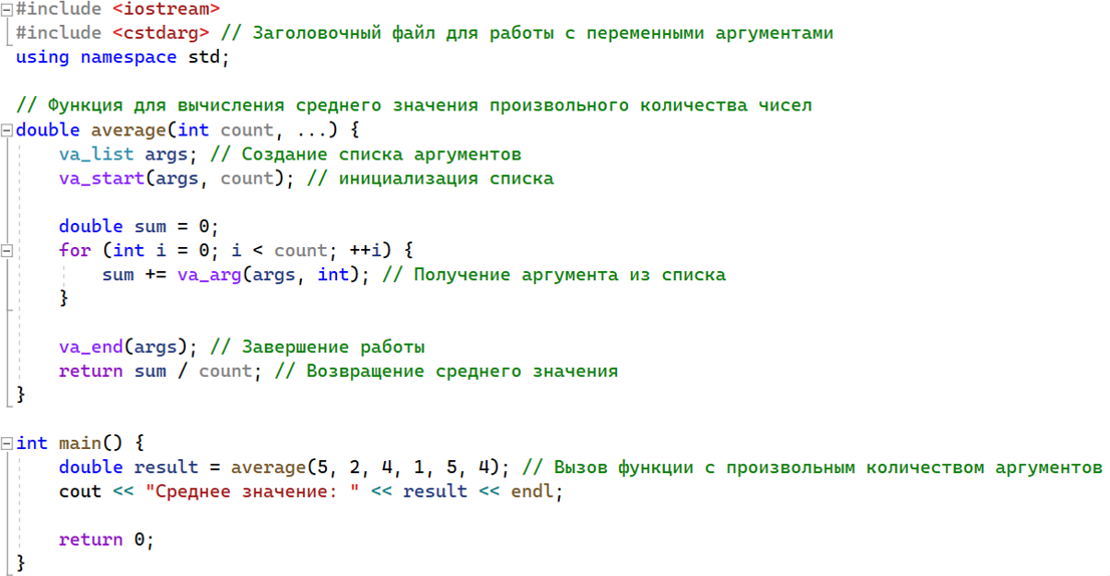


Рисунок 11.2 – Пример использования функции с переменным числом параметров

**11.2 Перегруженные функции**

Использование нескольких функций с одним и тем же именем, но с различными типами параметров, называется перегрузкой функций. Компилятор выбирает наиболее подходящую перегруженную функцию на основе переданных аргументов. На рисунке 11.3 определены 3 функции с одинаковым именем, но с разными типами параметров и возвращаемого значения.

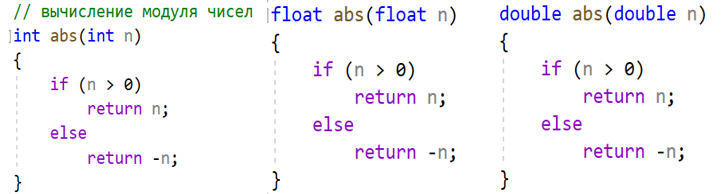


Рисунок 11.3 – Пример использования перегруженных функций

Правила описания перегруженных функций:

1. Перегруженные функции должны находиться в одной области видимости.
2. Функции перегружены, когда отличаются типы данных входных и выходных параметров.
3. Функции не могут быть перегружены, если описание их параметров отличается только модификатором const или использованием ссылки (например, int и const int или int и int &).

**11.3 Шаблоны функций**

Шаблоны – это конструкции языка программирования, которые позволяют создавать универсальный код, способный работать с различными типами данных. Они позволяют определить общую логику или алгоритм работы.  Шаблоны функций вводятся с целью автоматизировать процесс создания функций, которые могут работать с различными типами данных. Шаблоны позволяют создавать семейства функций, что обеспечивает единую логику (алгоритм) при работе с разными типами данных.[26] Тип данных, с которым работает шаблон, передается в качестве параметра.

Правила описания шаблона:

1. Описание любого шаблона начинается с ключевого слова *template*.
2. Параметры (аргументы) шаблона перечисляются в угловых скобках <…>.
3. Параметры шаблона обозначают типы данных, поэтому каждому параметру должно предшествовать ключевое слово *typename* или *class.*
4. Параметров шаблона может быть несколько.

Пример шаблонной функции, которая принимает два параметра с разными типами данных и выводит в консоль их размер в байтах, представлен на рисунке 11.4.

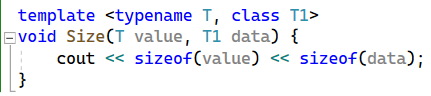


Рисунок 11.4 – Пример шаблонной функции

На рисунке 11.5 показано, что вместо 3-х перегруженных функций, которые вычисляют модуль значений разных типов данных, можно использовать одну шаблонную функцию, что значительно упрощает код.

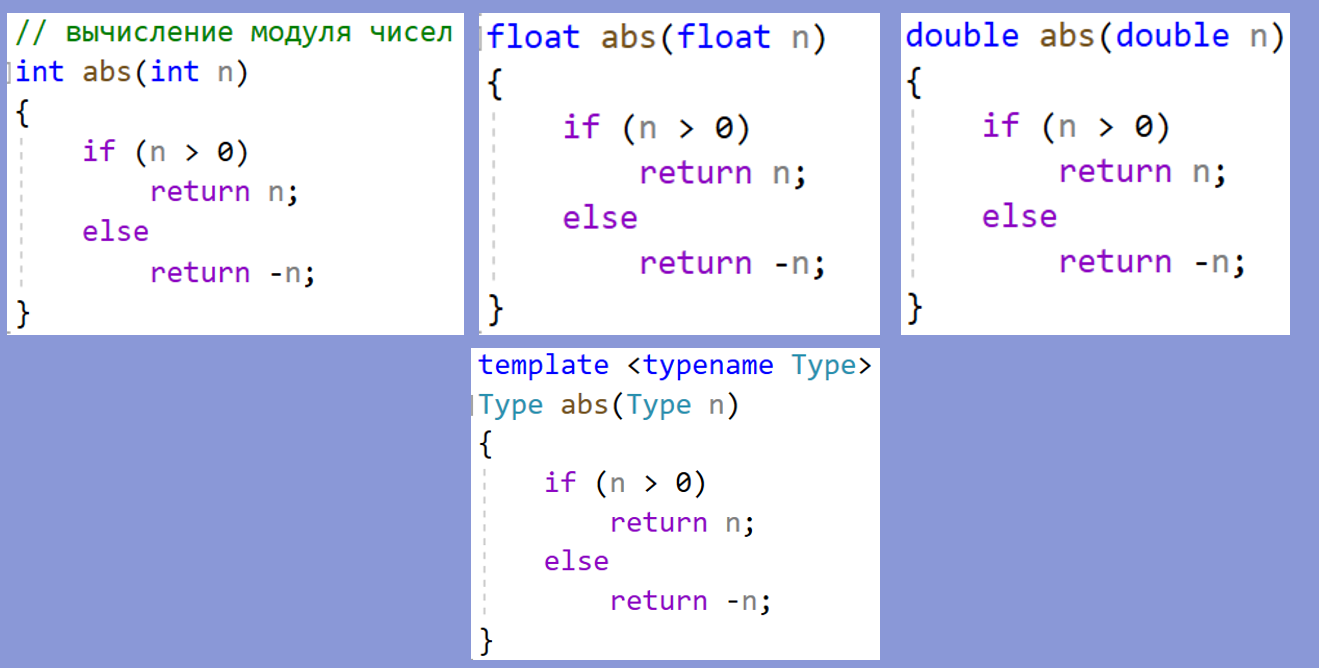


Рисунок 11.5 – Применение шаблонной функции

Основные свойства параметров шаблона функции:

1. В списке параметров шаблона может быть несколько параметров, каждому предшествует ключевое слово *typename* или *class*. При вызове шаблонной функции необходимо, чтобы типы фактических параметров соответствовали формальным параметрам.

На рисунке 11.6 в первом примере представлена шаблонная функция "maximum", которая принимает два аргумента – один с типом "T" и другой с типом "T1". В функции сравниваются значения аргументов и выводится максимальное значение. Здесь нет ошибки, потому что оба фактических параметра – "a" и "b" – имеют разные типы, которые соответствуют разным параметрам шаблона. То есть при создании функции по шаблону, вместо параметра T устанавливается тип int, а вместо параметра T1 устанавливается тип float [31].

Во втором примере реализована эта же самая функция, но принимает два аргумента одинакового типа "T". Здесь возникает ошибка. При вызове функции передаётся два параметра разных типов: "a" имеет тип "int", а "b" имеет тип "float". Фактические параметры не соответствуют формальным параметрам шаблона, что приводит к сбою компиляции, так как непонятно какой тип нужно установить вместо T.

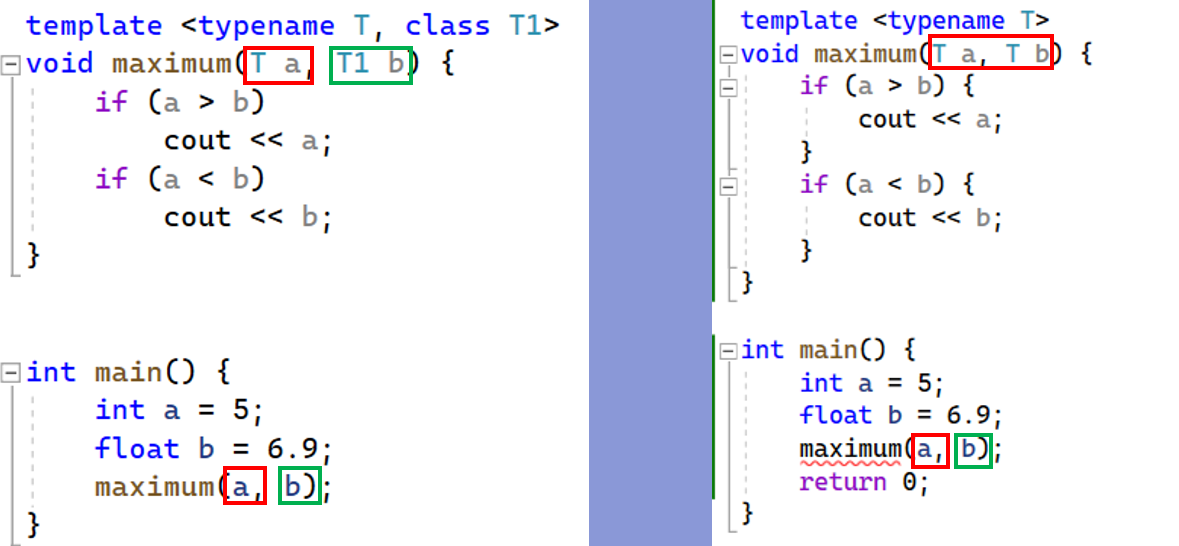


Рисунок 11.6 – Пример корректного и некорректного использования шаблонной функции

1. Имя параметра шаблона имеет в определяемой шаблоном функции все права имени типа. Это означает, что:

Имя параметра шаблона может быть использовано в качестве имени типа для определения параметров функции.

Имя параметра шаблона может быть использовано для определения типа возвращаемого функцией значения.

Имя параметра шаблона может быть использовано для определения переменных, констант или других объектов, локализованных в теле функции.

В примере на рисунке 11.7 в функции определены формальные параметры "a" и "b" с типом "T", который является именем параметра шаблона. Внутри функции объявлена переменная "c" типа "T". Затем происходит сравнение значений "a" и "b" и переменной "c" присваивается большее из них. Возвращаемое значение функции также имеет тип "T", который определяется параметром шаблона. Функция возвращает значение переменной "c", которое является максимальным из двух аргументов.

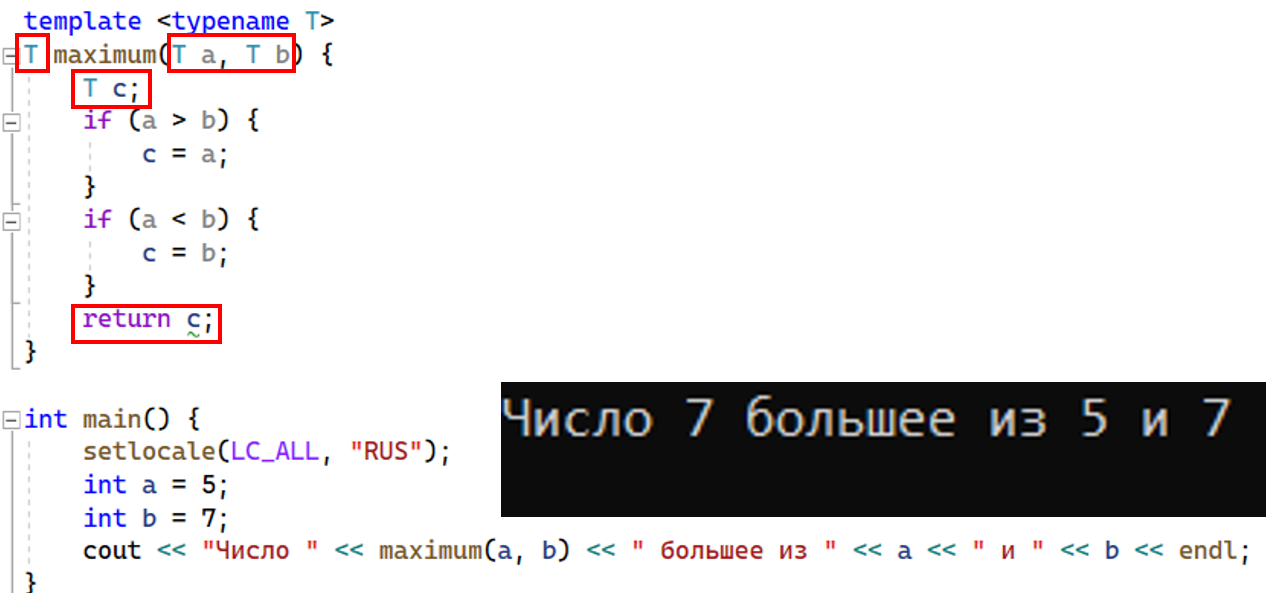


Рисунок 11.7 – Пример работы с шаблонной функцией

1. Параметризированная функция может иметь сколько угодно не параметризированных формальных параметров. В примере на рисунке 11.8 шаблонная функция "function 4" принимает три формальных параметра: "a" типа "T1", "b" типа "int" и "c" типа "bool". Обратите внимание, что типы параметров "b" и "c" не являются параметрами шаблона, они имеют стандартные типы данных.



Рисунок 11.8 – Пример стандартных параметров в шаблонной функции

1. Имена параметров шаблона не должны совпадать с именами параметров функции. В примере на рисунке 11.9 шаблонная функция принимает два параметра типа "T". Однако здесь возникает проблема: имя "T" имеет как параметр шаблона, так и параметр функции. Это приводит к путанице и ошибкам в коде. Для избегания таких ситуаций важно выбирать различные имена для параметров шаблона и формальных параметров функции.



Рисунок 11.9 – Пример некорректного использования параметров в шаблонной функции

Когда компилятор видит ключевое слово template и следующее за ним определение функции, то компилятор «запоминает» шаблон для будущего использования. Это означает, что компилятор распознает, что данная функция является шаблонной и должна быть обработана по-особому.

Генерация кода не происходит до тех пор, пока функция не будет реально вызвана в ходе выполнения программы. Компилятор не создает специализированный код для каждого возможного типа данных заранее, а откладывает эту задачу до момента фактического вызова функции с конкретными типами данных.

При первом вызове функции с конкретными типами параметров компилятор построит функцию с параметрами этого типа. Например, если есть шаблонная функция с параметром типа T, и вызывается для параметров с типом int, компилятор создаст специализированную функцию, куда вместо T подставит целочисленный тип данных. Этот процесс называется инстанцированием шаблона.

**11.4 Прототип функции**

Прототипы позволяют объявлять функции перед их определением. Это упрощает организацию кода и способствует читаемости программы. Обычно прототипы размещают в начале программы или в заголовочных файлах (.h). В примере на рисунке 11.10 можно увидеть прототип функции “summ”, которая выводит результат суммы двух целочисленных переменных, затем следует вызов функции и только потом определение, при этом компилятор не выдаёт ошибку. Если определение функции следует после вызова её, при этом не было бы прототипа, то программа бы не запустилась. Также стоит отметить, что в прототипе не обязательно указывать имена параметров, можно указать только типы, но при этом следование типов должно совпадать со следованием типов в определении.

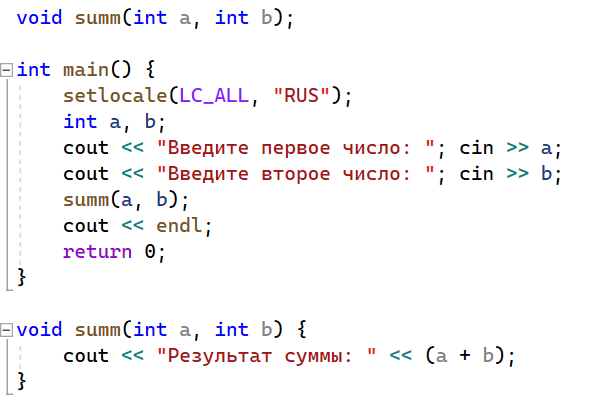


Рисунок 11.10 – Пример использования прототипа функции