# Лабораторная работа №1. Перегрузка функций. Шаблоны функций

**1 Цель работы**

Цель работы – изучить определение и варианты использования перегрузки функций и шаблонов функций в языке С++.

**2 Порядок выполнения работы**

* ознакомиться с описанием лабораторной работы;
* получить задание у преподавателя по вариантам;
* разработать и отладить программу;
* составить и защитить отчет по лабораторной работе у преподавателя.

**3 Содержание отчета**

* титульный лист;
* краткое теоретическое описание;
* задание на лабораторную работу, включающее математическую формулировку задачи;
* результаты выполнения работы, включающие схему алгоритма, тексты программ, результаты вычислений.

**4 Краткая теория**

**4.1 Перегрузка функций**

С++ позволяет определить несколько функций с одним и тем же именем, если эти функции имеют разные наборы параметров (по меньшей мере разные типы параметров). Эта особенность называется *перегрузкой функции*. При вызове перегруженной функции компилятор С++ определяет соответствующую функцию путем анализа количества, типов и порядка следования аргументов в вызове. Перегрузка функции обычно используется для создания нескольких функций с одинаковым именем, предназначенных для выполнения сходных задач, но с разными типами данных.

Перегруженные функции различаются компилятором с помощью их сигнатуры – комбинации имени функции и типов ее параметров. Компилятор кодирует идентификатор каждой функции по числу и типу ее параметров, чтобы иметь возможность осуществлять надежное связывание типов. Надежное связывание типов гарантирует, что вызывается надлежащая функция и что аргументы согласуются с параметрами.

При использовании перегруженных функций нужно с осторожностью задавать начальные значения их параметров. Предположим, мы следующим образом определили перегруженную функцию умножения разного количества параметров:

**double multy(double x) {return x\*x\*x;}**

**double multy(double x, double y) {return x\*y\*y;}**

**double multy(double x, double y, double z)**

**{return x\*y\*z;}**

Каждое из следующих обращений к функции **multy()** будет однозначно идентифицировано и правильно обработано:

**multy(0.4)**

**multy(4.0,12.3)**

**multy(0.1e-6,1.2e4,6.4)**

Однако добавление в программу такой функции прототипа с начальными значениями параметров:

**double multy(double a=1.0, double b=1.0,**

**double c=1.0, double d=1.0)**

**{return a\*b+c\*d;}**

навсегда запутает любой компилятор при попытках обработать, например, такой вызов:

**multy(0.1e-6,1.2e4)**

**4.2 Шаблоны функций**

Перегруженные функции обычно используются для выполнения сходных операций над различными типами данных. Если операции идентичны для каждого типа, это можно выполнить более компактно и удобно, используя *шаблоны функций*. Достаточно написать одно единственное определение шаблона функции. Основываясь на типах аргументов, указанных в вызовах этой функции, С++ автоматически генерирует разные функции для соответствующей обработки каждого типа. Таким образом, определение единственного шаблона определяет целое семейство решений.

Все определения шаблонов функций начинаются с ключевого слова **template**, за которым следует список формальных параметров функции, заключенный в угловые скобки (**<**) и (**>**). Каждый формальный тип параметра предваряется служебным словом **class**. Формальные типы параметров – это встроенные типы или типы, определяемые пользователем. Они используются для задания типов аргументов функции, для задания типов возвращаемого значения функции и для объявления переменных внутри тела функции. После шаблона следует обычное описание функции. Пример определения шаблон функций, вычисляющих абсолютные значения числовых величин разных типов:

**template <class type>**

**type abs (type x) { return x>0 ? x: -x; }**

Шаблон семейства функций состоит из двух частей – заголовка шаблона:

**template <список\_параметров\_шаблона>**

и из обыкновенного определения функции. Имена параметров шаблона могут использоваться и в теле определения функции для обозначения типов локальных объектов.

По существу, механизм шаблонов функций позволяет автоматизировать подготовку определений перегруженных функций. При использовании шаблонов уже нет необходимости готовить заранее все варианты функций с перегруженным именем. Компилятор автоматически, анализируя вызовы функций в тексте программы, формирует необходимые определения именно для таких типов параметров, которые использованы в обращениях. Дальнейшая обработка выполняется так же, как и для перегруженных функций.

Исполнение

Компиляция

Шаблон семейства функций

Типы фактических параметров в вызове функции

Текст конкретного определения функции

Значения фактических параметров в вызове функции

Исполнение функции

Рисунок 1.1 – Схема параметризации шаблона функций

Можно считать, что параметры шаблона функций являются формальными параметрами, а типы тех параметров, которые используются в конкретных обращениях к функции, служат фактическими параметрами шаблона. Именно по ним выполняется параметрическая настройка и с учетом этих типов генерируется конкретный текст определения функции (см. рис. 2).

Перечислим основные свойства параметров шаблона.

1. Имена шаблона должны быть уникальными во всем определении шаблона.
2. Список параметров шаблона не может быть пустым, так как при этом теряется возможность параметризации и шаблон функций становится обычным определением конкретной функции.
3. В списке параметров шаблона функции может быть несколько параметров. Каждый из них должен начинаться служебным словом **class**. Например, допустим такой заголовок шаблона:

**template <class type1, class type2>**

Соответственно, неверен заголовок:

**template <class type1, type2, type3>**

1. Недопустимо использовать в заголовке шаблона параметры с одинаковыми именами, т.е. ошибочен такой заголовок:

**template <class t, class t, class t>**

1. Имя параметра шаблона имеет в определяемой шаблоном функции все права имени типа, т.е. с его помощью могут специализироваться формальные параметры, определяться тип возвращаемого функцией значения и типы любых объектов, локализованных в теле функции. Имя параметра шаблона видно во всем определении и скрывает другие использования того же идентификатора в области, глобальной по отношению к данном шаблону функции. Все параметры шаблона функций должны быть обязательно использованы в спецификациях параметров определения функции. Определяемая с помощью шаблона функция может иметь любое количество непараметризованных формальных параметров. Может быть не параметризовано и возвращаемое функцией значение. При использовании шаблонов функций возможна перегрузка как шаблонов, так и функций. Могут быть шаблоны с одинаковыми именами, но разными параметрами. Или с помощью шаблона может создаваться функция с таким же именем, что и явно определенная функция. В обоих случаях «распознавание» конкретного вызова выполняется по сигнатуре, т.е. по типам, порядку и количеству фактических параметров.

**5 Примеры программ**

1. Программа, вычисляющая среднее значение элементов массивов типа **int** и **float**, а также матриц типа **int**.

**#include <iostream>**

**using namespace std;**

**//вычисление среднего для массивов типа int**

**float sred(int n, int x[])**

**{**

**float s = 0;**

**for (int i=0; i<n; i++)**

**s+=x[i];**

**s/=n;**

**return s;**

**}**

**//вычисление среднего для массивов типа float**

**float sred(int n, float x[])**

**{**

**float s = 0;**

**for (int i=0; i<n; i++)**

**s+=x[i];**

**s/=n;**

**return s;**

**}**

**//вычисление среднего для матриц типа int**

**float sred(int m, int n, int \*\*x)**

**{**

**float s = 0;**

**for (int i=0; i<m; i++)**

**for (int j=0; j<n; j++)**

**s+=x[i][j];**

**s/=(m\*n);**

**return s;**

**}**

**void main()**

**{**

**//формируем массив типа int**

**int a[] = {1,2,3,4,5};**

**//определяем количество элементов массива**

**int k = sizeof(a)/sizeof(a[0]);**

**//вызываем для него функцию**

**cout<<"For int sred = "<<sred(k, a)<<endl;**

**//аналогично для массива типа float**

**float b[] = {10.2,5.3,3.4,1.2};**

**k = sizeof(b)/sizeof(b[0]);**

**cout<<"For float sred = "<<sred(k, b)<<endl;**

**//формируем динамически матрицу**

**int m=3, n=4, \*\*matr;**

**matr = new int\* [m];**

**for (int i=0; i<m; i++)**

**{**

**matr[i] = new int[n];**

**for (int j=0; j<n; j++)**

**matr[i][j] = i+j;**

**}**

**//вызываем функцию для матрицы**

**cout<<"For matrix sred = "<<sred(m,n,matr)<<endl;**

**}**

2. Разработать шаблон функции поиска заданного значения в массиве. Функция должна возвращать индекс найденного элемента, либо –1, если аргумент не найден.

**#include <iostream>**

**using namespace std;**

**//шаблон функции поиска в массиве**

**template <class T>**

**int find(int n, T mas[], T arg)**

**{**

**for (int i=0; i<n; i++)**

**if (mas[i]==arg)**

**return i;**

**return -1;**

**}**

**void main()**

**{**

**//формируем целочисленный массив**

**int int\_arg, a[] = {1,2,3,4,5,6,7};**

**int k = sizeof(a)/sizeof(a[0]);**

**//вводим аргумент для поиска**

**cout<<"int\_arg = ";**

**cin>>int\_arg;**

**//вызываем функцию поиска**

**int idx = find(k,a,int\_arg);**

**//анализируем результат вызова**

**if (idx==-1)**

**cout<<"No element!\n";**

**else**

**cout<<"Index = "<<idx<<endl;**

**//аналогичные действия для массива типа float**

**float float\_arg, b[] = {1.0,2.0,3.0,4.0,5.0};**

**k = sizeof(a)/sizeof(a[0]);**

**cout<<"float\_arg = ";**

**cin>>float\_arg;**

**idx = find(k,b,float\_arg);**

**if (idx==-1)**

**cout<<"No element!\n";**

**else**

**cout<<"Index = "<<idx<<endl;**

**}**

**6 Контрольные вопросы**

1. Как и с какой целью используется перегрузка функций?
2. Каким образом выполняется распознавание перегруженных функций при их вызове?
3. Можно ли при использовании перегруженных функций задавать начальные значения их параметров?
4. Каким образом описывается шаблон семейства функций?
5. Опишите схему параметризации шаблона функций.
6. Перечислите основные свойства параметров шаблона.
7. Может ли созданная с помощью шаблона функция иметь то же имя, что и явно определенная функция?

**7 Варианты заданий для самостоятельного решения**

Во всех вариантах требуется перегрузить заданную функцию и описать шаблон функций.

1. Функция суммирования одномерных массивов типа **int** и **float** и сцепления строк.

2. Функция, которая меняет местами минимальный и максимальный элементы массива.

3. Функция, вычисляющая площадь круга по его радиусу, прямоугольника и треугольника по их сторонам.

4. Функция, возвращающая минимальный из трех передаваемых в нее параметров любого (но одинакового) типа.

5. Функция, вычисляющая периметр круга по его радиусу, прямоугольника и треугольника по их сторонам.

6. Функция, формирующая новый массив, упорядоченный по неубыванию, из двух массивов-параметров, уже упорядоченных по неубыванию.

7. Функции для ввода и вывода на экран массивов и матриц.

8. Функция, формирующая массив, каждый элемент которого равен минимальному из соответствующих значений двух других массивов – параметров.

9. Функция вычисления расстояния между двумя точками в двумерном и трехмерном пространстве.

10. Функция сортировки элементов массива методом пузырька.

11. Функция поиска минимального элемента в массивах и матрицах различного типа.

12. Функция, меняющая местами значения двух переданных в нее параметров.

13. Функция определения количества нулей в массивах и матрицах различного типа.

14. Функция, находящая минимальный элемент в массиве и заменяющая его заданным числом.

15. Функция определения пройденного расстояния при равномерном и равноускоренном движениях тела.

16. Функция, подсчитывающая сумму положительных элементов в массиве.

17. Функция вычисления площади треугольников: равностороннего (по стороне), равнобедренного (по основанию и высоте), обычного (по трем сторонам).

18. Функция, вычисляющая произведение отрицательных элементов массива.

19. Функция определения объемов: шара по его радиусу (*V=4/3πR2*), конуса по радиусу основания и высоте (*V=1/3πr2h*), усеченного конуса по радиусам двух оснований и высоте (*V=1/3πh(r2+r12+rr1)*).

20. Функция, находящая среднее арифметическое элементов массива.

21. Функции заполнения массивов и матриц случайными числами заданного диапазона и вывода их на экран.

22. Функция, которая считает сумму элементов главной диагонали матрицы.

23. Функция определения площади поверхности: шара по его радиусу (*S=4πR2*), цилиндра по радиусу основания и высоте (*S=2πr(r+h)*), усеченного конуса по радиусам двух оснований и образующей (*S=π((r+r1)+r2+r12)*).

24. Функция, находящая среднеквадратическое значение элементов матрицы.

25. Функция умножения массива и матрицы на число.

26. Функция сложения двух матриц.

27. Функция нахождения разности двух массивов и матриц.

28. Функция, подсчитывающая количество строк матрицы, сумма элементов которых больше некоторого заданного числа (это число имеет тот же тип, что и элементы матрицы).