Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение   
высшего образования   
«Владимирский государственный университет   
имени Александра Григорьевича и Николая Григорьевича Столетовых»  
(ВлГУ)

Институт прикладной математики, физики и информатики

Кафедра физики и прикладной математики

ЛАБОРАТОРНАЯ РАБОТА №4  
по дисциплине  
«Основы программирования»  
на тему:  
«Шаблоны функций»

Выполнил:  
ст. гр. ПМИ-123

Рушев А.М.  
  
  
Принял:  
ст. преподаватель   
каф. ФиПМ  
Шишкина М.В.

Владимир,2024

**Цель работы**

Расширение навыков работы с функциями на языке программирования С++, получение опыта создания шаблонов функций. Закрепления навыков написания кода на языке С++ с использованием функций, способов передачи параметров в функции, возвращения значения из функции.

**Постановка задачи**

Написать шаблон функции сортировки двумерного массива. Вызвать функцию для целочисленного, вещественного и символьного двумерных массивов. Результаты сортировки массивов отобразить на экране (вывести массивы в виде таблицы).

**Теоретическая часть**

Шаблоны (templates) в C++ представляют собой мощный механизм, который позволяет создавать обобщенные классы и функции, работающие с разными типами данных. Использование шаблонов позволяет писать универсальный код, который может быть использован для различных типов данных без необходимости создания отдельных версий для каждого типа.

Шаблоны функций позволяют объявить функцию без указания конкретного типа данных, которые она будет принимать или возвращать. Это делается с использованием шаблонного параметра (template parameter), который указывается перед именем функции или класса в угловых скобках. Например:

template <typename T>

T max(T a, T b) {

return a > b ? a : b;

}

Здесь T - шаблонный параметр, который будет заменен на конкретный тип данных при вызове функции.

Шаблоны классов позволяют создавать обобщенные классы, которые могут работать с различными типами данных. Шаблонный параметр указывается после имени класса. Пример:

template <typename T>

class Array {

private:

T\* data;

int size;

public:

// Конструктор и методы класса

};

При создании экземпляра шаблонного класса, необходимо указать конкретный тип данных, с которым будет работать класс:

Array<int> intArray(10); // Создание объекта Array для типа int

Array<double> doubleArray(5); // Создание объекта Array для типа double

Шаблоны в C++ позволяют увеличить гибкость и эффективность кода, уменьшить дублирование кода и сделать программу более обобщенной и легко поддерживаемой. Однако использование шаблонов также может повлечь за собой увеличение времени компиляции и сложности отладки.

**Практическая часть**

**Листинг кода программы:**

#include <iostream>

using namespace std;

const int n = 3;

const int m = 5;

template<class T> void SortMassiv(T\*\* mass, int n, int m) {

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

{

bool flag = false;

for (int j = 0; j < m - 1; j++)

{

for (int k = 0; k < m - j - 1; k++) {

if (mass[i][k] > mass[i][k + 1])

{

flag = true;

T temp = mass[i][k];

mass[i][k] = mass[i][k + 1];

mass[i][k + 1] = temp;

}

}

if (!flag) {

break;

}

}

}

}

template<class T> void Print(T arr, int n, int m) {

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

for (int j = 0; j < m; j++) {

cout << arr[i][j] << "\t";

}

cout << endl;

}

}

int main()

{

setlocale(LC\_ALL, "rus");

cout << "Лабораторная работа №4" << endl;

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

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

mass[i] = new int[m];

}

//Заполнение массива целых чисел

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

for (int j = 0; j < m; j++) {

mass[i][j] = 1 + rand() % 50;

}

}

cout << "Исходный массив:" << endl;

Print(mass, n, m);

SortMassiv(mass, n, m);

cout << "Отсортированный массив:" << endl;

Print(mass, n, m);

float\*\* mass2 = new float\* [m];

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

mass2[i] = new float[m];

}

//Заполнение массива вещественных чисел

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

for (int j = 0; j < m; j++) {

mass2[i][j] = static\_cast<float>(rand()) / (static\_cast<float>(RAND\_MAX / 100.0));

}

}

cout << "Исходный массив:" << endl;

Print(mass2, n, m);

SortMassiv(mass2, n, m);

cout << "Отсортированный массив:" << endl;

Print(mass2, n, m);

char\*\* mass3 = new char\* [m];

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

mass3[i] = new char[m];

}

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

for (int j = 0; j < m; j++) {

mass3[i][j] = rand() % 26 + 'A';

}

}

cout << "Исходный массив:" << endl;

Print(mass3, n, m);

SortMassiv(mass3, n, m);

cout << "Отсортированный массив:" << endl;

Print(mass3, n, m);

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

delete[] mass[i];

delete[] mass2[i];

delete[] mass3[i];

}

delete[] mass;

system("pause");

return 0;

}

**Результат выполнения кода программы:**

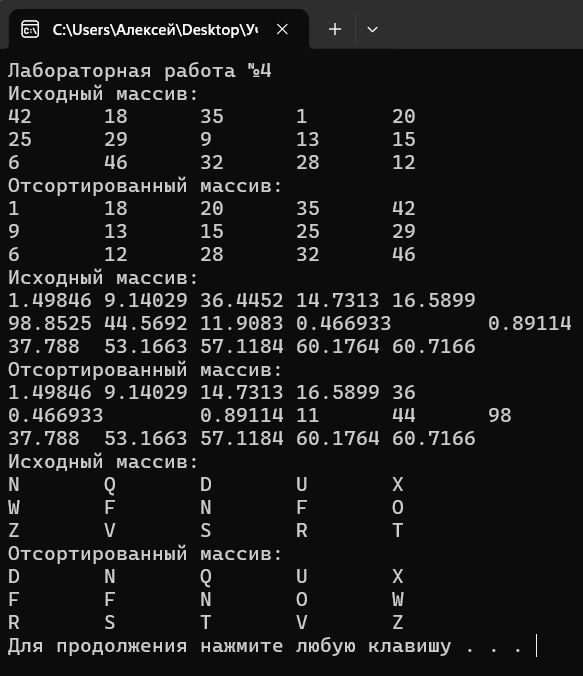


Рисунок 1 – Результат выполнения кода программы

**Вывод:** в результате выполнения задания удалось значительно улучшить свои навыки работы с функциями на языке программирования C++. Создание шаблонов функций стало доступным и понятным после практического применения. Закрепление навыков написания кода с использованием функций позволило более глубоко понять особенности передачи параметров в функции и возвращения значений из них. Полученный опыт безусловно будет полезен для дальнейшей работы над проектами на C++ и позволит эффективнее разрабатывать программное обеспечение.

Плюсы использования шаблонов функции в C++:

1. Универсальность: шаблоны позволяют создавать обобщенный код, который может работать с разными типами данных без необходимости написания отдельных функций для каждого типа.
2. Гибкость: использование шаблонов позволяет создавать функции, которые могут принимать различное количество аргументов или разные типы аргументов.
3. Увеличение производительности: компилятор генерирует отдельные версии функции для каждого типа данных, что может привести к более эффективной и оптимизированной работе программы.
4. Удобство использования: шаблоны позволяют создавать функции, которые могут быть использованы для различных типов данных без необходимости изменения кода.

Минусы использования шаблонов функции в C++:

1. Увеличение размера исполняемого файла: компилятор генерирует отдельные версии функций для каждого используемого типа данных, что может привести к увеличению размера исполняемого файла.
2. Увеличенные временные затраты на компиляцию: компилятор должен генерировать код для каждой версии функции, что может привести к увеличенному времени компиляции программы.
3. Усложнение отладки: ошибки в коде шаблона могут быть сложными для обнаружения и исправления из-за сложности генерируемого компилятором кода.

В целом, использование шаблонов функции в C++ имеет больше плюсов, чем минусов, и может значительно упростить и улучшить структуру программы.