Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Институт информационных технологий

Факультет компьютерных технологий

Кафедра информационных систем и технологий

**Лабораторная работа №3**

по дисциплине

«Конструирование ПО»

|  |  |
| --- | --- |
|  | Выполнил студент группы 181073  Наркевич Михаил Викторович |
|  | Проверил Сицко Владимир Александрович |

Минск 2022

**Лабораторная работа №3**

**Тема работы:** Шаблоны функций и классов.

**Цель работы:** Создание консольной программы для реализации шаблонов функций и классов, включая: явные специализации шаблонных функций и классов, стандартные типы в шаблонных классах.

**Общие требования к выполнению работы**

1.     Рассмотреть теоретические сведения по теме лабораторной работы.

2.     Создать проект консольной программы.

3.     Создать шаблонную функцию или класс согласно заданию.

4.     Реализовать явную специализацию шаблонной функции или класса для строк символов (char \*).

5.     В шаблонных классах использовать стандартные типы и аргументы по умолчанию. Данные в шаблонных классах хранить в статическом массиве. Набор методов класса должен обеспечить основные операции с элементами объекта (добавление, удаление, поиск, просмотр, упорядочивание и т.п.) и операции над однотипными объектами (сравнение, сложение и т.п.).

6.     Дополнительно рассмотреть реализацию наследования шаблонных классов (например, от простого базового класса).

7.     В функции main выполнить действия с шаблонной функцией или классом, которые продемонстрируют работу методов.

8.     Отладить и выполнить полученную программу. Проверить использование специализаций, стандартных типов и аргументов по умолчанию.

9.     Создать отчет, включающий задание, листинг программы и результаты ее выполнения.

Ход работы:

Тема: Класс «Множество»

Код программы:

#include <iostream>

using namespace std;

// шаблонный класс Матрица

template <typename T>

class MATRIX

{

private:

T\*\* M; // матрица

int m; // количество строк

int n; // количество столбцов

public:

// конструкторы

MATRIX()

{

n = m = 0;

M = nullptr; // необязательно

}

// конструктор с двумя параметрами

MATRIX(int \_m, int \_n)

{

m = \_m;

n = \_n;

// Выделить память для матрицы

// Выделить пам'ять для массива указателей

M = (T\*\*) new T \* [m]; // количество строк, количество указателей

// Выделить память для каждого указателя

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

M[i] = (T\*)new T[n];

// заполнить массив M нулями

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

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

M[i][j] = 0;

}

// Конструктор копирования - обязательный

MATRIX(const MATRIX& \_M)

{

// Создается новый объект для которого виделяется память

// Копирование данных \*this <= \_M

m = \_M.m;

n = \_M.n;

// Выделить память для M

M = (T\*\*) new T \* [m]; // количество строк, количество указателей

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

M[i] = (T\*) new T[n];

// заполнить значениями

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

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

M[i][j] = \_M.M[i][j];

}

// методы доступа

T GetMij(int i, int j)

{

if ((m > 0) && (n > 0))

return M[i][j];

else

return 0;

}

void SetMij(int i, int j, T value)

{

if ((i < 0) || (i >= m))

return;

if ((j < 0) || (j >= n))

return;

M[i][j] = value;

}

// метод, выводящий матрицу

void Print(const char\* ObjName)

{

cout << "Object: " << ObjName << endl;

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

{

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

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

cout << endl;

}

cout << "---------------------" << endl << endl;

}

// оператор копирования - обязательный

MATRIX operator=(const MATRIX& \_M)

{

if (n > 0)

{

// освободить память, выделенную ранее для объекта \*this

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

delete[] M[i];

}

if (m > 0)

{

delete[] M;

}

// Копирование данных M <= \_M

m = \_M.m;

n = \_M.n;

// Выделить память для M опять

M = (T\*\*) new T \* [m]; // количество строк, количество указателей

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

M[i] = (T\*) new T[n];

// заполнить значениями

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

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

M[i][j] = \_M.M[i][j];

return \*this;

}

// Деструктор - освобождает память, выделенную для матрицы

~MATRIX()

{

if (n > 0)

{

// освободить выделенную память для каждой строки

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

delete[] M[i];

}

if (m > 0)

delete[] M;

}

};

void main()

{

// тест для класса MATRIX

MATRIX<int> M(2, 3);

M.Print("M");

// Заполнить матрицу значеннями по формуле

int i, j;

for (i = 0; i < 2; i++)

for (j = 0; j < 3; j++)

M.SetMij(i, j, i + j);

M.Print("M");

MATRIX<int> M2 = M; // вызов конструктора копирования

M2.Print("M2");

MATRIX<int> M3; // вызов оператора копирования - проверка

M3 = M;

M3.Print("M3");

MATRIX<int> M4;

M4 = M3 = M2 = M; // вызов оператора копирования в виде "цепочки"

M4.Print("M4");

}

Результат выполнения программы представлен на рисунке 1.

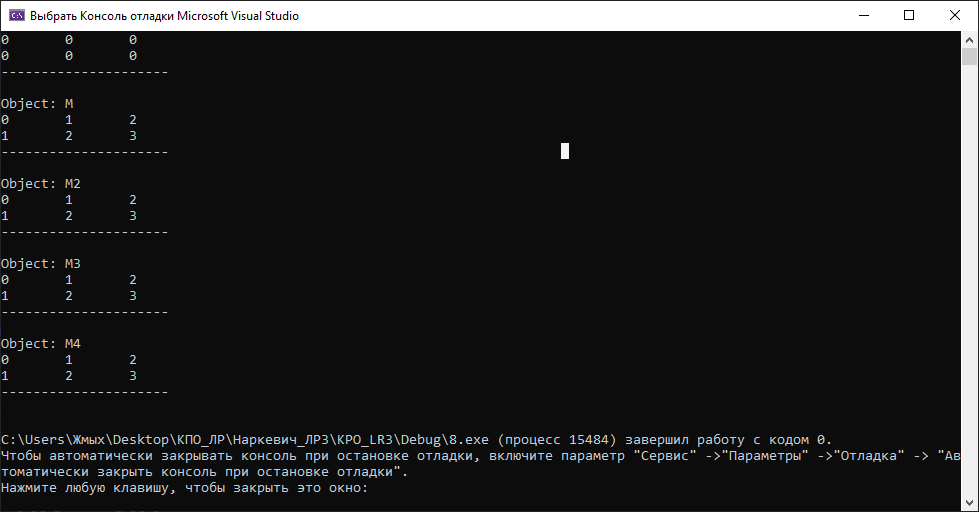


Рисунок 1 – Результат работы программы

**Контрольные вопросы**

1.     Шаблоны функций.

В языке C++ шаблоны функций — это функции, которые служат образцом для создания других подобных функций. Главная идея — создание функций без указания точного типа(ов) некоторых или всех переменных. Для этого мы определяем функцию, указывая тип параметра шаблона, который используется вместо любого типа данных. После того, как мы создали функцию с типом параметра шаблона, мы фактически создали «трафарет функции».

2.     Шаблоны классов.

Шаблон класса (class template) позволяет задать тип для объектов,

используемых в классе.

3.     Явные специализации шаблонных функций.

При создании [экземпляра шаблона функции](https://ravesli.com/urok-174-ekzemplyary-shablonov-funktsij/) для определенного типа данных компилятор копирует шаблон функции и заменяет параметр типа шаблона функции на фактический (передаваемый) тип данных. Это означает, что все экземпляры функции имеют одну реализацию, но разные типы данных. Хотя в большинстве случаев это именно то, что требуется, иногда может понадобиться, чтобы реализация шаблона функции для одного типа данных отличалась от реализации шаблона функции для другого типа данных. Специализация шаблонов именно для этого и предназначена.

4.     Явные специализации шаблонных классов.

Специализация шаблона класса (или «явная специализация шаблона

класса») позволяет специализировать шаблон класса для работы с определенным типом данных (или сразу с несколькими типами данных, если есть несколько параметров шаблона).

5.     Стандартные типы в шаблонных классах.

В шаблонных классах также можно использовать переменные стандартных типов (например, в качестве полей класса)

Вывод: в ходе проделанной лабораторной работы было изучено и применено на практике использование шаблонов, являющиеся мощным инструментом, позволяющим соблюдать DRY принцип при написании кода, а также имеет полезный механизм явной специализации для классов и функций.