Работа 4.

Тема: Шаблоны. Конструктор копирования. Перегрузка операций

Задача: Создание шаблонного класса Матрица

Создать шаблонный класс Матрица. Размер матрицы m*n. Тип элементов задается через параметр шаблона. Память для матрицы выделяется динамически.

Элементы класса:

- переменная M типа «указатель на указатель». Эта переменная определяет матрицу. Память для матрицы будет выделяться динамически;
- переменные m, n – это размерность матрицы M;
- конструктор по умолчанию (без параметров);
- конструктор с двумя параметрами – создает матрицу размером m*n. В конструкторе выделяется память для столбцов и строк матрицы. Значение каждого элемента матрицы устанавливается в 0;
- конструктор копирования Matrix (Matrix &). Этот конструктор необходим для создания копии объекта-матрицы из другого объекта-матрицы;
- методы чтения/записи элементов матрицы GetM(), SetM();
- метод Print() – вывод матрицы;
- оператор копирования operator=(Matrix &). Этот оператор перегружает оператор присваивания = и предназначен для корректного копирования объектов, например, obj2=obj1;
- деструктор.

II. В качестве демонстрационного примера написать 2 варианта программы:

Вариант 1

```
int main()
{

  Matrix <int> M(3, 4);
  M.Print("M");

  // Заполнить матрицу значениями по
формуле
  int i, j;
  for (i = 0; i < 2; i++)
    for (j = 0; j < 3; j++)
      M.SetM(i, j, i + j);

  M.Print("M");

  Matrix < int > M1 = M; // вызов
конструктора копирования
  M1.Print("M1");
```

Результат работы программы (для варианта 1):

```
Object: M
0     0     0     0
0     0     0     0
0     0     0     0
----------------------
Object: M
0     1     2     0
1     2     3     0
0     0     0     0
----------------------
```

```
  Matrix < int > M2;
  M2 = M; // вызов оператора копирования
- проверка
  M2.Print("M2");

  Matrix < int > M3;
  M3 = M2 = M1 = M; // вызов оператора
копирования в виде "цепочки"
  M3.Print("M3");
}
```

```
Object: M1
0     1     2     0
1     2     3     0
0     0     0     0
---------------------
Object: M2
0     1     2     0
1     2     3     0
0     0     0     0
---------------------
Object: M3
0     1     2     0
1     2     3     0
0     0     0     0
---------------------
```

Вариант 2

Такая же программа, как в варианте 1, но тип элементов матрицы - double:

```
int main()
{

  Matrix <double> M(3, 4);
  M.Print("M");

  // Заполнить матрицу значениями по
формуле
  int i, j;
  for (i = 0; i < 2; i++)
    for (j = 0; j < 3; j++)
      M.SetM(i, j, (i + j)*0.5);

  M.Print("M");

  Matrix <double> M1 = M; // вызов
конструктора копирования
  M1.Print("M1");

  Matrix <double> M2;
  M2 = M; // вызов оператора копирования
- проверка
  M2.Print("M2");

  Matrix <double> M3;
```

Результат работы программы (для варианта 2):

```
Object: M
0     0     0     0
0     0     0     0
0     0     0     0
---------------------
Object: M
0     0.5   1     0
0.5   1     1.5   0
0     0     0     0
---------------------
Object: M1
0     0.5   1     0
0.5   1     1.5   0
0     0     0     0
---------------------
Object: M2
0     0.5   1     0
0.5   1     1.5   0
0     0     0     0
---------------------
```

M3 = M2 = M1 = M; // вызов оператора копирования в виде "цепочки"
M3.Print("M3");
}
	Object: M3
	0 0.5 1 0
	0.5 1 1.5 0
	0 0 0 0
	---------- 	

Реализация класса для приложения типа Console Application имеет следующий вид

```
#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;
  }
};

int 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

Object: M

0 0 0

0 0 0

Object: M

0 1 2

1 2 3

Object: M2

0 1 2

1 2 3

Object: M3

0 1 2

1 2 3

Object: M4

0 1 2

1 2 3

Итог. Если память в классе выделяется динамически, то обязательно нужно реализовывать собственный конструктор копирования и оператор копирования.