Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Рязанский государственный радиотехнический университет
имени В. Ф. Уткина»

Кафедра «Вычислительная и прикладная математика»

Отчет
по лабораторной работе № 3

по дисциплине

«Объектно-ориентированное программирование»

на тему

«Наследование, статические члены, перегрузка операторов»

Выполнил:
студент гр. 143
Вербицкая И. С.

Проверил:
Антипов О.В.

Рязань 2022

Задание

В данной лабораторной работе необходимо изменить класс, созданный в лабораторной работе №2 следующим образом:

1. Добавить для методов параметры-ссылки там, где это необходимо.
2. Добавить константные методы и параметры там, где это оправданно.
3. Заменить методы, реализующие операции над сущностью, перегруженными операторами (соответствующими смыслу операции). Например операция add() заменяется оператором +.
4. Добавить статический метод для класса, возвращающий новый объект, заполненный случайными значениями из заданного диапазона (который задаётся через параметры данного метода). Для бинарных значений диапазон не указывается.
5. Добавить наследование от базового класса Object, в котором должна быть реализация следующего функционала:
    a. Подсчёт количества созданных объектов и количества активных объектов на данный момент через статические поля класса. Статический метод printTotalInfo для вывода этой информации в консоль.
    b. Функционал для создания **динамического** массива (через malloc или через new) – списка всех выполненных операций с объектом. При выполнении каждой операции её обозначение заносится в данный массив через метод addOp. Метод clearOp – очищает массив операций. Метод printOp – выводит полный список всех выполненных операций с объектом.
    c. Конструктор копии и оператор присваивания (=) для класса Object.
6. Дополнить класс юнит-тестирования **минимум 5 тестами** для проверки работы базового класса Object и статического метода для случайной генерации класса сущности.

Важно: при перегрузке оператора присваивания (=) необходимо возвращать ссылку на текущий объект (*this), чтобы можно было их использовать в цепочке вызовов. Также стоит всегда проверять на самоприсваивание (особенно при наличии выделения динамической памяти), и не выполнять лишних действий, сразу возвращая ссылку на текущий объект.

Описание структуры программы

Класс Object

Поля класса:

- static int All – количество созданных объектов
- static int Act - количество активных объектов
- int* Op – указатель на динамический массив операций, выполненных с объектом (каждой операции соответствует свой целочисленный номер)
- int Ops – количество операций в массиве Op

Методы класса:

- static void RestartObjInfo() – сбрасывает информацию о количестве объектов класса (созданных и активных)
- Object() – конструктор класса
- Object(const Object& Obj) – конструктор копирования класса
- ~Object() – деструктор класса
- static void PrintTotalInfo() – выводит в консоль информацию о количестве объектов класса (созданных и активных)
- static int GetAll() – геттер поля All
- static int GetAct() – геттер поля Act

- void AddOp(int k) – добавляет команду k в динамический массив операций Op, выделяя под неё память
- void PrintOp() – выводит список операций текущего объекта
- void ClearOp() – сбрасывает список операций над объектом, освобождая выделенную память
- Object& operator= (Object& Matr) – перегруженный оператор присваивания для класса
- bool Comp(int (&Spis)[N]) – вспомогательная функция, возвращающая 1 в случае, если массив операций Op совпадает с массивом Spis

Класс Matrix3x3 – класс-потомок класса Object

Поля класса:

- arr - двумерный массив действительных чисел размерностью 3 на 3
- id - логическое значение; 1 – если матрица единичная; 0 – если нет

Методы класса:

- bool Ident () - функция по определению поля id в структуре матрицы; возвращает 0 если матрица не единичная и 1 – если единичная
- Matrix3x3 (float(&Matr)[N][N]) – конструктор класса; заполняет матрицу значениями двумерного массива Matr
- Matrix3x3(const Matrix3x3& Matr) – конструктор копирования класса; копирует значения полей Matr в поля текущего объекта
- ~Matrix3x3() – деструктор класса
- static Matrix3x3 St(int A, int B) – статический метод, возвращающий новый объект – случайным образом сгенерированную матрицу
- Matrix3x3 operator+ (Matrix3x3& A) – возвращает новый объект класса – матрицу – результат сложения текущей матрицы с матрицей A; добавляет операцию сложения в списки операций текущей матрицы и матрицы A
- Matrix3x3 operator* (Matrix3x3& A) – возвращает новый объект класса – матрицу – результат умножения текущей матрицы и матрицы A; добавляет операцию умножения в списки операций текущей матрицы и матрицы A
- void operator* (const int& n) – умножает каждый элемент матрицы на скаляр n; добавляет операцию умножения на скаляр в список операций текущей матрицы
- void operator++ () – транспонирует матрицу; добавляет операцию транспонирования в список операций текущей матрицы
- float Det () – находит и возвращает определитель матрицы; добавляет операцию нахождения определителя в список операций текущей матрицы
- void operator! () - преобразует матрицу в обратную, если это возможно; добавляет операцию обращения в список операций текущей матрицы, если обратная матрица для данной существует
- void Out () – выводит матрицу в консоль
- bool Compare (float (&Matr)[N][N]) – сравнивает поля матрицы с матрицей Matr, переданной как двумерный массив
- Matrix3x3& operator= (Matrix3x3 Matr) – копирует значения полей матрицы Matr в поля текущей матрицы и возвращает указатель на текущий объект

Класс Test

Поля класса:

- Pass – целочисленное значение – количество успешно пройденных тестов
- Numb – целочисленный счётчик количества тестов в целом

Методы класса:

- void Start () – обнуляет счётчики в полях класса

- bool Expect (int& Actual, const int& Expected), bool Expect (float& Actual, const float& Expected) – сравнивают значения Actual и Expected между собой и возвращает 1, если они совпадают
- bool ExpectM(Matrix3x3& Actual, int (&Expected)[N]) – сравнивает список операций матрицы Actual со списком операций – одномерным массивом Expected – и возвращает 1, если они совпадают
- bool ExpectM (Matrix3x3& Actual, float (&Expected)[N][N]) сравнивает объект - матрицу Actual – с двумерным массивом - матрицей Expected - и возвращает 1, если они совпадают
- void Success (bool n) – выводит сообщение об успехе/неудаче теста
- void Results () – выводит результаты теста
- void Head() – выводит заголовок теста (его номер)
- void Run () – запускает Unit-тестирование
- void CountOfAct1(), void CountOfAct2() – вспомогательные процедуры для теста CountOfAct(), описанного ниже; инициализируют по одной матрице

Все нижеописанные методы возвращают 1 в случае совпадения полученного результата с ожидаемым и 0 – в противном случае.

- bool TSumm () – тестирование метода Summ класса Matrix3x3
- bool TMult () - тестирование метода Mult класса Matrix3x3
- bool TSclr () - тестирование метода Scalar класса Matrix3x3
- bool TTransp () - тестирование метода Transpos класса Matrix3x3
- bool TDet () – тестирование метода Det класса Matrix3x3
- bool TRev () - тестирование метода Reverse класса Matrix3x3
- bool TMath () – тестирование методов Reverse и Mult на основе соблюдения формулы линейной алгебры $A * A^{-1} = E$
- bool TDetNull () - тестирование метода Det класса Matrix3x3 (нахождение определителя пропорциональной матрицы)
- bool TSummTranspSclr () - тестирование методов Summ, Transpos, Scalar класса Matrix3x3
- bool TRevNever () - тестирование метода Reverse класса Matrix3x3 (попытка обращения матрицы с нулевым определителем)
- bool CountOfAll() – тестирование метода St класса Matrix3x3, правильности работы конструкторов копирования для классов Matrix3x3 и Object, правильности подсчёта общего количества созданных объектов (поле All класса Object)
- bool CountOfAct() - тестирование метода St класса Matrix3x3, правильности работы конструкторов копирования для классов Matrix3x3 и Object, правильности подсчёта общего количества активных объектов (поле Act класса Object)
- bool TrueCopy() – тестирование правильности перегрузки оператора присваивания для классов Matrix3x3 и Object
- bool TrueOperations1() – тестирование метода AddOp класса Object путем преобразований заданной матрицы
- bool TrueOperations2() - тестирование правильности перегрузки оператора присваивания для классов Matrix3x3 и Object и метода AddOp класса Object

Листинг программы:

Файл m3.h

```
#pragma once
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
#include <time.h>
#define N 3

class Object {

private:
    static int All; //кол-во созданных объектов
    static int Act; //кол-во активных объектов
    int* Op; //указатель для выделения памяти под массив
    int Ops; //счетчик числа операций в массиве

public:
    //обнуление статических полей класса
    static void RestartObjInfo();

    Object();
    //конструктор копирования
    Object(const Object& Obj);
    ~Object();

    //вывод All и Act
    static void PrintTotalInfo();
    //геттеры All и Act
    static int GetAll() { return All; };
    static int GetAct() { return Act; };

    //добавление операции в массив операций
    void AddOp(int k);
    //вывод истории операций
    void PrintOp();
    //очищение истории операций
    void ClearOp();

    //оператор присваивания
    Object& operator= (Object& Matr);

    //сравнение списка операций
    bool Comp(int (&Spis)[N]);
};
```

Файл m3.cpp

```cpp
#include "m3.h"

using namespace std;

int Object::All = 0;
int Object::Act = 0;

void Object::RestartObjInfo() {
    All = 0;
    Act = 0;
}

Object::Object() {
    All++;
    Act++;
    Ops = 0;
    Op = nullptr;
}

Object::Object(const Object& Obj) {
    int i;
    ///*this->*/ClearOp();
    Ops = Obj.Ops;
    if (Ops != 0) {
        Op = new int[Ops];
        for (i = 0; i < Ops; i++)
            Op[i] = Obj.Op[i];
    }
    else Op = nullptr;
}

Object::~Object() {
    Act--;
    ClearOp();
}

void Object::PrintTotalInfo() {
    cout << "Кол-во активных объектов:" << Act << endl;
    cout << "Всего было создано объектов:" << All << endl;
}

void Object::AddOp(int k) {
    int i;
    int* Dop;
    Ops++;
    if ((!Op) || (Ops==0)) {
        Op = new int[Ops];
    }
    else {
        Dop = new int[Ops - 1];
```

```
50          Dop = new int[Ops - 1];
51          for (i = 0; i < Ops - 1; i++)
52              Dop[i] = Op[i];
53          Op = new int[Ops];
54          for (i = 0; i < Ops - 1; i++)
55              Op[i] = Dop[i];
56          delete[] Dop;
57          Dop = nullptr;
58      }
59      Op[Ops - 1] = k;
60  }
61
62  void Object::PrintOp() {
63      int i;
64      if (!Op) cout << "С объектом не выполнялось операций" << endl;
65      else {
66          cout << "История операций с объектом:" << endl;
67          for (i = 0; i < Ops; i++)
68              switch (Op[i]) {
69              case 1: {cout << i + 1 << ") Сложение" << endl; break; }
70              case 2: {cout << i + 1 << ") Умножение" << endl; break; }
71              case 3: {cout << i + 1 << ") Умножение на скаляр" << endl; break; }
72              case 4: {cout << i + 1 << ") Транспонирование" << endl; break; }
73              case 5: {cout << i + 1 << ") Нахождение определителя" << endl; break; }
74              case 6: {cout << i + 1 << ") Преобразование в обратную" << endl; break; }
75              }
76      }
77  }
78
79  void Object::ClearOp() {
80      if (Op) {
81          Ops = 0;
82          delete[] Op;
83          Op = nullptr;
84      }
85  }
86
87  Object& Object::operator= (Object& Obj)
88  {
89      if (&Obj == this) {
90          cout << "Внимание: выполнено самоприсваивание" << endl;
91          return *this;
92      }
93      int i;
94      /*this->*/ClearOp();
95      if (Obj.Ops) {
96          Ops = Obj.Ops;
97          Op = new int[Ops];
98          for (i = 0; i <Ops; i++)
99              Op[i] = Obj.Op[i];
100     }
101     return *this;
102 }
103
104 bool Object::Comp(int (&Spis)[N]) {
105     if ((!Op) || (Ops>N)) return 0;
106     int i;
107     for (i = 0; i < Ops; i++)
108         if (Op[i]!=Spis[i]) return 0;
109     return 1;
110 }
```

Файл m2.h

```
#pragma once
#include "m3.h"
#define N 3 //размерность матриц
#define eps 1e-3 //погрешность при сравнении вещественных чисел

class Matrix3x3 : public Object
{
private:
    //поля
    float arr[N][N];
    bool id;
    //определение поля id
    bool Ident();
    //определитель двумерного массива
    float Determ(float(&A)[N][N], int n) const;

public:

    //конструктор копирования
    Matrix3x3(const Matrix3x3& Matr);
    Matrix3x3(float(&Matr)[N][N]);
    ~Matrix3x3();
    //статический метод, возвращающий новый объект
    static Matrix3x3 St(int A, int B);

    //сложение матриц
    Matrix3x3 operator+ (Matrix3x3& A);
    //умножение матриц
    Matrix3x3 operator* (Matrix3x3& A);
    //умножение на скаляр
    void operator* (const int& n);
    //транспонирование
    //void operator& ();
    void operator++ ();
    //нахождение определителя
    float Det ();
    //преобразование в обратную
    void operator! ();
    //вывод одной матрицы
    void Out();
    //сравнение двух матриц
    bool Compare(float(&Matr)[N][N]) const;
    //оператор присваивания
    Matrix3x3& operator= (Matrix3x3 Matr);
};
```

Файл m2.cpp

```cpp
#include "m2.h"
float Nulls[N][N] = { 0,0,0,0,0,0,0,0,0 };

bool Matrix3x3::Ident() {
    int i, j;
    //проверка главной диагонали
    for (i = 0; i < N; i++)
        if (fabs(arr[i][i] - 1) >= eps) return 0;
    //проверка остальных элементов
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            if ((fabs(arr[i][j]) >= eps) && (i != j)) return 0;
    return 1;
}

float Matrix3x3::Determ(float(&A)[N][N], int n) const {
    int i, j, k;
    float a, b;
    float D;
    float B[N][N];//вспомогательная матрица, вычисляя определитель
    //которой будем получать текущее алгебраическое дополнение
//если матрица 2x2 определитель находится элементарно
    if (n == 2)
        D = A[0][0] * A[1][1] - A[0][1] * A[1][0];
    //если НЕ 2x2 матрица раскладывается по первой строке, разложение происходит
    //до тех пор, пока все матрицы, определители которых ищем, не примут вид 2x2
    else {
        D = 0;
        //собственно разложение по первой строке
        for (k = 0; k < n; k++) {
            //заполнение текущей вспомогательной матрицы
            for (j = 0; j < n - 1; j++)
                for (i = 0; i < n - 1; i++) {
                    if (j < k) B[i][j] = A[i + 1][j];
                    else B[i][j] = A[i + 1][j + 1];
                }
            a = A[0][k]; //элемент первой строки, положение которого зависит от k
            b = Determ(B, n - 1); //определитель вспомогательной матрицы
            //определение знака алгебраического дополнения
            if ((2 + k) % 2 == 0) D = D + a * b;
            else D = D - a * b;
        }
    }
    return D;
}

Matrix3x3::Matrix3x3(const Matrix3x3& Matr) {
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            arr[i][j] = Matr.arr[i][j];
```

```
50              arr[i][j] = Matr.arr[i][j];
51          id = Ident();
52      }
53
54      Matrix3x3::Matrix3x3(float(&Matr)[N][N]) : Object()  {
55          for (int i = 0; i < N; i++)
56              for (int j = 0; j < N; j++)
57                  arr[i][j] = Matr[i][j];
58          id = Ident();
59      }
60
61      Matrix3x3::~Matrix3x3() {
62      }
63
64      Matrix3x3 Matrix3x3::St(int A, int B) {
65          Matrix3x3 Res(Nulls);
66          srand(time(NULL));
67          int i, j;
68          for (i = 0; i < N; i++)
69              for (j = 0; j < N; j++)
70                  Res.arr[i][j] = A + rand() % (B - A + 1);
71          return Res;
72      }
73
74      Matrix3x3 Matrix3x3::operator+ (Matrix3x3& A) {
75          Matrix3x3 Res(Nulls);
76          int i, j;
77          for (i = 0; i < N; i++)
78              for (j = 0; j < N; j++)
79                  Res.arr[i][j] = this->arr[i][j] + A.arr[i][j];
80          Res.id = Res.Ident();
81          AddOp(1);
82          A.AddOp(1);
83          return Res;
84      }
85
86      Matrix3x3 Matrix3x3::operator* (Matrix3x3& A) {
87          int i, j, k;
88          float S; //для накопления текущей суммы текущего элемента матрицы C
89          Matrix3x3 Res(Nulls);
90          for (i = 0; i < N; i++)
91              for (j = 0; j < N; j++) {
92                  S = 0;
93                  for (k = 0; k < N; k++)
94                      S += this->arr[i][k] * A.arr[k][j];
95                  Res.arr[i][j] = S;
96              }
97          Res.id = Res.Ident();
98          AddOp(2);
```

```
 99      A.AddOp(2);
100      return Res;
101  }
102
103  void Matrix3x3::operator* (const int& n) {
104      int i, j;
105      for (i = 0; i < N; i++)
106          for (j = 0; j < N; j++)
107              arr[i][j] *= n;
108      id = Ident();
109      AddOp(3);
110  }
111
112  void Matrix3x3::operator++ () {
113      int i, j;
114      float A[N][N]; //дополнительный массив
115      for (i = 0; i < N; i++)
116          for (j = 0; j < N; j++)
117              A[i][j] = arr[i][j];
118      for (i = 0; i < N; i++)
119          for (j = 0; j < N; j++)
120              arr[i][j] = A[j][i];
121      id = Ident();
122      AddOp(4);
123  }
124
125  float Matrix3x3::Det() {
126      int i, j;
127      float D;
128      float A[N][N]; //вспомогательная матрица
129      //переносим элементы структуры во вспомогательную матрицу
130      for (i = 0; i < N; i++)
131          for (j = 0; j < N; j++)
132              A[i][j] = arr[i][j];
133      D = Determ(A, N); //считаем её определитель
134      AddOp(5);
135      return D;
136  }
137
138  void Matrix3x3::operator! () {
139      int i, j, k, q;
140      float A[N][N]; //вспомогательная матрица
141      //переносим элементы структуры во вспомогательную матрицу
142      for (i = 0; i < N; i++)
143          for (j = 0; j < N; j++)
144              A[i][j] = arr[i][j];
145      float B[N][N]; //присоединенная матрица из алгебраических дополнений
146      float C[N][N]; //матрица для нахождения текущего минора
```

```
146         float C[N][N]; //матрица для нахождения текущего минора
147         //проходимся по всем элементам исходной матрицы для каждого - свой минор,
148         //поэтому заполняем C в зависимости от положения текущего элемента
149         for (k = 0; k < N; k++)
150             for (q = 0; q < N; q++) {
151                 //собственно заполнение C
152                 for (i = 0; i < N; i++)//
153                     for (j = 0; j < N; j++) {
154                         if (i < q) {
155                             if (j < k) C[i][j] = A[i][j];
156                             else if (j > k) C[i][j - 1] = A[i][j];
157                         }
158                         else if (i > q) {
159                             if (j < k) C[i - 1][j] = A[i][j];
160                             else if (j > k) C[i - 1][j - 1] = A[i][j];
161                         }
162                     }
163                 //считаем минор текущей матрицы, составленной для текущего элемента A[i][j]
164                 //тк присоединённая матрица должна быть транспонированной - [q][k], а не [k][q]
165                 B[q][k] = Determ(C, N - 1);
166                 //определяем знак, т.е. вносим алгебраическое дополнение в присоединённую матрицу B
167                 if ((k + q + 2) % 2 != 0) B[q][k] *= -1;
168             }
169
170         //в случае попытки найти обратную матрицу для матрицы с нулевым определителем
171         //матрица преобразована не будет
172         if (Determ(A, N) != 0) {
173             //коэффициент, на который домножаем присоединенную матрицу для получения обратной
174             float D = 1 / Determ(A, N);
175             //собственно домножаем
176             for (i = 0; i < N; i++)
177                 for (j = 0; j < N; j++)
178                     B[i][j] *= D;
179             //вносим полученную матрицу обратно в структуру
180             for (i = 0; i < N; i++)
181                 for (j = 0; j < N; j++)
182                     arr[i][j] = B[j][i];
183             id = Ident();
184             AddOp(6);
185         }
186     }
187
188     void Matrix3x3::Out() {
189         int i, j;
190         for (i = 0; i < N; i++) {
191             for (j = 0; j < N; j++) printf("%*.3f  ", 7, arr[i][j]);
192             printf("\n");
193         }
194         if (id == 0) printf("\t     НЕ единичная\n");
195         else printf("\t       Единичная\n");
196     }
197
198     bool Matrix3x3::Compare(float(&Matr)[N][N]) const {
199         int i, j;
200         for (i = 0; i < N; i++)
201             for (j = 0; j < N; j++)
202                 if (fabs(Matr[i][j] - arr[i][j]) >= eps) return 0;
203         return 1;
204     }
205
206     Matrix3x3& Matrix3x3::operator= (Matrix3x3 Matr) {
207         int i, j;
208         for (i = 0; i < N; i++)
209             for (j = 0; j < N; j++)
210                 arr[i][j] = Matr.arr[i][j];
211         id = Ident();
212         return *this;
213     }
```

Файл t1.h

```
#pragma once
#include "m2.h"

//класс теста:
class Test
{
    int Pass; //количество пройденных тестов
    int Numb; //общее количество тестов

    //обнуление счетчиков
    void Start();
    //сравнение результатов
    //для величин
    bool Expect(float& Actual, const float& Expected);
    bool Expect(int& Actual, const int& Expected);
    //для матриц/массивов
    bool ExpectM(Matrix3x3& Actual, int (&Expected)[N]);
    bool ExpectM(Matrix3x3& Actual, float(&Expected)[N][N]);
    //вывод сообщения об успехе/неудаче
    void Success(bool n);
    //вывод результатов теста
    void Results();
    //вывод заголовка теста
    void Head();

    //возможные тесты:
    bool TSumm();
    bool TMult();
    bool TSclr();
    bool TTransp();
    bool TDet();
    bool TRev();
    bool TMath();
    bool TDetNull();
    bool TSummTranspSclr();
    bool TRevNever();

    //НОВЫЕ ТЕСТЫ:
    bool CountOfAll();

    bool CountOfAct();
    void CountOfAct1(); //Вспомогательная функция к CountOfAct()
    void CountOfAct2(); //Вспомогательная функция к CountOfAct()

    bool TrueCopy();

    bool TrueOperations1();
    bool TrueOperations2();

public:
    //запуск Unit-тестирования
    void Run();
};
```

Файл t1.cpp

```cpp
#include "t1.h"
using namespace std;

//нулевая матрица для заполнения
float MNull[N][N] = { 0,0,0,0,0,0,0,0,0 };

//результаты и исходные данные тестов:

float Test1M1[N][N] = { 3,9,10,-11,23,0,4,7,9 };
float Test1M2[N][N] = { -9,3,11,5,7,8,4,0,55 };
//сумма 1 и 2
float Test1R[N][N] = { -6,12,21,-6,30,8,8,7,64 };

float Test2M1[N][N] = { 3,9,10,-11,23,0,4,7,9 };
float Test2M2[N][N] = { -9,3,11,5,7,8,4,0,55 };
//произведение 1 и 2
float Test2R[N][N] = { 58,72,655,214,128,63,35,61,595 };

float Test3M[N][N] = { -9,3,11,5,7,8,4,0,55 };
const int Test3Sclr = 2;
//умножение на Sclr
float Test3R[N][N] = { -18,6,22,10,14,16,8,0,110 };

float Test4M[N][N] = { 3,9,10,-11,23,0,4,7,9 };
//транспонирование
float Test4R[N][N] = { 3,-11,4,9,23,7,10,0,9 };

float Test5M[N][N] = { 3,-11,4,9,23,7,10,0,9 };
//определитель
float Test5R = -178;

float Test6M[N][N] = { 58,72,655,214,128,63,35,61,595 };
//преобразование в обратную
float Test6R[N][N] = { 0.0902,-0.0036,-0.0989,-0.1561,0.0145,0.1704,0.0107,-0.0013,-0.0099 };

//единичная матрица
float Test7R[N][N] = { 1,0,0,0,1,0,0,0,1 };

//пропорциональная матрица
float Test8M[N][N] = { 1,2,3,4,5,6,7,8,9 };

//сумма T1M1 и T1M2 + транспонирование результата +умножение на Sclr
float Test9R[N][N] = { 18,18,-24,-36,-90,-21,-63,-24,-192 };
int Test9Sclr = -3;

void Test::Start() {
    Pass = 0;
    Numb = 0;
}

bool Test::Expect(float& Actual, const float& Expected) {
    cout << "\nОжидаемый результат: " << Expected << endl;
    cout << "Фактический результат: " << Actual << endl;
```

```
54      return fabs(Actual - Expected) < eps;
55  }
56
57  bool Test::Expect(int& Actual, const int& Expected) {
58      cout << "\nОжидаемый результат: " << Expected << endl;
59      cout << "Фактический результат: " << Actual << endl;
60      return (Actual==Expected);
61  }
62
63  bool Test::ExpectM(Matrix3x3& Actual, int (&Expected)[N]) {
64      int i;
65      cout << "\nОжидаемый результат: " << endl;
66      for (i = 0; i < N; i++)
67          switch (Expected[i]) {
68          case 1: {cout << i + 1 << ") Сложение" << endl; break; }
69          case 2: {cout << i + 1 << ") Умножение" << endl; break; }
70          case 3: {cout << i + 1 << ") Умножение на скаляр" << endl; break; }
71          case 4: {cout << i + 1 << ") Транспонирование" << endl; break; }
72          case 5: {cout << i + 1 << ") Нахождение определителя" << endl; break; }
73          case 6: {cout << i + 1 << ") Преобразование в обратную" << endl; break; }
74          }
75      cout << "Фактический результат: " << endl;
76      Actual.PrintOp();
77      return Actual.Comp(Expected);
78  }
79
80  bool Test::ExpectM(Matrix3x3& Actual, float(&Expected)[N][N]) {
81      cout << "\n\tОЖИДАНИЕ:" << endl;
82      int i, j;
83      bool I; //проверка на единичность двумерного массива
84      I = 1;
85      for (i = 0; i < N; i++) {
86          for (j = 0; j < N; j++) {
87              printf("%*.3f  ", 7, Expected[i][j]);
88              if ((fabs(Expected[i][i] - 1) >= eps || ((fabs(Expected[i][j]) >= eps) && (i != j)))
89                  I = 0;
90          }
91          printf("\n");
92      }
93      if (!I) printf("\t     НЕ единичная");
94      else printf("\t        Единичная");
95      cout << "\n\tРЕЗУЛЬТАТ:" << endl;
96      Actual.Out();
97      return Actual.Compare(Expected);
98  }
99
100 void Test::Success(bool n) {
101     ++Numb;
102     if (n) ++Pass, cout << "\nУспех!" << endl;
103     else cout << "\nНеудача!" << endl;
104 }
```

```
105
106     void Test::Results()
107     {
108         cout << "\n_____" << endl;
109         cout << "\nРЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ КЛАССА MATRI3X3" << endl;
110         cout << "Тестов пройдено: " << Pass << " из " << Numb << endl;
111     }
112
113     void Test::Head() {
114         cout << "\n________ТЕСТ №" << Pass + 1 << "__________\n" << endl;
115     }
116
117     bool Test::TSumm() {
118         //arrange
119         this->Head();
120         cout << "СЛОЖЕНИЕ МАТРИЦ 1 И 2\n" << endl;
121         Matrix3x3 A(Test1M1);
122         Matrix3x3 B(Test1M2);
123         Matrix3x3 C(MNull);
124         printf("\tМАТРИЦА 1\n");
125         A.Out();
126         printf("\tМАТРИЦА 2\n");
127         B.Out();
128         //act
129     //  C=A.Summ(B);
130         C = (A + B);
131         //assert
132         return ExpectM(C, Test1R);
133     }
134
135     bool Test::TMult() {
136         //arrange
137         this->Head();
138         cout << "УМНОЖЕНИЕ МАТРИЦ 1 и 2\n" << endl;
139         Matrix3x3 A(Test2M1);
140         Matrix3x3 B(Test2M2);
141         Matrix3x3 C(MNull);
142         printf("\tМАТРИЦА 1\n");
143         A.Out();
144         printf("\tМАТРИЦА 2\n");
145         B.Out();
146         //act
147         C = A * B;
148         //assert
149         return ExpectM(C, Test2R);
150     }
151
152     bool Test::TSclr() {
153         //arrange
154         this->Head();
155         cout << "УМНОЖЕНИЕ МАТРИЦЫ 1 НА ЧИСЛО " << Test3Sclr << "\n" << endl;
```

```
156         Matrix3x3 A(Test3M);
157         printf("\tМАТРИЦА 1\n");
158         A.Out();
159         //act
160         //A.Scalar(Test3Sclr);
161         A* Test3Sclr;
162         //assert
163         return ExpectM(A, Test3R);
164     }
165
166     bool Test::TTransp() {
167         //arrange
168         this->Head();
169         cout << "ТРАНСПОНИРОВАНИЕ МАТРИЦЫ 1\n" << endl;
170         Matrix3x3 A(Test4M);
171         printf("\tМАТРИЦА 1\n");
172         A.Out();
173         //act
174         //A.Transpos();
175         ++A;
176         //assert
177         return ExpectM(A, Test4R);
178     }
179
180     bool Test::TDet() {
181         //arrange
182         this->Head();
183         cout << "НАХОЖДЕНИЕ ОПРЕДЕЛИТЕЛЯ МАТРИЦЫ 1\n" << endl;
184         Matrix3x3 A(Test5M);
185         printf("\tМАТРИЦА 1\n");
186         A.Out();
187         //act
188         float D;
189         //D=A.Det();
190         D = A.Det();
191         //assert
192         return Expect(D, Test5R);
193     }
194
195     bool Test::TRev() {
196         //arrange
197         this->Head();
198         cout << "ПРЕОБРАЗОВАНИЕ МАТРИЦЫ 1 В ОБРАТНУЮ\n" << endl;
199         Matrix3x3 A(Test6M);
200         printf("\tМАТРИЦА 1\n");
201         A.Out();
202         //act
203         //A.Reverse();
204         !A;
205         //assert
206         return ExpectM(A, Test6R);
```

```
207 }
208
209 bool Test::TMath() {
210     //arrange
211     this->Head();
212     cout << "ПРЕОБРАЗОВАНИЕ МАТРИЦЫ 1 В ОБРАТНУЮ, А ЗАТЕМ ПЕРЕМНОЖЕНИЕ МАТРИЦ 1 И 2" << endl;
213     cout << "(ПРОВЕРКА ФОРМУЛЫ ИЗ ЛИНЕЙНОЙ АЛГЕБРЫ A*A^(-1)=E)\n" << endl;
214     Matrix3x3 A(Test6M);
215     Matrix3x3 B(Test6M);
216     Matrix3x3 C(MNull);
217     printf("\tМАТРИЦА 1\n");
218     A.Out();
219     printf("\tМАТРИЦА 2\n");
220     B.Out();
221     //act
222     //A.Reverse();
223     !A;
224     C = A * B;
225     //assert
226     return ExpectM(C, Test7R);
227 }
228
229 bool Test::TDetNull() {
230     //arrange
231     this->Head();
232     cout << "НАХОЖДЕНИЕ ОПРЕДЕЛИТЕЛЯ ПРОПОРЦИОНАЛЬНОЙ МАТРИЦЫ 1\n" << endl;
233     Matrix3x3 A(Test8M);
234     printf("\tМАТРИЦА 1\n");
235     A.Out();
236     //act
237     float D;
238     //D=A.Det();
239     D = A.Det();
240     //assert
241     return Expect(D, 0);
242 }
243
244 bool Test::TSummTranspSclr() {
245     //arrange
246     this->Head();
247     cout << "СЛОЖЕНИЕ МАТРИЦ 1 и 2, А ЗАТЕМ ТРАНСПОНИРОВАНИЕ РЕЗУЛЬТАТА И УМНОЖЕНИЕ НА ЧИСЛО " <
248     Matrix3x3 A(Test1M1);
249     Matrix3x3 B(Test1M2);
250     Matrix3x3 C(MNull);
251     printf("\tМАТРИЦА 1\n");
252     A.Out();
253     printf("\tМАТРИЦА 2\n");
254     B.Out();
255     //act
256     C = (A + B);
257     //C.Transpos();
```

```
258         //C.Scalar(Test9Sclr);
259         ++C;
260         C* Test9Sclr;
261         //assert
262         return ExpectM(C, Test9R);
263     }
264
265     bool Test::TRevNever() {
266         //arrange
267         this->Head();
268         cout << "ПРЕОБРАЗОВАНИЕ МАТРИЦЫ 1 В ОБРАТНУЮ" << endl;
269         cout << "(ПОПЫТКА ПРЕОБРАЗОВАНИЯ МАТРИЦЫ С НУЛЕВЫМ ОПРЕДЕЛИТЕЛЕМ В ОБРАТНУЮ)\n" << endl;
270         Matrix3x3 A(Test8M);
271         printf("\tМАТРИЦА 1\n");
272         A.Out();
273         //act
274         //A.Reverse();
275         !A;
276         //assert
277         return ExpectM(A, Test8M);
278     }
279
280     bool Test::CountOfAll() {
281         //arrange
282         Object::RestartObjInfo();
283         this->Head();
284         cout << "СОЗДАНИЕ ДВУХ МАТРИЦ СТАТИЧЕСКИМ МЕТОДОМ ГЕНЕРАЦИИ" << endl;
285         cout << "И ПОДСЧЁТ ДЛЯ НИХ КОЛИЧЕСТВА СОЗДАННЫХ ОБЪЕКТОВ" << endl;
286         //act
287         Matrix3x3 A = Matrix3x3::St(-100,100);
288         Matrix3x3 B = Matrix3x3::St(-100,100);
289         printf("\tМАТРИЦА 1\n");
290         A.Out();
291         printf("\tМАТРИЦА 2\n");
292         B.Out();
293         //assert
294         int T=2;
295         int R=Object::GetAll();
296         return Expect(R,T);
297     }
298
299     void Test::CountOfAct1() {
300         Matrix3x3 A = Matrix3x3::St(-100,100);
301         printf("\tМАТРИЦА 1\n");
302         A.Out();
303     }
304     void Test::CountOfAct2() {
305         Matrix3x3 B(Test4M);
306         printf("\tМАТРИЦА 2\n");
307         B.Out();
308     }
```

```
309 bool Test::CountOfAct() {
310     //arrange
311     Object::RestartObjInfo();
312     this->Head();
313     cout << "СОЗДАНИЕ ДВУХ МАТРИЦ В ДВУХ ОТДЕЛЬНЫХ ФУНКЦИЯХ И ОДНОЙ - НЕПОСРЕДСТВЕННО В ФУНКЦИИ
314     cout << "ПОДСЧЁТ КОЛИЧЕСТВА АКТИВНЫХ ОБЪЕКТОВ ПОСЛЕ ИХ СОЗДАНИЯ" << endl;
315     //act
316     CountOfAct1();
317     CountOfAct2();
318     Matrix3x3 C(Test7R);
319     printf("\tМАТРИЦА 3\n");
320     C.Out();
321     //assert
322     int T=1;
323     int R=Object::GetAct();
324     return Expect(R,T);
325 }
326
327 bool Test::TrueCopy() {
328     //arrange
329     this->Head();
330     cout << "ПРИРАВНИВАНИЕ К НОВОЙ МАТРИЦЕ - УЖЕ ИНИЦИАЛИЗИРОВАННОЙ" << endl;
331     cout << "ИЗМЕНЕНИЕ ЗНАЧЕНИЙ В ИЗНАЧАЛЬНОЙ МАТРИЦЕ И ПРОВЕРКА, НЕ ИЗМЕНИЛИСЬ ЛИ ЗНАЧЕНИЯ" <<
332     float T[N][N] = { 18,18,-24,-36,-90,-21,-63,-24,-192 };
333     Matrix3x3 A(T);
334     printf("\tИЗНАЧАЛЬНАЯ МАТРИЦА\n");
335     A.Out();
336     //act
337     Matrix3x3 B = A;
338     ++A; !A; A*3;
339     printf("\tИЗМЕНЁННАЯ ИЗНАЧАЛЬНАЯ\n");
340     A.Out();
341     //assert
342     return ExpectM(B,T);
343 }
344
345 bool Test::TrueOperations1() {
346     //arrange
347     this->Head();
348     cout << "ТРАНСПОНИРОВАНИЕ МАТРИЦЫ, ОБРАЩЕНИЕ, УМНОЖЕНИЕ НА СКАЛЯР" << endl;
349     cout << "ПРОВЕРКА СПИСКА ИСТОРИИ ОПЕРАЦИЙ ЭТОЙ МАТРИЦЫ." << endl;
350     float R[N][N] = { 3,9,10,-11,23,0,4,7,9 };
351     Matrix3x3 A(R);
352     printf("\tМАТРИЦА\n");
353     A.Out();
354     //act
355     ++A;
356     !A;
357     A*2;
358     printf("  ПРЕОБРАЗОВАННАЯ МАТРИЦА\n");
359     A.Out();
```

```
360         //assert
361         int T[N]={4,6,3};
362         return ExpectM(A,T);
363     }
364     bool Test::TrueOperations2() {
365         //arrange
366         this->Head();
367         cout << "СОЗДАНИЕ МАТРИЦЫ A И ЕЁ ТРАНСПОНИРОВАНИЕ. ИНИЦИАЛИЗАЦИЯ МАТРИЦЫ B ЧЕРЕЗ A" << endl;
368         cout << "УМНОЖЕНИЕ МАТРИЦЫ B НА НЕКОТОРУЮ МАТРИЦУ C И СЛОЖЕНИЕ С МАТРИЦЕЙ D." << endl;
369         cout << "ПРОВЕРКА СПИСКА ОПЕРАЦИЙ МАТРИЦЫ B." << endl;
370         //act
371         Matrix3x3 A = Matrix3x3::St(-10,10);
372         ++A;
373         Matrix3x3 B = A;
374         Matrix3x3 C(Test3R);
375         Matrix3x3 D(Test9R);
376         B*C;
377         B+D;
378         //assert
379         int T[N]={2,1};
380         return ExpectM(B,T);
381     }
382
383     //запуск Unit-тестирования
384     void Test::Run() {
385         Start();
386
387         Success(TSumm());
388         Success(TMult());
389         Success(TSclr());
390         Success(TTransp());
391         Success(TDet());
392         Success(TRev());
393         Success(TMath());
394         Success(TDetNull());
395         Success(TSummTranspSclr());
396         Success(TRevNever());
397
398         Success(CountOfAll());
399         Success(CountOfAct());
400         Success(TrueCopy());
401         Success(TrueOperations1());
402         Success(TrueOperations2());
403
404
405         Results();
406     }
```

Файл main.cpp

```
1   #include "t1.h"
2   int main() {
3       system("chcp 1251");
4       Test T;
5       T.Run();
6       return 0;
7   }
```

Результаты работы программы:

Итак, после запуска программа вывела в консоль результаты всех 15-ти тестов (рисунки 1, 2, 3 – приведены результаты выполнения последних 5ти тестов, добавленных в этой лабораторной работе – остальные тесты не изменились), значения которых соответствуют действительности и совпадают с ранее выполненными проверками. Программа работает корректно.

```
________ТЕСТ №11__________

СОЗДАНИЕ ДВУХ МАТРИЦ СТАТИЧЕСКИМ МЕТОДОМ ГЕНЕРАЦИИ
И ПОДСЧЁТ ДЛЯ НИХ КОЛИЧЕСТВА СОЗДАННЫХ ОБЪЕКТОВ
        МАТРИЦА 1
  1.000   48.000  -88.000
 -1.000  -29.000   46.000
-74.000   -2.000    7.000
            НЕ единичная
        МАТРИЦА 2
  1.000   48.000  -88.000
 -1.000  -29.000   46.000
-74.000   -2.000    7.000
            НЕ единичная

Ожидаемый результат: 2
Фактический результат: 2

Успех!

________ТЕСТ №12__________

СОЗДАНИЕ ДВУХ МАТРИЦ В ДВУХ ОТДЕЛЬНЫХ ФУНКЦИЯХ И ОДНОЙ - НЕПОСРЕДСТВЕННО В ФУНКЦИИ ТЕСТА
ПОДСЧЁТ КОЛИЧЕСТВА АКТИВНЫХ ОБЪЕКТОВ ПОСЛЕ ИХ СОЗДАНИЯ
        МАТРИЦА 1
  1.000   48.000  -88.000
 -1.000  -29.000   46.000
-74.000   -2.000    7.000
            НЕ единичная
        МАТРИЦА 2
  3.000    9.000   10.000
-11.000   23.000    0.000
  4.000    7.000    9.000
            НЕ единичная
        МАТРИЦА 3
  1.000    0.000    0.000
  0.000    1.000    0.000
  0.000    0.000    1.000
               Единичная

Ожидаемый результат: 1
Фактический результат: 1

Успех!
```

Рисунок 1

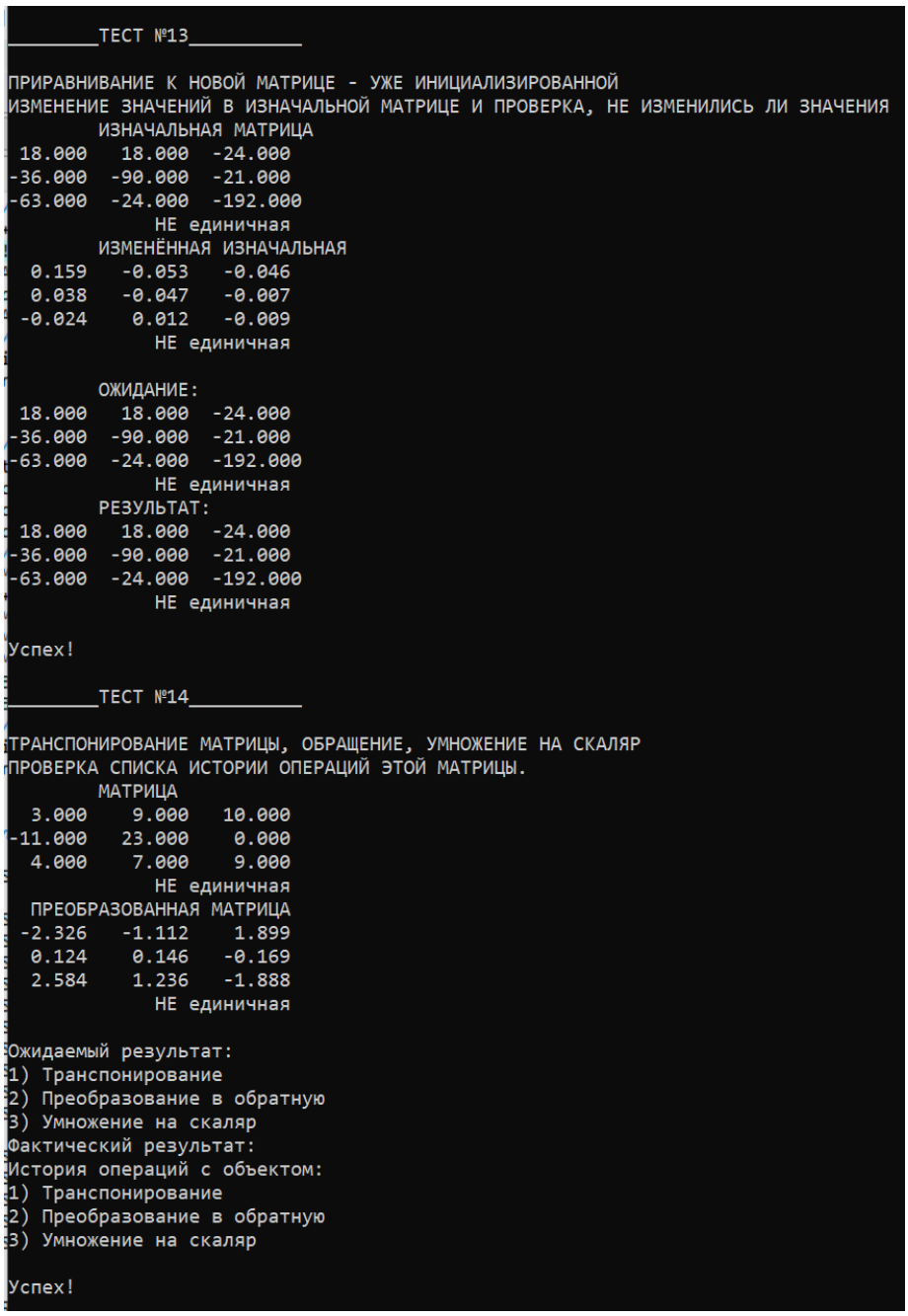

Рисунок 2

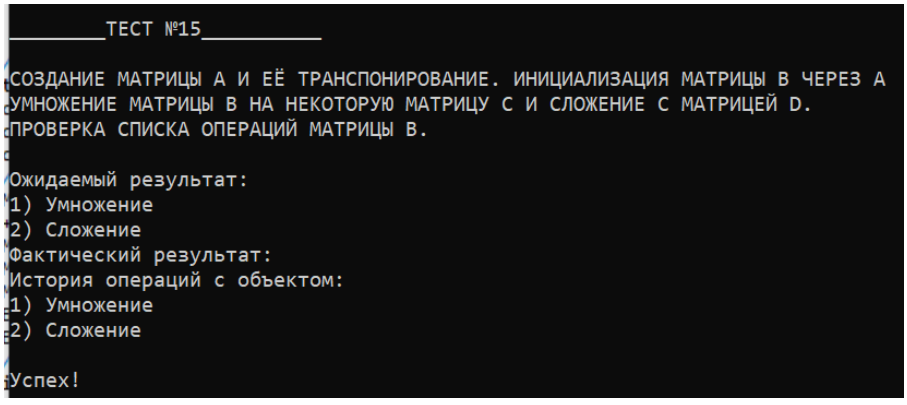

Рисунок 3