ГУАП
КАФЕДРА №14

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

Должность, уч. степень, звание подпись, дата инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1

по курсу: ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ
СТУДЕНТ ГР. 1441 М.И. Лубинец
подпись, дата инициалы, фамилия

Санкт-Петербург
2015

1. Формализация задачи

Задача №1:

Для заданного целого N (0 <= N <= 40) создает квадратную матрицу
размера N на N такую, что каждый недиагональный элемент матрицы
представляет собой сумму индексов элемента (нумерация индексов начинается с
единицы), а каждый диагональный элемент - сумму элементов, находящихся с
ним в одной строке и одном столбце.
Отобразить матрицу на экране
Попернуть матрицу на 90 градусов по часовой стрелке и отобразить ее на экране

Задача №2

Реализовать класс, выполняющий условия задачи №1,
Добавить "безопасные" методы по работе с данными матрицы.
Добавить методы по установке и изменению размера матриц.
Добавить метод разворота матрицы на угол 90 * n градусов, где n - целое.
Добавить метод Assign, который должен использоваться для копирования содержимого
одной матрицы в другую.
При реализации необходимо учитывать возможные ошибки при выделении памяти, некорректные
входные параметры и т.д. Программа должны вести себя адекватным образом и в случае
возникновения ошибок. Использование программы не должно приводить к утечке памяти.

Задача №3

Для класса матрицы из задачи №2, ереопределить бинарные математические операторы так, чтобы в
арифметических операциях могли участвовать матрицы, натуральные числа.

Задача №5

Для класса матрицы, определить метод расчета определителя матрицы

2. Листинги

Задача 1.

Файл main.cpp

```
#include <iostream>
#include <stdint.h>
#include <assert.h>
#include <string>

/// Function computing elements in row and collumn
inline uint16_t
row_col_sum(uint16_t** matrix, size_t msize, uint16_t num) {
    assert(matrix != NULL && msize <= 40);

    uint16_t sum = 0;

    /// Go through collumn
    for(uint16_t y = 0; y < msize; y++) {
        sum += matrix[y][num];
    }

    /// Go through row
    for(uint16_t x = 0; x < msize; x++) {
        sum += matrix[num][x];
    }

    return sum;
}

uint16_t**
init_matrix(size_t msize) {
    assert(msize <= 40 && "0 <= size <= 40");

    /// Creating array of pointers (collumns)
    uint16_t** matrix = new uint16_t*[msize];

    /// Initializing values in matrix
    for(uint16_t i = 0; i < msize; i++) {
        matrix[i] = new uint16_t[msize];

        for(uint16_t j = 0; j < msize; j++) {
            if(i == j) {
                matrix[i][j] = 0;
            } else {
                matrix[i][j] = (i+1) + (j+1);
            }
        }
    }

    /// Initializing main diagonal
    for(uint16_t i = 0; i < msize; i++) {
        matrix[i][i] = row_col_sum(matrix, msize, i);
    }

    return matrix;
}

void
rotate_90(uint16_t** matrix, size_t msize) {
    assert(matrix != NULL && msize <= 40);

    uint16_t temp;
    for(uint16_t i = 0; i < msize/2; i++) {
        for(uint16_t j = 0; j < msize/2; j++) {
            temp                        = matrix[msize-1-j][i];
            matrix[msize-1-j][i]        = matrix[msize-1-i][msize-1-j];
            matrix[msize-1-i][msize-1-j] = matrix[j]        [msize-1-i];
            matrix[j]        [msize-1-i] = matrix[i]        [j];
            matrix[i]        [j]         = temp;
        }
    }
}

void
print_matrix(uint16_t** matrix, size_t msize) {
    assert(matrix != NULL && msize <= 40);

    std::string out;
    for(uint16_t i = 0; i < msize; i++) {
        for(uint16_t j = 0; j < msize; j++) {
            out = std::to_string(matrix[i][j]);
            out.resize(4, ' ');
            std::cout << out;
        }
        std::cout<< "\n\n";
    }
}

int main() {
    const size_t msize = 40;
    uint16_t** matrix = init_matrix(msize);
    print_matrix(matrix, msize);
    rotate_90(matrix, msize);
    print_matrix(matrix, msize);
    return 0;
}
```

Задача 2.
Файл matrix.hpp

```
/*****************************************************************************

ModuleName:
    Matrix

Abstract:
    Matrix class, designed for Programming Technology class, extended version.

Author:
    Lubinets Mike 1441 (Лубинец Михаил 1441)

Date:
    2016-03-20

*****************************************************************************/

#ifndef PT_L1_T2_MATRIX_H
#define PT_L1_T2_MATRIX_H

#include <string>
#include <stdint.h>
#include <stddef.h>

namespace PT_l1_t2 {

using std::string;

class Matrix {
public:
    Matrix();
    virtual ~Matrix();

    /* @brief Create and initialize matrix with passed size
     * @param size -- desired matrix size */
    void Init(size_t size);

    /* @brief Rotate matrix by 90 degrees */
    void Rotate();

    /* @brief Output matrix to stdout */
    void Print();

    /* Getters */
    size_t size() const;

protected:
    void InitMemory();
    void FreeMemory();

    int ColRowSum(size_t n);

    size_t _size;
    int** _matrix;        ///< Array of pointers to _memory (columns)
    int*  _memory;        ///< Continuous chunk of memory (to prevent memory fragmentation)
};

}

#endif // PT_L1_T2_MATRIX_H
```

Файл matrix2.hpp

```
/*****************************************************************************

ModuleName:
    Matrix2

Abstract:
    Matrix class, designed for Programming Technology class, extended version.

Author:
    Lubinets Mike 1441 (Лубинец Михаил 1441)

Date:
    2016-03-20

Changes:
    Addded safe operator[][] semantics
    Resize method
    Rotate by n*90
    Copy constructor and Assign operator

*****************************************************************************/

#ifndef PT_L1_T2_MATRIX2_H
#define PT_L1_T2_MATRIX2_H
#include "matrix.hpp"

namespace PT_l1_t2 {

class Matrix2 : public Matrix {
public:
    Matrix2();
    Matrix2(const Matrix2& copy_from);
    Matrix2(const Matrix2* copy_from);

    /* @brief Copy matrix */
    void Assign(const Matrix2& assign_from);
    void Assign(const Matrix2* assign_from);

    /* @brief Resize matrix
     * @param new_size -- new matrix size */
    void Resize(size_t new_size);

    /* @brief Rotate matrix by n*90 grad
     * n can be positive or negative, but it only makes sence to pass 1 <= n < 3,
     * as n > 3 will be translated to n % 4,
     * -3 => 1, -2 => 2, -1 => 3 */
    void Rotate(int n);

    /* Proxy class for safe implementation of operator[][] semantics */
    struct MatrixRow {
        explicit MatrixRow(int* row_ptr, size_t size);
        int& operator[](size_t ix);
    private:
        int*  _ptr;
        size_t _size;
    };

    /* Safe access to matrix elements */
    MatrixRow operator[](size_t iy);
};

}

#endif // PT_L1_T2_MATRIX2_HPP
```

Файл matrix.cpp

```
#include "matrix.hpp"
#include <iostream>
#include <stdexcept>

namespace PT_l1_t2 {

Matrix::Matrix() :
    _size(0),
    _matrix(nullptr),
    _memory(nullptr) {}

Matrix::~Matrix() {
    FreeMemory();
}

void
Matrix::Init(size_t size) {
    if(size > 40) {
        throw std::logic_error("ERROR: Size range must be: 0 <= size <= 40");
    }

    _size = size;
    InitMemory();

    /// Do values initialization
    for(uint i = 0; i < size; i++) {
        for(uint j = 0; j < size; j++) {
            if(i == j) {
                _matrix[i][j] = 0;
            } else {
                _matrix[i][j] = (i+1) + (j+1);
            }
        }
    }

    /// Initializing main diagonal
    for(uint i = 0; i < size; i++) {
        _matrix[i][i] = ColRowSum(i);
    }
}

void
Matrix::Rotate() {
    if(!_matrix || !_memory) {
        throw std::logic_error("ERROR: Matrix was not initialized, can't rotate");
    }

    for(uint y = 0; y < _size/2; y++) {
        size_t s = _size;
        for(uint x = 0; x < s/2; x++) {
            int temp            = _matrix[s-1-x][y];
            _matrix[s-1-x][y]     = _matrix[s-1-y][s-1-x];
            _matrix[s-1-y][s-1-x] = _matrix[x]    [s-1-y];
            _matrix[x]    [s-1-y] = _matrix[y]    [x];
            _matrix[y]    [x]     = temp;
        }
    }
}

void
Matrix::Print() {
    if(!_matrix || !_memory) {
        throw std::logic_error("ERROR: Matrix was not initialized, can't print");
    }

    std::string out;
    for(uint16_t i = 0; i < _size; i++) {
        for(uint16_t j = 0; j < _size; j++) {
            /// Pretty print
            out = std::to_string(_matrix[i][j]);
            out.resize(4, ' ');

            std::cout << out;
        }
        std::cout << "\n\n";
    }
}

size_t
Matrix::size() const {
    return _size;
}
```

```
void
Matrix::InitMemory() {
    /// Checking if matrix was initialized previously to prevent memory leak
    if(_matrix || _memory) {
        FreeMemory();
    }

    /// Allocating memory for the matrix and collumns
    _memory = new int [_size*_size];
    _matrix = new int*[_size];

    /// Setting up collumn pointers
    for(size_t i = 0; i < _size; i++) {
        _matrix[i] = _memory + i*_size;
    }
}
void
Matrix::FreeMemory() {
    if(_matrix != nullptr) {
        std::clog << "TRACE: Freeing collumn pointers memory" << std::endl;
        delete[] _matrix;
    }

    if(_memory != nullptr) {
        std::clog << "TRACE: Freeing matrix memory" << std::endl;
        delete[] _memory;
    }
}

int
Matrix::ColRowSum(size_t n) {
    uint16_t sum = 0;

    /// Go through collumn
    for(uint y = 0; y < _size; y++) {
        sum += _matrix[y][n];
    }

    /// Go through row
    for(uint x = 0; x < _size; x++) {
        sum += _matrix[n][x];
    }

    return sum;
}


}
```

Файл matrix2.cpp

```
#include "matrix2.hpp"
#include <stdexcept>

namespace PT_l1_t2 {

using std::to_string;

Matrix2::Matrix2()                          : Matrix() {}
Matrix2::Matrix2(const Matrix2* copy_from) : Matrix2(*copy_from) {}
Matrix2::Matrix2(const Matrix2& copy_from) : Matrix(copy_from) {
    Assign(copy_from);
}

void
Matrix2::Assign(const Matrix2* from) {
    return Assign(*from);
}

void
Matrix2::Assign(const Matrix2& from) {
    _size = from.size();
    InitMemory();

    size_t mem_size = _size*_size;
    for(size_t i = 0; i < mem_size; i++) {
        _memory[i] = from._memory[i];
    }
}

void
Matrix2::Resize(size_t new_size) {
    _size = new_size;
    InitMemory();
}

void
Matrix2::Rotate(int n) {
    bool neg = (n < 0) ? true : false;

    n = abs(n) % 4;
    if(n == 0 || n == 4) {
        return;
    }

    n = (neg) ? 4 - n
              : n;

    for(int i = 0; i < n; i++) {
        Matrix::Rotate();
    }
}

Matrix2::MatrixRow::MatrixRow(int* row_ptr, size_t size)
    : _ptr(row_ptr), _size(size) {}

int&
Matrix2::MatrixRow::operator[](size_t ix) {
    if(ix >= _size) {
        throw std::out_of_range (
            "ERROR: Row index " + to_string(ix) + " is out-of-range (0.." + to_string(_size-1) + ")"
        );
    }
    return _ptr[ix];
}

Matrix2::MatrixRow
Matrix2::operator[](size_t iy) {
    if(iy >= _size) {
        throw std::out_of_range (
            "ERROR: Col index " + to_string(iy) + " is out-of-range (0.." + to_string(_size-1) + ")"
        );
    }
    return MatrixRow(_matrix[iy], _size);
}

}
```

Файл task2.cpp

```
#include <iostream>
#include "matrix.hpp"
#include "matrix2.hpp"
using namespace std;
using namespace PT_l1_t2;
void demo_matrix() {
    Matrix m1;
    /// Trying to print and rotate non-initialized matrix
    try { m1.Rotate(); } catch(std::exception& e) {
        std::cerr << e.what();
    }
    try { m1.Print(); } catch(std::exception& e) {
        std::cerr << e.what();
    }
    /// Trying to initialize woth wrong values
    try { m1.Init(-1); } catch(std::exception& e) {
        std::cerr << e.what();
    }
    try { m1.Init(41); } catch(std::exception& e) {
        std::cerr << e.what();
    }
    /// Finished trolling matrix class, testing main functionality
    m1.Init(10);

    cout << "Matrix with size " << m1.size() << endl;
    m1.Print();

    cout << endl << endl
         << "Rotated by 90 deg matrix with size " << m1.size() << endl;
    m1.Rotate();
    m1.Print();

    /// Reinitializing the matrix with size 15
    m1.Init(15);
    cout << "Matrix with size " << m1.size() << endl;
    m1.Print();

    cout << endl << endl
         << "Rotated by 90 deg matrix with size " << m1.size() << endl;
    m1.Rotate();
    m1.Print();
}
void demo_matrix2() {
    Matrix2 m1;
    Matrix2 m2;
    Matrix2 m3;

    m1.Init(2);
    m2.Init(4);
    m3.Init(10);

    cout << "Original matrix 2:\n";
    m2.Print();

    cout << "\nMatrix 2 rotated 2 times:\n";
    m2.Rotate(2);
    m2.Print();

    cout << "\nOriginal matrix 3:\n";
    m3.Print();

    cout << "\nMatrix 3 rotated 400000001 times:\n";
    m3.Rotate(400000001);
    m3.Print();

    cout << "\nOriginal matrix 1:\n";
    m1.Print();

    cout << "\nSet 0:1 and 1:1 in matrix 1 to 0:\n";
    m1[0][1] = 0;
    m1[1][1] = 0;
    m1.Print();

    cout << "\nTrying to set 10:1 of matrix 1:\n";
    try {
        m1[10][1] = 0;
    } catch(std::out_of_range& e) {
        std::cerr << e.what();
    }

    cout << "\nTrying to set 1:10 of matrix 1:\n";
    try {
        m1[1][10] = 0;
    } catch(std::out_of_range& e) {
        std::cerr <<  e.what();
    }

    cout << "\nCopy matrix 1 to matrix 3:\n";
    m3.Assign(m1);
    m3.Print();

}
```

```
int main() {
    demo_matrix();
    demo_matrix2();

    size_t matrix_size;

    cout << "Enter matrix size: ";
    cin  >> matrix_size;

    Matrix m1;
    try {
        m1.Init(matrix_size);
    } catch (std::exception& e) {
        std::cout << e.what();
        return 1;
    }

    cout << "Matrix with size " << m1.size() << endl;
    m1.Print();

    cout << endl << endl
         << "Rotated by 90 deg matrix with size " << m1.size() << endl;
    m1.Rotate();
    m1.Print();
}
```

Задача 3
Файл martix3.hpp

```
/*****************************************************************************

ModuleName:
    Matrix

Abstract:
    Matrix class, designed for Programming Technology class, double extended version.

Author:
    Lubinets Mike 1441 (Лубинец Михаил 1441)

Date:
    2016-03-20

Changes
    Arithmetic operators

*****************************************************************************/

#ifndef MATRIX3_H
#define MATRIX3_H
#include "matrix2.hpp"

namespace PT_l1_t3 {

class Matrix3 : public PT_l1_t2::Matrix2 {
public:
    using Matrix2::Matrix2;

    Matrix3& operator= (Matrix3& b);
    Matrix3& operator= (int b);

    /* Operations with matrices */
    friend Matrix3 operator+ (const Matrix3& a, const Matrix3& b);
    friend Matrix3 operator- (const Matrix3& a, const Matrix3& b);
    friend Matrix3 operator* (const Matrix3& a, const Matrix3& b);

    /* Operations with natural numbers */
    friend Matrix3 operator* (const Matrix3& a, int b);

};

}

#endif // MATRIX3_H
```

Файл matrix3.cpp

```
#include "matrix3.hpp"
#include <stdexcept>
namespace PT_l1_t3 {

Matrix3&
Matrix3::operator= (Matrix3& b) {
    Assign(b);
    return *this;
}

Matrix3 operator+ (const Matrix3& a, const Matrix3& b) {
    if(a.size() != b.size()) {
        throw std::logic_error("a.size() != b.size()");
    }

    size_t size  = a.size();
    size_t msize = size*size;

    Matrix3 c;
    c.Resize(size);

    for(uint i = 0; i < msize; i++) {
        c._memory[i] = a._memory[i] + b._memory[i];
    }

    return c;
}

Matrix3 operator- (const Matrix3& a, const Matrix3& b) {
    if(a.size() != b.size()) {
        throw std::logic_error("a.size() != b.size()");
    }

    size_t size  = a.size();
    size_t msize = size*size;

    Matrix3 c;
    c.Resize(size);

    for(uint i = 0; i < msize; i++) {
        c._memory[i] = a._memory[i] - b._memory[i];
    }

    return c;
}

Matrix3 operator* (const Matrix3& a, const Matrix3& b) {
    if(a.size() != b.size()) {
        throw std::logic_error("a.size() != b.size()");
    }

    size_t size  = a.size();

    Matrix3 c;
    c.Resize(size);

    for(uint i = 0; i < size; i++) {
        for(uint j = 0; j < size; j++) {
            int c_ij = 0;
            for(uint k = 0; k < size; k++) {
                c_ij += a._matrix[i][k] + a._matrix[k][j];
            }
            c[i][j] = c_ij;
        }
    }

    return c;
}

Matrix3 operator* (const Matrix3& a, int b) {
    size_t size  = a.size();
    size_t msize = size*size;

    Matrix3 c;
    c.Resize(size);

    for(uint i = 0; i < msize; i++) {
        c._memory[i] = a._memory[i] * b;
    }

    return c;
}

}
```

Файл task3.cpp

```cpp
#include <iostream>
#include "matrix3.hpp"

using namespace std;
using namespace PT_l1_t3;

int main() {
    Matrix3 m1;
    Matrix3 m2;

    m1.Init(5);
    m2.Init(5);

    cout << "M1:\n";
    m1.Print();

    cout << "M2:\n";
    m2.Print();

    cout << "M1+M2=\n";
    (m1+m2).Print();

    cout << "M1-M2=\n";
    (m1-m2).Print();

    cout << "M1*M2=\n";
    (m1*m2).Print();

    cout << "M1*10=\n";
    (m1*10).Print();

    cout << "M3 = (M1 + M2)\n";
    Matrix3 m3 = (m1 + m2);
    m3.Print();

    return 0;
}
```

Задание 5.1
Файл matrix4.hpp

```
/*****************************************************************************

ModuleName:
    Matrix

Abstract:
    Matrix class, designed for Programming Technology class, triple extended version.

Author:
    Lubinets Mike 1441 (Лубинец Михаил 1441)

Date:
    2016-03-21

Changes
    Determinant calculation

*****************************************************************************/


#ifndef MATRIX4_H
#define MATRIX4_H
#include "matrix3.hpp"

namespace PT_l1_t5 {

class Matrix4 : public PT_l1_t3::Matrix3 {
public:
    using Matrix3::Matrix3;

    int Determinant();

private:
    static int Determinant(int** matrix, size_t size);
};

}

#endif // MATRIX4_H
```

Файл matrix4.cpp

```
#include "matrix4.hpp"
#include <stdexcept>
#include <cmath>

namespace PT_l1_t5 {

int
Matrix4::Determinant() {
    if(!_memory || !_matrix) {
        throw std::logic_error("Matrix must be initialized first");
    }

    if(_size < 1) {
        throw std::logic_error("Matrix size mast be >= 1 to calculate discriminant");
    }



    return Determinant(_matrix, _size);
}

int
Matrix4::Determinant(int** matrix, size_t size) {
    if(size == 1) {
        return matrix[0][0];
    }

    if(size == 2) {
        return matrix[0][0] * matrix[1][1] - matrix[1][0] * matrix[0][1];
    }

    int d = 0, k = 1;

#ifndef __GNUG__
    int** p = new int*[size-1];
#else
    int* p[size-1];
#endif


    uint j;
    for(uint i = 0; i < size; i++) {
        for(j = 0; j < size-1; j++) {
            if(j < i) {
                p[j] = matrix[j];
            } else {
                p[j] = matrix[j+1];
            }
        }

        k = ((i+j) % 2) ? -1 : 1;
        d += k * Determinant(p, size-1) * matrix[i][size-1];

    }

#ifndef __GNUG__
    delete[] p;
#endif

    return d;
}

}
```

Файл task5.1.cpp

```cpp
#include <iostream>
#include "matrix4.hpp"

using namespace std;
using namespace PT_l1_t5;


int main() {
    Matrix4 m;

    m.Init(5);
    m.Print();

    cout << "Det: " << m.Determinant() << endl;
}
```

3. Примеры

Задача 1:

Матрица 5x5

```
36  3   4   5   6
3   42  5   6   7
4   5   48  7   8
5   6   7   54  9
6   7   8   9   60
```

Повернутая матрица

```
6   5   4   3   36
7   6   5   42  3
4   5   48  7   8
9   54  7   6   5
60  9   8   7   6
```

Задача 2:
Ввод данных: 2

```
6   3
3   6
```

```
3   6
6   3
```

Задача 3:

M1

```
14  3   4
3   16  5
4   5   18
```

M2

```
14  3   4
3   16  5
4   5   18
```

M1+M2

```
28  6   8
6   32  10
8   10  36
```

M1-M2

```
0   0   0
0   0   0
0   0   0
```

M1*M2

```
42  45  48
45  48  51
48  51  54
```

M1*10

```
140 30  40
30  160 50
40  50  180
```

Задача 5.1:

1x1

```
0
```

Det = 0

2x2

```
6   3
3   6
```

Det = 27

3x3

```
14  3   4
3   16  5
4   5   18
```

Det = 3384

4x4

```
24  3   4   5
3   28  5   6
4   5   32  7
5   6   7   36
```

Det = 669456

5x5

```
36  3   4   5   6
3   42  5   6   7
4   5   48  7   8
5   6   7   54  9
6   7   8   9   60
```

Det = 206132400