



Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э.
Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУК "Информатика и управление"

КАФЕДРА ИУК5 "Системы обработки информации"

ЛАБОРАТОРНАЯ РАБОТА №2

«Одномерные и двумерные массивы.»

ДИСЦИПЛИНА: «Вычислительные алгоритмы»

Выполнил: студент гр.ИУК5-41Б

_____ (____ Шиндин А.О.____)
(Подпись) (Ф.И.О.)

Проверил:

_____ (____ Вершинин В.Е.____)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

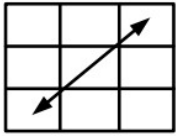
Калуга, 2023

Цель: выработать навыки реализации типовых алгоритмов обработки одномерных и двумерных массивов.

Задачи: В лабораторной работе необходимо:

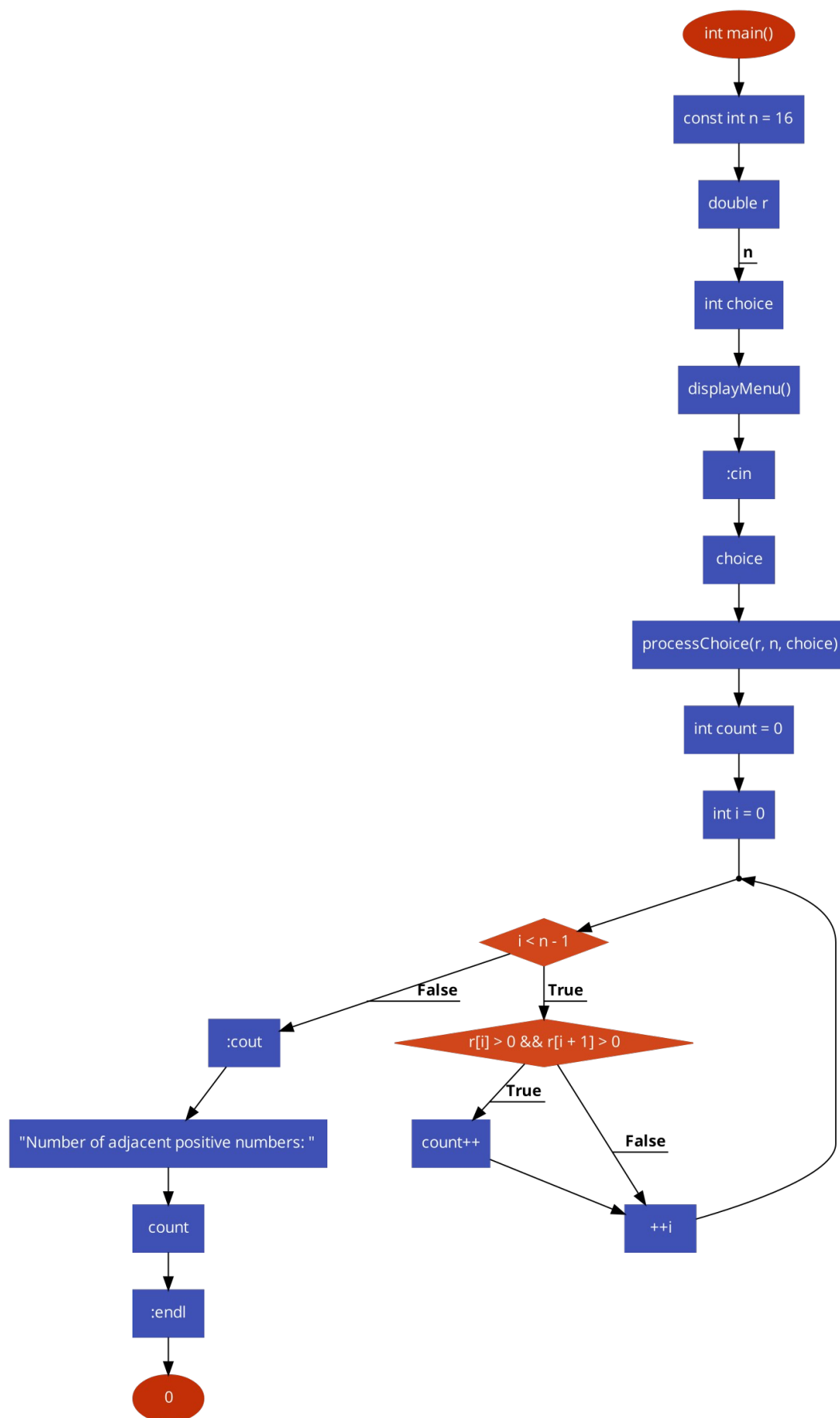
1. создать блок-схему алгоритма (см. лабораторную работу № 1)
2. реализовать заданный алгоритм Алгоритм выбирается в соответствии с вариантом задания, полученным от преподавателя.

Задание:

Вариант		Задания			
12		Даны: действительные числа $r_1, r_2, r_3, \dots, r_{16}$. Определить количество соседств двух положительных чисел в последовательности $r_1 \dots r_{16}$.			
12	$\frac{(i+j)\sin(i)}{\cos(j)},$ (6×6)	упорядочить элементы в строках по убыванию	$\begin{matrix} 1 & 2 & 3 & \dots & n \\ 1 & 2 & 3 & \dots & n \\ 1 & 2 & 3 & \dots & n \\ \dots & \dots & \dots & \dots & \dots \\ 1 & 2 & 3 & \dots & n \end{matrix}$		количество положительных элементов главной диагонали

Ход работы:

- 1) Блок-схема алгоритма:



Программа написанная на языке с++:

```
#include <iostream>
```

```
#include <cstdlib>
```

```
#include <ctime>
```

```
void displayMenu() {
```

```
    std::cout << "Choose the method of filling the sequence:\n";
```

```
    std::cout << "1. Enter numbers manually\n";
```

```
    std::cout << "2. Random generation\n";
```

```
    std::cout << "3. Use predefined numbers\n";
```

```
    std::cout << "Your choice: ";
```

```
}
```

```
void processChoice(double r[], int n, int choice) {
```

```
    switch (choice) {
```

```
        case 1:
```

```
            std::cout << "Enter " << n << " real numbers:\n";
```

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

```
                std::cout << "r" << i + 1 << ": ";
```

```
                std::cin >> r[i];
```

```
            }
```

```
            break;
```

```
        case 2:
```

```
            // Random generation
```

```
            std::cout << "Generating random numbers...\n";
```

```
            srand(time(0)); // Initialize random number generator
```

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

```
                r[i] = (rand() % 1000) / 100.0; // Generate a random number in the range [0, 10)
```

```
            }
```

```
            break;
```

```
        case 3:
```

```
            double predefined[] = {1.2, -3.5, 0.8, 2.1, 4.5, -0.6, 7.2, 8.3, -2.0, 1.9, 3.4, 5.6, -4.3,  
                                   6.7, 9.0, 10.1};
```

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

```
                r[i] = predefined[i];
```

```
            }
```

```
            break;
```

```
        }
```

```
}
```

```
int main() {  
    const int n = 16;  
    double r[n];
```

```
    int choice;  
    displayMenu();  
    std::cin >> choice;
```

```
    processChoice(r, n, choice);
```

```
    int count = 0;  
    for (int i = 0; i < n - 1; ++i) {  
        if (r[i] > 0 && r[i + 1] > 0)  
            count++;  
    }
```

```
    std::cout << "Number of adjacent positive numbers: " << count << std::endl;  
    return 0;  
}
```

Результат:

```
Choose the method of filling the sequence:  
1. Enter numbers manually  
2. Random generation  
3. Use predefined numbers  
Your choice: 3  
Number of adjacent positive numbers: 7  
alex@fedora:~/Documents/code/alg$
```

2) Программа написанная на языке c++:

```
#include <iostream>  
  
#include <vector>
```

```

#include <cmath>
#include <algorithm>

class Matrix {
private:
    std::string name;
    int rows;
    int cols;
    std::vector<std::vector<double>> data;

public:
    // Constructor to initialize the matrix with given dimensions
    Matrix(std::string _name, int _rows, int _cols)
    : name(_name), rows(_rows), cols(_cols),
    data(_rows, std::vector<double>(_cols, 0.0)) {}

    // Function to create a matrix based on the given formula
    void createMatrix() {
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < cols; ++j) {
                data[i][j] = ((i + j) * sin(i)) / cos(j);
            }
        }
    }

    // Function to create a matrix where each row contains a sequence from 1 to n
    void createMatrixWhereEachRowInSimpleSequence1toN() {
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < cols; ++j) {
                data[i][j] = j + 1; // Adding 1 to the index to start numbers from 1 instead of 0
            }
        }
    }

    // Function to swap specified elements in the matrix
    void swapElements(int row1, int col1, int row2, int col2) {
        std::swap(data[row1][col1], data[row2][col2]);
    }

```

```
}
```

```
// Function to sort elements in each row of the matrix in descending order
void sortRowsDescending() {
for (auto& row : data) {
std::sort(row.begin(), row.end(), std::greater<double>()); // Sort the row in
descending order
}
}
```

```
// Function to check if two matrices can be multiplied
bool canMultiply(const Matrix& other) const {
return cols == other.rows;
}
```

```
// Function to multiply two matrixes
Matrix multiply(const Matrix& other , std::string _name) const {
if (!canMultiply(other)) {
std::cerr << "Error: Matrices cannot be multiplied. Number of columns in first matrix
must be equal to number of rows in second matrix." << std::endl;
return Matrix("Error", 0, 0);
}
```

```
Matrix result(_name, rows, other.cols);
```

```
for (int i = 0; i < rows; ++i) {
for (int j = 0; j < other.cols; ++j) {
for (int k = 0; k < cols; ++k) {
result.data[i][j] += data[i][k] * other.data[k][j];
}
}
}
```

```
return result;
}
```

```

// Function to copy the matrix
Matrix copy(std::string _name) const {
    Matrix copyMatrix(_name, rows, cols);
    copyMatrix.data = data;
    return copyMatrix;
}

// Function to swap specified elements in matrix E
void swapElementsInMatrixE(Matrix& E, int m, int n) {

    for (int i = 0; i < (int)(m / 3); i++)
    {
        for (int j = 0; j < (int)(n / 3); j++)
        {
            E.swapElements(i, (int)(n-(j+1)), (int)(n-(j+1)), i);
        }
    }

    // E.swapElements(0, 5, 5, 0); // Swap elements (1, 6) and (6, 1)
    // E.swapElements(0, 4, 4, 0); // Swap elements (1, 5) and (5, 1)
    // E.swapElements(1, 4, 4, 1); // Swap elements (2, 5) and (5, 2)
    // E.swapElements(1, 5, 5, 1); // Swap elements (2, 6) and (6, 2)
}

// Function to count positive elements on the main diagonal and print matrix info
void countPositiveDiagonalElements() const {
    std::cout << "\nMatrix " << name << " (" << rows << "x" << cols << "):" <<
    std::endl;
    std::cout << "Elements on the main diagonal:" << std::endl;
    int count = 0;
    for (int i = 0; i < std::min(rows, cols); ++i) {
        std::cout << data[i][i] << " ";
        if (data[i][i] >= 0) {
            ++count;
        }
    }
    std::cout << std::endl;
    std::cout << "Number of positive elements on the main diagonal: " << count <<
    std::endl;
}

```



```
}
```

```
// Function to print the matrix
void printMatrix() const {
    std::cout << "\nMatrix " << name << " (" << rows << "x" << cols << "):" <<
    std::endl;
    for (const auto& row : data) {
        for (double element : row) {
            std::cout << element << " ";
        }
        std::cout << std::endl;
    }
}
};
```

```
int main() {
    const int m = 6; const int n = m;
```

```
// Create matrix A with user input
Matrix A("A", m, n);
A.createMatrix(); // Fill matrix A with values based on the formula
```

```
// Print matrix A
A.printMatrix();
```

```
// Copy matrix A to matrix B
Matrix B = A.copy("B");
```

```
// Sort elements in each row of matrix B in descending order
B.sortRowsDescending();
// Print matrix B with sorted rows
B.printMatrix();
```

```
Matrix C("C", m, n);
C.createMatrixWhereEachRowInSimpleSequence1toN();
```

```

// Print matrix C
C.printMatrix();

// Multiply matrices B and C if possible
// if (B.canMultiply(C)) {
// Matrix D = B.multiply(C, "D");
// // Print matrix D
// D.printMatrix();
// }

Matrix D = B.multiply(C, "D");
// Print matrix D
D.printMatrix();

// Create matrix E as a copy of matrix B
Matrix E = D.copy("E");

// Swap [2*2] elements in diagonal matrix E
E.swapElementsInMatrixE(E, m, n);

// Print matrix E with swapped elements
E.printMatrix();

// Count positive elements on the main diagonal of matrix E and print matrix info
E.countPositiveDiagonalElements();

return 0;
}

```

Результат:

Matrix A (6x6):

```
0 0 -0 -0 -0 0
0.841471 3.11482 -6.06616 -3.39991 -6.43677 17.7987
1.81859 5.04883 -8.74016 -4.59245 -8.34673 22.439
0.42336 1.04475 -1.69556 -0.855279 -1.51128 3.97995
-3.02721 -7.00351 10.9116 5.35117 9.26257 -24.0117
-4.79462 -10.6488 16.1301 7.74894 13.2034 -33.8052
```

Matrix B (6x6):

```
0 0 -0 -0 -0 0
17.7987 3.11482 0.841471 -3.39991 -6.06616 -6.43677
22.439 5.04883 1.81859 -4.59245 -8.34673 -8.74016
3.97995 1.04475 0.42336 -0.855279 -1.51128 -1.69556
10.9116 9.26257 5.35117 -3.02721 -7.00351 -24.0117
16.1301 13.2034 7.74894 -4.79462 -10.6488 -33.8052
```

Matrix C (6x6):

```
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
```

Matrix D (6x6):

```
0 0 0 0 0 0
5.85217 11.7043 17.5565 23.4087 29.2609 35.113
7.62704 15.2541 22.8811 30.5082 38.1352 45.7622
1.38594 2.77187 4.15781 5.54375 6.92969 8.31562
-8.51715 -17.0343 -25.5515 -34.0686 -42.5858 -51.1029
-12.1661 -24.3323 -36.4984 -48.6645 -60.8307 -72.9968
```

```
Matrix D (6x6):
0 0 0 0 0 0
5.85217 11.7043 17.5565 23.4087 29.2609 35.113
7.62704 15.2541 22.8811 30.5082 38.1352 45.7622
1.38594 2.77187 4.15781 5.54375 6.92969 8.31562
-8.51715 -17.0343 -25.5515 -34.0686 -42.5858 -51.1029
-12.1661 -24.3323 -36.4984 -48.6645 -60.8307 -72.9968
```

```
Matrix E (6x6):
0 0 0 0 -8.51715 -12.1661
5.85217 11.7043 17.5565 23.4087 -17.0343 -24.3323
7.62704 15.2541 22.8811 30.5082 38.1352 45.7622
1.38594 2.77187 4.15781 5.54375 6.92969 8.31562
0 29.2609 -25.5515 -34.0686 -42.5858 -51.1029
0 35.113 -36.4984 -48.6645 -60.8307 -72.9968
```

```
Matrix E (6x6):
Elements on the main diagonal:
0 11.7043 22.8811 5.54375 -42.5858 -72.9968
Number of positive elements on the main diagonal: 4
alex@fedora:~/Documents/code/alg$
```

Вывод: в результате выполнения лабораторной работы были приобретены практические навыки и реализации типовых алгоритмов обработки одномерных и двумерных массивов.