**Аннотация**

В курсовой работе преследуется цель создания прикладного программного обеспечения, функционал которого соответствует предлагаемому варианту задания.

Для разработки программного обеспечения используется интегрированная среда разработки программного обеспечения VSCodium (open-source Visual Studio Code без надстроек от Microsoft). Создание приложения производится с использованием современных технологий программирования на языке C++.

Пояснительная записка состоит из теоретической и практической частей.

Теоретическая часть включает в себя описания метода решения задачи, обоснование выбор метода, с помощью которого осуществляется решение задачи, его преимущества и недостатки по сравнению с другим методами. Также описываются, какие стандартные функции или функции пользователя будут использоваться в программе, какова будет структура программы.

Практическая часть содержит блок-схему, листинг программы в среде VSCodium, результаты работы программы.

Оглавление

[ИСХОДНЫЕ ДАННЫЕ 3](#_Toc71916517)

[ПОДРОНОЕ ОПИСАНИЕ ИГРЫ 4](#_Toc71916518)

[РЕШЕНИЕ. 4](#_Toc71916519)

[ОПИСАНИЕ СТРУКТУРЫ ПРОГРАММЫ 5](#_Toc71916520)

[ЛИСТИНГ ПРОГРАММЫ 5](#_Toc71916521)

[БЛОК СХЕМА АЛГОРИТМА 8](#_Toc71916522)

[РЕЗУЛЬТАТ РАБОТЫ 9](#_Toc71916523)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 11](#_Toc71916524)

# ИСХОДНЫЕ ДАННЫЕ

Необходимо реализовать класс «Матрица». Матрица является квадратной по условию, её размерность N задаётся пользователем через консоль после запуска программы. После этого на экран выводится некая информация о матрице, а именно:

* Количество строк, в которых нет ни единого нулевого элемента
* Неединственный(!) наибольший элемент, если такой существует

Заполняется матрица случайными числами из отрезка [-N - 2, N - 2].

# ПОДРОНОЕ ОПИСАНИЕ ИГРЫ

При запуске приложения пользователю предлагается ввести число N, которое является размером нашей квадратной матрицы, дальнейшие шаги производятся приложением без участия пользователя.

Как уже было сказано – после завершения работы приложения пользователь увидит на экране данные, которые были указаны в задании.

# РЕШЕНИЕ.

Задача решена с помощью использования высокоуровневого языка С++. Был выбран именно этот язык, потому что он кроссплатформенный, а главное – достаточно быстро выполняет всё, что напишет пользователь без лишних затрат при должных умениях программиста. Язык является Тьюринг-полным, так что нет ни единой причины выбирать что-то другое.

После ввода размерности матрицы приложение создаёт объект std::vector<std::vector<int64\_t>> внутри класса «Matrix», где std::vector – один из контейнеров из STL (Standard Template Library), а int64\_t – 64битное целое знаковое число. Выбран был именно этот контейнер, потому что он динамически выделит нужную память в «куче», автоматически почистит её и расходы на него не являются критичными в нашей ситуации.

Для проверки на наличие нулей в строках используется алгоритм std::find(), который принимает итераторы начала вектора и конца. Он возвращает итератор на первый найденный элемент в векторе или же возвращает указатель на его конец. Алгоритм легко воспроизводим руками, но в нашем случае зачем писать то, что до нас уже было написано, оптимизированно и введено в стандарт.

Для нахождения наибольшего неединственного числа было решено пойти по пути «больше затрат памяти, меньше затрат по времени» и был разработан алгоритм по «сжатию» квадратной матрицы в вектор. После чего он сортируется быстрой сортировкой, которая используется алгоритмом std::sort(), который прекрасно работает в нашем случае. На выходе мы имеем отсортированный вектор, пройдя по которому мы можем без лишних затрат по времени (за О(N)) найти наш искомый элемент сравнением двух соседних элементов. Попарно проверяя элементы с конца, мы ищем момент, в который будет хотя-бы одно совпадение и выводим его. Очевидно, что такой элемент будет наибольшим, поскольку вектор уже отсортирован по возрастанию. В случае отсутствия таких элементов пользователь увидит соответствующее сообщение.

# ОПИСАНИЕ СТРУКТУРЫ ПРОГРАММЫ

Был разработан отдельный класс Matrix, в котором автоматически динамически выделяется нужная нам память. В файле «Matrix.h» объявлен сам класс и его функции, в «Matrix.ccp» находится их реализация, а файл «main .cpp» -файл с функцией int main(), где и происходит создание объекта класса Matrix и выполнение его функций.

# ЛИСТИНГ ПРОГРАММЫ

//main.cpp

#include "matrix.hpp"

#include <iostream>

int main() {

    std::cout << "Input N: ";

    uint32\_t N;

    std::cin >> N;

    Matrix m(N);

    std::cout << "Rows without 0: " << m.CountNotEmptyRows() << '\n';

    m.NotAloneMaxElement(0, std::cout);

}

//matrix.hpp

#pragma once

#include <algorithm>

#include <ctime>

#include <iostream>

#include <ostream>

#include <random>

#include <utility>

#include <vector>

//Для быстрой смены размера, если нужно

using intize = int32\_t;

using nsize = uint32\_t;

class Matrix {

private:

  std::vector<std::vector<intize>> elements;

public:

  Matrix(nsize N) noexcept;

  //Подсчёт строк без 0 элементов

  intize CountNotEmptyRows() const;

  //Нахождение неединственного наибольшего

  void NotAloneMaxElement(bool, std::ostream &) noexcept;

  const std::vector<intize> &operator[](size\_t i) const { return elements[i]; }

  std::vector<intize> &operator[](uint64\_t i) { return elements[i]; }

  //Оператор вывода

  friend std::ostream &operator<<(std::ostream &out, const Matrix &matrix);

};

//matrix.cpp

#include "matrix.hpp"

 Matrix::Matrix(nsize N) noexcept {

    srand(time(NULL));

    elements.assign(N, std::vector<intize>(N));

    for (nsize i = 0; i < N; i++) {

      for (nsize j = 0; j < N; j++) {

        elements[i][j] = rand() % N \* 2 - N;

      }

    }

  }

 intize Matrix::CountNotEmptyRows() const {

    intize result = 0;

    for (nsize i = 0; i < elements.size(); i++) {

      auto number = std::find(elements[i].begin(), elements[i].end(), 0);

      if (number == elements[i].end())

        result++;

    }

    return result;

  }

  void Matrix::NotAloneMaxElement(bool showsorted = 0,

                          std::ostream &out = std::cout) noexcept {

    //Сжатие матрицы в вектор

    auto size = elements.size() \* elements.size();

    std::vector<intize> all\_elements(size);

    for (nsize i = 0; i < elements.size(); i++) {

      for (nsize j = 0; j < elements.size(); j++) {

        all\_elements[i \* elements.size() + j] = elements[i][j];

      }

    }

    //Сортировка

    std::sort(all\_elements.begin(), all\_elements.end());

    if (showsorted) {

      for ( const auto &a : all\_elements)

        std::cout << a << ' ';

      std::cout << '\n';

    }

    //Нахождение искомого

    for (nsize i = size - 1; i > 1; i--) {

      if (all\_elements[i] == all\_elements[i - 1]) {

        out << "Not alone biggest number is: " << all\_elements[i] << '\n';

        return;

      }

    }

    out << "No same numbers\n";

  }

  std::ostream &operator<<(std::ostream &out, const Matrix &matrix) {

    for (int row = 0; row < matrix.elements.size(); row++) {

      for (int column = 0; column < matrix.elements.size(); column++) {

        if (column > 0) {

          out << '\t';

        }

        out << matrix[row][column];

      }

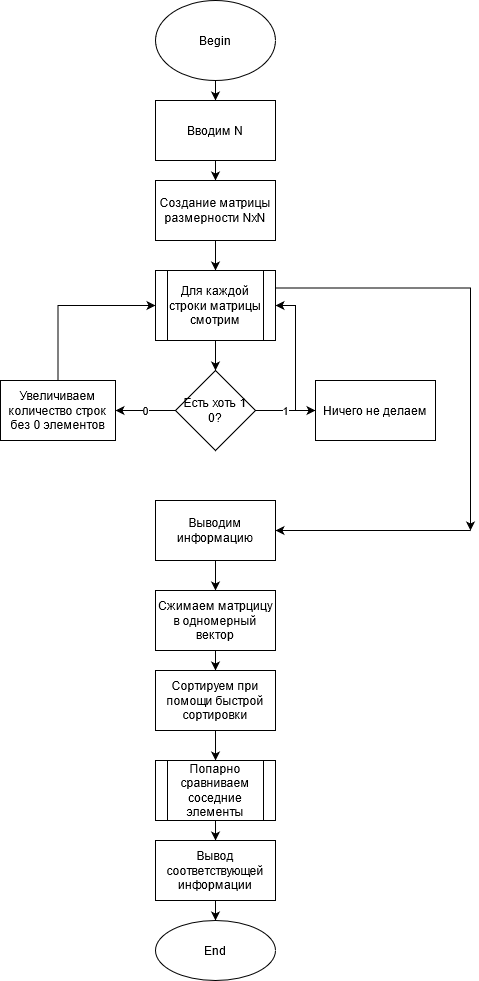
      out << '\n';

    }

    return out;

  }

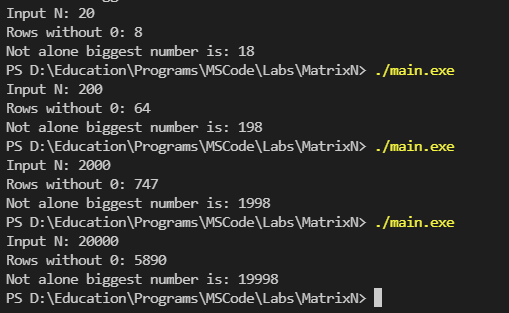
# БЛОК СХЕМА АЛГОРИТМА



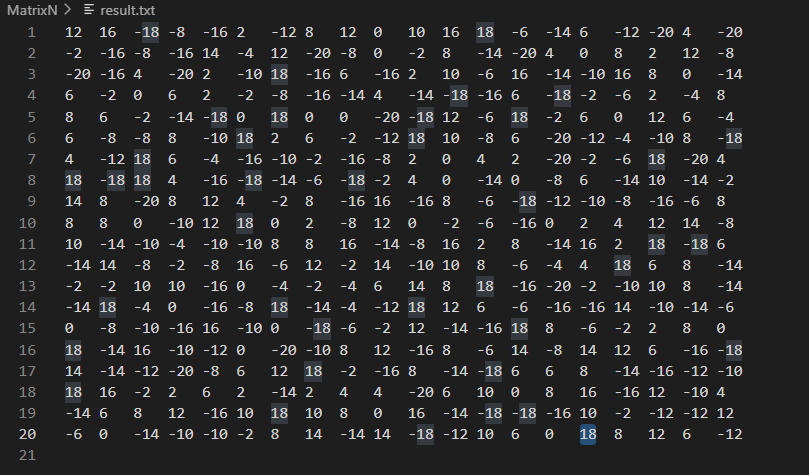
# РЕЗУЛЬТАТ РАБОТЫ

После запуска приложения пользователю нужно ввести N, после чего программа продолжит работу:

Как пример будут рассмотрены случаи для N = 20, 200, 2000 и 20000.

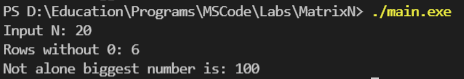


Рассмотрим случай для 20 элементов, что-бы проверить правильность работы программы, для этого запишем нашу матрицу в файл и вручную сравним данные.



Как видно со снимка экрана –максимальный элемент 18 встречается достаточно часто, а больше него не может быть по условию. Для уменьшения вероятности выпадения одинаковых элементов можно увеличить выборку.

В итоге мы видим, что программа работает без ошибок. Для полного удостоверения в этом вручную изменим данные. Пускай 2 наугад выбранных элемента точно будут больше, к примеру зададим им значение 100.



Что опять же подтверждает правильность работы.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Мейерс, Скотт. М45 Эффективный и современный С++: 42 рекомендации по использованию С++ 11 и С++14 -304 с.