**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Информационных систем**

отчет

**по курсовой работе**

**по дисциплине «Программирование»**

Тема: **Основы алгоритмизации и программирования на языке С++**

|  |  |  |
| --- | --- | --- |
| Студент гр. 0323 |  | Кольцов К.Э |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2020

**Цель работы.**

Необходимо объединить все 4 лабораторные работы в единый проект. Нужно добавить инфраструктуру переключения между заданиями (интерактивное меню).

В первой работе необходимо разработать алгоритм и написать программу, которая позволяет:

1) Вывести, сколько памяти (в байтах) на вашем компьютере отводится под различные типы данных со спецификаторами и без: int, short int, long int, float, double, long double, char и bool.

2) Вывести на экран двоичное представление в памяти (все разряды) целого числа. При выводе необходимо визуально обозначить знаковый разряд и значащие разряды отступами или цветом.

3) Вывести на экран двоичное представление в памяти (все разряды) типа float. При выводе необходимо визуально обозначить знаковый разряд мантиссы, знаковый разряд порядка (если есть), мантиссу и порядок.

4) Вывести на экран двоичное представление в памяти (все разряды) типа double. При выводе необходимо визуально обозначить знаковый разряд мантиссы, знаковый разряд порядка (если есть), мантиссу и порядок.

Во второй необходимо написать программу, которая:

1)    Создает целочисленный массив размерности *N* = 100. Элементы массивы должны принимать случайное значение в диапазоне от -99 до 99.

2)    Отсортировать заданный в пункте 1 элементы массива […] Quick sort (быстрой сортировкой). Определить время, затраченное на сортировку, используя библиотеку chrono.

3)    Найти максимальный и минимальный элемент массива. Подсчитайте время поиска этих элементов в отсортированном массиве и неотсортированном, используя библиотеку chrono.

4)    Выводит среднее значение (если необходимо, число нужно округлить) максимального и минимального значения. Выводит индексы всех элементов, которые равны этому значению, и их количество.

5)    Выводит количество элементов в отсортированном массиве, которые меньше числа *a*, которое инициализируется пользователем.

6)    Выводит количество элементов в отсортированном массиве, которые больше числа *b*, которое инициализируется пользователем.

7)    Выводит информацию о том, есть ли введенное пользователем число в отсортированном массиве. Реализуйте алгоритм бинарного поиска. Сравните скорость его работы с обычным перебором.

8)     Меняет местами элементы массива, индексы которых вводит пользователь. Выведите скорость обмена, используя библиотеку chrono.

Должна присутствовать возможность запуска каждого пункта многократно.

В третьей необходимо написать программу, которая:

1)    Используя арифметику указателей, заполняет квадратичную целочисленную матрицу порядка *N* (6,8,10) случайными числами от 1 до  N\*N согласно схемам (смю рис. 1), приведенным на рисунках. Пользователь должен видеть процесс заполнения квадратичной матрицы.



Рисунок 1 – Схемы заполнения матриц

2)    Получает новую матрицу, из матрицы п. 1, переставляя ее блоки в соответствии со схемами (см. рис. 2).



Рисунок 2 – Схемы перестановки матрицы по квадратам 2 на 2

3)    Используя арифметику указателей, сортирует элементы любой сортировкой.

4)    Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число.

В четвертой необходимо написать программу, которая реализует поставленную задачу:

1)    С клавиатуры или из файла (пользователь сам может выбрать способ ввода) вводится последовательность, содержащая от 1 до 50 слов, в каждом из которых от 1 до 10 строчных латинских букв и цифр. Между соседними словами произвольное количество пробелов. За последним символом стоит точка.

2)    Необходимо отредактировать входной текст:

·        удалить лишние пробелы;

·        удалить лишние знаки препинания (под «лишними» подразумевается несколько подряд идущих знаков (обратите внимание, что «…» - корректное использование знака) в тексте);

·        исправить регистр букв, если это требуется (пример некорректного использования регистра букв: пРиМЕр);

3) Выполнить задание по варианту:

Вывести на экран слова последовательности, не содержащие цифр.

4) Выполнить задание по варианту:

Вывести на экран ту же последовательность, удалив из всех слов заданный набор букв и (или) цифр.

5)  Необходимо найти подстроку, которую введёт пользователь в имеющейся строке. Реализуйте два алгоритма: первый алгоритма – Линейный поиск, а второй алгоритм согласно вашему номеру в списке. Четные номера должны реализовать алгоритм КНМ, а нечетные – Бойера-Мура.

**Основные теоретические положения.**

Числа в памяти компьютера представлены в виде целого двоичного числа. Для знаковых чисел старший разряд числа используется для обозначения знака: ноль для неотрицательных чисел и единица для отрицательных. Для простоты внутренней арифметики последние представлены в дополненном коде. Он получается прибавлением единицы к обратному коду числа (в котором все разряды модуля числа инвертированы).

Вещественные типы бывают только знаковыми. Для них отличается представление в памяти. Они переводятся в нормализованную форму двоичной системы счисления. Так как нормализованная мантисса в двоичной системе счисления больше либо равна единице и меньше двух, значит она всегда начинается с единицы и её можно не записывать в память, освободив место для еще одного разряда порядка или мантиссы. У порядка знакового бита нет, вместо этого к нему прибавляется половина диапазона. Например, порядок равный нулю будет начинаться нулём, а все остальные разряды будут единицами.

Для нахождения размера занимаемой памяти используется функция sizeof, возвращающая количество байт для соответствующего типа данных. Для вывода представления чисел в памяти используется & (оператор побитового И) вместе с числом с единственной единицей в двоичной записи (два в неотрицательной степени), а для вещественных типов данных union (объединение), для получения доступа в области памяти через целочисленные типы (int для float и long long для double).

Массив представляет собой индексированную последовательность однотипных элементов с заранее определенным количеством элементов. Наглядно одномерный массив можно представить, как набор пронумерованных ячеек, в каждой из которых содержится определенное значение.

Все массивы можно разделить на две группы: одномерные и многомерные. Описание массива в программе отличается от объявления обычной переменной наличием размерности массива, которая задается в квадратных скобках после имени.

Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы (класс памяти, const и инициализатор), что и для простых переменных.

Аналогом одномерного массива из математики может служить последовательность некоторых элементов с одним индексом: a\_i*ai*​ при  i = 0, 1, 2, … n – одномерный вектор. Каждый элемент такой последовательности представляет собой некоторое значение определенного типа данных.

Быстрая сортировка (quick sort) – одна из самых быстрых сортировок. Эта сортировка по сути является существенно улучшенной версией алгоритма пузырьковой сортировки.

Общая идея алгоритма состоит в том, что сначала выбирается из массива элемент, который называется опорным. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность. Затем необходимо сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующие друг за другом: меньше опорного, раны опорному и больше опорного. Для меньших и больших значений необходимо выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят на две части: «меньше опорного» и «равные и большие» или «меньше опорного или равные» и «большие». Такой поход в общем случае эффективнее, ведь упрощается алгоритм разделения.

Бинарный поиск работает только в топ случае, если массив отсортирован. Например, если бы искомое минимальное значение стояло не на своем положенном месте, а на месте максимального элемента, то мы бы откинули его на первой же итерации. Сам алгоритм имеет вид:

1)    Определение значения в середине массива (или иной структуры данных). Полученное значение сравнивается с ключом (значением, которое необходимо найти).

2)    Если ключ меньше значения середины, то необходимо осуществлять поиск в первой половине элементов, иначе – во второй.

3)    Поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с ключом.

4)    Процесс продолжается до тех пор, пока не будет определен элемент, равный значению ключа или не станет пустым интервал для поиска.

Тип данных string является классом для работы со строками. Работать с ним можно как с массивом символов, но также имеются собственные методы, упрощающие работу. В отличии от массива имеет свой собственный адрес, поэтому для создания указателя \*char нужно приравнивать его к нулевому элементу simpleString[0], также при передачи строки в метод передается копия строки, поэтому для работы со строкой можно использовать указатель на нее.

Для объединения в большой проект в языке c++ существует определенный механизм – заголовочные файлы. В них содержатся прототипы функций, реализованных в файле. Так в этой работе есть четыре основных файла с лабораторными работами по дисциплине «программирование», для каждого из них существует заголовочный файл, который добавляется в начало программы и позволяет скомпилировать общий файл с этими функциями. В процессе работы программы она может обратиться к функциям из другого файла за счет прототипов функций.

Используется это для создания коротких и простых main программ, за которыми могут скрываться десятки и сотни исполнительных классов и библиотек. Так в большом проекте много людей могут заниматься разными участками программы, не затрагивая остальные. Принцип черного ящика (мы жестко определяем, что в него входит и что выходит, а внутрь не лезем, если лишь используем его) позволяет находить проблемы в отдельных методах и файлах, облегчая тестирование и масштабирование проекта.

**Экспериментальные результаты.**

Программа выводит на экран приветствие и считывает номер работы, которую следует запустить, после этого работает отдельный файл, и по закрытию лабораторной курсовая выводит приветствие и снова считывает команду

После этого программа завершает работу (см. рис. 3)

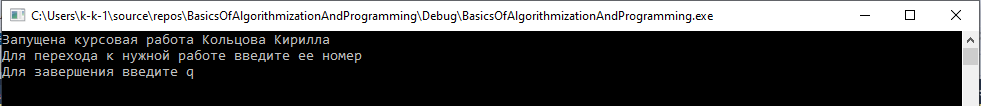


Рисунок 3 – Начало работы файла курсовой работы

Первая программа выводит на экран, сколько памяти (в байтах) на компьютере отводится под различные типы данных со спецификаторами и без: int, short int, long int, float, double, long double, char и bool.

Затем по порядку считываются, обрабатываются и выводятся двоичные представления в памяти типов int, float, double.

После этого программа завершает работу (см. рис. 4)

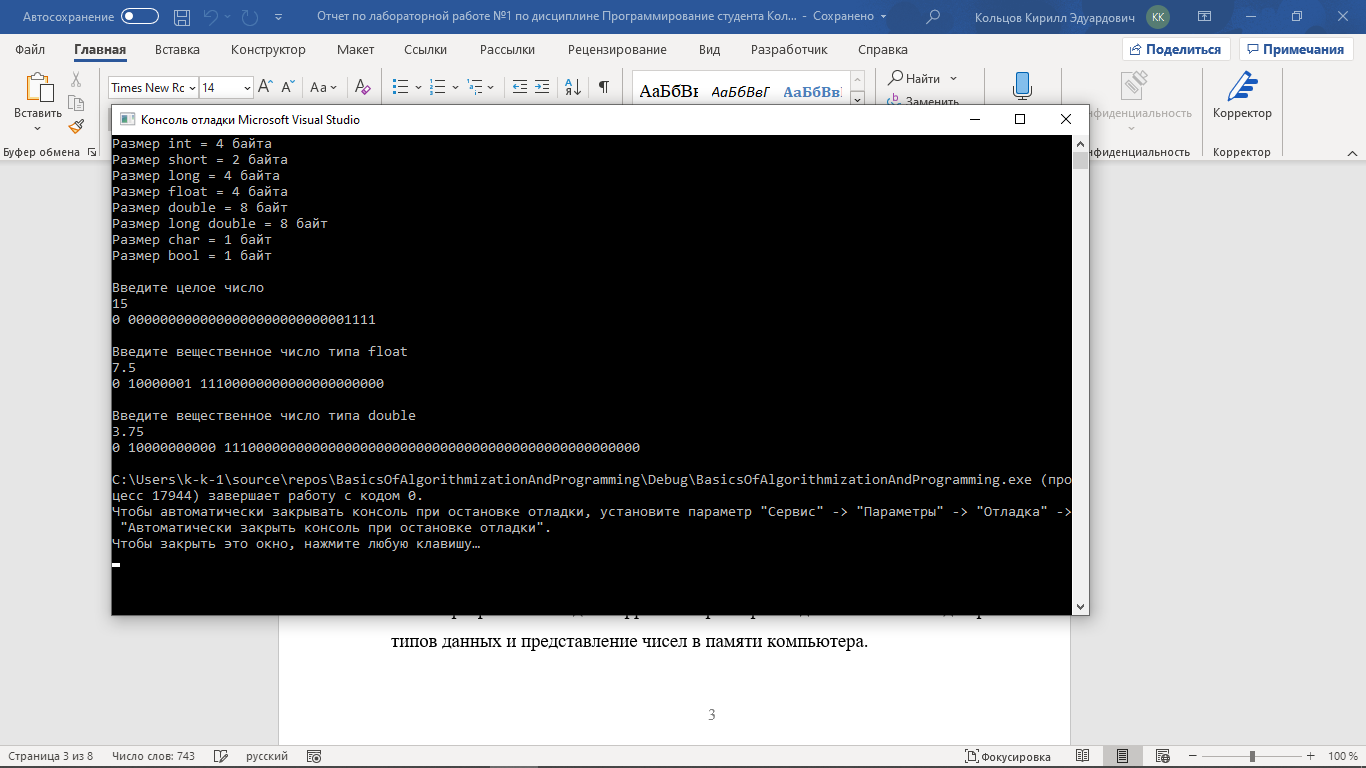


Рисунок 4 – Пример работы первой программы

Вторая программа выполняет команды, которые пишет пользователь, всего команд 7:

Команда h выводит список команд, команда e завершает программу, остальные команды вызывают соответствующие методы в коде программы.

Команда s позволяет заново сгенерировать массив, через функцию newArr, которая вызывается в начале программы для создания массива.

Команда b считывает число с ввода и выдает количество элементов больше заданного. Команда l работает аналогично для чисел меньше заданного.

Команда m находит количество и индексы элементов в массиве, равных половине суммы максимального и минимального элементов массива.

Команда w запрашивает пару чисел и меняет местами два элемента массива с соответствующими индексами. Если индексы не совпадают, то массив становится неотсортированным, из-за чего перед каждой функцией, требующей порядка в массиве проводится сортировка.

Команда f находит число, введённое пользователем, в массиве бинарным поиском и простым перебором, для сравнения скорости работы.

Команда e заканчивает работу программы.

Работу каждой функции можно просмотреть отдельно (см. рис. 5, 6, 7, 8)

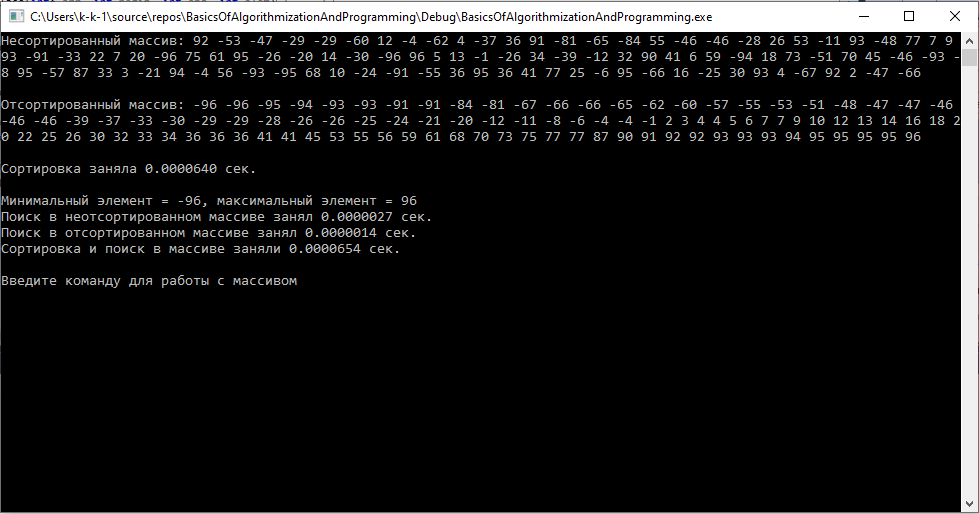


Рисунок 5 – Начало работы

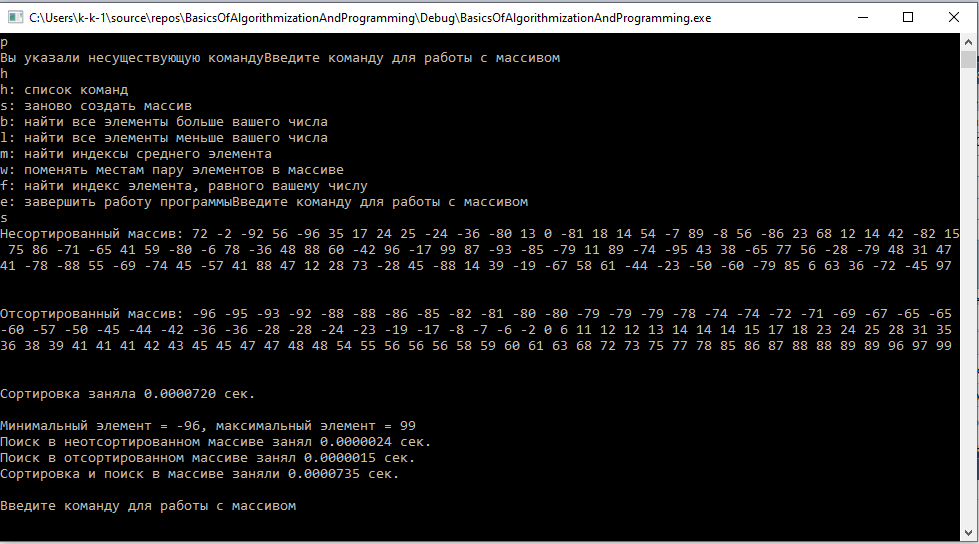


Рисунок 6 – Неверная команда, Команды h и s

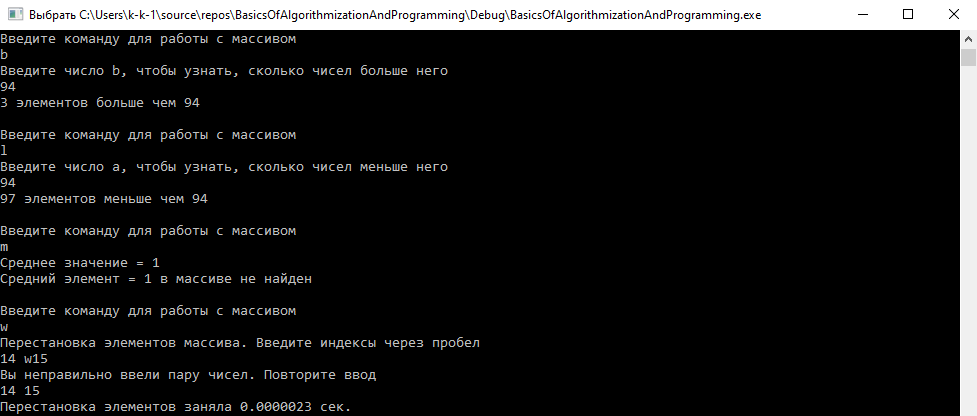


Рисунок 7 – Команды b и l

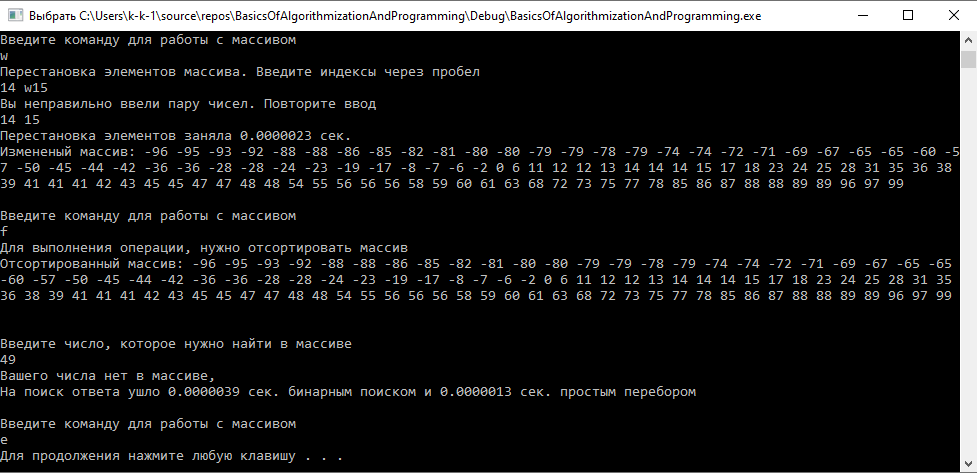


Рисунок 8 – Команды w,f и e, завершение второй программы

Программа выполняет команды, которые пишет пользователь, всего команд 6:

Команда h выводит список команд, команда e завершает программу, остальные команды вызывают соответствующие методы в коде программы.

Команда m позволяет заново сгенерировать матрицу.

Команда с считывает оператор (+, -, \* или :) и затем число, для изменения всей матрицы.

Команда s сортируют матрицу быстрой сортировкой по схеме создания.

Команда p переставляет элементы матрицы по схемам квадратов 2 на 2.

Работу каждой функции можно просмотреть отдельно (см. рис. 9, 10, 11)

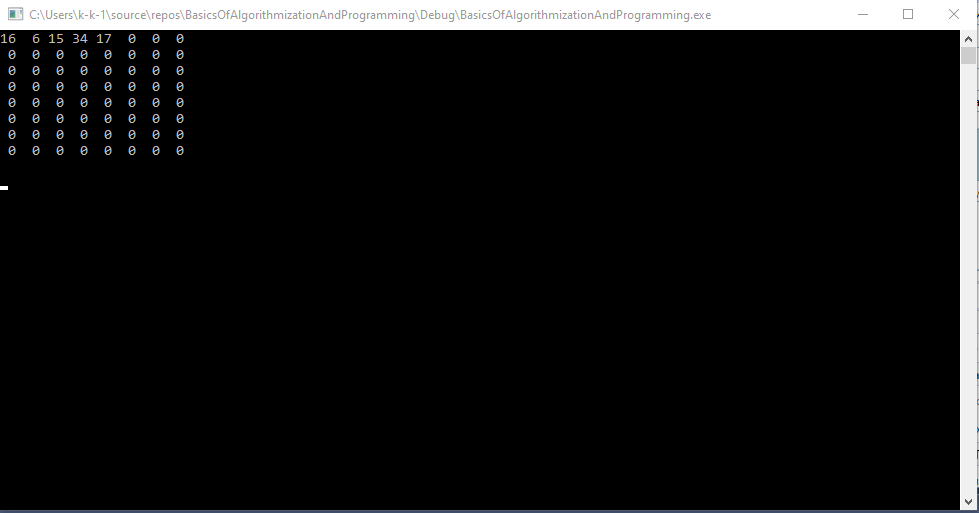


Рисунок 9 – Процесс заполнения матрицы

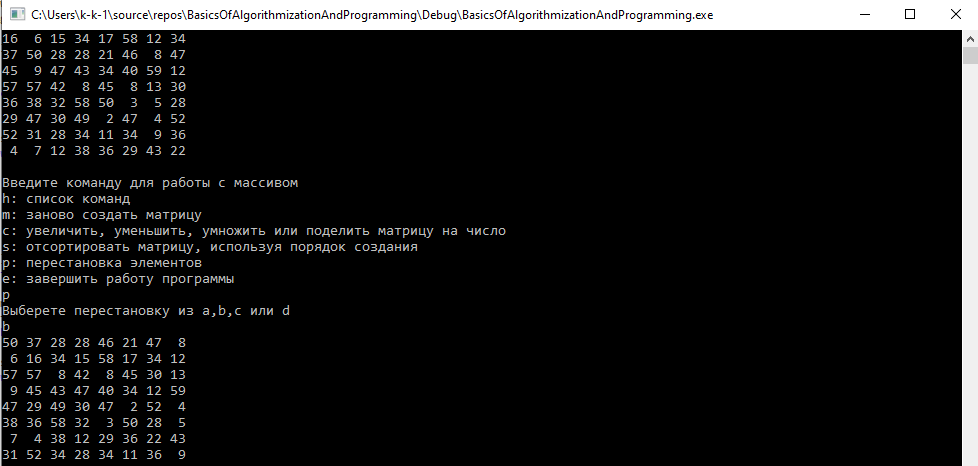


Рисунок 10 – Готовая матрица и перестановка элементов

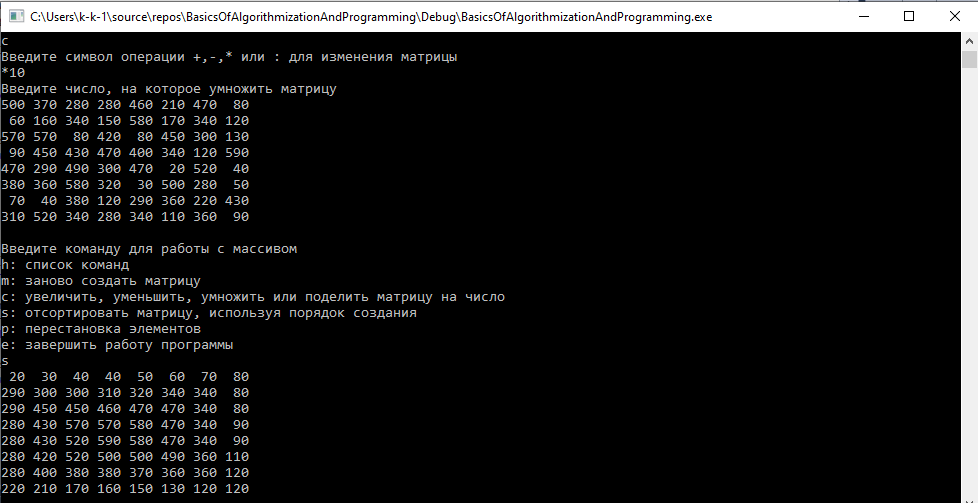


Рисунок 11 – Умножение матрицы на число и сортировка по схеме создания

Программа выполняет команды, которые пишет пользователь, всего команд 5:

Команда h выводит список команд, команда e завершает программу, остальные команды вызывают соответствующие методы в коде программы.

Команда n позволяет заново задать предложение тремя способами:

f: из файла; r: создать случайную c: ввести с клавиатуры.

Команда d создает список символов, который нельзя печатать и выводит предложение, пропуская в нем эти символы.

Команда w считывает подстроку пользователя ищет ее с помощью двух алгоритмов и выводит скорость их работы.

Команда e заканчивает работу программы.

Работу каждой функции можно просмотреть отдельно (см. рис. 12, 13, 14)

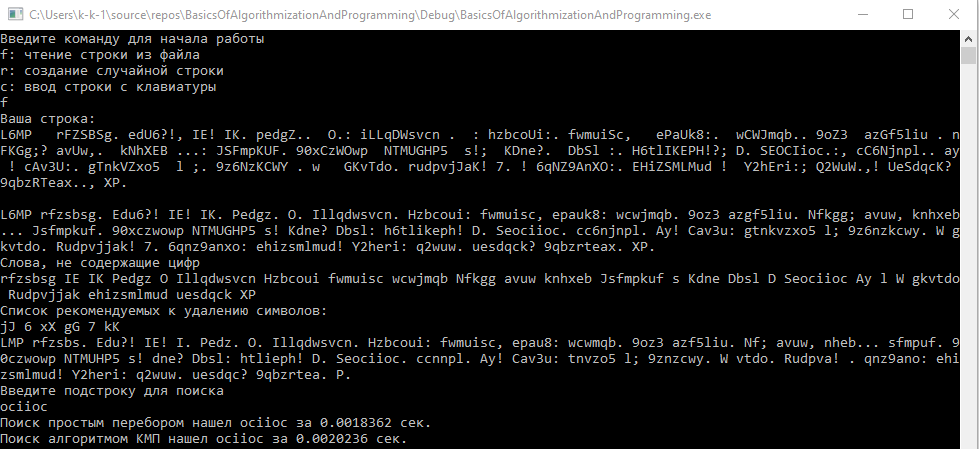


Рисунок 12 – Начало работы программы

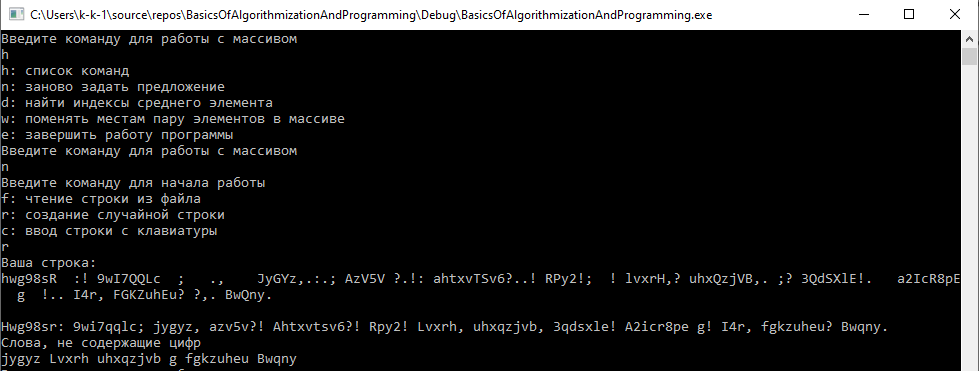


Рисунок 13 – Команды h и n

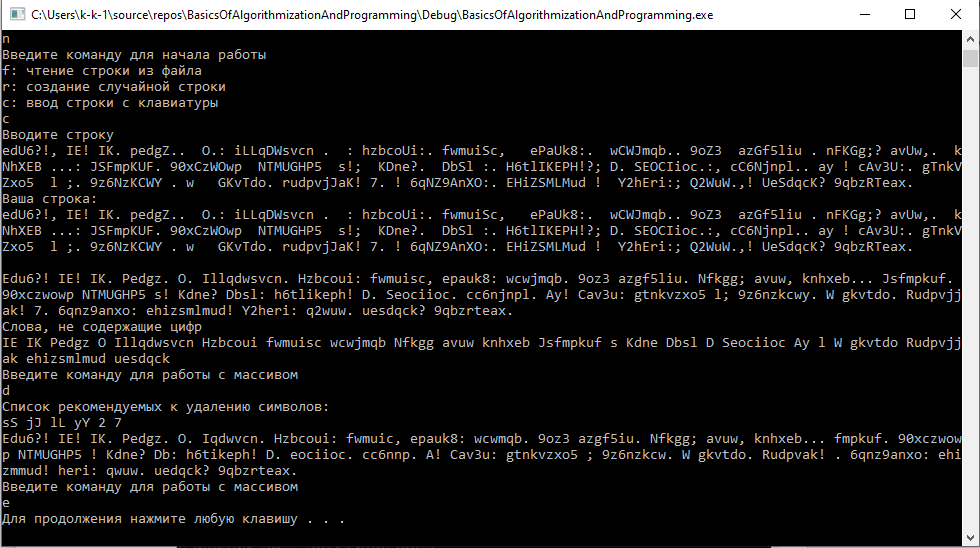


Рисунок 14 – Варианты команды n и завершение четвертой программы

**Обработка результатов эксперимента.**

Первая программа выводит корректные размеры выделяемой памяти для разных типов данных и представление чисел в памяти компьютера.

Вторая программа выводит результат корректной сортировки и правильно находит элементы в отсортированном массиве.

Третья программа корректно работает с двумерным массивом с помощью указателей, сортирует и заполняет.

Четвертая программа выводит результат корректной обработки строки и правильно находит подстроку в предложении.

Головная программа корректно осуществляет переход между программами дисциплины «программирование».

**Выводы.**

В ходе данной лабораторной работы я изучил внутреннее представление различных типов данных в памяти компьютера; различные способы сортировки массивов и работы с ними; указатели и их арифметику; различные способы работы со строчным типом данных в C++; а также методы работы с программами, состоящими из нескольких файлов.

ПРИЛОЖЕНИЕ А

Листинг файла CurseWork.cpp

#include <iostream>

#include "PracticalWork1.h"

#include "PracticalWork2.h"

#include "PracticalWork3.h"

#include "PracticalWork4.h"

int main()

{

setlocale(LC\_ALL, "Russian"); //для корректного вывода руских букв;

std::cout << "Запущена курсовая работа Кольцова Кирилла\n";

bool curseWorkWork = true;

char curseWorkAnswer;

while (curseWorkWork) {

std::cout << "Для перехода к нужной работе введите ее номер\n" \

"Для завершения введите q\n";

std::cin >> curseWorkAnswer;

switch (curseWorkAnswer)

{

case '1':

std::cin.ignore(INT\_MAX, '\n');

system("cls");

practicalWork1();

system("cls");

std::cout << "Запущена курсовая работа Кольцова Кирилла\n";

break;

case '2':

std::cin.ignore(INT\_MAX, '\n');

system("cls");

practicalWork2();

system("cls");

std::cout << "Запущена курсовая работа Кольцова Кирилла\n";

break;

case '3':

std::cin.ignore(INT\_MAX, '\n');

system("cls");

practicalWork3();

system("cls");

std::cout << "Запущена курсовая работа Кольцова Кирилла\n";

break;

case '4':

std::cin.ignore(INT\_MAX, '\n');

system("cls");

practicalWork4();

system("cls");

std::cout << "Запущена курсовая работа Кольцова Кирилла\n";

break;

case 'q':

curseWorkWork = false;

break;

default:

std::cout << "Вы указали несуществующую команду\n";

break;

}

}

system("pause");

return 0;

}

ПРИЛОЖЕНИЕ Б

Листинг файла PracticalWork1.h

#ifndef PRACTICALWORK1

#define PRACTICALWORK1

#include <iostream>

#include <typeinfo> //нужен для получения имени типа данных в форме строки

/\*

Первая функция нужна для вывода на консоль строки с размером соответствующего типа,

перегрузка используется для вывода размеров заданных в задании типов

и нужна для малого количества строк в main

\*/

template <typename T> void printMemorySize();

void printMemorySize();

//Функция принимает переменную целого, вещественного одинарной или двойной точности

void printMemoryShowing(double);

void printMemoryShowing(float);

void printMemoryShowing(int);

int practicalWork1();

#endif

ПРИЛОЖЕНИЕ В

Листинг файла PracticalWork2.h

#ifndef PRACTICALWORK2

#define PRACTICALWORK2

#include <iostream>

#include <ctime>

#include <chrono>

#define N2 100 //позволяет как бы переменно задавать размер массивов, так как компилятор не дает использовать переменные, кроме констант

extern bool wasArrSort; /\*нужна, чтобы знать, отсортирован ли массив arrSort, если пользователь перестановкой изменил его,

то перед большинством функций ему предложат пересортировать массив, иначе функция не выплнится\*/

void newArr(int\*, int\*); //создание массивов, повторный вызов пересоздаст массивы

void quicksort(int\*, int, int); //сортировка, использующая опорное число

int findElem(int\*, int, int, int); //выдает индекс элемента, если он есть в массиве, и -1, если такого элемента нет

int simpFindElem(int\*, int, int, int); //делает тоже самое, но простым перебором, а не бинарным поиском

void findAllMed(int\*, int, int); //находит все элементы, равные половине суммы максимального и минимального элементов

double findPlace(int\*, int, int, int); //находит позицию элемента, если элемента нет, выдает позицию между которых он бы был

void userFindLess(int\*); //пользователь вводит элемент, после чего выводится кол-во элементов меньше этого числа

void userFindMore(int\*); //пользователь вводит элемент, после чего выводится кол-во элементов больше этого числа

void userSwap(int\*); //пользователь вводит пару индексов, после чего в отсортированном массиве соответсвующие элементы меняются местами

void userFindNumber(int\*); //пользователь вводит число, после чего находится соответствующий элемент бинарным поиском и простым перебором

int genQuicksort(int\*, int, int, int); //функция для сортировки с выводом коментариев

void generator(); //функция для доп задания 10: генератор заданий

void variant(int\*); //функция для доп задания 5: работа с четными числами

int practicalWork2();

#endif

ПРИЛОЖЕНИЕ Г

Листинг файла PracticalWork3.h

#ifndef PRACTICALWORK3

#define PRACTICALWORK3

#include <iostream>

#include <iomanip>

#include <ctime>

#include <windows.h>

#define N 8 //размер матриц

void printMatrix(int [N][N]); //метод, для печати матриц

void makeMatrixA(int[N][N], int\*[N\*N]); //методы для создания матриц

void makeMatrixB(int[N][N], int\*[N\*N]);

void changeMatrixA(int[N][N]); //методы для перестановки элементов матрицы

void changeMatrixB(int[N][N]);

void changeMatrixC(int[N][N]);

void changeMatrixD(int[N][N]);

void quicksort(int\*\*, int, int); //сортировка матрицы из прошлой работы, работает через порядок создания

void userChangeMatrix(int[N][N]); //уведичение, уменьшение, умножение или деление матрицы на число пользователя

int maxValue; //число, для ровного вывода матрицы

int practicalWork3();

#endif

ПРИЛОЖЕНИЕ Д

Листинг файла PracticalWork4.h

#ifndef PRACTICALWORK4

#define PRACTICALWORK4

#include <iostream>

#include <string>

#include <ctime>

#include <vector>

#include <fstream>

#include <chrono>

std::string makeSentence(int); //принимает указатель на строку и меняет ее на случайную строку по условиям работы

void changeSentence(std::string\*); //удаляет лишние пробелы, приводит в порядок знаки препинания и регистры

void printNotNumberWord(std::string); //печатает слова из предложения, не содержащие цифры

void printDeathNote(std::string); //печатает предложение, исключая из слова набор случайных символов

void userFindWord(std::string); //поиск подстроки пользователя

int practicalWork4();

#endif