МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра информационных систем

ОТЧЕТ

по Курсовой работе

по дисциплине «Программирование»

Студентка гр. 0324 _____ Жигалова Д.А.

Преподаватель _____ Глущенко А.Г

Санкт-Петербург

2020

Цель работы

Формирование общей программы по всем четырем лабораторным работам с инфраструктурой переключения между заданиями (интерактивное меню).

Основные теоретические положения

Любая программа предназначена для обработки данных. Данные бывают различного типа, причем хранятся и обрабатываются они по-разному. Любые данные хранятся в памяти компьютера в виде двоичных кодов. Память компьютера представляет собой непрерывную последовательность двоичных ячеек, каждая из которых может находиться в двух состояний, которые можно условно обозначить 0 и 1. Каждая такая двоичная ячейка называется битом. Последовательность, условно разбитая по 8 бит, называется байтами. Из этого следует, что 1 байт = 8 бит. Байт в свою очередь является основной единицей измерения объема памяти. В любом алгоритмическом языке каждая константа, переменная, результата вычисления выражения или функции должны иметь определённый тип, который определяет:

- характер данных (знаковое или число без знака, целое или с дробной частью, последовательность символов или одиночный символ и т.д.);
- объем памяти, который занимают в памяти эти данные;
- диапазон или множество возможных значений;
- операции и функции, которые можно применять к величинам этого типа

В языке C++ определено шесть основных типов данных для представления целых, вещественных, символьных и логических величин. К ним относятся массивы, перечисления, функции, структуры, ссылки, указатели, объединения и классы.

Для описания основных типов определены следующие ключевые слова:

- int (целый);
- char (символьный);
- bool (логический);

- float (вещественный);
- double (вещественный тип с двойной точностью).

Существует четыре спецификатора типа, уточняющих внутреннее представление и диапазон значений стандартных типов:

- short (короткий);
- long (длинный);
- signed (знаковый);
- unsigned (беззнаковый)

Массивы.

При использовании простых переменных каждой области памяти для хранения данных соответствует свое имя. Если с группой величин одинакового типа требуется выполнить однообразные действия, им дают одно имя, а различают по порядковому номеру (индексу). Это дает возможность компактно записать множество операций с использованием циклов.

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: ai при i = 0, 1, 2, … n – одномерный вектор. Каждый элемент такой последовательности представляет собой некоторое значение определенного типа данных. Наглядно

одномерный массив можно представить как набор пронумерованных ячеек, в каждой из которых содержится определенное значение.

Объявление в программах одномерных массивов выполняется в соответствии со следующим правилом:

<Базовый тип элементов> <Идентификатор массива> [<Количество элементов>]

Значения индексов элементов массивов всегда начинается с 0. Поэтому максимальное значение индекса элемента в массиве всегда на единицу меньше количества элементов в массиве.

Обращение к определенному элементу массива осуществляется с помощью указания значения индекса этого элемента: A[8]. При обращении к конкретному элементу массива этот элемент можно рассматривать как обычную переменную, тип которой соответствует базовому типу элементов массива, и осуществлять со значением этого элемента любые операции, которые характерны для базового типа. Например, поскольку базовым типом массива A является тип данных int, с любым элементом этого массива можно выполнять любые операции, которые можно выполнять над значениями типа int.

Значения всех элементов массива в памяти располагаются в непрерывной области одно за другим. Общий объем памяти, выделяемый компилятором для массива, определяется как произведение объема одного элемента массива на количество элементов в массиве и равно:

sizeof(<Базовый тип>) * <Количество элементов>

Виды сортировок.

Сортировка – процесс размещения элементов заданного множества объектов в определенном порядке. Когда элементы отсортированы, их проще найти, производить с ними различные операции. Сортировка напрямую влияет на скорость алгоритма, в котором нужно обратиться к определенному элементу массива.

Простейшая из сортировок – сортировка обменом (пузырьковая сортировка). Вся суть метода заключается в попарном сравнении элементов и

последующем обмене. Таким образом, если следующий элемент меньше текущего, то они меняются местами, максимальный элемент массива постепенно смещается в конец массива, а минимальный – в начало. Один полный проход по массиву может гарантировать, что в конце массива находится максимальный элемент.

Затем процесс повторяется до тех пор, пока вся последовательность не будет упорядочена. Важно заметить, что после первого прохода по массиву, уже имеется один упорядоченный элемент, он стоит на своем месте, и менять его не надо. Таким образом на следующем шаге будут сравниваться N-1 элемент.

Очевидно, что хуже всего алгоритм будет работать, когда на вход подается массив, отсортированный в обратную сторону (от большего к меньшу). Быстрее же всего алгоритм работает с уже отсортированным массивом.

Но стандартный алгоритм пузырьковой сортировки предполагает полный циклический проход по массиву. Если изначально подается упорядоченная последовательность, то работа алгоритма все равно продолжиться. Исправить это можно, добавив условие проверки: если на текущей итерации ни один элемент не изменил свой индекс, то работа алгоритма прекращается.

Shaker sort – модификация пузырьковой сортировки. Принцип работы этой сортировки аналогичен bubble sort: попарное сравнение элементов и последующий обмен местами. Но имеется существенное отличие. Как только максимальный элемент становится на свое место, алгоритм не начинает новую итерацию с первого элемента, а запускает сортировку в обратную сторону. Алгоритм гарантирует, что после выполнения первой итерации, минимальный и максимальный элемент будут в начале и конце массива соответственно.

Затем процесс повторяется до тех пор, пока массив не будет отсортирован. За счет того, что сортировка работает в обе стороны, массив сортируется на порядок быстрее. Очевидным примером этого был бы случай, когда в начале массива стоит максимальный элемент, а в конце массива – минимальный. Shaker sort справится с этим за 1 итерацию, при условии, что другие элементы стоят на правильном месте.

Кажется, что bubble sort теряет свою эффективность по сравнению с shaker sort. Сортировка проходит в массиве в обоих направлениях, а не только от его начала к концу. Но в работе с большими массивами преимущество шейкер-сортировки уменьшается как раз из-за использования двух циклов.

Сортировка вставками (insert sort) – алгоритм сортировки, в котором элементы массива просматриваются по одному, и каждый новый элемент размещается в подходящее место среди ранее упорядоченных элементов.

Общая суть сортировки вставками такова:

1) Перебираются элементы в неотсортированной части массива.

2) Каждый элемент вставляется в отсортированную часть массива на то место, где он должен находиться.

Сортировка вставками делить массив на 2 части – отсортированную и неотсортированную. С каждым новым элементом отсортированная часть будет увеличиваться, а неотсортированная уменьшаться. Причем найти нужное место для очередного элемента в отсортированном массиве достаточно легко.

Рассмотрим самый простой способ (рис. 3.5). Необходимо пройти массив слева направо и обработать каждый элемент. Слева будет наращиваться отсортированная часть массива, а справа – уменьшаться неотсортированная. В отсортированной части массива ищется точка вставки для очередного элемента. Сам элемент отправляется в буфер, что освобождает место в массиве и позволяет сдвинуть элементы и освободить точку вставки.

Существует множество модификаций сортировки вставками, некоторые из них затрагивают именно способ вставки элемента в отсортированную часть. Одна из лучших модификаций – сортировка простыми вставками с бинарным поиском. Бинарный поиск будет описан позже.

Лучше всего сортировка вставками работает при обработке почти отсортированных массивов. В таком случае insert sort работает быстрее других сортировок.

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.

Указатель – переменная, значением которой является адрес ячейки памяти. То есть указатель ссылается на блок данных из области памяти, причём на самое его начало. Указатель может ссылаться на переменную или функцию. Для этого нужно знать адрес переменной или функции. Так вот, чтобы узнать адрес конкретной переменной в C++ существует унарная операция взятия адреса &. Такая операция извлекает адрес объявленных переменных, для того, чтобы его присвоить указателю.

Указатели используются для передачи по ссылке данных, что намного ускоряет процесс обработки этих данных (в том случае, если объём данных большой), так как их не надо копировать, как при передаче по значению, то есть, используя имя переменной. В основном указатели используются для организации динамического распределения памяти, например при объявлении массива, не надо будет его ограничивать в размере. Любой указатель необходимо объявить перед использованием, как и любую переменную. При использовании простых переменных каждой области памяти для хранения данных соответствует свое имя. Если с группой величин одинакового типа требуется выполнить однообразные действия, им дают одно имя, а различают по порядковому номеру (индексу). Это дает возможность компактно записать множество операций с использованием циклов.

Принцип объявления указателей такой же, как и принцип объявления переменных. Отличие заключается только в том, что перед именем ставится символ звёздочки *. Визуально указатели отличаются от переменных только одним символом. При объявлении указателей компилятор выделяет несколько байт памяти, в зависимости от типа данных отводимых для хранения некоторой информации в памяти. Чтобы получить значение, записанное в некоторой области, на которое ссылается указатель нужно воспользоваться операцией разыменования указателя *. Необходимо поставить звёздочку перед именем и получим доступ к значению указателя.

Формально указатели представляют собой обычные целые значения типа int и занимают в памяти 4 байта не зависимо от базового типа указателя. Значения указателей при их выводе на экран представляются как целые значения в шестнадцатеричном формате.

Указатели поддерживают ряд операций: присваивание, получение адреса указателя, получение значения по указателю, некоторые арифметические операции и операции сравнения.

К указателям можно применять некоторые арифметические операции. К таким операциям относятся: +, -, ++, --. Результаты выполнения этих операций по отношению к указателям существенно отличаются от результатов соответствующих арифметических операций, выполняющихся с обычными числовыми данными.

Указатели – это очень мощное, полезное, но и очень опасное средство. Ошибки, которые возникают при неправильном использовании указателей, кроме того, что могут приводить к серьезным и непредсказуемым ошибкам в работе программы, еще и очень трудно диагностировать (обнаруживать).

Класс string предназначен для работы со строками типа char, которые представляют собой строчку с завершающим нулем (символ ‘\0’). Класс string был введен как альтернативный вариант для работы со строками типа char.

Класс string обладает широким функционалом:

- функция compare() сравнивает одну часть строки с другой;

· функция length() определяет длину строки;

· функции find() и rfind() служат для поиска подстроки в строке (отличаются функции лишь направлением поиска);

· функция erase() служит для удаления символов;

· функция replace() выполняет замену символов;

· функция insert() необходима, чтобы вставить одну строку в заданную позицию другой строки;

Но весь функционал string накладывает и свой негативный отпечаток. Основным недостатком string в сравнении с типом char является замедленная скорость обработки данных.

При работе со строками часто будет возникать потребность в поиске набора символа или слов (поиска подстроки в строке). При условии, что текст может быть крайне большим, хочется, чтобы алгоритм поиска подстроки работал быстро.

Самый простой способ подстроки в строке – Линейный поиск – циклическое сравнение всех символов строки с подстрокой. Действительно, этот способ первый приходит в голову, но очевидно, что он будет самым долгим.

На первых двух итерациях цикла сравниваемые буквы не будут совпадать. На третьей же итерации, совпал символ ‘L’, это означает, что теперь нужно сравнивать следующий символ подстроки со следующим символом строки. Видно, что символы отличаются, поэтому алгоритм продолжает свою работу. На четвертой же итерации подстрока была найдена.

Если представить, что исходная строка непорядок больше и подстрока находится в конце строки (или вовсе отсутствует), то сразу видны минусы данного алгоритма.

Одним из самых популярных алгоритмов, который работает быстрее, чем приведенный выше алгоритм, является алгоритм Кнута-Морриса-Пратта (КМП). Идея заключается в том, что не нужно проходить и сравнивать абсолютно все символы строки, если известны символы, которые есть и в строке, и в подстроке.

Суть алгоритма: дана подстрока S и строка T. Требуется определить индекс, начиная с которого образец S содержится в строке T. Если S не содержится в T, необходимо вернуть индекс, который не может быть интерпретирован как позиция в строке.

Хоть алгоритм и работает быстрее, по-прежнему необходимо сначала пройти всю строку, чтобы определить префиксы или суффиксы (вхождение (индексы) символов).

Алгоритм Бойера-Мура в отличие от КМП полностью не зависим и не требует заранее проходить по строке. Этот алгоритм считается наиболее быстрым среди алгоритмов общего назначения, предназначенных для поиска подстроки в строке.

Преимущество этого алгоритма в том, что ценной некоторого количества предварительных вычислений над подстрокой (но не над исходной строкой, в которой ведётся поиск), подстрока сравнивается с исходным текстом не во всех позициях (пропускаются позиции, которые точно не дадут положительный результат).

Поиск подстроки ускоряется благодаря созданию таблиц сдвигов. Сравнение подстроки со строки начинается с последнего символа подстроки, а затем происходит прыжок, длина которого определяется по таблице сдвигов. Таблица сдвигов строится по подстроке так чтобы перепрыгнуть максимальное количество символов строки и не пропустить вхождение подстроки в строку.

Постановка задачи

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

Выполнение работы

Для решения поставленных была создана программа на языке программирования C++. Итоговый код программы представлен в приложении А, а результат работы в приложении В.

Переключение между четырьмя практическими работами происходит при помощи инструкции множественного выбора.

Вывод

В ходе проделанной работы были изучены следующие темы:

- бинарные операции;
- представление разных типов данных в памяти;
- одномерные статические массивы;
- быстрая сортировка;
- метод бинарного поиска;
- двумерные статические массивы;
- указатели и их арифметика;
- способы обработки текстовых данных;
- алгоритм поиска подстроки в строке.

ПРИЛОЖЕНИЕ А
ПОЛНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <ctime>
#include <chrono>
#include <thread>
#include <string>
#include <fstream>
using namespace std;

void PW1()
{
    int answer;
    answer = 1;
    while (answer != 0) {
        cout << "\nWhat tasks do you wanna check? (Write task's number)\n";
        cout << "1. Print how much memory(in bytes) on your computer is allocated for
various data types withand without specifiers;\n";
        cout << "2. Display the binary representation in memory (all digits) of an
integer;\n";
        cout << "3. Display a binary representation in memory (all bits) of the float
type;\n";
        cout << "4. Display a binary representation in memory (all digits) of the
double type.\n";
        cout << "To exit, enter 0 \n";
        cin >> answer;
        system("CLS");

        switch (answer)
        {
        case (1):
            int a;
            short int b;
            long int c;
            float d;
            double e;
            long double f;
            char g;
            bool h;
            cout << "int: " << sizeof(a) << "; short int: " << sizeof(b) << "; long
int: " << sizeof(c) << "; float: " << sizeof(d) << "; double: " << sizeof(e) << "; long
double: " << sizeof(f) << "; char: " << sizeof(g) << "; bool: " << sizeof(h) << "\n";
            break;
        case (2):
            int toBin;
            int mask;
            mask = 1;
            int num[32];
            bool answer2;

            while (true) {
                cout << "\nEnter integer: \n";
                cin >> toBin;

                for (int i = 0; i < 32; i++) {
                    num[i] = toBin & mask ? 1 : 0;
```

```
            toBin = toBin >> 1;
        }

        for (int i = 31; i >= 0; i--) {
            if (i == 30)
                cout << " ";
            cout << num[i];
        }
        cout << "\nExit? 1/0";
        cin >> answer2;
        if (answer2) {
            system("CLS");
            break;
        }
    }
    break;
case (3):
    int mask3;
    mask3 = 1;
    int num3[32];
    bool answer3;
    union {
        int tool;
        float toBin2;
    };
    while (true) {
        cout << "\nEnter float: \n";
        cin >> toBin2;
        for (int i = 0; i < 32; i++) {
            num3[i] = tool & mask3 ? 1 : 0;
            tool = tool >> 1;
        }
        for (int i = 31; i >= 0; i--) {

            cout << num3[i];
            if (i == 31 || i == 23) {
                cout << " ";
            }
        }
        cout << "\nExit? 1/0";
        cin >> answer3;
        if (answer3) {
            system("CLS");
            break;
        }
    }
    break;
case (4):
    bool num4[64];
    bool answer4;
    int index;
    union {
        double toBin3;
        int tool2[2];
    };
    while (true) {
        cout << "\nEnter double: \n";
```

```
                cin >> toBin3;
                index = 0;

                for (int k = 0; k < 2; k++) {
                    for (int i = 0; i < 32; i++) {
                        num4[index] = 1 & tool2[k];
                        tool2[k] >>= 1;
                        index++;
                    }
                }
                for (int i = 63; i >= 0; i--) {

                    cout << num4[i];
                    if (i == 63 || i == 52) {
                        cout << " ";
                    }
                }
                cout << "\nExit? 1/0";
                cin >> answer4;
                if (answer4) {
                    system("CLS");
                    break;
                }
            }
            break;
        case (0):
            cout << "Have a nice day!\n";
            break;
        default:
            cout << "Oh, no! This task doesn't exist. Let's try again.\n";
            break;
        };
    }
}

void quicksort(int* arr, int  end, int begin) {
    int mid;
    int f = begin;
    int l = end;
    mid = arr[(f + l) / 2];
    while (f < l)
    {
        while (arr[f] < mid) f++;
        while (arr[l] > mid) l--;
        if (f <= l)
        {
            swap(arr[f], arr[l]);
            f++;
            l--;
        }
    }
    if (begin < l) quicksort(arr, l, begin);
    if (f < end) quicksort(arr, end, f);
}

int binarySearch(int array[], int x, int low, int high) {
```

```
    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (array[mid] == x)
            return mid;

        if (array[mid] < x)
            low = mid + 1;

        else
            high = mid - 1;
    }
    return -1;
}

void PW2()
{
    const int N = 100;
    int unsortArr[N];
    int arr[N];
    cout << "\nLet's make some array and sort it to realize other tasks\n";

    int answer;
    answer = 2;

    while (answer != 0) {
        cout << "\nTo make new array, enter 1\n";
        cout << "To exit, enter 0 \n";
        cin >> answer;
        system("CLS");

        switch (answer)
        {
        case (1):
        {
            // создаем рандомный массив из 100 чисел от -99 до 99
            for (int i = 0; i < N; i++) {
                unsortArr[i] = -99 + (rand() % 200);
            }

            for (int i = 0; i < N; ++i)
                arr[i] = unsortArr[i];

            // сортируем массив с помощью quick sort
            auto startQuick = std::chrono::high_resolution_clock::now();
            int end, begin;
            end = N - 1;
            begin = 0;
            quicksort(arr, end, begin);
            auto endQuick = std::chrono::high_resolution_clock::now();
            auto timeQuick =
std::chrono::duration_cast<std::chrono::nanoseconds>(endQuick - startQuick);

            cout << "\nTime spent on sorting (nanoseconds): " << std::fixed <<
timeQuick.count() << endl;

            int answer2;
```

```
            answer2 = 9;

            while (answer2 != 0) {
                cout << "\nUnsorted array: \n";
                for (int i = 0; i < N; i++)
                    cout << unsortArr[i] << " ";

                cout << "\n\nSorted array: \n";
                for (int i = 0; i < N; i++)
                    cout << arr[i] << " ";

                cout << "\n\nWhat tasks do you wanna check? (Write task's number)\n";
                cout << "3. Find the maximum and minimum element of the array.
Calculate the search time in sorted and unsorted arrays using the chrono library;\n";
                cout << "4. Output the average value of the maximum and minimum values.
Print the indexes of all elements that are equal to this value, and their number.\n";
                cout << "5. Print the number of elements in the sorted array that are
less than the number that is initialized by the user;\n";
                cout << "6. Print the number of elements in the sorted array that are
greater than the number that is initialized by the user;\n";
                cout << "7. Output information about whether the number entered by the
user is in the sorted array;\n";
                cout << "8. Swaps array elements whose indexes are entered by the
user.\n";
                cout << "To make new array or exit, enter 0 \n";
                cin >> answer2;
                system("CLS");

                switch (answer2)
                {
                case (3):
                {
                    int minInUnsorted, maxInUnsorted;
                    minInUnsorted = 0;
                    maxInUnsorted = 0;

                    auto startFindingUn = std::chrono::high_resolution_clock::now();
                    for (int i = 0; i < N; i++)
                    {
                        if (unsortArr[i] <= minInUnsorted)
                        {
                            minInUnsorted = unsortArr[i];
                        }
                        if (unsortArr[i] >= maxInUnsorted)
                        {
                            maxInUnsorted = unsortArr[i];
                        }
                    }
                    cout << "\nMinimum element of an unsorted array: " <<
minInUnsorted;
                    cout << "\nMaximum element of an unsorted array: " <<
maxInUnsorted;
                    auto endFindingUn = std::chrono::high_resolution_clock::now();
                    auto timeFindingUn =
std::chrono::duration_cast<std::chrono::nanoseconds>(endFindingUn - startFindingUn);
                    cout << "\nTime spent on unsorted finding (nanoseconds): " <<
std::fixed << timeFindingUn.count() << endl;
```

```
                    auto startFindingSo = std::chrono::high_resolution_clock::now();
                    cout << "\nMinimum element of the sorted array: " << arr[0];
                    cout << "\nMaximum element of the sorted array: " << arr[99];
                    auto endFindingSo = std::chrono::high_resolution_clock::now();
                    auto timeFindingSo =
std::chrono::duration_cast<std::chrono::nanoseconds>(endFindingSo - startFindingSo);
                    cout << "\nTime spent on sorted finding (nanoseconds): " <<
std::fixed << timeFindingSo.count() << endl;

                    break;
                }

                case (4):
                    int average;

                    average = round((arr[0] + arr[99]) / 2);
                    cout << "\nThe average value of the maximum and minimum values: "
<< average;
                    cout << "\nIndexes of all elements that are equal to this value: ";
                    int numberInArray;
                    numberInArray = 0;
                    for (int i = 0; i < 100; i++) {
                        if (unsortArr[i] == average) {
                            numberInArray += 1;
                            cout << "\n" << i << "\n";
                        }
                    }
                    cout << "Total of these in the array: " << numberInArray << "\n";

                    break;

                case (5):
                    cout << "To search for elements in the sorted array that are less
than the number a, enter a: ";
                    int a;
                    int lowerThanA;
                    lowerThanA = 0;
                    cin >> a;
                    for (int i = 0; i < N; i++) {
                        if (arr[i] < a) {
                            lowerThanA += 1;
                        }
                    }
                    cout << "\nElements less than a: " << lowerThanA << endl;
                    break;

                case (6):
                    cout << "To search for elements in the sorted array that are
greater than the number b, enter b: ";
                    int b;
                    int higherThanB;
                    higherThanB = 0;
                    cin >> b;
                    for (int i = 0; i < N; i++) {
                        if (arr[i] > b) {
                            higherThanB += 1;
```

```
                        }
                    }
                    cout << "\nElements greater than  b: " << higherThanB << endl;
                    break;

                case (7):
                {
                    int x;
                    bool answer7;
                    bool isExists = false;

                    cout << "\nUnsorted array: \n";
                    for (int i = 0; i < N; i++)
                        cout << unsortArr[i] << " ";

                    cout << "\n\nSorted array: \n";
                    for (int i = 0; i < N; i++)
                        cout << arr[i] << " ";

                    while (true) {
                        cout << "\nWhat integer do you want to find? \n";
                        cin >> x;

                        auto startSort = std::chrono::high_resolution_clock::now();
                        for (int i = 0; i < N; i++) {
                            if (arr[i] == x) {
                                isExists = true;
                                break;
                            }
                        }
                        auto endSort = std::chrono::high_resolution_clock::now();
                        auto timeSort =
std::chrono::duration_cast<std::chrono::nanoseconds>(endSort - startSort);
                        cout << "\nTime spent on regular sorted finding (nanoseconds):
" << std::fixed << timeSort.count() << endl;

                        auto startBinary = std::chrono::high_resolution_clock::now();
                        int result = binarySearch(arr, x, 0, N - 1);
                        auto endBinary = std::chrono::high_resolution_clock::now();
                        auto timeBinary =
std::chrono::duration_cast<std::chrono::nanoseconds>(endBinary - startBinary);
                        cout << "\nTime spent on binary sorted finding (nanoseconds): "
<< std::fixed << timeBinary.count() << endl;

                        if (result == -1)
                            cout << "Not found" << "\n";
                        else
                            cout << "Element is found at index " << result << "\n";

                        cout << fixed << "\nDifference between Binary search and
regular search (nanoseconds): " << timeSort.count() - timeBinary.count() << "\n";

                        cout << "\nExit? 1/0";
                        cin >> answer7;
                        if (answer7) {
                            system("CLS");
                            break;
```

```
                        }
                    }
                    break;
                }

                case (8):
                {
                    bool answer8;
                    cout << "\nUnsorted array: \n";
                    for (int i = 0; i < N; i++)
                        cout << unsortArr[i] << " ";

                    while (true) {
                        cout << "\nTo swap two array elements, enter their indexes:
\n";
                        int first;
                        int second;
                        cout << "First: ";
                        cin >> first;
                        cout << "Second: ";
                        cin >> second;
                        cout << "\n";

                        auto startSwap = std::chrono::high_resolution_clock::now();
                        swap(unsortArr[first], unsortArr[second]);
                        auto endSwap = std::chrono::high_resolution_clock::now();
                        auto timeSwap =
std::chrono::duration_cast<std::chrono::nanoseconds>(endSwap - startSwap);
                        cout << "\nTime spent on swaping(nanoseconds): " << std::fixed
<< timeSwap.count() << endl;

                        cout << "\nUnsorted array: \n";
                        for (int i = 0; i < N; i++)
                            cout << unsortArr[i] << " ";

                        cout << "\nExit? 1/0";
                        cin >> answer8;
                        if (answer8) {
                            system("CLS");
                            break;
                        }
                    }
                    break;
                }

                case (0):
                    break;

                default:
                    cout << "Oh, no! This task doesn't exist. Let's try again.\n";
                    break;
                }
            }
            break;
        }
        case (0):
            cout << "Have a nice day!\n";
```

```
            break;

        default:
            cout << "Oh, no! This task doesn't exist. Let's try again.\n";
            break;
        }
    }
}

void animation(int* p, int s) {
    system("CLS");
    for (int i = 0; i < s; ++i) {
        for (int j = 0; j < s; ++j) {
            cout << *(p + s * i + j) << " ";
        }
        cout << endl;
    }
    this_thread::sleep_for(chrono::milliseconds(40));
}

void withoutAnimation(int* p, int s) {
    for (int i = 0; i < s; ++i) {
        for (int j = 0; j < s; ++j) {
            cout << *(p + s * i + j) << " ";
        }
        cout << endl;
    }
    cout << endl;
}

void fillSnake(int* p, int s) {

    int counter = 1;

    for (int i = 0; i < s; ++i) {
        for (int j = 0; j < s; ++j) {
            *(p + s * j + i) = 0;
        }
    }
    for (int i = 0; i < s; i++) {
        if ((i % 2) != 0) {
            for (int j = (s - 1); j >= 0; j--) {
                *(p + s * j + i) = rand() % (s * s) + 1;
                //*(p + s * j + i) = counter; для проверки корректной работы
                animation(p, s);
                counter++;
            }
        }
        else {
            for (int j = 0; j < s; j++) {
                *(p + s * j + i) = rand() % (s * s) + 1;
                //*(p + s * j + i) = counter; для проверки корректной работы
                animation(p, s);
                counter++;
            }
        }
    }
```

```
}

void fillSpiral(int* p, int s) {

    for (int i = 0; i < s; ++i) {
        for (int j = 0; j < s; ++j) {
            *(p + s * j + i) = 0;
        }
    }
    int c = 1, j, k = 0, d = 1;

    while (c < (s * s + 1))
    {
        k++;
        for (j = k - 1; j < s - k + 1; j++)
        {
            *(p + (k - 1) * s + j) = rand() % (s * s) + 1;
            //*(p + (k - 1) * s + j) = c;
            animation(p, s);
            c++;
        }   //верх

        for (j = k; j < s - k + 1; j++)
        {
            *(p + j * s + (s - k)) = rand() % (s * s) + 1;
            //*(p + j * s + (s-k)) = c;
            animation(p, s);
            c++;
        }   //право

        for (j = s - k - 1; j >= k - 1; j--)
        {
            *(p + (s - k) * s + j) = rand() % (s * s) + 1;
            //*(p + (s - k) * s + j) = c;
            animation(p, s);
            c++;
        }   //низ

        for (j = s - k - 1; j >= k; j--)
        {
            *(p + j * s + (k - 1)) = rand() % (s * s) + 1;
            //*(p + j * s + (k-1)) = c;
            animation(p, s);
            c++;
        }   //лево
    }
}

void maths(int* p, int s) {

    int answer3 = 0;
    int n;

    while (answer3 != 5) {

        cout << "\n What exectly do you want?\n";
        cout << "\nTo reduce, enter 1\n";
```

```
cout << "\nTo increase, enter 2\n";
cout << "\nTo multiplie, enter 3\n";
cout << "\nTo divide, enter 4\n";
cout << "\nTo exit, enter 5 \n";
cin >> answer3;

switch (answer3) {

case 1:
    cout << "And enter a number:";
    cin >> n;

    cout << "\nBefore:\n";
    withoutAnimation(p, s);
    for (int i = 0; i < s; ++i) {
        for (int j = 0; j < s; ++j) {
            *(p + s * j + i) = *(p + s * j + i) - n;
        }
    }
    cout << "\nAfter:\n";
    withoutAnimation(p, s);
    break;

case 2:
    cout << "And enter a number:";
    cin >> n;

    cout << "\nBefore:\n";
    withoutAnimation(p, s);
    for (int i = 0; i < s; ++i) {
        for (int j = 0; j < s; ++j) {
            *(p + s * j + i) = *(p + s * j + i) + n;
        }
    }
    cout << "\nAfter:\n";
    withoutAnimation(p, s);
    break;

case 3:
    cout << "And enter a number:";
    cin >> n;

    cout << "\nBefore:\n";
    withoutAnimation(p, s);
    for (int i = 0; i < s; ++i) {
        for (int j = 0; j < s; ++j) {
            *(p + s * j + i) = *(p + s * j + i) * n;
        }
    }
    cout << "\nAfter:\n";
    withoutAnimation(p, s);
    break;

case 4:
    cout << "And enter a number:";
    cin >> n;
```

```
            cout << "\nBefore:\n";
            withoutAnimation(p, s);
            for (int i = 0; i < s; ++i) {
                for (int j = 0; j < s; ++j) {
                    *(p + s * j + i) = *(p + s * j + i) / n;
                }
            }
            cout << "\nAfter:\n";
            withoutAnimation(p, s);
            break;

        case 5:
            break;

        default:
            cout << "Oh, no! This task doesn't exist. Let's try again.\n";
            break;
        }
    }
    system("CLS");
}

void bubbleSort(int* p, int s) {

    bool sw = 1;

    while (sw != 0) {
        sw = 0;
        for (int i = 0; i < (s * s - 1); i++) {
            if ((*(p + i)) > (*(p + i + 1))) {
                swap((*(p + i)), (*(p + i + 1)));
                sw = 1;
            }
        }
    }
}

void blocks(int* p, int s) {

    int answer1 = 0;
    int half = s / 2;
    int t;

    while (answer1 != 5) {

        cout << "\n What exectly do you want?\n";
        cout << "\na)To rearrange with a snake, enter 1\n";
        cout << "\nb)To rearrange the diagonal, enter 2\n";
        cout << "\nc)To rearrange the vertical, enter 3\n";
        cout << "\nd)To rearrange the horizontal, enter 4\n";
        cout << "\nTo exit, enter 5 \n";
        cin >> answer1;

        switch (answer1) {

        case 1:
            cout << "\nBefore:\n";
```

```
    withoutAnimation(p, s);
    for (int i = 0; i < half; ++i) {
        for (int j = 0; j < half; ++j) {
            t = *(p + s * j + i);
            *(p + s * j + i) = *(p + s * (j + half) + i);
            *(p + s * (j + half) + i) = *(p + s * (j + half) + (i + half));
            *(p + s * (j + half) + (i + half)) = *(p + s * j + (i + half));
            *(p + s * j + (i + half)) = t;
        }
    }
    cout << "\nAfter:\n";
    withoutAnimation(p, s);
    break;

case 2:
    cout << "\nBefore:\n";
    withoutAnimation(p, s);
    for (int i = 0; i < half; ++i) {
        for (int j = 0; j < half; ++j) {
            t = *(p + s * j + i);
            *(p + s * j + i) = *(p + s * (j + half) + (i + half));
            *(p + s * (j + half) + (i + half)) = t;
        }
    }
    for (int i = 0; i < half; ++i) {
        for (int j = 0; j < half; ++j) {
            t = *(p + s * (j + half) + i);
            *(p + s * (j + half) + i) = *(p + s * j + (i + half));
            *(p + s * j + (i + half)) = t;
        }
    }
    cout << "\nAfter:\n";
    withoutAnimation(p, s);
    break;

case 3:
    cout << "\nBefore:\n";
    withoutAnimation(p, s);
    for (int i = 0; i < s; ++i) {
        for (int j = 0; j < half; ++j) {
            t = *(p + s * j + i);
            *(p + s * j + i) = *(p + s * (j + half) + i);
            *(p + s * (j + half) + i) = t;
        }
    }
    cout << "\nAfter:\n";
    withoutAnimation(p, s);
    break;

case 4:
    cout << "\nBefore:\n";
    withoutAnimation(p, s);
    for (int i = 0; i < half; ++i) {
        for (int j = 0; j < s; ++j) {
            t = *(p + s * j + i);
            *(p + s * j + i) = *(p + s * j + (i + half));
            *(p + s * j + (i + half)) = t;
```

```
                }
            }
            cout << "\nAfter:\n";
            withoutAnimation(p, s);
            break;

        case 5:
            break;

        default:
            cout << "Oh, no! This task doesn't exist. Let's try again.\n";
            break;
        }
    }
    system("CLS");
}

void PW3()
{
    srand(time(0));

    const int size = 6;
    int A[size][size];
    int* p = &A[0][0];

    int end = size * size - 1;
    int begin = 0;

    cout << "\nLet's make some matrix (6*6)\n";
    int answer;
    answer = 3;

    while (answer != 0) {
        cout << "\nTo make new matrix with Snake animation, enter 1\n";
        cout << "\nTo make new matrix with Spiral animation, enter 2\n";
        cout << "\nTo exit, enter 0 \n";
        cin >> answer;
        system("CLS");

        if (answer == 1 || answer == 2) {
            if (answer == 1)
                fillSnake(p, size);

            if (answer == 2)
                fillSpiral(p, size);

            while (answer != 4) {
                cout << "\nTo make new matrix, rearranging its blocks in accordance
with the schemes(a, b, c, d), enter 1\n";
                cout << "\nTo sort elements by bubble sort, using pointer arithmetic,
enter 2\n";
                cout << "\nTo reduce, increase, multiplie, or divide all matrix
elements by the number entered by the user, enter 3\n";
                cout << "\nTo make new matrix, enter 4 \n";
                cin >> answer;
                system("CLS");
```

```
                if (answer == 1 || answer == 2 || answer == 3) {
                    if (answer == 1)
                        blocks(p, size);

                    if (answer == 2) {
                        cout << "\nUnsorted array:\n";
                        withoutAnimation(p, size);
                        bubbleSort(p, size);
                        cout << "\nSorted array:\n";
                        withoutAnimation(p, size);
                    }

                    if (answer == 3)
                        maths(p, size);
                }

                else if (answer == 4) {
                    cout << "Let's make some matrix (6*6)\n";
                }

                else {
                    cout << "Oh, no! This task doesn't exist. Let's try again.\n";
                }
            }
        }

        else if (answer == 0) {
            cout << "Have a nice day!\n";
        }

        else {
            cout << "Oh, no! This task doesn't exist. Let's try again.\n";
        }
    }
}

string note() {
    int answer2;
    string text0, text;
    cout << "\nSelect the input method\n";
    cout << "\n1) Keyboard\n";
    cout << "\n2) File (D:\\hello.txt)\n";
    cin >> answer2;

    if (answer2 == 1) {
        cin.ignore();
        getline(cin, text0);
        text = text0;
    }
    else if (answer2 == 2) {
        ifstream in("D:\\hello.txt"); // окрываем файл для чтения
        if (in.is_open())
        {
            while (getline(in, text))
            {
                cout << "You entered: " << endl;
                cout << text << endl;
```

```
            }
        }
        else cout << "File doesn't exist\n";
        in.close();
    }
    return text;
}

string cleaning(string text, int len) {

    while (text[0] == '.' || text[0] == ' ') {
        text.erase(0, 1);
        len--;
    }
    while (text[len] == ' ') {
        text.erase(len, 1);
        len--;
    }
    for (int i = 0; i < len; i++) {
        if (text[i] == ' ' && (text[i + 1] == ' ' || text[i + 1] == ',' || text[i + 1]
== '.' || text[0] == ' ')) {
            text.erase(i, 1);
            len--;
        }
    }
    for (int i = 0; i < len; i++) {
        if ((text[i] == '.' && text[i + 1] == '.') && text[i + 2] == '.')
            i += 3;
        else if ((text[i] == '.' || text[i] == ',' || text[i] == ';' || text[i] == ':'
|| text[i] == '!' || text[i] == '?') && (text[i + 1] == '.' || text[i + 1] == ',' ||
text[i + 1] == ';' || text[i + 1] == ':' || text[i + 1] == '!' || text[i + 1] == '?'))
{
            text.erase(i, 1);
            len--;
            i--;
        }
    }
    for (int i = 0; i < len; i++) {
        if (i == 0)
            text[i] = toupper(text[i]);
        else
            text[i] = tolower(text[i]);
    }
    for (int i = 0; i < len; i++) {
        if (text[i] == '.' && text[i + 1] == ' ')
            text[i + 2] = toupper(text[i + 2]);
    }
    return text;
}

void exerciseThree(string text) {
    int End = 0;
    string str, word;
    char letters[26] =
{ 'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','
v','w','x','y','z' };
    char numbers[10] = { '0','1','2','3','4','5','6','7','8','9' };
```

```
    int len = text.length();
    for (int i = 0; i < len; i++) {
        if ((text[i] == ';' || text[i] == ',' || text[i] == '.' || text[i] == ':' ||
text[i] == '!' || text[i] == '?')) {
            text.erase(i, 1);
            len--;
            i--;
        }
    }
    str = text;
    cout << "\nAll words containing only letters - ";

    while (End >= 0) {
        bool foundNum = false;
        End = str.find(' ');
        word = str.substr(0, End);
        str = str.erase(0, End + 1);
        for (int i = 0; i < word.length(); i++) {
            for (auto j : numbers)
                if (word[i] == j) {
                    foundNum = true;
                    break;
                }
        }
        if (!foundNum) {
            cout << word << " ";
        }
    }
    End = 0;
    str = text;
    cout << "\n\nAll words containing only numbers - ";
    while (End >= 0) {
        bool foundNum = false;
        End = str.find(' ');
        word = str.substr(0, End);
        str = str.erase(0, End + 1);
        for (int i = 0; i < word.length(); i++) {
            for (auto j : letters) {
                if (word[i] == j) {
                    foundNum = true;
                    break;
                }
            }
        }
        if (!foundNum) cout << word << " ";
    }
    End = 0;
    str = text;
    cout << "\n\nAll words containing only numbers and letters - ";
    while (End >= 0) {
        bool foundNum = false;
        End = str.find(' ');
        word = str.substr(0, End);
        str = str.erase(0, End + 1);
        for (int i = 0; i < word.length(); i++) {
            for (auto j : letters) {
```

```
                if (word[i] == j) {
                    for (int m = i; m < word.length(); m++) {
                        for (auto k : numbers) {
                            if (word[m] == k) {
                                foundNum = true;
                                break;
                            }
                        }
                    }
                }
            }
        }
        for (int i = 0; i < word.length(); i++) {
            for (auto j : numbers) {
                if (word[i] == j) {
                    for (int m = i; m < word.length(); m++) {
                        for (auto k : letters) {
                            if (word[m] == k) {
                                foundNum = true;
                                break;
                            }
                        }
                    }
                }
            }
        }
        if (foundNum) cout << word << " ";
    }
}

void exerciseFour(string text) {
    int len = text.length();
    for (int i = 0; i < len; i++) {
        if ((text[i] == ';' || text[i] == ',' || text[i] == '.' || text[i] == ':' ||
text[i] == '!' || text[i] == '?')) {
            text.erase(i, 1);
            len--;
            i--;
        }
    }
    string word, text2, line;
    text2 = "";
    text += " ";
    int index = 0;
    //cout << text << endl;
    int End = 0;

    while (End >= 0) {
        End = text.find(' ');
        word = text.substr(0, End);
        text = text.erase(0, End + 1);
        if (word.length() < 10) {
            int k = 10 - word.length();
            for (int j = 0; j < k; j++) {
                word += " ";
            }
        }
```

```
        text2 += word;
    }
    text2 = text2.erase(text2.length() - 10, text2.length() - 1);

    for (int j = 0; j < 10; j++) {
        for (int i = 0; i < text2.length(); i += 10) {
            line += text2[i + j];
            line += " ";
        }
        cout << line << endl;
        line = "";
    }
}

void exerciseFive(string text) {
    string subText0, subText;
    bool foundOne = false;
    bool found = true;
    int answer = 1;
    while (answer == 1) {

        cout << "\nEnter substring:" << endl;
        cin.ignore();
        getline(cin, subText0);

        subText = subText0;

        for (int i = 0; i < text.length(); i++) {
            if (text[i] == subText[0]) {
                for (int j = 1; j < subText.length(); j++) {
                    if (text[i + j] != subText[j])
                        found = false;
                }
                if (found == true) {
                    cout << "First index of substring - " << i << "\n";
                    foundOne = true;
                }
            }
            found = true;
        }
        if (!foundOne)
            cout << "This substring doesn't exist";
        cout << "\n\nDo you wanna repeat? (yes - 1; no - 0) (please repeat entering if
it isn't true)" << endl;
        cin >> answer;
    }
}

void PW4()
{
    int answer1, len;
    answer1 = 1;
    len = 0;
    string text;

    text = note(); // используется функция записи введенного текста (1 задание)
```

```
    while (text[len])
        ++len;
    cout << "\nNumber of symbols in the text: " << len << endl;

    text = cleaning(text, len); // 2 задание
    cout << "\nEdited input text:" << endl;
    cout << text << endl;

    exerciseThree(text);

    cout << "\n\nDisplays all words in the original sequence vertically on the
screen:\n" << endl;
    exerciseFour(text);

    exerciseFive(text);
}

int main()
{
    int practicalWork = 1;
      while (practicalWork != 0) {
            system("CLS");
        cout << "To choose Practical work, Enter the number of practical work or enter
0 for exit: \n";
        cout << "\t1) Data types and their internal representation in memory \n";
        cout << "\t2) One-dimensional static arrays \n";
        cout << "\t3) Pointers \n";
        cout << "\t4) String \n\n";
            cin >> practicalWork;
            switch (practicalWork) {
            case 1:
                  system("CLS");
                  PW1();
                  break;
            case 2:
                  system("CLS");
                  PW2();
                  break;
            case 3:
                  system("CLS");
                  PW3();
                  break;
            case 4:
                  system("CLS");
                  PW4();
                  break;
            case 0:
                  cout << "Have a nice day!\n";
                  break;
            default:
                  cout << "Oh, no! This task doesn't exist. Let's try again.\n";
                  break;
            }
      }
      return 0;
}
```

ПРИЛОЖЕНИЕ В
РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

Интерактивное меню выбора практической работы:

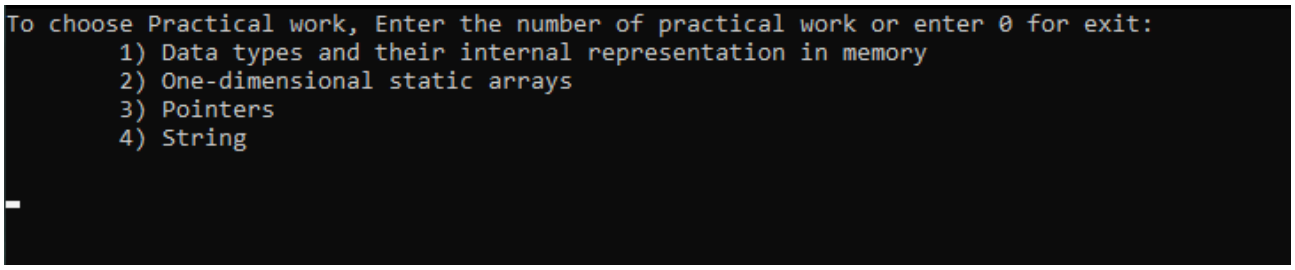

Рисунок 1 - интерактивное меню

Первая практическая работа:

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

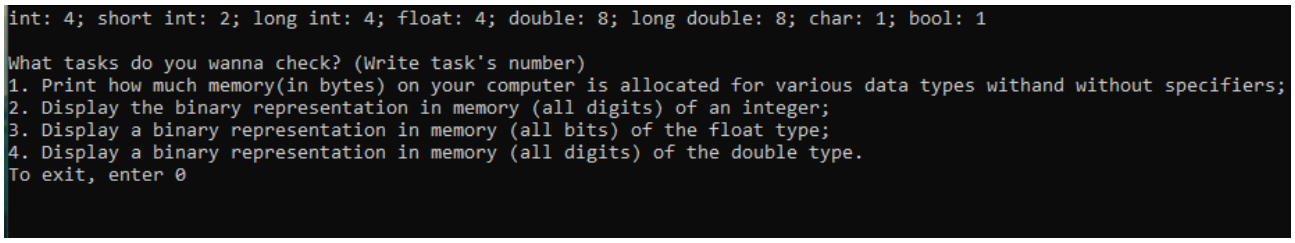

Рисунок 2 - Задание 1

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.

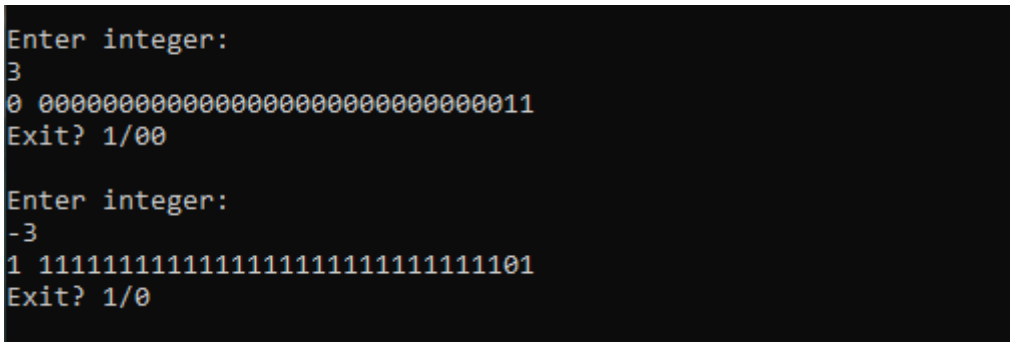

Рисунок 3 - Задание 2

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.

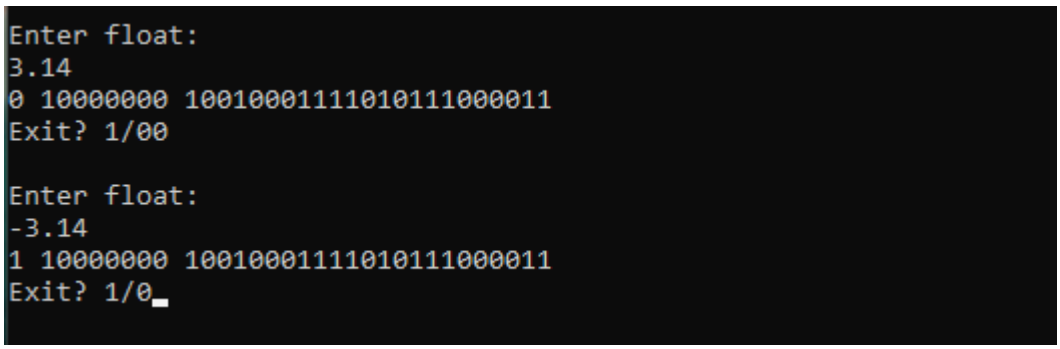

Рисунок 4 - Задание 3

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.

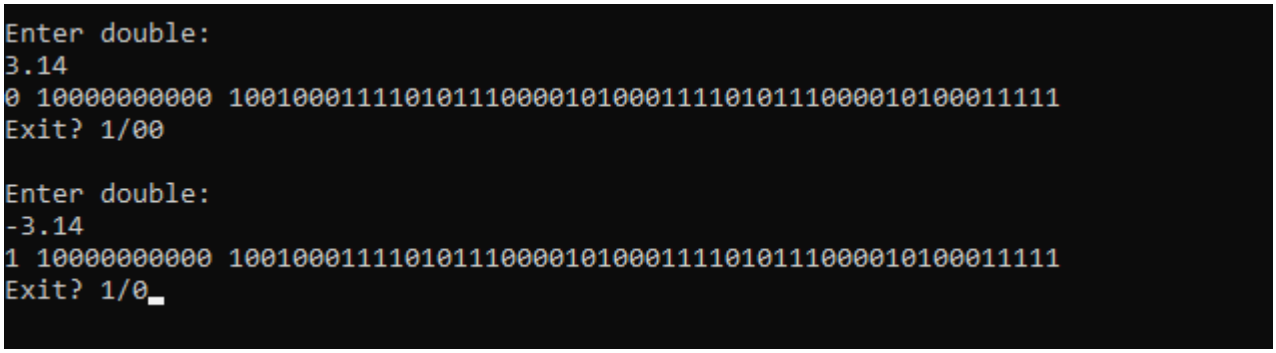

Рисунок 5 - Задание 4

Вторая практическая работа:

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

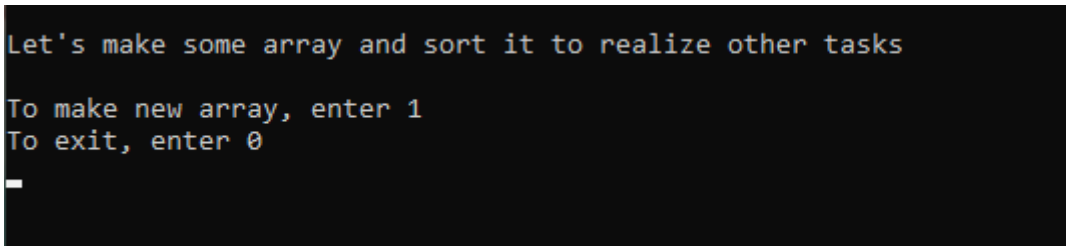

Рисунок 6 - Создание массива

2. Отсортировать заданный в пункте 1 элементы массива […] сортировкой (от меньшего к большему). Определить время, затраченное на сортировку, используя библиотеку chrono.

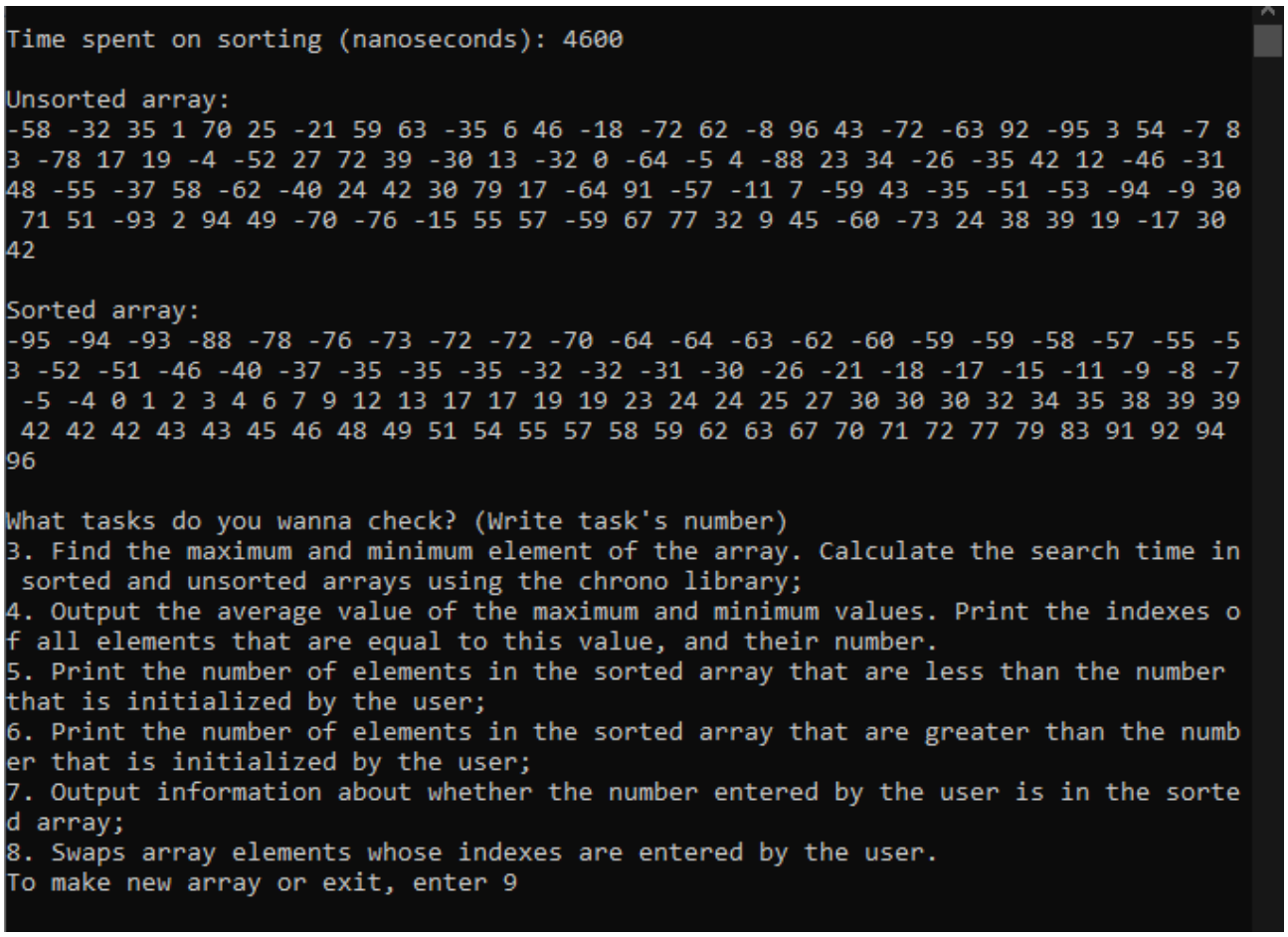

Рисунок 7 - Реализация первого и второго задания, а также меню

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

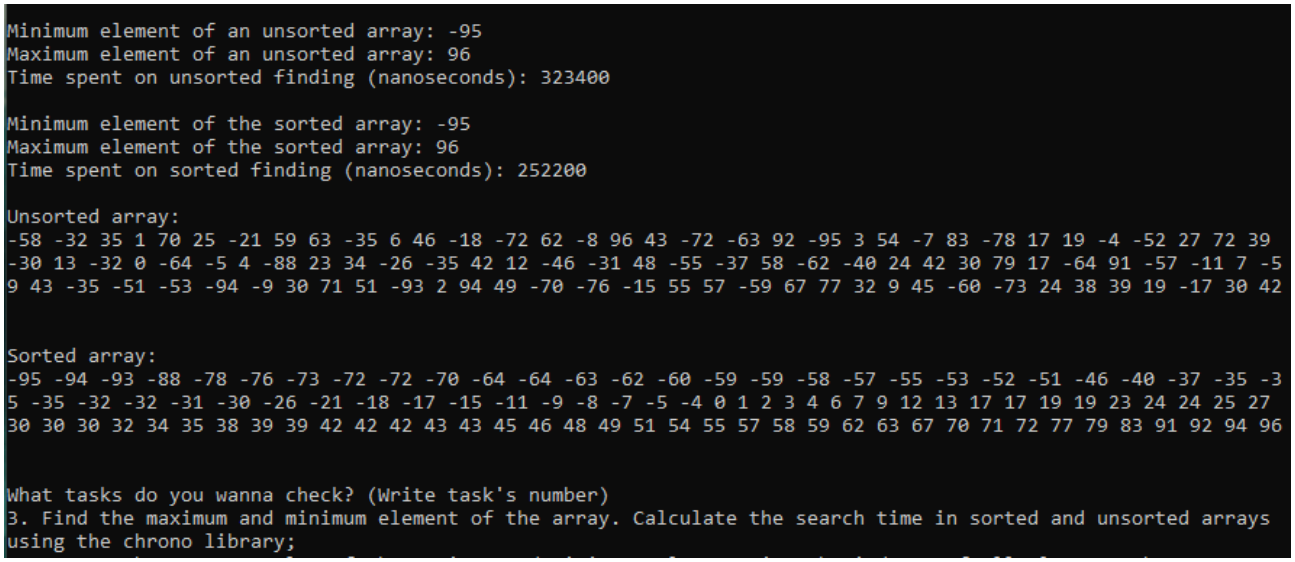

Рисунок 8 - Реализация 3 задания

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

```
The average value of the maximum and minimum values: 0
Indexes of all elements that are equal to this value:
37
Total of these in the array: 1

Unsorted array:
-58 -32 35 1 70 25 -21 59 63 -35 6 46 -18 -72 62 -8 96 43 -72 -63 92 -95 3 54 -7 8
3 -78 17 19 -4 -52 27 72 39 -30 13 -32 0 -64 -5 4 -88 23 34 -26 -35 42 12 -46 -31
48 -55 -37 58 -62 -40 24 42 30 79 17 -64 91 -57 -11 7 -59 43 -35 -51 -53 -94 -9 30
 71 51 -93 2 94 49 -70 -76 -15 55 57 -59 67 77 32 9 45 -60 -73 24 38 39 19 -17 30
42

Sorted array:
-95 -94 -93 -88 -78 -76 -73 -72 -72 -70 -64 -64 -63 -62 -60 -59 -59 -58 -57 -55 -5
3 -52 -51 -46 -40 -37 -35 -35 -35 -32 -32 -31 -30 -26 -21 -18 -17 -15 -11 -9 -8 -7
 -5 -4 0 1 2 3 4 6 7 9 12 13 17 17 19 19 23 24 24 25 27 30 30 30 32 34 35 38 39 39
 42 42 42 43 43 45 46 48 49 51 54 55 57 58 59 62 63 67 70 71 72 77 79 83 91 92 94
96

What tasks do you wanna check? (Write task's number)
3. Find the maximum and minimum element of the array. Calculate the search time in
 sorted and unsorted arrays using the chrono library;
```

Рисунок 9 - Реализация 4 задания

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

```
To search for elements in the sorted array that are less than the number a, enter
a: 57

Elements less than a: 84

Unsorted array:
-58 -32 35 1 70 25 -21 59 63 -35 6 46 -18 -72 62 -8 96 43 -72 -63 92 -95 3 54 -7 8
3 -78 17 19 -4 -52 27 72 39 -30 13 -32 0 -64 -5 4 -88 23 34 -26 -35 42 12 -46 -31
48 -55 -37 58 -62 -40 24 42 30 79 17 -64 91 -57 -11 7 -59 43 -35 -51 -53 -94 -9 30
 71 51 -93 2 94 49 -70 -76 -15 55 57 -59 67 77 32 9 45 -60 -73 24 38 39 19 -17 30
42

Sorted array:
-95 -94 -93 -88 -78 -76 -73 -72 -72 -70 -64 -64 -63 -62 -60 -59 -59 -58 -57 -55 -5
3 -52 -51 -46 -40 -37 -35 -35 -35 -32 -32 -31 -30 -26 -21 -18 -17 -15 -11 -9 -8 -7
 -5 -4 0 1 2 3 4 6 7 9 12 13 17 17 19 19 23 24 24 25 27 30 30 30 32 34 35 38 39 39
 42 42 42 43 43 45 46 48 49 51 54 55 57 58 59 62 63 67 70 71 72 77 79 83 91 92 94
96

What tasks do you wanna check? (Write task's number)
3. Find the maximum and minimum element of the array. Calculate the search time in
 sorted and unsorted arrays using the chrono library;
```

Рисунок 10 - Реализация 5 задания, где a = 57

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

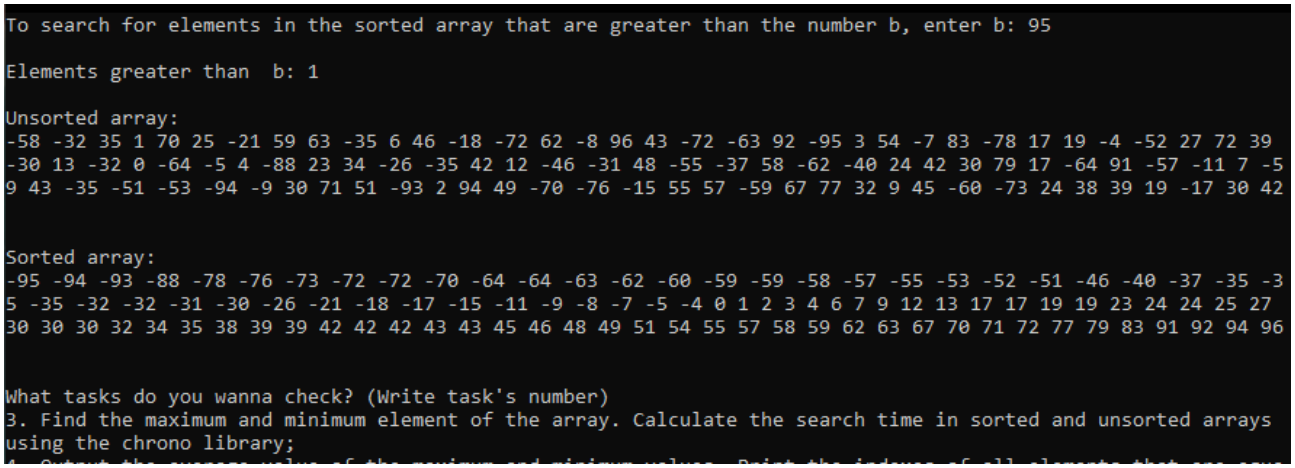

Рисунок 11 - Реализация 6 задания, где b = 95

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

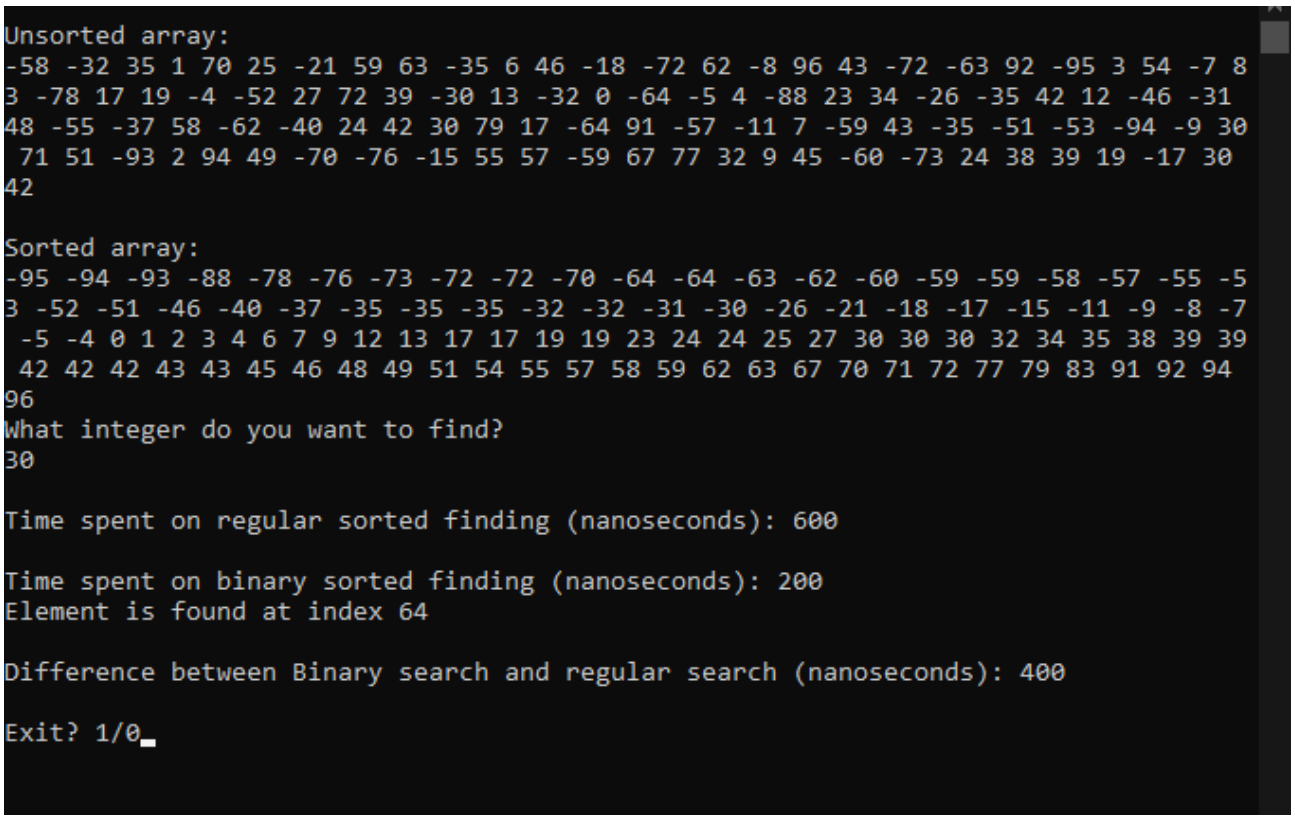

Рисунок 12 - Реализация 7 задания

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

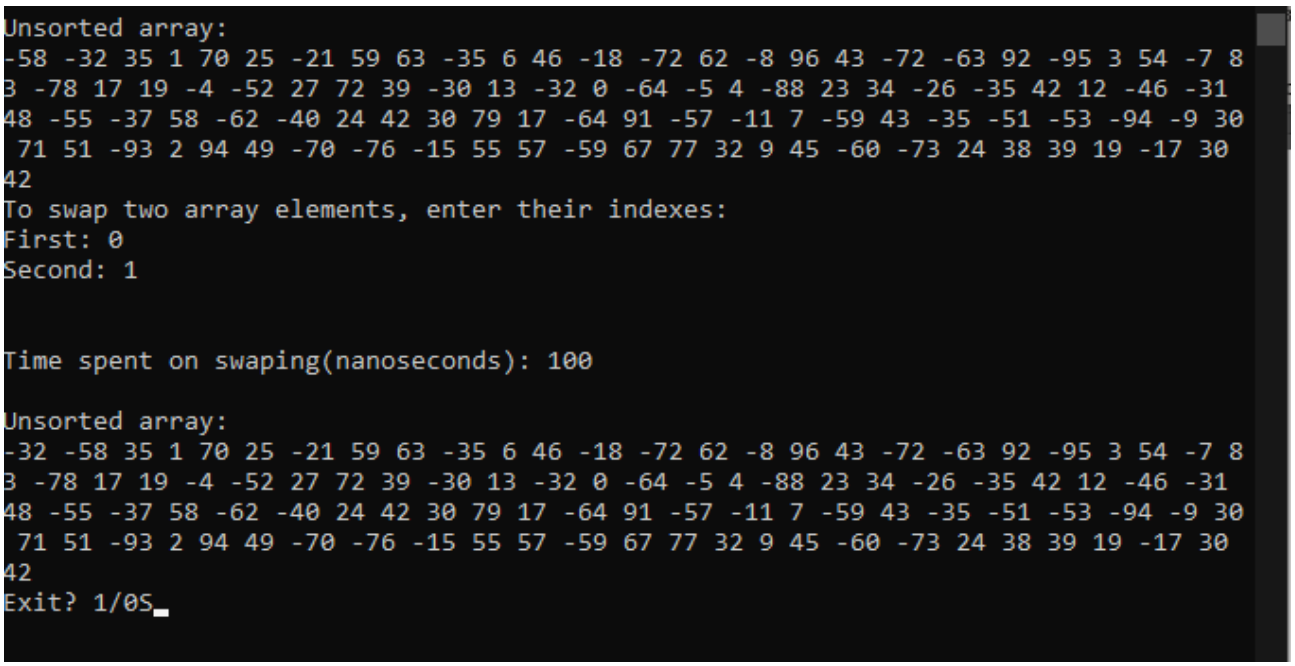

Рисунок 13 - Реализация 8 задания

Третья практическая работа:

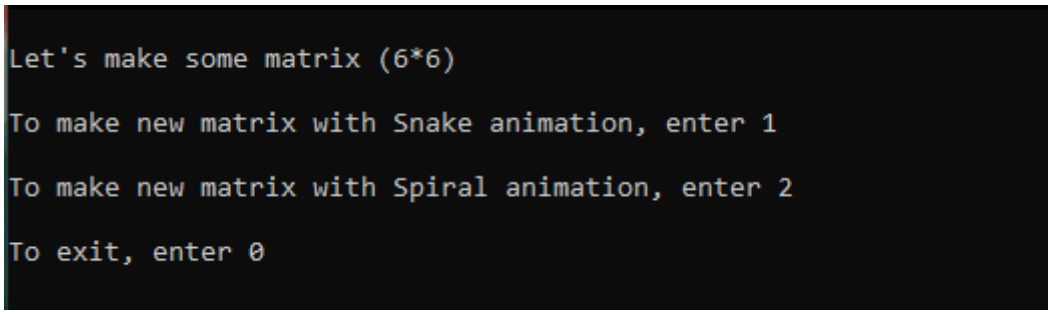

Рисунок 14 - Создание матрицы

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

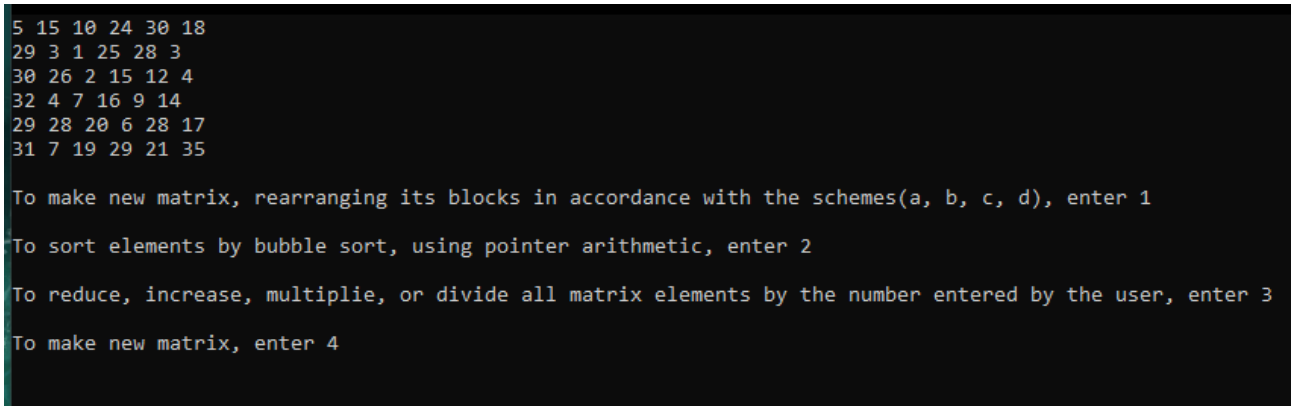

Рисунок 15 - Реализация первого задания, а также меню

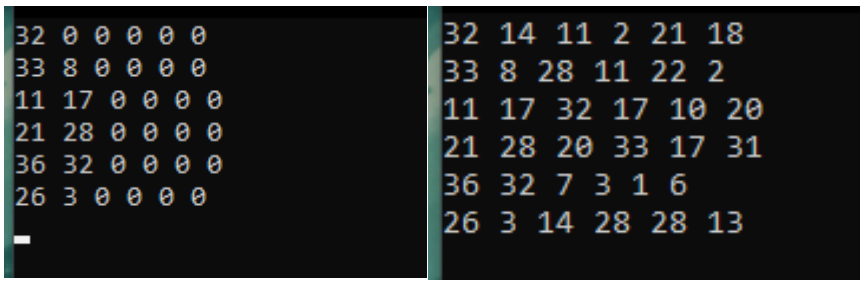

Рисунок 16 – Процесс реализации 1 задания в соответствии со схемой (б)

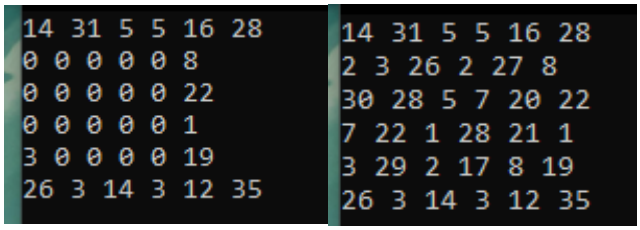

Рисунок 17 – Процесс реализации 1 задания в соответствии со схемой (а)

2. Используя арифметику указателей, сортирует элементы любой сортировкой.

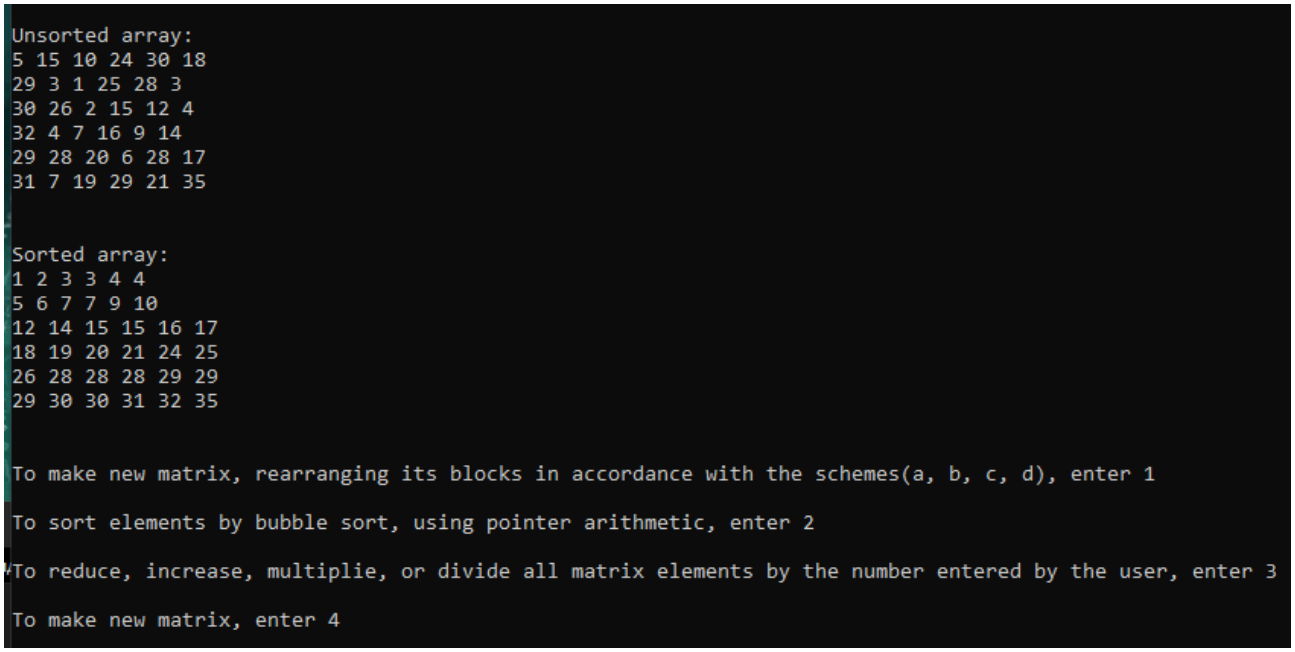

Рисунок 18 - Реализация 3 задания (сортировка выполнена в первую очередь для более наглядного представления других заданий)

3. Получает новую матрицу, из матрицы п. 1, переставляя ее блоки в соответствии со схемами:

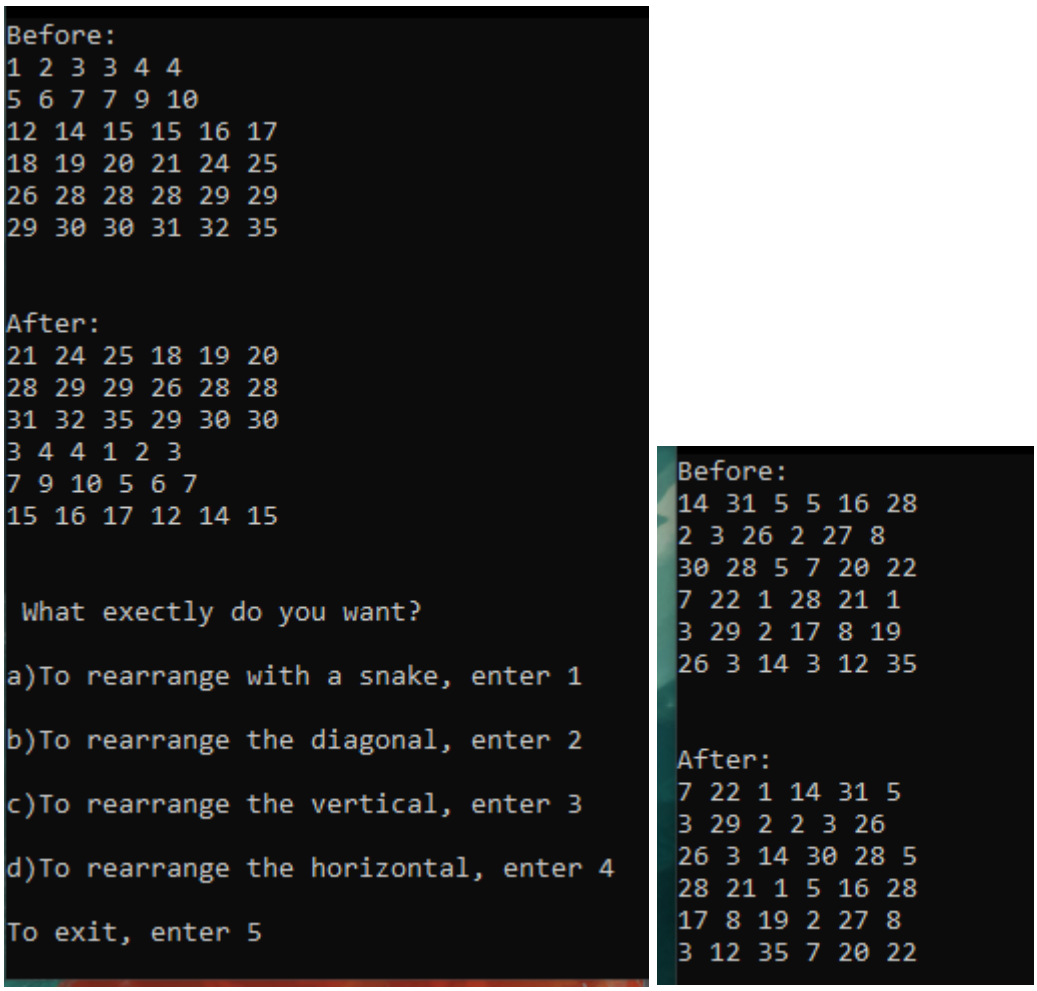

Рисунок 19 – Реализация 2 задания (пункт B и A(взята другая матрица), также работают C и D)

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

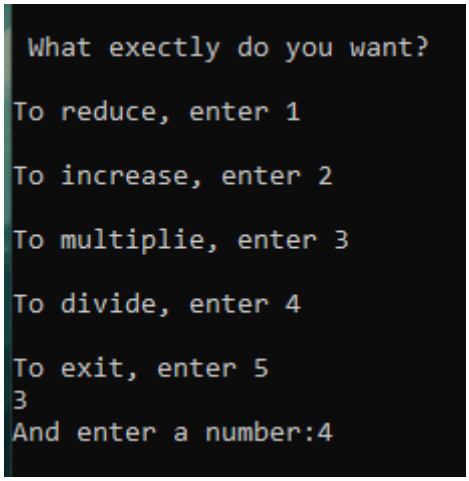

Рисунок 20 – Меню выполнения 4 задания (было выбрано умножение всех элементов на 4)

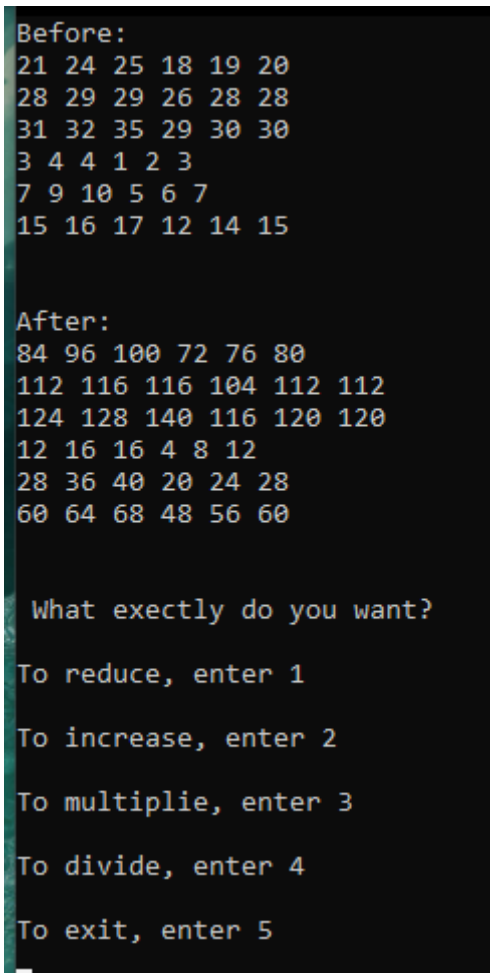

Рисунок 21 – Результат выполнения 4 задания (было выбрано умножение всех элементов на 4)

Четвертая практическая работа:

1. С клавиатуры или с 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: 6

После окончания ввода последовательности вывести на экран сначала все слова, содержащие только буквы, затем слова, содержащие только цифры, а потом слова, содержащие и буквы, и цифры.

4. Выполнить задание по варианту: 6

Вывести все слова исходной последовательности на экран вертикально.

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. (*)

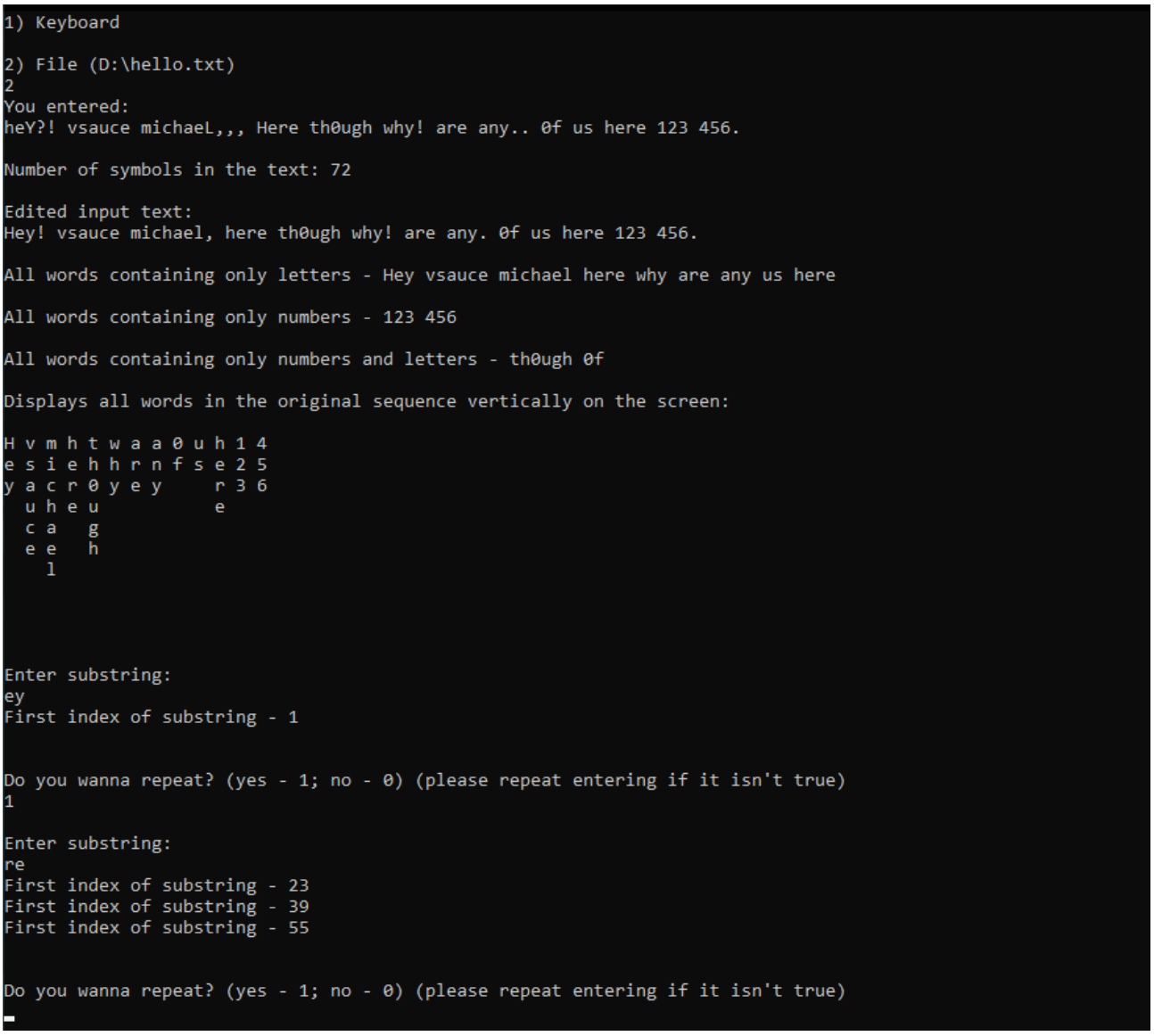

Рисунок 16 – Результат работы программы (4 практическая работа)