

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. 0323

Наумов А.С.

Преподаватель

Глущенко А.Г

Санкт-Петербург

2020

Цель работы

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

Основные теоретические положения

Каждая программа используется для обработки данных. Существуют разные типы данных, которые хранятся и обрабатываются по-разному. Все данные хранятся в памяти компьютера в виде двоичных кодов. Компьютерная память - это непрерывная последовательность двоичных ячеек, каждая из которых может находиться в двух состояниях, которые условно можно обозначить как 0 и 1. Каждая из этих двоичных ячеек называется битом. Последовательность, условно разделенная на 8 бит, называется байтами. Следовательно, 1 байт = 8 бит. Байт, в свою очередь, является базовой единицей измерения объема памяти. На любом алгоритмическом языке каждая константа, переменная, результат вычисления выражения или функции должны иметь определенный тип, который определяет:

- о характер данных (знаковое или число без знака, целое или с дробной частью, последовательность символов или одиночный символ и т.д.);
- о объем памяти, который занимают в памяти эти данные;
- о диапазон или множество возможных значений;
- о 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:

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

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

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

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

Массивы.

Для простых переменных каждая область памяти для хранения данных имеет собственное имя. Если для выполнения одних и тех же действий требуется группа значений одного типа, они получают одинаковое имя и отличаются своим порядковым номером (индексом). Это позволяет вам компактно записывать несколько операций с помощью цикла.

Массив - это проиндексированная последовательность подобных элементов с заранее определенным количеством элементов. Одномерный массив можно визуализировать как набор пронумерованных ячеек, каждая из которых содержит 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 при $i = 0, 1, 2, \dots, n$ – 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 меньшему). 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 продолжится. 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 делит 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, равны 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 строку 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 не порядок 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.

```

#include <iostream>
#include <ctime>
#include <chrono>
#include <algorithm>
#include <set>
#include <numeric>
#include <iomanip>
#include <fstream>
#include <cstring>

using namespace std;
using namespace chrono;

const int gl_SIZESHORTINTBITS = 8 * sizeof(signed short int);

void binary()
{
    cout << "Memory in byte on the your computer !\n";
    cout << " " << endl;
    bool bool_1;
    char char_1;
    signed char signed_char;
    //signed short int ssi;
    unsigned char unsigned_char;
    wchar_t wchar_t_1;
    char16_t char16;
    char32_t char32;
    short short_1; //short int
    unsigned short unsigned_short; //unsigned short int
    int int_1;
    unsigned int unsigned_int;
    long long_1;
    unsigned long unsigned_long; //unsigned long int
    long long long_long; // long long int, signed long long int и signed long long.
    unsigned long long unsigned_long_long; //unsigned long long int
    float float_1;
    double d_2;
    long double long_double;

    signed short int x; // переменная служит для ввода целых чисел!

    //void: тип без значения

    cout << "1) " << endl;
    cout << "bool: " << sizeof(bool_1) << " bytes" << endl;
    cout << "char: " << sizeof(char_1) << " bytes" << endl;
    cout << "signed char: " << sizeof(signed_char) << " bytes" << endl;
    cout << "unsigned char: " << sizeof(unsigned_char) << " bytes" << endl;

    cout << "wchar_t: " << sizeof(wchar_t_1) << " bytes" << endl;
    cout << "char16_t: " << sizeof(char16) << " bytes" << endl;
    cout << "char32_t: " << sizeof(char32) << " bytes" << endl;
    cout << "short: " << sizeof(short_1) << " bytes" << endl;

    cout << "unsigned short: " << sizeof(unsigned_short) << " bytes" << endl;
    cout << "int: " << sizeof(int_1) << " bytes" << endl;
    cout << "unsigned int: " << sizeof(unsigned_int) << " bytes" << endl;

```

```

cout << "long: " << sizeof(long_1) << " bytes" << endl;
cout << "unsigned long: " << sizeof(unsigned_long) << " bytes" << endl;

cout << "long long: " << sizeof(long_long) << " bytes" << endl;
cout << "unsigned long long: " << sizeof(unsigned_long_long) << " bytes" << endl;
cout << "float: " << sizeof(float_1) << " bytes" << endl;
cout << "double: " << sizeof(d_2) << " bytes" << endl;
cout << "long double: " << sizeof(long_double) << " bytes" << endl;
cout << "ssi: " << sizeof(x) << " bytes" << endl;
cout << " " << endl;
cout << "===== " <<
endl;
cout << " " << endl;
cout << "2) " << endl;

cout << "Enter the number: ";
cin >> x;

int numbit = 0;
cout << "Enter the number byte: ";
cin >> numbit;

int bits[gl_SIZESHORTINTBITS];

if (x < 0)
{
    bits[gl_SIZESHORTINTBITS - 1] = 1;
}

//цикл для получения числа
for (int i = 0; i < gl_SIZESHORTINTBITS - 1; ++i)
{
    if (x % 2 == 0)
    {
        bits[i] = 0;
    }

    else
    {
        bits[i] = 1;
    }

    x = x / 2;
}

//циклы для инверсии
for (int i = 0; i < gl_SIZESHORTINTBITS - 1; ++i)
{
    if (bits[i] == 1)
    {
        bits[i] = 0;
    }
    else
    {
        bits[i] = 1;
    }
}

//bool carrier = true;

```

```

for (int i = 0; i < gl_SIZESHORTINTBITS; ++i)
{
    if (bits[i] == 1)
    {
        bits[i] = 0;
    }

    else
    {
        bits[i] = 1;
        //carrier = false;
        //break;
    }
}

for (int i = gl_SIZESHORTINTBITS - 1; i >= 0; --i)
{
    cout << bits[i];
}

cout << endl;
cout << "Number bits " << numbit << " : " << bits[numbit] << endl;
cout << endl;
}
void bubbleSort(int a[])
{
    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10 - i - 1; j++)
        {
            if (a[j] > a[j + 1])
            {
                //swap(a[j] , a[j+1]);
                int temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }
}
void sort()
{
    const int SIZE = 10;
    int N[SIZE], min, max, midle;

    srand(time(NULL)); // инициализация генерации случайных чисел

    cout << endl;

    for (int i = 0; i < 10; i++)
    {
        N[i] = (rand() % 100 - 35);
    }

    cout << "Before Sorting: " << " ";
    for (int i = 0; i < 10; i++)
    {

```

```

        cout << N[i] << " ";
    }

    cout << endl;
    cout << endl;

    bubbleSort(N);

    cout << "After Sorting: " << " ";
    for (int i = 0; i < 10; i++)
    {
        cout << N[i] << " ";
    }

    system_clock::time_point start = system_clock::now();

    cout << "\n";

    system_clock::time_point end = system_clock::now();
    duration<double> sec = end - start;

    cout << endl;

    cout << "seconds: " << sec.count() << endl;
    /*
    cout << "Task 3" << endl;
    N[SIZE];
    int size1 = sizeof(N[SIZE] / sizeof(int));
    int summ = accumulate(&N[0], &N[size1], 0);
    int mean = summ / size1;

    cout << mean << endl;
    for (int i = 0; i < size1; i++)
    {
        cout << N[i] << " ";
    }
    */
    /*
    int a1[5];
    for (int i = 0; i < 5; i++)
    {
        if (a1[i] == N[i]);
    }

    cout << endl;

    int b1[5];
    for (int i = 0; i < 5; i++)
    {
        if (b1[i] == N[i+1])
    }
    */

    cout << endl;
    cout << endl;

    int a;
    cout << "Insert the number: " << " ";
    cin >> a;

```

```

cout << "less than a : " << " ";
for (int i = 0; i < 10; ++i)
{
    if (a < N[i]) { cout << N[i] << " "; }
}

cout << endl;
cout << endl;

int b;
cout << "Insert the number: " << " ";
cin >> b;
cout << "number more a : " << " ";
for (int i = 0; i < 10; ++i)
{
    if (b > N[i]) { cout << N[i] << " "; }
}

cout << endl;
cout << endl;

cout << endl;
cout << "-----" << endl;

min = N[0];
max = N[0];

for (int i = 1; i < SIZE; i++)
{
    if (N[i] < min) { min = N[i]; }
    if (N[i] > max) { max = N[i]; }
}

midle = N[0];
for (int i = 1; i < SIZE; i++)
{
    if (N[i] = max / min) { midle = N[i]; }
}

cout << "Min: " << min << endl;
cout << "Max: " << max << endl;
cout << "Midle: " << midle << endl;

cout << endl;

//cout << "Number a: " << a << endl;

cout << endl;
}
void mySwap(int& value1, int& value2)
{
    int temp = value1;
    value1 = value2;
    value2 = temp;
}
void matrixReflection(int** a, int n)

```



```

{
    int** ab = new int* [n]; // Создаем массив указателей
    for (int i = 0; i < n; i++)
    {
        ab[i] = new int[n]; // Создаем элементы
    }

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            ab[i][j] = a[i][j];
        }
    }

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n / 2; j++)
        {
            mySwap(ab[i][j], ab[i][j + n / 2]);
        }
    }

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << setw(4) << ab[i][j] << ' ';
        }
        cout << '\n';
    }
}

void matrixReflectionTwo(int** a, int n)
{
    int** ab1 = new int* [n]; // Создаем массив указателей
    for (int i = 0; i < n; i++)
    {
        ab1[i] = new int[n]; // Создаем элементы
    }

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            ab1[i][j] = a[i][j];
        }
    }

    for (int i = 0; i < n / 2; i++)
    {
        for (int j = 0; j < n / 2; j++)
        {
            mySwap(ab1[i][j], ab1[i + n / 2][j + n / 2]);
        }
    }

    for (int i = n / 2; i < n; i++)
    {
        for (int j = 0; j < n / 2; j++)

```

```

        {
            mySwap(ab1[i][j], ab1[i - n / 2][j + n / 2]);
        }
    }

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << setw(4) << ab1[i][j] << ' ';
        }
        cout << '\n';
    }
}

void matrixReflectionThree(int** a, int n)
{
    int** ab2 = new int* [n]; // Создаем массив указателей
    for (int i = 0; i < n; i++)
    {
        ab2[i] = new int[n]; // Создаем элементы

        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                ab2[i][j] = a[i][j];
            }
        }

        for (int i = n / 2; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                mySwap(ab2[i][j], ab2[i - n / 2][j]);
            }
        }

        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                cout << setw(4) << ab2[i][j] << ' ';
            }
            cout << '\n';
        }
    }
}

void matrixRound(int** a, int n)
{
    int** ab3 = new int* [n]; // Создаем массив указателей
    for (int i = 0; i < n; i++)
    {
        ab3[i] = new int[n]; // Создаем элементы

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

```

```

        {
            ab3[i][j] = a[i][j];
        }
    }

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n / 2; j++)
        {
            mySwap(ab3[i][j], ab3[i][j + n / 2]);
        }
    }

    for (int i = 0; i < n / 2; i++)
    {
        for (int j = 0; j < n / 2; j++)
        {
            mySwap(ab3[i][j], ab3[i + n / 2][j + n / 2]);
        }
    }

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << setw(4) << ab3[i][j] << ' ';
        }
        cout << '\n';
    }
}

void matrixPointers(int** a, int n) {

    int** ab = new int* [n]; // Создаем массив указателей
    for (int i = 0; i < n; i++)
    {
        ab[i] = new int[n]; // Создаем элементы
    }

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            ab[i][j] = a[i][j];
        }
    }

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << setw(4) << ab[i][j] << ' ';
        }
        cout << '\n';
    }
}

void pointers()
{
    const int N = 10;
    int n;

```

```

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

int a[N][N];

int k = 1;
int i = 0;
int j = 0;

while (k <= n * n) {

    a[i][j] = k; //действие заполняет массив

    if (i <= j + 1 && i + j < n - 1)
        ++j;
    else if (i < j && i + j >= n - 1)
        ++i;
    else if (i >= j && i + j > n - 1)
        --j;
    else
        --i;
    ++k;
}

cout << "Before\n";

for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j)
    {
        //a[i][j] = 0;
        cout << setw(4) << a[i][j];

    }
    cout << endl;
}

cout << "=====\n";

int** a1 = new int* [n]; // Создаем массив указателей
for (int i = 0; i < n; i++)
{
    a1[i] = new int[n]; // Создаем элементы
}

int** sort = new int* [n]; // Создаем массив указателей
for (int i = 0; i < n; i++)
{
    sort[i] = new int[n]; // Создаем элементы
}

while (k <= n * n)
{
    a1[i][j] = k; //действие заполняет массив

    if (i <= j + 1 && i + j < n - 1)
        ++j;

```

```

        else if (i < j && i + j >= n - 1)
            ++i;
        else if (i >= j && i + j > n - 1)
            --j;
        else
            --i;
        ++k;
    }

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j)
        {
            a1[i][j] = rand() % 50;
            cout << setw(4) << a1[i][j];

        }

        cout << endl;
    }

    cout << "=====\n";

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (a1[j] > a1[j + 1])
            {
                swap(a1[j], a1[j + 1]);
                /*int t = sort[j];
                sort[j] = sort[j + 1];
                sort[j + 1] = t;*/
            }
        }
    }

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j)
        {
            cout << setw(4) << a1[i][j];
        }
        cout << endl;
    }

    //cout << "====Variant2=====\n";

    //for (int i = 0; i < n; ++i) {
    //    for (int j = 0; j < n; ++j)
    //    {
    //        a1[i][j] = rand() % 100;
    //        cout << setw(4) << a1[i][j];

    //    }

    //    cout << endl;

```

```

//}

//cout << "=====\n";

//

//for (int i = 0; i < n; i++)
//{
//    int maxi = 0;
//    int maxj = 0;

//    for (int j = 0; j < n; j++)
//    {
//        if (a1[i][j] > a1[maxi][maxj]) {

//            maxi = i;
//            maxj = j;
//        }
//    }

//    if (a[0][0] != a1[maxi][maxj])
//    {
//        int tmp;
//        if (maxi != 0)
//            for (int j = 0; j < n; j++)
//            {
//                tmp = a1[maxi][j];
//                a1[maxi][j] = a1[0][j];
//                a1[0][j] = tmp;
//            }

//        if (maxj != 0)
//            for (int i = 0; i < n; i++)
//            {
//                tmp = a1[i][maxj];
//                a1[i][maxj] = a1[i][0];
//                a1[i][0] = tmp;
//            }
//    }
//}

//cout << endl;

//cout << "|";
//for (int i = 0; i < n; ++i) {

//    for (int j = 0; j < n; ++j)
//    {
//        //a1[i][j] = rand() % 50;
//        cout << setw(4) << a1[i][j] << "|";
//    }

//    cout << endl;
//}

```

```

//cout << "=====Variant2=====\\n";

cout << "=====\\n";

matrixReflection(a1, n);

cout << "=====\\n";

//matrixReflectionTwo(a1,n);

cout << "=====\\n";

matrixReflectionThree(a1, n);

cout << "=====\\n";

//matrixPointers(a1,n);

//matrixRound(a1,n);
}

void print(const string& item)
{
    cout << item << endl;
}

void file()
{
    string path = "string.txt";
    fstream fs;
    fs.open(path, fstream::in | fstream::out | fstream::app);

    if (!fs.is_open()) {
        cout << "Error!";
    }

    else {

        string msg;
        int size = 0;

        int value;
        cout << "File is open \\n";
        cout << "=====\\n";
<< endl;
        cout << "Enter the number 1 for write text to file or 2 our text from file: ";
        cin >> value;

        if (value == 1) {

            cout << "Enter the size of an array: ";
            cin >> size;
            cin.ignore(); // очищает буфер

```

```

for (int i = 0; i < size; i++)
{

    cout << "Enter the text: ";
    cin >> msg;

    for (int i = 1; i < msg.length(); i++)
    {
        if (msg[i] == ' ') {
            msg.erase(i--, 1);
        }

        if (ispunct(msg[i])) {
            msg.erase(i++, 1);
        }

        if (ispunct(msg[i])) {
            msg.erase(i++, 1);
        }

        if ('a' <= msg[i] && msg[i] <= 'z') {
            msg[i] = char(((int)msg[i]) - 32);
        }

        if ('A' <= msg[i] && msg[i] <= 'Z') {
            msg[i] = char(((int)msg[i]) + 32);
        }

        /*for (int i = 0; i < msg.length() - 1; i++)
        {
            for (int j = 1; j < msg.length() - i; j++)
            {
                if ((msg[j-1], msg[j]) == 1)
                {
                    string temp = msg[j - 1];
                    msg[j - 1] = msg[j];
                    msg[j] = temp;
                }
            }
        }*/

        /*string temp;
        bool yes = true;
        while (yes)
        {
            yes = false;

            for (int i = 0; i < msg.length(); i++)
            {
                for (int j = i + 1; j < msg.length(); j++)
                {
                    if (strcmp(msg[i], msg[j]) > 0)
                    {
                        strcpy(temp, msg[i]);
                        strcpy(msg[i], msg[j]);
                        strcpy(msg[j], temp);
                        yes = true;
                    }
                }
            }
        }
    }
}

```



```

        }
    }*/

    /*string temp;
    bool yes = true;
    while (yes)
    {
        yes = false;

        for (int i = 0; i < msg.length() - 1; i++)
        {
            for (int j = i + 1; j < msg.length() - i; j++)
            {
                if ((msg[i], msg[j]) > 0)
                {
                    (temp, msg[i]);
                    (msg[i], msg[j]);
                    (msg[j], temp);
                    yes = true;
                }
            }
        }
    }*/

    }

    fs << msg << " ";
}

}

if (value == 2) {
    cout << "Reading data from a file";
    while (!fs.eof()) {
        msg = "";
        fs >> msg;
        cout << msg << " ";
    }
}

}

fs.close();
}

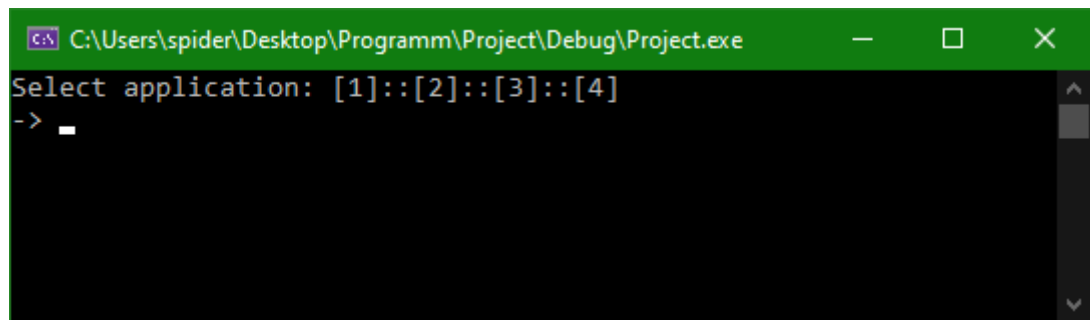
int main()
{
    int value;
    cout << "Select application: [1]::[2]::[3]::[4] \n";
    cout << "-> ";
    cin >> value;

    if (value == 1)
    {
        binary();
    }
}

```

```
    }  
  
    if (value == 2)  
    {  
        sort();  
    }  
  
    if (value == 3)  
    {  
        pointers();  
    }  
  
    if (value == 4)  
    {  
        file();  
    }  
  
    else  
    {  
        cout << "error";  
    }  
  
    system("pause");  
}
```

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.

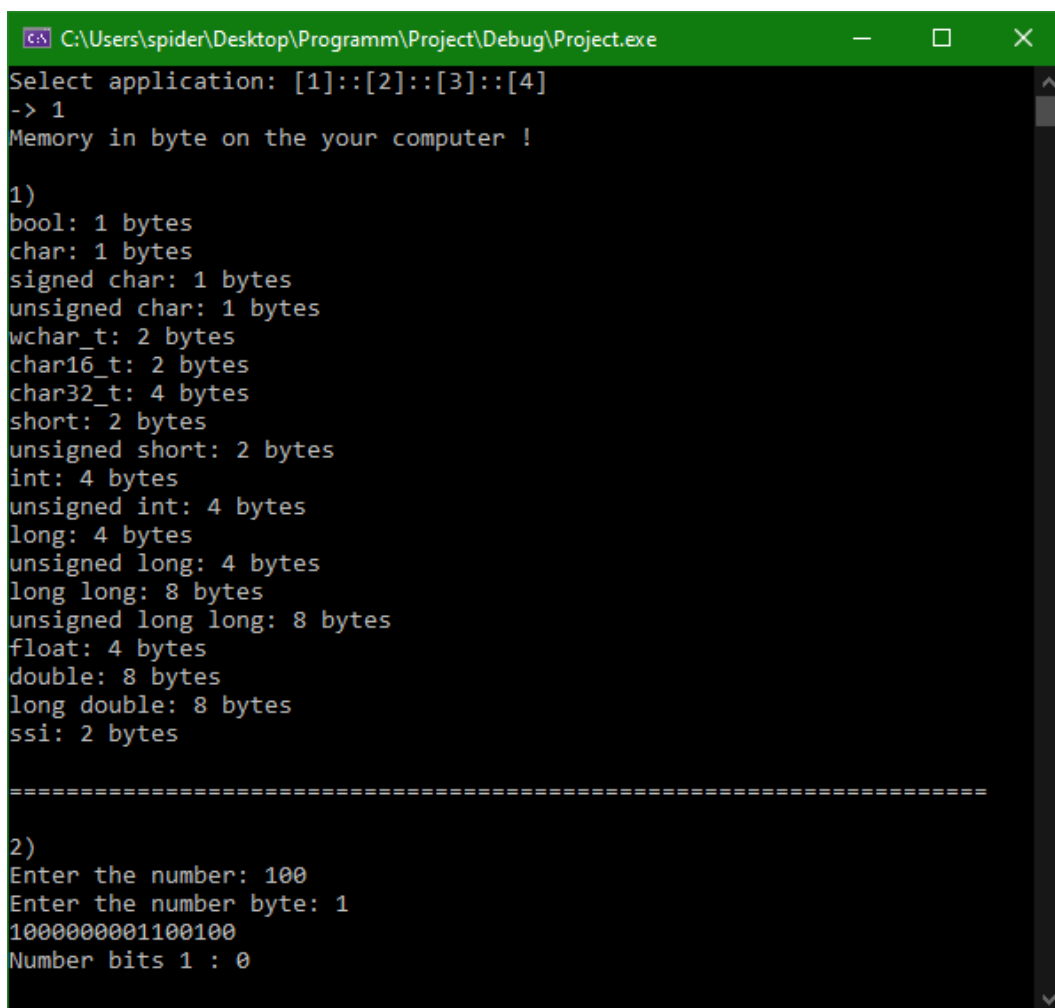


Рисунок 2 - Задание 1

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.

```
=====
2)
Enter the number: 100
Enter the number byte: 1
1000000001100100
Number bits 1 : 0
```

Рисунок 3 - Задание 2

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

```

Enter float:
3.14
0 10000000 10010001111010111000011
Exit? 1/00

Enter float:
-3.14
1 10000000 10010001111010111000011
Exit? 1/0_

```

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.

```

Enter double:
3.14
0 100000000000 1001000111101011100001010001111010111000010100011111
Exit? 1/00

Enter double:
-3.14
1 100000000000 1001000111101011100001010001111010111000010100011111
Exit? 1/0_

```

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.

```

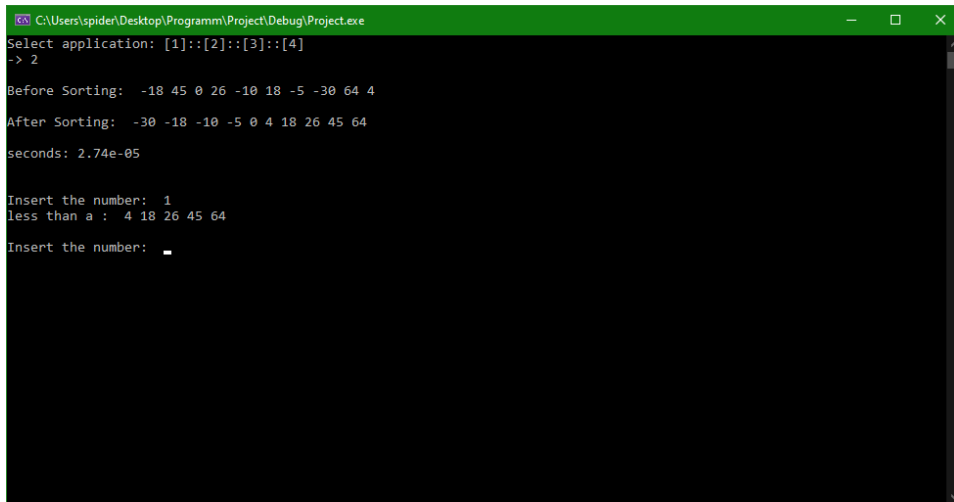
C:\Users\spider\Desktop\Programm\Project\Debug\Project.exe
Select application: [1]::[2]::[3]::[4]
-> 2
Before Sorting: -18 45 0 26 -10 18 -5 -30 64 4
After Sorting: -30 -18 -10 -5 0 4 18 26 45 64
seconds: 2.74e-05
Insert the number: 1
less than a : 4 18 26 45 64
Insert the number: _

```

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.



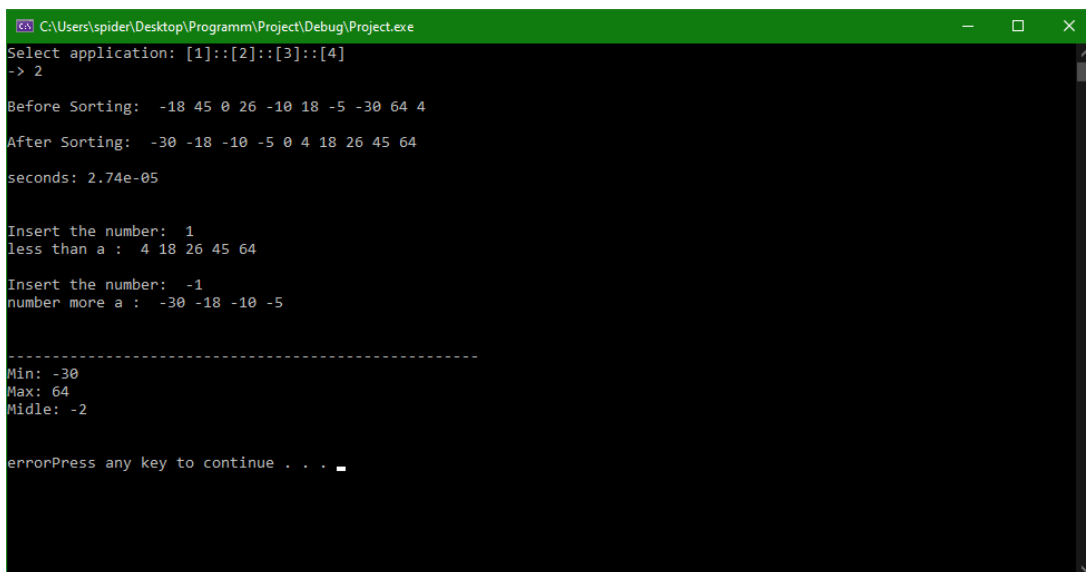
```
C:\Users\spider\Desktop\Programm\Project\Debug\Project.exe
Select application: [1]::[2]::[3]::[4]
-> 2

Before Sorting: -18 45 0 26 -10 18 -5 -30 64 4
After Sorting: -30 -18 -10 -5 0 4 18 26 45 64
seconds: 2.74e-05

Insert the number: 1
less than a : 4 18 26 45 64
Insert the number: _
```

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.



```
C:\Users\spider\Desktop\Programm\Project\Debug\Project.exe
Select application: [1]::[2]::[3]::[4]
-> 2

Before Sorting: -18 45 0 26 -10 18 -5 -30 64 4
After Sorting: -30 -18 -10 -5 0 4 18 26 45 64
seconds: 2.74e-05

Insert the number: 1
less than a : 4 18 26 45 64

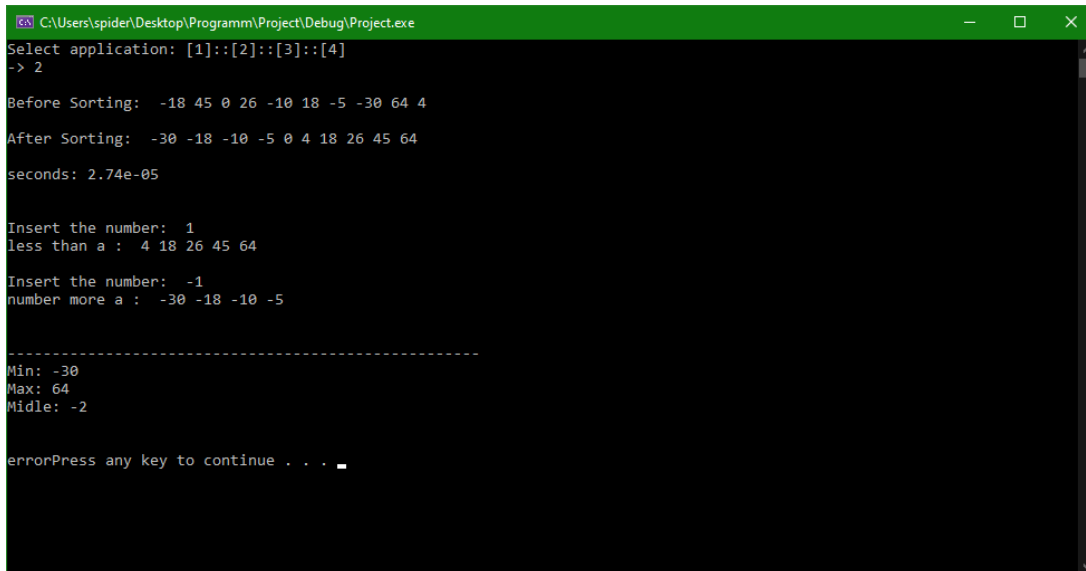
Insert the number: -1
number more a : -30 -18 -10 -5

-----
Min: -30
Max: 64
Middle: -2

errorPress any key to continue . . . _
```

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.



```
C:\Users\spider\Desktop\Programm\Project\Debug\Project.exe
Select application: [1]::[2]::[3]::[4]
-> 2

Before Sorting: -18 45 0 26 -10 18 -5 -30 64 4
After Sorting: -30 -18 -10 -5 0 4 18 26 45 64
seconds: 2.74e-05

Insert the number: 1
less than a : 4 18 26 45 64

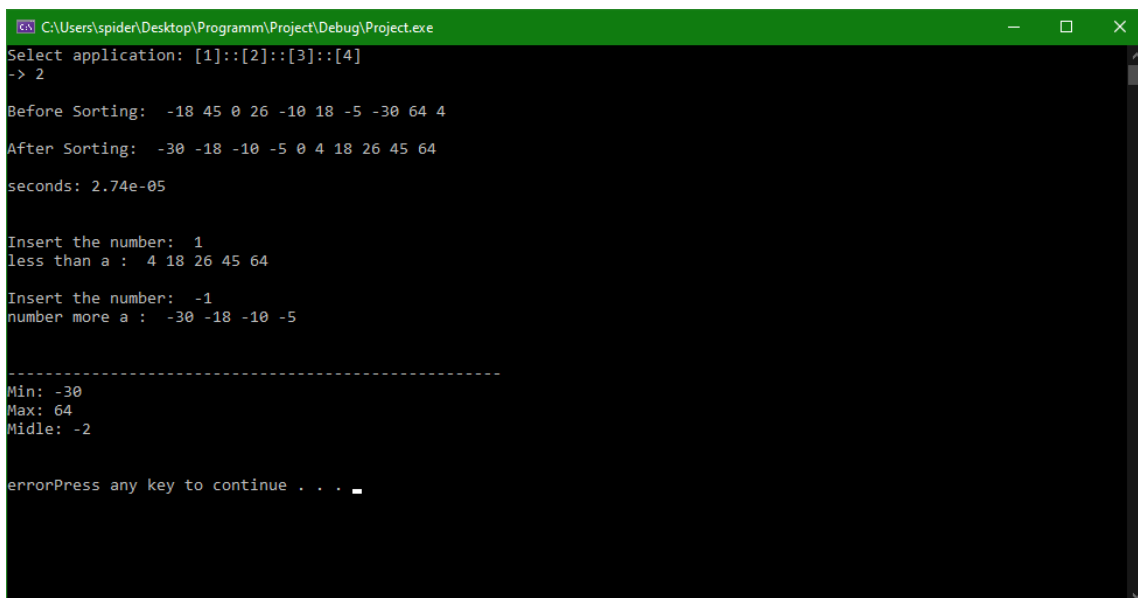
Insert the number: -1
number more a : -30 -18 -10 -5

-----
Min: -30
Max: 64
Midle: -2

errorPress any key to continue . . .
```

Рисунок 9 - Реализация 4 задания

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



```
C:\Users\spider\Desktop\Programm\Project\Debug\Project.exe
Select application: [1]::[2]::[3]::[4]
-> 2

Before Sorting: -18 45 0 26 -10 18 -5 -30 64 4
After Sorting: -30 -18 -10 -5 0 4 18 26 45 64
seconds: 2.74e-05

Insert the number: 1
less than a : 4 18 26 45 64

Insert the number: -1
number more a : -30 -18 -10 -5

-----
Min: -30
Max: 64
Midle: -2

errorPress any key to continue . . .
```

Рисунок 10 - Реализация 5 задания

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

```
C:\Users\spider\Desktop\Programm\Project\Debug\Project.exe
Select application: [1]::[2]::[3]::[4]
-> 2

Before Sorting: -18 45 0 26 -10 18 -5 -30 64 4
After Sorting: -30 -18 -10 -5 0 4 18 26 45 64
seconds: 2.74e-05

Insert the number: 1
less than a : 4 18 26 45 64

Insert the number: -1
number more a : -30 -18 -10 -5

-----
Min: -30
Max: 64
Middle: -2

errorPress any key to continue . . .
```

Рисунок 11 - Реализация 6 задания

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

```
C:\Users\spider\Desktop\Programm\Project\Debug\Project.exe
Select application: [1]::[2]::[3]::[4]
-> 2

Before Sorting: -18 45 0 26 -10 18 -5 -30 64 4
After Sorting: -30 -18 -10 -5 0 4 18 26 45 64
seconds: 2.74e-05

Insert the number: 1
less than a : 4 18 26 45 64

Insert the number: -1
number more a : -30 -18 -10 -5

-----
Min: -30
Max: 64
Middle: -2

errorPress any key to continue . . .
```

Рисунок 12 - Реализация 7 задания

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


```

C:\Users\spider\Desktop\Programm\Project\Debug\Project.exe
Select application: [1]::[2]::[3]::[4]
-> 2

Before Sorting: -18 45 0 26 -10 18 -5 -30 64 4
After Sorting: -30 -18 -10 -5 0 4 18 26 45 64
seconds: 2.74e-05

Insert the number: 1
less than a : 4 18 26 45 64

Insert the number: -1
number more a : -30 -18 -10 -5

-----
Min: -30
Max: 64
Midle: -2

errorPress any key to continue . . .

```

Рисунок 13 - Реализация 8 задания

Третья практическая работа:

```

C:\Users\spider\Desktop\Programm\Project\Debug\Project.exe
Select application: [1]::[2]::[3]::[4]
-> 3
Enter matrix size: 6
Before
 1  2  3  4  5  6
20 21 22 23 24  7
19 32 33 34 25  8
18 31 36 35 26  9
17 30 29 28 27 10
16 15 14 13 12 11
-----
41 17 34  0 19 24
28  8 12 14  5 45
31 27 11 41 45 42
27 36 41  4  2  3
42 32 21 16 18 45
47 26 21 38 19 12
-----
31 27 11 41 45 42
47 26 21 38 19 12
28  8 12 14  5 45
41 17 34  0 19 24
42 32 21 16 18 45
27 36 41  4  2  3
-----
41  45  42  31  27  11
38  19  12  47  26  21
14  5  45  28  8  12
 0  19  24  41  17  34
16  18  45  42  32  21
 4  2  3  27  36  41
-----
41  17  34  0  19  24
42  32  21  16  18  45
27  36  41  4  2  3
31  27  11  41  45  42
47  26  21  38  19  12
28  8  12  14  5  45
-----
errorPress any key to continue . . .

```

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.

```
C:\Users\spider\Desktop\Programm\Project\Debug\Project.exe
Select application: [1]::[2]::[3]::[4]
-> 3
Enter matrix size: 6
Before
 1  2  3  4  5  6
20 21 22 23 24  7
19 32 33 34 25  8
18 31 36 35 26  9
17 30 29 28 27 10
16 15 14 13 12 11
=====
41 17 34  0 19 24
28  8 12 14  5 45
31 27 11 41 45 42
27 36 41  4  2  3
42 32 21 16 18 45
47 26 21 38 19 12
=====
31 27 11 41 45 42
47 26 21 38 19 12
28  8 12 14  5 45
41 17 34  0 19 24
42 32 21 16 18 45
27 36 41  4  2  3
=====
41 45 42 31 27 11
38 19 12 47 26 21
14  5 45 28  8 12
 0 19 24 41 17 34
16 18 45 42 32 21
 4  2  3 27 36 41
=====
41 17 34  0 19 24
42 32 21 16 18 45
27 36 41  4  2  3
31 27 11 41 45 42
47 26 21 38 19 12
28  8 12 14  5 45
=====
errorPress any key to continue . . .
```

Рисунок 15 - Реализация первого задания, а также меню

Before											
1	2	3	4	5	6	41	17	34	0	19	24
20	21	22	23	24	7	28	8	12	14	5	45
19	32	33	34	25	8	31	27	11	41	45	42
18	31	36	35	26	9	27	36	41	4	2	3
17	30	29	28	27	10	42	32	21	16	18	45
16	15	14	13	12	11	47	26	21	38	19	12

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

2. Используя арифметику указателей, сортирует элементы любой сортировкой.

```

Select C:\Users\spider\Desktop\Programm\Project\Debug\Project.exe
Select application: [1]::[2]::[3]::[4]
-> 3
Enter matrix size: 6
Before
  1  2  3  4  5  6
20 21 22 23 24 7
19 32 33 34 25 8
18 31 36 35 26 9
17 30 29 28 27 10
16 15 14 13 12 11
=====
41 17 34 0 19 24
28 8 12 14 5 45
31 27 11 41 45 42
27 36 41 4 2 3
42 32 21 16 18 45
47 26 21 38 19 12
=====
42 32 21 16 18 45
41 17 34 0 19 24
31 27 11 41 45 42
27 36 41 4 2 3
47 26 21 38 19 12
28 8 12 14 5 45
=====
16 18 45 42 32 21
0 19 24 41 17 34
41 45 42 31 27 11
4 2 3 27 36 41
38 19 12 47 26 21
14 5 45 28 8 12
=====
27 36 41 4 2 3
47 26 21 38 19 12
28 8 12 14 5 45
42 32 21 16 18 45
41 17 34 0 19 24
31 27 11 41 45 42
=====
errorPress any key to continue . . .

```

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

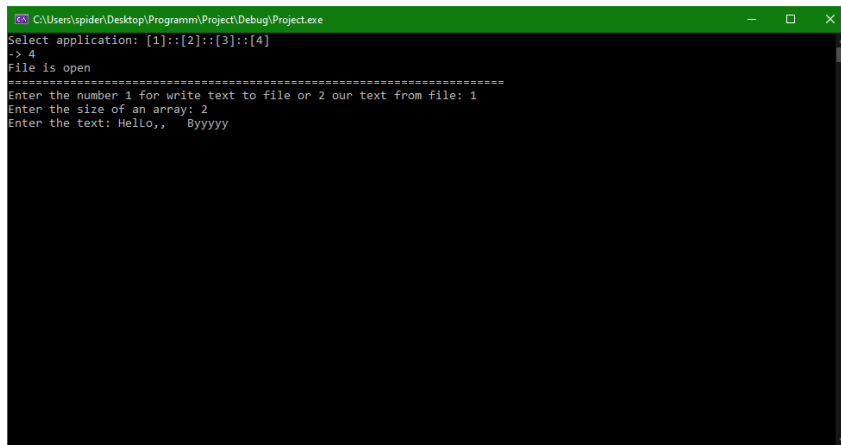
3. Получает новую матрицу, из матрицы п. 1, переставляя ее блоки в соответствии со схемами:

```
Select C:\Users\spider\Desktop\Programm\Project\Debug\Project.exe
Select application: [1]::[2]::[3]::[4]
-> 3
Enter matrix size: 6
Before
  1  2  3  4  5  6
20 21 22 23 24  7
19 32 33 34 25  8
18 31 36 35 26  9
17 30 29 28 27 10
16 15 14 13 12 11
=====
41 17 34  0 19 24
28  8 12 14  5 45
31 27 11 41 45 42
27 36 41  4  2  3
42 32 21 16 18 45
47 26 21 38 19 12
=====
42 32 21 16 18 45
41 17 34  0 19 24
31 27 11 41 45 42
27 36 41  4  2  3
47 26 21 38 19 12
28  8 12 14  5 45
=====
16 18 45 42 32 21
 0 19 24 41 17 34
41 45 42 31 27 11
 4  2  3 27 36 41
38 19 12 47 26 21
14  5 45 28  8 12
=====
27 36 41  4  2  3
47 26 21 38 19 12
28  8 12 14  5 45
42 32 21 16 18 45
41 17 34  0 19 24
31 27 11 41 45 42
=====
errorPress any key to continue . . .
```

Рисунок 19 – Реализация 2 задания (пункт В и А(взята другая матрица), также работают С и D)

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.



```
C:\Users\spider\Desktop\Programm\Project\Debug\Project.exe
Select application: [1]::[2]::[3]::[4]
-> 4
File is open
=====
Enter the number 1 for write text to file or 2 our text from file: 1
Enter the size of an array: 2
Enter the text: Hello,, Byyyyy
```

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);

```
C:\Users\spider\Desktop\Programm\Project\Debug\Project.exe
Select application: [1]::[2]::[3]::[4]
-> 4
File is open
=====
Enter the number 1 for write text to file or 2 our text from file: 2
Reading data from a fileHello, Byyyyy Press any key to continue . . .
```

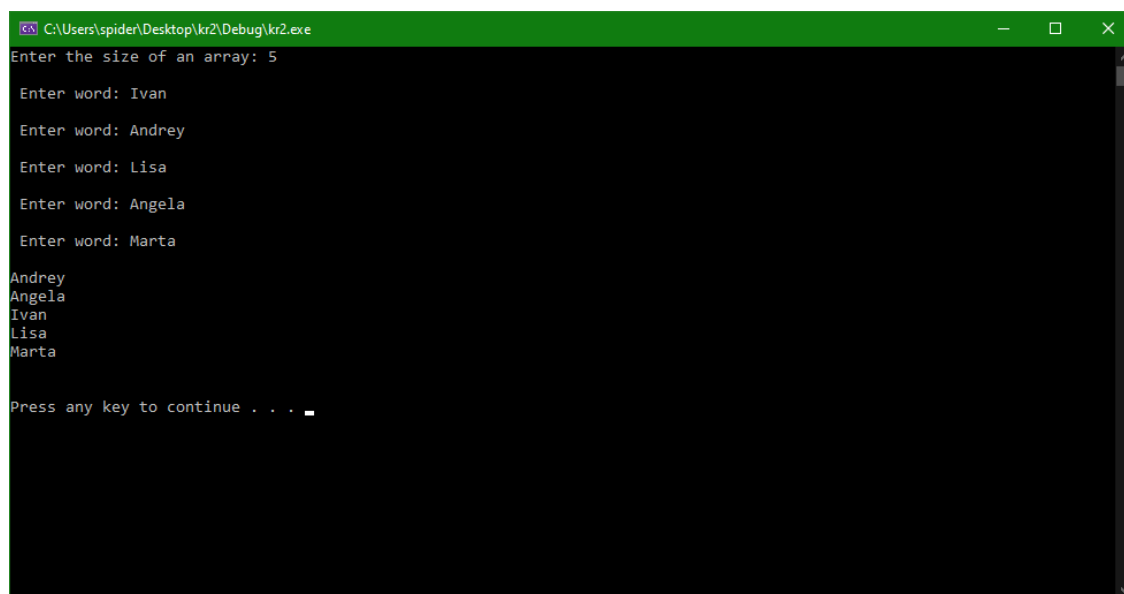
```
Microsoft Visual Studio Debug Console
Select application: [1]::[2]::[3]::[4]
-> 4
File is open
=====
Enter the number 1 for write text to file or 2 our text from file: 1
Enter the size of an array: 5
Enter the text: Hello,, how aRe yoU,, oK
Enter the text: Enter the text: Enter the text: Press any key to continue . . .

C:\Users\spider\Desktop\Programm\Project\Debug\Project.exe (process 388) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

```
C:\Users\spider\Desktop\Programm\Project\Debug\Project.exe
Select application: [1]::[2]::[3]::[4]
-> 4
File is open
=====
Enter the number 1 for write text to file or 2 our text from file: 2
Reading data from a fileHello, Byyyyy Hello, how are you, ok Press any key to continue . . .
```

3. Выполнить задание по варианту: 2

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.



```
C:\Users\spider\Desktop\kr2\Debug\kr2.exe
Enter the size of an array: 5
Enter word: Ivan
Enter word: Andrey
Enter word: Lisa
Enter word: Angela
Enter word: Marta
Andrey
Angela
Ivan
Lisa
Marta
Press any key to continue . . .
```