**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №1**

**по дисциплине «Программирование»**

Тема: Типы данных и их внутреннее представление в памяти

|  |  |  |
| --- | --- | --- |
| Студент(ка) гр. |  | Саевич И. И. 4372 |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2021

**Цель работы.**

Изучение стандартных типов данных; получение практических навыков работы с ними и их внутренним представлением в памяти компьютера;

**Основные теоретические положения.**

C++ — язык со статической типизацией. У каждой переменной на этапе компиляции должен быть чётко определённый тип данных. Про каждый тип данных заранее известно, сколько места в памяти занимает переменная такого типа.

Основные типы данных часто называют арифметическими, поскольку их можно использовать в арифметических операциях. Для описания основных типов определены следующие ключевые слова:

· **int** (целый);

· **float** (вещественный);

· **double** (вещественный тип с двойной точностью);

· **bool** (логический);

· **char** (символьный).

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

· **short** (короткий);

· **long** (длинный);

· **signed** (знаковый);

· **unsigned** (беззнаковый).

Размер типа int не определяется стандартом, а зависит от компьютера и компилятора. Для 16-разрядного процессора под величины этого типа отводится 2 байта, а для 32-разрядного - 4 байта.

Спецификатор short перед именем типа указывает компилятору, что под число требуется отвести 2 байта независимо от разрядности процессора.

Спецификатор long означает, что целая величина будет занимать 4 байта. Таким образом, на 16-разрядном компьютере эквиваленты int и short int, а на 32-разрядном — int и long int.

По умолчанию все целочисленные типы считаются знаковыми, то есть спецификатор signed можно опускать. В знаковых типах старший бит числа интерпретируется как знаковый (О — положительное число, 1 — отрицательное). Для представления отрицательных чисел в С++ используется дополнительный код. Чтобы получить его, надо инвертировать все биты кроме знакового, а затем к младшему биту прибавить единицу.

Спецификатор unsigned позволяет представлять только положительные числа, при этом старший разряд рассматривается как часть кода числа.

Вещественные типы данных хранятся в памяти компьютера иначе, чем целочисленные. Внутреннее представление вещественного числа состоит из двух частей – мантиссы и порядка.

Для 32-разрядного процессора для float под мантиссу отводится 23 бита, под экспоненту – 8, под знак – 1. Для double под мантиссу отводится 52 бита, под экспоненту – 11, под знак – 1. Спецификатор unsigned неприменим к вещественным типам.

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.

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

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

Например:

int ArrInt [10], A1 [20];

double D [100];

char Chars [50];

bool B [200];

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:

A1 [8] = -2000;

cout << A1 [8]; // На экран выведено -2000

В этом примере, обратившись к элементу массива A1 с индексом 8, мы, фактически, обратились к его 9-му элементу.

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 A1 является тип данных int, с любым элементом этого массива можно выполнять любые операции, которые можно выполнять над значениями типа int.

При объявлении массива его можно инициализировать определенными значениями:

short S [5] = {1, 4, 9, 16, 25};

или так:

short S [ ] = {1, 4, 9, 16, 25};

Во втором случае мы не указываем количество элементов массива S. Автоматически создается массив на 5 элементов в соответствии с инициализирующими значениями.

Эти инициализации будут эквивалентны следующим операциям присваивания:

S[0] = 1;

S[1] = 4;

S[2] = 9;

S[3] = 16;

S[4] = 25;

Количество значений, указанных в фигурных скобках (инициализирующих значений) не должно превышать количества элементов в массиве (в нашем примере - 5).

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>

Для предыдущего примера объем массива S будет равен sizeof( short) \* 5 = 2 \* 5 = 10 байтам.

Поскольку все элементы массивов располагаются в памяти один за другим без разрывов, обращение к элементам массива по их индексам (какой бы длины не был этот массив) осуществляется очень эффективно путем вычисления адреса нужного элемента. Пусть, например, адрес памяти, где начинается массив S, равен 100, тогда адрес элемента этого массива с индексом 3 будет равен 100 + sizeof( short) \* 3 = 100 + 2 \* 3 = 106. Обращаемся по этому адресу и считываем 2 байта. Это и будет значением элемента с индексом 3 массива S.

В языке C++ не осуществляется проверка выхода за границы массивов. То есть, вполне корректно (с точки зрения компилятора) будет обращение к элементу массива S, индекс которого равен 10. Это может привести к возникновению весьма серьезных отрицательных последствий. Например, если выполнить присвоение S[10] = 1000 будут изменены данные, находящиеся за пределами массива, а это может быть значение какой-нибудь другой переменной программы. После этого предсказать поведение программы будет невозможно. Единственный выход – быть предельно внимательным при работе с индексами элементов массивов

Указатели и ссылки являются одними из самых важных и достаточно сложных для понимания и использования средств языка программирования. Они ориентированы на прямую работу с памятью компьютера. С помощью этих средств реализуется работа с динамической памятью и динамическими объектами, возвращение из функций измененных данных и многое другое. К использованию указателей и ссылок мы будем неоднократно возвращаться в последующих разделах.

Все данные (переменные, константы и др.) хранятся в памяти. Память представляет собой непрерывную последовательность ячеек (байтов), каждая из которых имеет свой номер – адрес:

При определении, например, некоторой переменной, она располагается в памяти по определенному адресу и занимает столько ячеек, сколько требует тип этой переменной. Пусть, например, имеется переменные int A = 2351 и double B = 3.1 и пусть они располагаются в памяти так:

Говорят, что переменная А располагается по адресу 102 и занимает 4 байта, а переменная B имеет адрес 106 и занимает 8 байт памяти.

Для получения адреса какого-либо программного объекта используется оператор &. Например, если выполнить фрагмент следующей программы (в предположении, что переменные A и B располагаются в памяти, как это показано на предыдущем рисунке):

int A = 2351;

double B = 3.14;

cout << “Значение переменной А: ” << A << endl;

cout << “Адрес переменной А: ” << &A << endl;

cout << “Значение переменной В: ” << В << endl;

cout << “Адрес переменной В: ” << &В << endl;

получим следующий результат:

Значение переменной А: 2351

Адрес переменной А: 102

Значение переменной В: 3.14

Адрес переменной В: 106

Правда, значения адресов переменных будут выведены в шестнадцатеричном формате.

Указатели – это тоже обычные переменные, но они служат для хранения адресов памяти.

Указатели определяются в программе следующим образом:

<тип данных> \*<имя переменной>

Здесь <тип данных> определяет так называемый базовый тип указателя.

<Имя переменной> является идентификатором переменной-указателя.

Признаком того, что это переменная указатель, является символ \*, располагающийся между базовым типом указателя и именем переменной-указателя.

Например:

int \*p1;

double \*p2;

Здесь определены две переменные-указатели (или просто – два указателя). Указатель p1 является переменной-указателем на базовый тип int (или, как говорят, переменная p1 указывает на int - значение), а указатель p2 указывает на double – значение.

Иными словами, переменная p1 предназначена для хранения адресов участков памяти, размер которых соответствует типу int (4 байта), а переменная p2 - для хранения адресов участков памяти, размер которых соответствует типу double (8 байт).

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.

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.

Рассмотрим следующий пример:

int A = 20, B = 30;

int \*p1 = &A;

Пусть переменные A и B расположены в памяти, например, так, как это показано на следующем рисунке:

Указатель p1 содержит адрес переменной A, который равен 100 и \*p1 будет равно значению переменной A, то есть 20. Выполним следующую операцию:

p1 = p1 + 1;

или, что то же самое:

p1++;

Значение указателя изменится и станет равным 104, а не 101, как, наверное, ожидалось. То есть теперь указатель ссылается уже на переменную B и значение \*p1 будет равно 30.

Таким образом, добавление или вычитание 1 из указателя приводит к изменению его значения на размер базового типа указателя. В общем случае, например, при выполнении следующей операции:

p1 = p1 + N; // N – некоторое целое значение

значение указателя увеличится на sizeof(<базовый тип указателя>) \* N и в нашем случае это приращение будет равно sizeof(int) \* N = 4 \* N. Так, если N = 4, а p1 = 100 , то значение указателя p1 увеличится на 16 и станет равно 116, и указатель будет ссылаться на данные, расположенные по адресу 116.

Внимание. Добавлять к указателям или вычитать из указателей можно только целые значения.

Поскольку упомянутые арифметические операции выполняются по-разному при их применении к указателям и обычным арифметическим типам данных, а также учитывая высший приоритет операции \*, при использовании указателей в составе выражений следует внимательно обращаться со скобками. Например, выражения (см. предыдущий рисунок)

\*(p1 + 1) и \*p1 + 1

имеют совершенно разный смысл. Первое выражение даст значение 30, а второе выражение будет равно 21 (в первом выражении сначала изменяется адрес, а затем осуществляется обращение в память по этому измененному адресу; во втором выражении мы обращаемся по старому адресу и к значению, хранящемуся по этому адресу добавляем 1).

В изучаемых нами языках программирования между массивами и указателями имеется очень тесная связь.

Кода мы определяем в программе некоторый массив, например,

int Arr[10];

переменная Arr без индексов представляет собой указатель на первый элемент массива в данном случае из 10 целых чисел (содержит адрес первого элемента массива). Если вывести на экран значение переменной Arr

cout << Arr:

мы увидим некоторое целое значение в шестнадцатеричном формате, соответствующее адресу первого элемента этого массива.

Замечание. Именно по этой причине в языке C++ отсутствует операция присвоения сразу всех значений одного массива другому (в некоторых других языках, например, в Pascal такая возможность имеется). Действительно, если имеются два массива

int A1[10], A2[10];

то попытка выполнить присвоение A1 = A2 привела бы к тому, что переменная A1 стала бы указывать на ту же область памяти, что и переменная A2 (мы скопировали адрес из A2 в A1, а не содержимое одного массива в другой). Адрес, который хранился ранее в переменной A1, был бы утерян, что привело бы к утечке памяти (для десяти элементов массива A1 в памяти было выделено место, но теперь мы “забыли”, где оно находится, то есть потеряли память). По этой причине подобные операции с массивами в языке C++ запрещены. Более того, запрещены любые изменения значения переменной массива.

Указателю, имеющему такой же базовый тип, как и элементы массива, можно присвоить массив следующим образом:

int Arr[10];

int \*p;

p = Arr;

Но обратное присвоение выполнить невозможно:

Arr = p; // Ошибка

Такое присвоение невозможно, поскольку переменная массива – это константа, изменение которой запрещено.

Так как переменная массива является указателем на первый элемент массива, появляются дополнительные возможности по работе с массивами на основе использования арифметики указателей. Например, чтобы получить 5–й элемент массива Arr можно воспользоваться одним из следующих выражений:

Arr[4] или \*(Arr + 4) или \*( p + 4)

Первое выражение – это пример обычной индексации элементов массива. Во втором и третьем выражениях мы использовали арифметику указателей и с помощью операции + получили адрес пятого элемента массива. Затем с помощью операции \* взяли значение по этому адресу и получили значение 5-го элемента массива. Обратите внимание на скобки в этих выражениях, если их не поставить и написать \*Arr + 4 или \*p + 4, то эти выражения будут равны значению первого элемента массива увеличенного на 4, так как операция \* имеет больший приоритет, чем операция +.

Вот пример фрагмента программы для работы с массивом с помощью обычной индексации элементов массива. Этот фрагмент обеспечивает ввод элементов целочисленного массива с клавиатуры, вычисление квадратов значений элементов массива, а затем вывод элементов массива на экран:

int A[10];

for (int i = 0; i < 10; ++ i)

{

cin >> A[i];

A[i] = A[i] \* A[i];

}

for (int i = 0; i < 10; ++ i)

cout << A[i] << “ “;

cout << endl;

…..

А вот тот же фрагмент, но с использованием арифметики указателей:

int A[10];

for (int \*Next = A, \*End = Next + 9; Next <= End; ++ Next)

{

cin >> \*Next;

\*Next = \*Next \* \*Next; // \*Next = (\*Next) \* (\*Next);

}

for (int \*Next = A, \*End = Next + 9; Next <= End; ++ Next)

cout << \*Next << “ “;

cout << endl;

…

Использование арифметики указателей при работе с массивами приводит обычно к уменьшению объема генерируемого кода программы и к уменьшению времени ее выполнения, то есть к увеличению быстродействия.

Поскольку указатель и имя массива, в большой степени, взаимозаменяемы, указатели можно индексировать, как обычные массивы:

int A[10], \*P = A;

for (int i = 0; i < 10; ++ i)

cout << P[i] << “ “;

Можно создавать и массивы указателей. Например:

int a = 1, b = 2, c = 3, \*M[3];

M[0] = & a; // Элементам массива М присваиваются адреса переменных a, b и c

M[1] = & b;

M[2] = & c;

for (int i = 0; i < 3; ++ i)

cout << \*M[i] << “ ”;

cout << endl;

Массив M – это трехэлементный массив указателей на целые значения, то есть каждый элемент этого массива представляет собой указатель на целое.

С помощью массивов указателей можно моделировать различные интересные конструкции данных. Например, пусть имеется квадратная матрица размерности 5 х 5 симметричная относительно главной диагонали. Для ее однозначного представления достаточно хранить в памяти не все 25 элементов этой матрицы, а только 15 (например, элементы под главной диагональю вместе с элементами главной диагонали). Для этого можно предложить следующую конструкцию:

int A1[1], A2[2], A3[3], A4[4], A5[5], \*A[5] = { A1, A2, A3, A4, A5 };

// Вводим 15 целых значений - элементы под главной диагональю и диагональные

// элементы матрицы

for (int i = 0; i < 5; ++i)

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

cin >> A[i][j];

cout << endl;

// Выводим симметричную матрицу 5 на 5 на экран

for (int i = 0; i < 5; ++i)

{

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

cout << A[i][j] << " ";

for (int j = i + 1; j < 5; ++ j)

cout << A[j][i] << " ";

cout << endl;

}

cout << endl;

А это пятиэлементный массив указателей на символы, инициализированный некоторыми текстовыми строками:

char \* Words[5] = { "Слово1", "Слово2", "Слово3", "Слово4", "Слово5" }

Как это работает: когда компилятор встречает в программе некоторый текст, заключенный в кавычки, в памяти создается символьный массив соответствующей этому тексту длины и адрес этого символьного массива присваивается соответствующему элементу – указателю массива Words.

Текстовые строки представляются с помощью одномерных массивов символов. В языке C++ текстовая строка представляет собой набор символов, обязательно заканчивающийся нулевым символом (‘\0’). Поэтому, если вы хотите создать текстовый массив для хранения 10 (N) символов, нужно выделить память под 11(N+1) символов.

Объявленный таким образом массив может использоваться для хранения текстовых строк, содержащих не более 10 символов. Нулевой символ позволяет определить границу между содержащимся в строке текстом и неиспользованной частью строки.

При определении строковых переменных их можно инициализировать конкретными значениями с помощью строковых литералов:

char S1[15] = “This is text”;

char S2[] = “Пример текста”;

Последние два элемента переменной просто не используются, а строка автоматически подстраивается под длину инициализирующего текста.

При работе со строками можно обращаться к отдельным символам строки как в обычном одномерном массиве с помощью индексов:

cout << S1[0]; // На экране будет выведен символ ‘T’

Если строка формируется при помощи цикла (или иного способа), то необходимо в ее конец обязательно записать нулевой символ '\0'.

**Постановка задачи.**

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

**Выполнение работы.**

Код программы представлен в приложении A.

**Выводы.**

В ходе работы мы смогли объединить 4 практические работы в единый проект.

Приложение А

рабочий код

﻿#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <Windows.h>

#include <fstream>

#include <cstring>

#include <cctype>

#include <vector>

#include <chrono>

using namespace std;

const int maxWords = 50;

const int maxLength = 11;

const int maxInputSize = 1000;

void InputCheck(int& number)

{

while (cin.fail())

{

cin.clear();

cin.sync();

cin.ignore(1000, '\n');

cout << "Введите ЧИСЛО" << endl;

cin >> number;

}

}

void RemoveExtraSpaces(char\* str)

{

int n = strlen(str);

int j = 0;

int count = 0;

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

{

if (str[i] != ' ' || (i > 0 && str[i - 1] != ' '))

{

str[j] = str[i];

j++;

}

}

str[j] = '\0';

}

void RemoveExtraPunctuation(char\* str)

{

int n = strlen(str);

int j = 0;

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

{

if (!ispunct(str[i]) || (i > 0 && !ispunct(str[i - 1])))

{

str[j++] = str[i];

}

}

str[j] = '\0';

}

void FixLetterCase(char\* str)

{

int n = strlen(str);

for (int i = 1; i < n; i++)

{

if (str[i - 1] != ' ' && isupper(str[i]))

str[i] = tolower(str[i]);

}

}

void EditText(char\* text)

{

RemoveExtraSpaces(text);

RemoveExtraPunctuation(text);

FixLetterCase(text);

}

int TextToWords(const char\* text, char(&words)[maxWords][maxLength])

{

int wordCount = 0;

char text1[maxInputSize];

strcpy(text1, text);

char\* token = strtok(text1, " .,:;-?!()[]{}<>|/+=\*&^%$#@~`№");

while (token != nullptr && wordCount < maxWords)

{

strcpy(words[wordCount], token);

wordCount++;

token = strtok(nullptr, " .,:;-?!()[]{}<>|/+=\*&^%$#@~`№");

}

return wordCount;

}

void ChangeRegister(char(&words)[maxWords][maxLength], int count)

{

for (int i = 0; i < count; i++)

words[i][0] = toupper(words[i][0]);

}

int LinearSearch(char\* text, char\* pattern)

{

int n = strlen(text);

int m = strlen(pattern);

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

int j;

for (j = 0; j < m; j++) {

if (text[i + j] != pattern[j])

break;

}

if (j == m)

return i;

}

return -1;

}

vector<int> Suffixes(const char\* pattern) {

int length = strlen(pattern);

vector<int> suffix(length, 0);

int g, f;

g = f = length - 1;

for (int i = length - 2; i >= 0; --i)

{

if (i > g && suffix[i + length - 1 - f] < i - g)

suffix[i] = suffix[i + length - 1 - f];

else

{

if (i < g)

g = i;

f = i;

while (g >= 0 && pattern[g] == pattern[g + length - 1 - f])

--g;

suffix[i] = f - g;

}

}

return suffix;

}

vector<int> boyerMoore(const char\* text, const char\* pattern) {

int n = strlen(text);

int m = strlen(pattern);

std::vector<int> occurrences;

std::vector<int> shift(256, m);

for (int i = 0; i < m - 1; ++i) {

shift[pattern[i]] = m - i - 1;

}

std::vector<int> suffix = Suffixes(pattern);

int i = 0;

while (i <= n - m) {

int j = m - 1;

while (j >= 0 && pattern[j] == text[i + j]) {

--j;

}

if (j < 0) {

occurrences.push\_back(i);

i += shift[text[i + m]];

}

else {

int x = suffix[j];

int y = j - x;

int z = m - 1 - j;

i += max(shift[text[i + j]] - y, z);

}

}

return occurrences;

}

COORD GetConsoleCursorPosition(HANDLE hConsoleOutput)

{

CONSOLE\_SCREEN\_BUFFER\_INFO cbsi;

if (GetConsoleScreenBufferInfo(hConsoleOutput, &cbsi))

{

return cbsi.dwCursorPosition;

}

else

{

COORD invalid = { 0, 0 };

return invalid;

}

}

void FillMatrixA(int\* matrix, int N, const int& delay, HANDLE hStdout, COORD destCoord) {

COORD current = GetConsoleCursorPosition(hStdout);

int Q = current.Y;

int num;

for (int j = 0; j < N; ++j)

{

destCoord.X = j \* 3 + 1;

if (j % 2 == 0)

{

for (int i = 0; i < N; ++i)

{

num = rand() % (N \* N + 1);

\*(matrix + i \* N + j) = num;

destCoord.Y = i + Q;

SetConsoleCursorPosition(hStdout, destCoord);

cout << num << '\r';

cout.flush();

Sleep(delay);

}

}

else

{

for (int i = N - 1; i >= 0; --i)

{

num = rand() % (N \* N + 1);

\*(matrix + i \* N + j) = num;

destCoord.Y = i + Q;

SetConsoleCursorPosition(hStdout, destCoord);

cout << num << '\r';

cout.flush();

Sleep(delay);

}

}

}

destCoord.X = 0;

destCoord.Y = Q + N;

SetConsoleCursorPosition(hStdout, destCoord);

}

void FillMatrixB(int\* matrix, int N, const int& delay, HANDLE hStdout, COORD destCoord) {

COORD current = GetConsoleCursorPosition(hStdout);

int Q = current.Y;

int num, k = 1;

int top = 0, bottom = N - 1, left = 0, right = N - 1;

while (k <= N \* N)

{

destCoord.Y = top + Q;

for (int j = left; j <= right; ++j)

{

num = rand() % (N \* N + 1);

\*(matrix + top \* N + j) = num;

destCoord.X = j \* 3 + 1;;

SetConsoleCursorPosition(hStdout, destCoord);

cout << num << '\r';

cout.flush();

Sleep(delay);

k++;

}

top++;

destCoord.X = right \* 3 + 1;

for (int i = top; i <= bottom; ++i)

{

num = rand() % (N \* N + 1);

\*(matrix + i \* N + right) = num;

destCoord.Y = i + Q;

SetConsoleCursorPosition(hStdout, destCoord);

cout << num << '\r';

cout.flush();

Sleep(delay);

k++;

}

right--;

destCoord.Y = bottom + Q;

if (top <= bottom)

{

for (int j = right; j >= left; --j)

{

num = rand() % (N \* N + 1);

\*(matrix + bottom \* N + j) = num;

destCoord.X = j \* 3 + 1;

SetConsoleCursorPosition(hStdout, destCoord);

cout << num << '\r';

cout.flush();

Sleep(delay);

k++;

}

bottom--;

}

destCoord.X = left \* 3 + 1;

if (left <= right)

{

for (int i = bottom; i >= top; --i)

{

num = rand() % (N \* N + 1);

\*(matrix + i \* N + left) = num;

destCoord.Y = i + Q;

SetConsoleCursorPosition(hStdout, destCoord);

cout << num << '\r';

cout.flush();

Sleep(delay);

k++;

}

left++;

}

}

destCoord.X = 0;

destCoord.Y = N + Q;

SetConsoleCursorPosition(hStdout, destCoord);

}

void ReplaceBlocksA(int\* matrix, int\* newMatrix, int N)

{

for (int i = 0; i < N \* N; ++i)

{

if ((i + 1) % N <= N / 2 && (i + 1) % N != 0 && i + 1 <= N \* N / 2)

{

\*(newMatrix + i + N / 2) = \*(matrix + i);

}

else if (((i + 1) % N > N / 2 || (i + 1) % N == 0) && i + 1 <= N \* N / 2)

{

\*(newMatrix + i + N \* N / 2) = \*(matrix + i);

}

else if ((i + 1) % N <= N / 2 && (i + 1) % N != 0 && i + 1 > N \* N / 2)

{

\*(newMatrix + i - N \* N / 2) = \*(matrix + i);

}

else if (((i + 1) % N > N / 2 || (i + 1) % N == 0) && i + 1 > N \* N / 2)

{

\*(newMatrix + i - N / 2) = \*(matrix + i);

}

}

}

void ReplaceBlocksB(int\* matrix, int\* newMatrix, int N)

{

for (int i = 0; i < N \* N; ++i)

{

if ((i + 1) % N <= N / 2 && (i + 1) % N != 0 && i + 1 <= N \* N / 2)

{

\*(newMatrix + i + N / 2 + N \* N / 2) = \*(matrix + i);

}

else if (((i + 1) % N > N / 2 || (i + 1) % N == 0) && i + 1 <= N \* N / 2)

{

\*(newMatrix + i + N \* N / 2 - N / 2) = \*(matrix + i);

}

else if ((i + 1) % N <= N / 2 && (i + 1) % N != 0 && i + 1 > N \* N / 2)

{

\*(newMatrix + i + N / 2 - N \* N / 2) = \*(matrix + i);

}

else if (((i + 1) % N > N / 2 || (i + 1) % N == 0) && i + 1 > N \* N / 2)

{

\*(newMatrix + i - N / 2 - N \* N / 2) = \*(matrix + i);

}

}

}

void ReplaceBlocksC(int\* matrix, int\* newMatrix, int N)

{

for (int i = 0; i < N \* N; ++i)

{

if ((i + 1) % N <= N / 2 && (i + 1) % N != 0 && i + 1 <= N \* N / 2)

{

\*(newMatrix + i + N \* N / 2) = \*(matrix + i);

}

else if (((i + 1) % N > N / 2 || (i + 1) % N == 0) && i + 1 <= N \* N / 2)

{

\*(newMatrix + i + N \* N / 2) = \*(matrix + i);

}

else if ((i + 1) % N <= N / 2 && (i + 1) % N != 0 && i + 1 > N \* N / 2)

{

\*(newMatrix + i - N \* N / 2) = \*(matrix + i);

}

else if (((i + 1) % N > N / 2 || (i + 1) % N == 0) && i + 1 > N \* N / 2)

{

\*(newMatrix + i - N \* N / 2) = \*(matrix + i);

}

}

}

void ReplaceBlocksD(int\* matrix, int\* newMatrix, int N)

{

for (int i = 0; i < N \* N; ++i)

{

if ((i + 1) % N <= N / 2 && (i + 1) % N != 0 && i + 1 <= N \* N / 2)

{

\*(newMatrix + i + N / 2) = \*(matrix + i);

}

else if (((i + 1) % N > N / 2 || (i + 1) % N == 0) && i + 1 <= N \* N / 2)

{

\*(newMatrix + i - N / 2) = \*(matrix + i);

}

else if ((i + 1) % N <= N / 2 && (i + 1) % N != 0 && i + 1 > N \* N / 2)

{

\*(newMatrix + i + N / 2) = \*(matrix + i);

}

else if (((i + 1) % N > N / 2 || (i + 1) % N == 0) && i + 1 > N \* N / 2)

{

\*(newMatrix + i - N / 2) = \*(matrix + i);

}

}

}

void InsertSortMatrix(int\* matrix, int N)

{

int key, j;

for (int i = 1; i != N; ++i)

{

key = \*(matrix + i);

j = i - 1;

while (j >= 0 && key < \*(matrix + j))

{

\*(matrix + j + 1) = \*(matrix + j);

j -= 1;

}

\*(matrix + j + 1) = key;

}

}

void Multiply(int\* matrix, int N, int X)

{

for (int i = 0; i < N \* N; ++i)

\*(matrix + i) \*= X;

}

void Divide(int\* matrix, int N, int X)

{

for (int i = 0; i < N \* N; ++i)

\*(matrix + i) /= X;

}

void Add(int\* matrix, int N, int X)

{

for (int i = 0; i < N \* N; ++i)

\*(matrix + i) += X;

}

void Subtract(int\* matrix, int N, int X)

{

for (int i = 0; i < N \* N; ++i)

\*(matrix + i) -= X;

}

void BubbleSort(int Array[], int N)

{

bool swapped = true;

int j = 0;

while (swapped)

{

swapped = false;

for (int i = 0; i != N - j - 1; ++i)

{

if (Array[i] > Array[i + 1])

{

swap(Array[i], Array[i + 1]);

swapped = true;

}

}

j++;

}

}

void ShakerSort(int Array[], int N)

{

int start = 0, end = N - 1;

bool swapped = true;

while (swapped)

{

swapped = false;

for (int i = start; i != end; ++i)

{

if (Array[i] > Array[i + 1])

{

swap(Array[i], Array[i + 1]);

swapped = true;

}

}

if (!swapped)

break;

swapped = false;

--end;

for (int i = end; i != start; --i)

{

if (Array[i] < Array[i - 1])

{

std::swap(Array[i], Array[i - 1]);

swapped = true;

}

}

++start;

}

}

void CombSort(int Array[], int N)

{

bool swapped = true;

int j = 0;

float S = N - 1, k = 1.247;

while (S >= 1)

{

for (int i = 0; i + S < N; ++i)

{

if (Array[i] > Array[int(i + S)])

swap(Array[i], Array[int(i + S)]);

}

S /= k;

}

while (swapped)

{

swapped = false;

for (int i = 0; i != N - j - 1; ++i)

{

if (Array[i] > Array[i + 1])

{

swap(Array[i], Array[i + 1]);

swapped = true;

}

}

j++;

}

}

void InsertSort(int Array[], int N)

{

int key, j;

for (int i = 1; i != N; ++i)

{

key = Array[i];

j = i - 1;

while (j >= 0 && key < Array[j])

{

Array[j + 1] = Array[j];

j -= 1;

}

Array[j + 1] = key;

}

}

void QuickSort(int Array[], int start, int end)

{

int mid, l = start, r = end;

mid = Array[(l + r) / 2];

while (l < r)

{

while (Array[l] < mid)

l++;

while (Array[r] > mid)

r--;

if (l <= r)

{

swap(Array[l], Array[r]);

l++;

r--;

}

}

if (start < r)

QuickSort(Array, start, r);

if (end > l)

QuickSort(Array, l, end);

}

int BinarySearch(int Array[], int value, int start, int end)

{

if (end >= start)

{

int mid = start + (end - start) / 2;

if (Array[mid] == value)

return mid;

if (Array[mid] > value)

return BinarySearch(Array, value, start, mid - 1);

return BinarySearch(Array, value, mid + 1, end);

}

return -1;

}

int BinarySearchFirstOccurrence(int Array[], int value, int start, int end)

{

if (end >= start)

{

int mid = start + (end - start) / 2;

if (Array[mid] == value)

{

while (Array[mid - 1] == value && mid - 1 >= 0)

mid -= 1;

return mid;

}

if (Array[mid] > value)

return BinarySearchFirstOccurrence(Array, value, start, mid - 1);

return BinarySearchFirstOccurrence(Array, value, mid + 1, end);

}

return start;

}

int BinarySearchLastOccurrence(int Array[], int value, int start, int end)

{

int mid = start + (end - start) / 2;

if (end >= start)

{

if (Array[mid] == value)

{

while (Array[mid + 1] == value && mid + 1 <= end)

mid += 1;

return mid;

}

if (Array[mid] > value)

return BinarySearchLastOccurrence(Array, value, start, mid - 1);

return BinarySearchLastOccurrence(Array, value, mid + 1, end);

}

return start - 1;

}

int ElemSearch(int Array[], int value, int N)

{

int X = -1;

for (int i = 0; i < N; ++i)

{

if (Array[i] == value)

{

X = value;

break;

}

}

return X;

}

int MinSearch(int Array[], int N)

{

int min = 101;

for (int i = 0; i < N; ++i)

min = Array[i] < min ? Array[i] : min;

return min;

}

int MaxSearch(int Array[], int N)

{

int max = 0;

for (int i = 0; i < N; ++i)

max = Array[i] > max ? Array[i] : max;

return max;

}

void First()

{

cout << "int: " << sizeof(int) << endl;

cout << "short int: " << sizeof(short int) << endl;

cout << "long int: " << sizeof(long int) << endl;

cout << "long long int: " << sizeof(long long int) << endl;

cout << "float: " << sizeof(float) << endl;

cout << "double: " << sizeof(double) << endl;

cout << "long double: " << sizeof(long double) << endl;

cout << "char: " << sizeof(char) << endl;

cout << "bool: " << sizeof(bool) << endl;

}

void Second()

{

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

int color = 9;

int A;

int order = sizeof(int) \* 8;

unsigned int mask = 1 << order - 1;

cout << "Введите число типа int" << endl;

cin >> A;

while (cin.fail())

{

cin.clear();

cin.ignore(1000, '\n');

cout << "Введите ЧИСЛО" << endl;

cin >> A;

}

for (int i = 1; i <= order; ++i)

{

SetConsoleTextAttribute(hConsole, color);

putchar(mask & A ? '1' : '0');

mask >>= 1;

if (i % 8 == 0 || i == 1)

{

putchar(' ');

color++;

}

}

SetConsoleTextAttribute(hConsole, 7);

cout << endl;

}

void Third()

{

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

int color = 11;

union

{

int tool;

float F;

};

int order = sizeof(float) \* 8;

unsigned int mask = 1 << order - 1;

cout << "Введите число типа float" << endl;

cin >> F;

while (cin.fail())

{

cin.clear();

cin.ignore(1000, '\n');

cout << "Введите ЧИСЛО" << endl;

cin >> F;

}

for (int i = 1; i <= order; ++i)

{

SetConsoleTextAttribute(hConsole, color);

putchar(mask & tool ? '1' : '0');

mask >>= 1;

if (i == 9 || i == 1)

{

putchar(' ');

color++;

}

}

SetConsoleTextAttribute(hConsole, 7);

cout << endl;

}

void Fourth()

{

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

int color = 11;

union {

int tool[2];

double D;

};

int order = sizeof(D) \* 8;

unsigned int mask = 1;

mask <<= order - 1;

cout << "Введите число типа double" << endl;

cin >> D;

while (cin.fail())

{

cin.clear();

cin.ignore(1000, '\n');

cout << "Введите ЧИСЛО" << endl;

cin >> D;

}

for (int i = 1; i >= 0; --i)

{

for (int j = 0; j < order / 2; ++j)

{

SetConsoleTextAttribute(hConsole, color);

putchar(mask & tool[i] ? '1' : '0');

mask >>= 1;

if ((i == 1 && j == 0) || (i == 1 && j == 11))

{

putchar(' ');

color++;

}

}

mask = 1 << order - 1;

}

SetConsoleTextAttribute(hConsole, 7);

cout << endl;

}

void PracticalWork1()

{

setlocale(0, "");

int number;

cout << "Введите номер задания: ";

cin >> number;

switch (number)

{

case 1:

First();

break;

case 2:

Second();

break;

case 3:

Third();

break;

case 4:

Fourth();

break;

case 666:

return;

default:

cout << "Задания с таким номером нет" << endl;

}

}

void PracticalWork2()

{

srand(time(0));

setlocale(0, "");

const int N = 100;

int Array[N], sortedArray[N], select = 0;

bool isSorted = false;

chrono::nanoseconds duration;

chrono::steady\_clock::time\_point start, end;

cout << "Неотсортированный массив:" << endl;

for (int i = 0; i < N; ++i)

{

Array[i] = -99.5 + rand() % 200;

cout << Array[i] << " ";

}

cout << endl;

cout << "1 - создать новый неотсортированный массив\n" <<

"2 - отсортировать массив\n" <<

"3 - найти максимальный и минимальный элементы массива\n" <<

"4 - найти среднее значение максимального и минимального элементов, а также вывести индексы элементов, равных этому и значению и их количество\n" <<

"5 - найти количество элементов в отсортированном массиве, которые меньше числа A\n" <<

"6 - найти количество элементов в отсортированном массиве, которые больше числа B\n" <<

"7 - вывести информацию о том, есть ли число X в массиве\n" <<

"8 - поменять местами два элемента\n" <<

"9 - показать текущее состояние массивов" << endl;

while (true)

{

cin >> select;

InputCheck(select);

switch (select)

{

case 1:

cout << "Неотсортированный массив:" << endl;

for (int i = 0; i < N; ++i)

{

Array[i] = -99.5 + rand() % 200;

cout << Array[i] << " ";

}

cout << endl;

isSorted = false;

break;

case 2:

{

isSorted = false;

cout << "1 - Bubble sort\n2 - Shaker sort\n3 - Comb sort\n4 - Insert sort\n5 - Quick sort" << endl;

for (int i = 0; i < N; ++i)

sortedArray[i] = Array[i];

cout << "Введите номер алгоритма" << endl;

int numberOfAlgorithm;

cin >> numberOfAlgorithm;

InputCheck(numberOfAlgorithm);

while (!isSorted)

{

switch (numberOfAlgorithm)

{

case 1:

isSorted = true;

start = chrono::steady\_clock::now();

BubbleSort(sortedArray, N);

end = chrono::steady\_clock::now();

duration = chrono::duration\_cast<chrono::nanoseconds>(end - start);

break;

case 2:

isSorted = true;

start = chrono::steady\_clock::now();

ShakerSort(sortedArray, N);

end = chrono::steady\_clock::now();

duration = chrono::duration\_cast<chrono::nanoseconds>(end - start);

break;

case 3:

isSorted = true;

start = chrono::steady\_clock::now();

CombSort(sortedArray, N);

end = chrono::steady\_clock::now();

duration = chrono::duration\_cast<chrono::nanoseconds>(end - start);

break;

case 4:

isSorted = true;

start = chrono::steady\_clock::now();

InsertSort(sortedArray, N);

end = chrono::steady\_clock::now();

duration = chrono::duration\_cast<chrono::nanoseconds>(end - start);

break;

case 5:

isSorted = true;

start = chrono::steady\_clock::now();

QuickSort(sortedArray, 0, N - 1);

end = chrono::steady\_clock::now();

duration = chrono::duration\_cast<chrono::nanoseconds>(end - start);

break;

default:

cout << "Алгоритма с такими номером нет. Введите другое число." << endl;

}

}

if (isSorted)

{

cout << "Отсортированный массив:" << endl;

for (int i = 0; i != N; ++i)

cout << sortedArray[i] << " ";

cout << endl << "Алгоритм занял " << duration.count() << " нс" << endl;

}

break;

}

case 3:

{

if (isSorted)

{

start = chrono::steady\_clock::now();

int min = MinSearch(Array, N);

end = chrono::steady\_clock::now();

duration = chrono::duration\_cast<chrono::nanoseconds>(end - start);

cout << "Поиск минимального элемента в неотсортированном массиве занял " << duration.count() << " нс" << endl;

start = chrono::steady\_clock::now();

int max = MaxSearch(Array, N);

end = chrono::steady\_clock::now();

duration = chrono::duration\_cast<chrono::nanoseconds>(end - start);

cout << "Поиск максимального элемента в неотсортированном массиве занял " << duration.count() << " нс" << endl;

start = chrono::steady\_clock::now();

min = sortedArray[0];

end = chrono::steady\_clock::now();

duration = chrono::duration\_cast<chrono::nanoseconds>(end - start);

cout << "Поиск минимального элемента в отсортированном массиве занял " << duration.count() << " нс" << endl;

start = chrono::steady\_clock::now();

max = sortedArray[N - 1];

end = chrono::steady\_clock::now();

duration = chrono::duration\_cast<chrono::nanoseconds>(end - start);

cout << "Поиск максимального элемента в отсортированном массиве занял " << duration.count() << " нс" << endl;

cout << "Максимальный элемент: " << max << "\nМинимальный элемент: " << min << endl;

}

else

cout << "Сначала надо отсортировать массив" << endl;

break;

}

case 4:

{

if (isSorted)

{

int number = 0, mid;

double dMid = static\_cast<double>((sortedArray[N - 1] + sortedArray[0])) / 2;

cout << "Точное среднее значение: " << dMid << endl;

if (dMid > 0)

mid = dMid + 0.5;

else

mid = dMid - 0.5;

cout << "Округленное среднее значение: " << mid << endl;

cout << "Индексы элементов, равных этому значению в неотсортированном массиве: ";

start = chrono::steady\_clock::now();

for (int i = 0; i < N; ++i)

{

if (Array[i] == mid)

{

cout << i << " ";

number++;

}

}

end = chrono::steady\_clock::now();

if (!number)

cout << "-" << endl;

else cout << endl;

duration = chrono::duration\_cast<chrono::nanoseconds>(end - start);

cout << "Время поиска: " << duration.count() << " нс" << endl;

number = 0;

cout << "Индексы элементов, равных этому значению в отсортированном массиве: ";

int j = 0;

start = chrono::steady\_clock::now();

while (sortedArray[j] <= mid)

{

if (sortedArray[j] == mid)

{

cout << j << " ";

number++;

}

j++;

}

end = chrono::steady\_clock::now();

if (!number)

cout << "-" << endl;

else cout << endl;

duration = chrono::duration\_cast<chrono::nanoseconds>(end - start);

cout << "Время поиска: " << duration.count() << " нс" << endl;

cout << "Количество таких элементов: " << number << endl;

}

else

cout << "Сначала надо отсортировать массив" << endl;

break;

}

case 5:

{

if (isSorted)

{

int A;

cout << "Введите число a" << endl;

cin >> A;

InputCheck(A);

int indexA = BinarySearchFirstOccurrence(sortedArray, A, 0, N - 1);

cout << "Количество элементов, которые меньше a: " << indexA << endl;

}

else

cout << "Сначала надо отсортировать массив" << endl;

break;

}

case 6:

{

if (isSorted)

{

int B;

cout << "Введите число B" << endl;

cin >> B;

InputCheck(B);

int indexB = BinarySearchLastOccurrence(sortedArray, B, 0, N - 1);

cout << "Количество элементов, которые больше b: " << N - indexB - 1 << endl;

}

else

cout << "Сначала надо отсортировать массив" << endl;

break;

}

case 7:

{

if (isSorted)

{

int X;

cout << "Введите число X" << endl;

cin >> X;

InputCheck(X);

start = chrono::steady\_clock::now();

int xInArray = BinarySearch(sortedArray, X, 0, N - 1);

end = chrono::steady\_clock::now();

duration = chrono::duration\_cast<chrono::nanoseconds>(end - start);

if (xInArray == -1)

cout << "Числа X нет в массиве" << endl;

else

cout << "Число X есть в массиве" << endl;

cout << "Алгоритм бинарного поиска занял " << duration.count() << " нс" << endl;

start = chrono::steady\_clock::now();

xInArray = ElemSearch(Array, X, N);

end = chrono::steady\_clock::now();

duration = chrono::duration\_cast<chrono::nanoseconds>(end - start);

cout << "Алгоритм перебора занял " << duration.count() << " нс" << endl;

}

else

cout << "Сначала надо отсортировать массив" << endl;

break;

}

case 8:

{

int p, q;

cout << "Введите индекс первого элемента" << endl;

cin >> p;

InputCheck(p);

cout << "Введите индекс второго элемента" << endl;

cin >> q;

InputCheck(q);

start = chrono::steady\_clock::now();

swap(sortedArray[p], sortedArray[q]);

end = chrono::steady\_clock::now();

duration = chrono::duration\_cast<chrono::nanoseconds>(end - start);

for (int i = 0; i < N; ++i)

cout << sortedArray[i] << " ";

cout << endl << "Обмен занял " << duration.count() << " нс" << endl;

if (p != q)

isSorted = false;

break;

}

case 9:

cout << "Неотсортированный массив:" << endl;

for (int i = 0; i < N; ++i)

cout << Array[i] << " ";

cout << endl;

cout << "Отсортированный массив:" << endl;

for (int i = 0; i < N; ++i)

cout << sortedArray[i] << " ";

cout << endl;

break;

case 666:

return;

default:

cout << "Нет команды с таким номером" << endl;

}

}

}

void PracticalWork3() {

setlocale(0, "");

srand(time(0));

HANDLE hStdout;

COORD destCoord;

destCoord.X = 0;

destCoord.Y = 0;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

int N, delay = 10;

cout << "Введите порядок матрицы (6, 8 или 10): ";

cin >> N;

InputCheck(N);

int newMatrix6[6][6];

int matrix6[6][6];

int newMatrix8[8][8];

int matrix8[8][8];

int newMatrix10[10][10];

int matrix10[10][10];

int\* matrix;

int\* newMatrix;

switch (N)

{

case 6:

matrix = \*matrix6;

newMatrix = \*newMatrix6;

break;

case 8:

matrix = \*matrix8;

newMatrix = \*newMatrix8;

break;

case 10:

matrix = \*matrix10;

newMatrix = \*newMatrix10;

break;

default:

cout << "Порядок матрицы должен быть 6, 8 или 10." << endl;

return;

}

int choice;

cout << "Введите номер схемы заполнения матрицы: ";

cin >> choice;

InputCheck(choice);

switch (choice)

{

case 1:

FillMatrixA(matrix, N, delay, hStdout, destCoord);

break;

case 2:

FillMatrixB(matrix, N, delay, hStdout, destCoord);

break;

default:

cout << "Нет такого варианта" << endl;

}

cout << "Введите номер схемы перестановки блоков матрицы: ";

cin >> choice;

InputCheck(choice);

switch (choice)

{

case 1:

ReplaceBlocksA(matrix, newMatrix, N);

break;

case 2:

ReplaceBlocksB(matrix, newMatrix, N);

break;

case 3:

ReplaceBlocksC(matrix, newMatrix, N);

break;

case 4:

ReplaceBlocksD(matrix, newMatrix, N);

break;

default:

cout << "Нет такого варианта" << endl;

}

int i = 0;

for (int\* ptr = newMatrix; ptr < newMatrix + N \* N; ++ptr)

{

cout << \*ptr << " ";

if ((i + 1) % N == 0)

cout << "\n";

i++;

}

InsertSortMatrix(newMatrix, N \* N);

cout << "Отсортированная матрица: " << endl;

i = 0;

for (int\* ptr = newMatrix; ptr < newMatrix + N \* N; ++ptr)

{

cout << \*ptr << " ";

if ((i + 1) % N == 0)

cout << "\n";

i++;

}

int X;

cout << "Введите число: ";

cin >> X;

InputCheck(X);

cout << "1 - сложение\n2 - вычитание\n3 - умножение\n4 - деление" << endl;

cout << "Введите номер операции: ";

cin >> choice;

InputCheck(choice);

switch (choice)

{

case 1:

Add(newMatrix, N, X);

break;

case 2:

Subtract(newMatrix, N, X);

break;

case 3:

Multiply(newMatrix, N, X);

break;

case 4:

Divide(newMatrix, N, X);

break;

default:

cout << "Нет такого варианта." << endl;

}

i = 0;

for (int\* ptr = newMatrix; ptr < newMatrix + N \* N; ++ptr)

{

cout << \*ptr << " ";

if ((ptr - newMatrix + 1) % N == 0)

cout << "\n";

i++;

}

}

void PracticalWork4()

{

setlocale(0, "");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

char text[maxInputSize];

char words[maxWords][maxLength];

cout << "Введите последовательность слов, заканчивающуюся точкой:\n";

cin.ignore(1000, '\n');

cin.get(text, maxInputSize);

EditText(text);

cout << "Отредактированный текст:\n";

cout << text;

int count = TextToWords(text, words);

cout << endl << "Количество слов: " << count << endl;

cout << "Слова в обратном порядке: " << endl;

for (int i = count - 1; i >= 0; --i)

cout << words[i] << " ";

cout << endl;

ChangeRegister(words, count);

for (int i = 0; i < count; i++)

cout << words[i] << " ";

cout << endl;

char pattern1[maxLength];

char pattern2[maxLength];

cout << "Алгоритм линейного поиска\nВведите подстроку: ";

cin.ignore(1000, '\n');

cin.get(pattern1, maxLength);

int result = LinearSearch(text, pattern1);

if (result == -1)

cout << "Подстрока не найдена" << endl;

else

cout << "Подстрока найдена в позиции: " << result << endl;

cout << "Алгоритм Бойера-Мура\nВведите подстроку: ";

cin.ignore(1000, '\n');

cin.get(pattern2, maxLength);

vector<int> occurrences = boyerMoore(text, pattern2);

if (!occurrences.empty())

{

cout << "Подстрока найдена на позициях:";

for (int i = 0; i < occurrences.size(); ++i)

cout << occurrences[i] << " ";

}

else

cout << "Подстрока не найдена";

cout << endl;

}

int main()

{

setlocale(0, "");

int choice;

while (1)

{

cout << "Введите номер практической работы: ";

cin >> choice;

InputCheck(choice);

switch (choice)

{

case 1:

PracticalWork1();

break;

case 2:

PracticalWork2();

break;

case 3:

PracticalWork3();

break;

case 4:

PracticalWork4();

break;

case 666:

return 0;

default:

cout << "Нет практической работы с таким номером";

}

}

}