**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра «Информационные Системы и Технологий (ИСиТ)»**

отчет

**по лабораторной работе №4**

**по дисциплине «Программирование»**

Тема: «Текстовые строки как массивы символов, организация работы с файлами»

|  |  |  |
| --- | --- | --- |
| Студент гр. 2003 |  | Зимацкий С. Н. |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2020

**Цель работы.**

Необходимо написать программу, которая реализует поставленную задачу:

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

2)    Необходимо отредактировать входной текст:

·        удалить лишние пробелы;

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

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

3) Вывести на экран слова последовательности в обратном порядке.

4) Вывести на экран  все слова последовательности в две или три колонки (в зависимости от количества слов) с выравниванием слов по правой границе колонки.

5)  Необходимо найти подстроку, которую введёт пользователь в имеющейся строке. Реализуйте два алгоритма: первый алгоритма – Линейный поиск, а второй алгоритм – Бойера-Мура. (\*)

**Основные теоретические положения.**

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'.

При выводе строк можно использовать форматирование (манипуляторы или функции потока вывода). Вывод текстовых строк на экран крайне простая задача:

char Str[21] = “Это пример текста”;

cout  <<  Str << endl;

cout  <<  “Это текстовый литерал.” << endl;

Ввод текста с клавиатуры можно осуществлять разными способами, каждый из которых имеет определенные особенности.

Непосредственное чтение текстовых строк из потока вывода осуществляется до первого знака пробела.

Такой способ чтения обеспечивает ввод символов до первого пробельного символа (не до конца строки). Остальные символы введенного с клавиатуры остаются в потоке ввода и могут быть прочитаны из него следующими операторами >>.

Для того чтобы прочесть всю строку полностью, можно воспользоваться одной из функций gets или gets\_s (для этого в программу должен быть включен заголовочный файл <stdio.h>).

Функция gets имеет один параметр, соответствующий массиву символов, в который осуществляется чтение. Вторая функция (gets\_s) имеет второй параметр, задающий максимальную длину массива символов .

Ввод текста, длина которого (вместе с нулевым символом) превышает значение второго параметра (то есть длины символьного массива ), приводит к возникновению ошибки при выполнении программы

Предпочтительно использование функции потока ввода cin.getline:

const int N = 21;

char Str [N];

cin.getline (Str, N);      // Пусть введена строка “Это пример текста”

cout  <<  Str << endl;  // На экран будет выведено “ Это пример текста”

Если длина введенного с клавиатуры текста превышает максимальную длину массива , в него будет записано (в нашем примере) 20 символов вводимого текста и нулевой символ. Остальные символы введенного текста остаются во входном потоке и могут быть взяты из него следующими инструкциями ввода.

Функция cin.getline может иметь третий параметр, задающий символ, при встрече которого чтение строки из потока прекращается:

cin.getline (Str, N,  ‘.’);

Иногда чтение из потока невозможно (например, попытка считать слишком длинный текст). Для того чтобы продолжить чтение из потока, необходимо восстановить его нормальное состояние. Этого можно достигнуть с помощью функции потока cin.clear(), которая сбрасывает состояние потока в нормальное. Если забирать остатки данных из потока ввода не надо, то следует очистить его с помощью функции cin.sync().

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

Чтобы использовать возможности класса string, нужно подключить библиотеку <string> и пространство имен std. Объявление же переменной типа string осуществляется схоже с обычной переменной:

string S1; // Переменная с именем s1 типа string

string S2 = “Пример”; // объявление с инициализацией

Создание нового типа string было обусловлено недостатками работы с строками символов, который показывал тип char. В сравнении с этим типом string имеет ряд основных преимуществ:

·        возможность использования для обработки строк стандартные операторы С++(=,+,<,==,>,+=,!=,<=,>=,[])(=,+,<,==,>,+=,!=,<=,>=,[]). Использование типа char приводило требовало написание чрезмерного программного кода;

·        обеспечение лучшей надежности программного кода;

·        обеспечение строки, как самостоятельного типа данных.

/\* string.cpp: Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы \*/

#include <iostream>

#include <string>

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

string A = "Пример"; // Объявляем и инициализируем строку А

string B; // Объявляем строку В

string C = "текста"; // Объявляем и инициализируем строку С

B = A; // Копируем текст из строки А в строку В

C = B + " " + C;

if(C == "Пример текста") // Проверяем корректно ли произошло присоединение строк

cout << C;

return 0;

}

Результат выполнения программы – вывод в консоль сообщения:

Пример текста

С string можно использовать оператор индексации и получать значения символа. Принцип действия такой же, как и у типа char:

string s1 = “hello!”;

cout << s1[0]; // Будет выведен символ ‘h’

Класс string обладает широким функционалом:

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

·        функция length() определяет длину строки;

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

·        функция erase() служит для удаления символов;

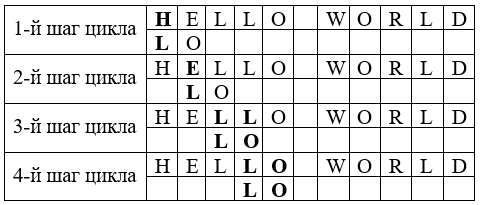
·        функция replace() выполняет замену символов;

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

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

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

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



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

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

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

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



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

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

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

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

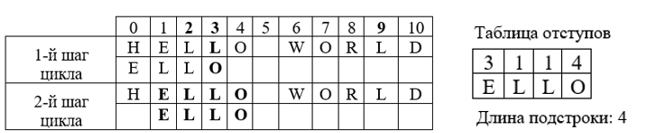
Правила построения таблицы сдвигов:

1)    Значение элемента таблицы равно удаленности соответствующего символа от конца шаблона (подстроки).

2)    Если символ встречается более одного раза, то применятся значение, соответствующее символу, наиболее близкому к концу шаблона.

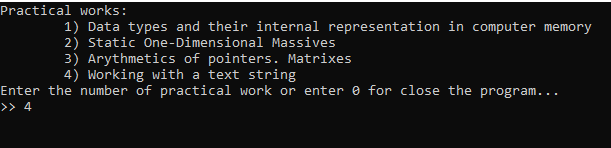
3)    Если символ в конце шаблона встречается 1 раз, ему соответствует значение, равное длине образа; если более одного раза – значение, соответствующее символу, наиболее близкому к концу образа.

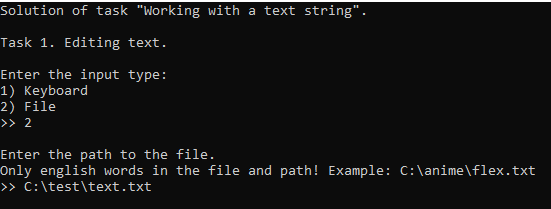
4)    Для символов, отсутствующих в образе, применяется значение, равное длине шаблона.

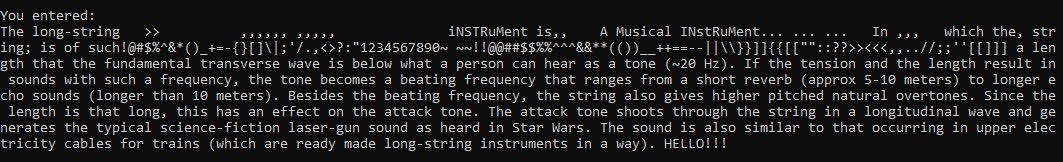


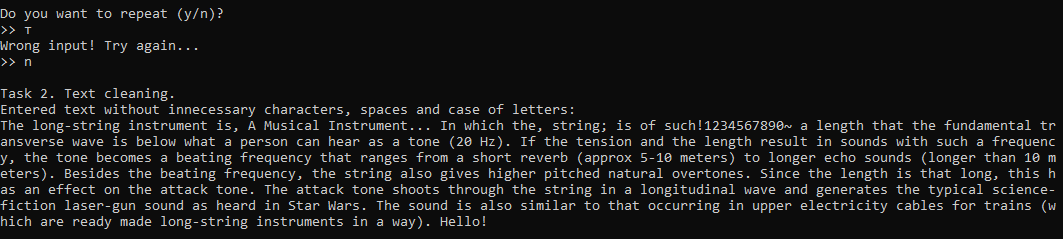
Сначала была построена таблица отступов и подсчитана длина подстроки. Затем начинается алгоритм поиска подстроки в строке. Сравнивает символ ‘L’ строки и ‘O’ подстроки. Элементы не совпадают, поэтому необходимо определить длину отступа. Символ ‘L’ присутствует в таблице отступа, длина отступа равняется 1. Подстрока смещается на 1 символ вперед. На следующей итерации подстрока найдена.

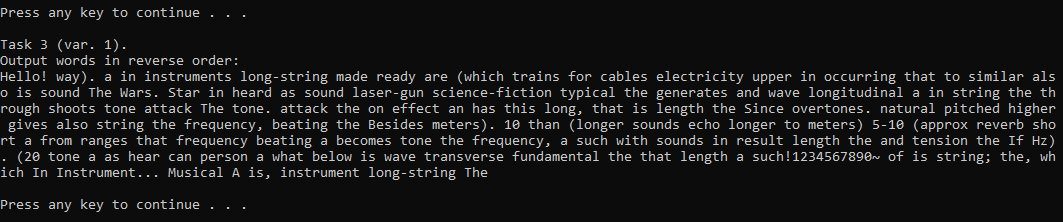
**Экспериментальные результаты.**

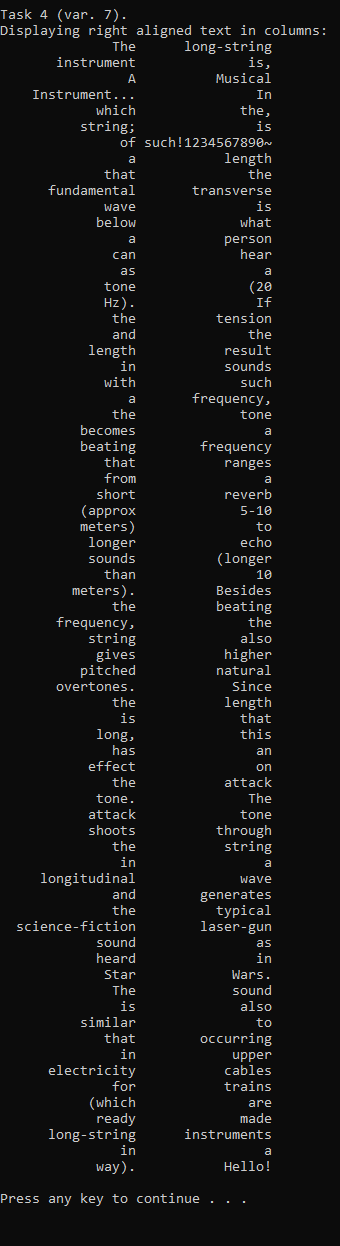


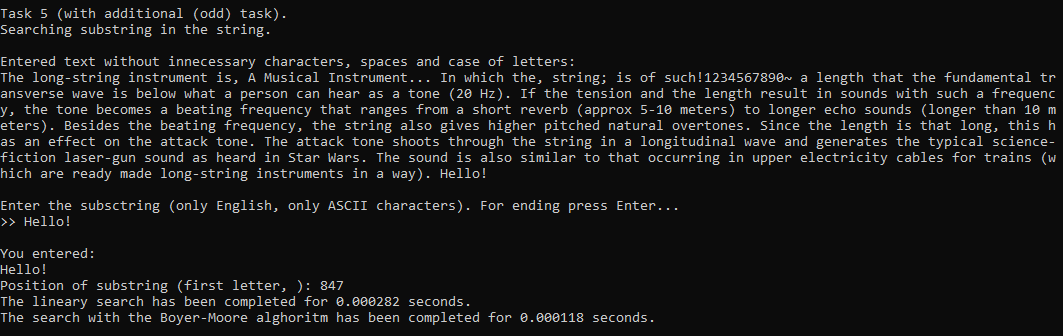


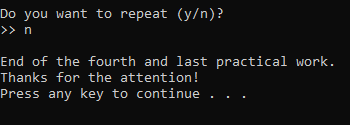










  
Рисунок 5 – Результат работы программы (полный код программы представлен в приложении А)

Приложение а  
полный код программы

#include <iostream>

#include <ctime> // Нужно для рандома

#include <chrono> // Нужно для засечения времени

#include <Windows.h> // Нужно для получения координат консоли и их изменения

#include <iomanip> // нужно для std::setw

#include <string>

#include <fstream> // нужно для чтения файлов

using namespace std::chrono; // Нужно для засечения времени

using namespace std;

void practicalWork1();

void practicalWork2();

void practicalWork3();

void practicalWork4();

// Функции, использующиеся в первой практической работе

void intToBinary(int);

void floatToBinary(int);

void doubleToBinary(int, int);

int readInt();

// Функции, использующиеся во второй практической работе

void createAndPrintRandomArray(int[]);

void bubbleSort(int[]);

void shakerSort(int[]);

void combSort(int[]);

void insertSort(int[]);

void quickSort(int[], int, int);

void Merge(int[], int, int, int, int[]);

void InternalMergeSort(int[], int, int, int[]);

void MergeSort(int[], int, int);

int getMinArrayElement(int[]);

int getMaxArrayElement(int[]);

int binarySearch(int[], int, int, int);

void copyArray(int[], int[]);

void printArray(int[]);

bool choiseNextAction();

float stopSecondsTimer(time\_point<steady\_clock>);

long long stopNanoSecondsTimer(time\_point<steady\_clock>);

// Функции, использующиеся в третьей практической работе

void snakeAnimation(int[], int, short cellSize = 4, short delay = 100);

void spiralAnimation(int[], int, short cellSize = 4, short delay = 100);

void consistentPermutation(int[], int);

void diagonalPermutation(int[], int);

void rowPermutation(int[], int);

void columnPermutation(int[], int);

// Функции, использующиеся в четвёртой практической работе

char \* readFile(string);

string textCleaner(string);

bool isSymbol(char);

string deleteChar(string, int, int);

void reverseOutput(string);

int countOfWords(string);

void wordOutput(string, int, int);

int longestWord(string);

int linearSubscringSearch(string, string);

int bouyerMooreSubstringSearch(string, string);

void badCharHeuristic(string, int, int[]);

int main()

{

while (true) {

system("CLS");

cout << "Practical works: \n"

<< "\t1) Data types and their internal representation in computer memory \n"

<< "\t2) Static One-Dimensional Massives \n"

<< "\t3) Arythmetics of pointers. Matrixes \n"

<< "\t4) Working with a text string \n"

<< "Enter the number of practical work or enter 0 for close the program... \n>> ";

int input;

cin >> input;

switch (input) {

case 1:

system("CLS");

practicalWork1();

break;

case 2:

system("CLS");

practicalWork2();

break;

case 3:

system("CLS");

practicalWork3();

break;

case 4:

system("CLS");

practicalWork4();

break;

default:

goto Exit;

}

}

Exit:

cout << "\nClosing the program... \n";

system("pause");

return 0;

}

void practicalWork1() {

cout << "Solution of task \"Data types and their internal representation in computer memory\". \n\n";

do {

cout << "\tTask 1. Size of data types. \n"

<< "int: \t\t" << sizeof(int) << " bytes. \n"

<< "short int \t" << sizeof(short int) << " bytes. \n"

<< "long int \t" << sizeof(long int) << " bytes. \n"

<< "float \t\t" << sizeof(float) << " bytes. \n"

<< "double \t\t" << sizeof(double) << " bytes. \n"

<< "long double \t" << sizeof(long double) << " bytes. \n"

<< "char \t\t" << sizeof(char) << " bytes. \n"

<< "bool \t\t" << sizeof(bool) << " bytes. \n";

} while (choiseNextAction());

system("CLS");

cout << "\tTask 2. Binary representation of integer number. \n\n";

int input;

do {

cout << "Enter the integer number... \n>> ";

// Считывание числа с проверкой на принадлежность к целым числам.

input = readInt();

cout << "Binary representation of number " << input << ": ";

intToBinary(int(input));

} while (choiseNextAction());

system("CLS");

cout << "\tTask 3. Binary representation of float number. \n";

union {

float inputFloat;

int inputInt1;

};

do {

cout << "Enter the float number... \n>> ";

cin >> inputFloat;

cout << "Binary representation of number " << inputFloat << ": ";

floatToBinary(inputInt1);

} while (choiseNextAction());

system("CLS");

cout << "\tTask 4. Binary representation of double number. \n";

union {

double inputDouble;

int inputInt2[2] = { NULL, NULL };

};

do {

cout << "Enter the double number... \n>> ";

cin >> inputDouble;

cout << "Binary representation of number " << inputDouble << ":\n";

doubleToBinary(inputInt2[1], inputInt2[0]);

} while (choiseNextAction());

cout << "\nEnd of the first practical work. \n";

system("pause");

}

void intToBinary(int value) {

int mask = 1 << 31; // Маска представляет собой 2^32, то есть 10...00, где нулей 31.

for (int i = 1; i <= 32; i++) {

/\*

Побитовое сравнение введённого числа с маской.

Если у числа и маски есть хотя бы одна общая единица (в данном случае бОльший бит числа), то выводится "1", иначе "0".

Далее побитовый сдвиг введённого числа влево и так 32 раза (именно столько бит определяется под int).

Например:

0 1000000 00000000 00000000 00000000 - двоичное представление в памяти компьютера введённого числа 2^31

1 0000000 00000000 00000000 00000000 - маска (2^32)

Вывод: 0

Затем побитовый сдвиг введённого числа влево:

1 0000000 00000000 00000000 00000000

1 0000000 00000000 00000000 00000000 - маска (2^32)

Вывод: 1

Далее следуют нули, в выводе: 0 1000000 00000000 00000000 00000000

\*/

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

value <<= 1;

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

putchar(' '); // отделение друг от друга байтов, содержащих значение числа, а также знакового бита

}

}

void floatToBinary(int value) {

int mask = 1 << 31;

for (int i = 1; i <= 32; i++) {

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

value <<= 1;

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

putchar(' '); // отделение друг от друга битов мантиссы и порядка, а также знакового бита

}

}

void doubleToBinary(int firstPart, int secondPart) {

int mask = 1 << 31;

for (int i = 1; i <= 32; i++) {

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

firstPart <<= 1;

if (i == 1 || i == 12)

putchar(' '); // отделение друг от друга битов мантиссы и порядка, а также знакового бита

}

for (int i = 1; i <= 32; i++) {

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

secondPart <<= 1;

}

}

int readInt() {

double input;

while (true) {

cin >> input;

// Проверка на целое число

if (static\_cast<int>(input) != input) {

cout << "Need enter the integer number! Try again...\n";

continue;

}

return int(input);

}

}

const int sizeOfArray = 100;

void practicalWork2() {

srand((unsigned)time(NULL));

time\_point<steady\_clock> startTimer = steady\_clock::now();

float stopTimeInSeconds = stopSecondsTimer(startTimer);

long long stopTimeInNanoSeconds = stopNanoSecondsTimer(startTimer);

int count = 0;

cout << "Solution of task \"Static One-Dimensional Massives\". \n\n";

// РЕШЕНИЕ ПЕРВОГО ЗАДАНИЯ

int mainArray[sizeOfArray];

do {

cout << "\tTask 1. Initial array:" << endl;

createAndPrintRandomArray(mainArray);

} while (choiseNextAction());

do { // РЕШЕНИЕ ВТОРОГО ЗАДАНИЯ

cout << "\n\tTask 2. Array sorting." << endl

<< "\t\t1) Bubble sort" << endl

<< "\t\t2) Shaker sort" << endl

<< "\t\t3) Comb sort" << endl

<< "\t\t4) Insert sort" << endl

<< "\t\t5) Quick sort" << endl

<< "\t\t6) Merge sort" << endl

<< "Enter the number of sorting... \n>> ";

int input;

cin >> input;

int copiedArray[sizeOfArray];

copyArray(mainArray, copiedArray); // Копирование массива в другой массив для возможности в дальнейшем на том же массиве проверить другие типы сортировки.

switch (input) {

case 1: // РЕШЕНИЕ 2.1 ЗАДАНИЯ (bubble sort)

startTimer = steady\_clock::now(); // Старт отсчёта времени

bubbleSort(copiedArray);

stopTimeInSeconds = stopSecondsTimer(startTimer);

cout << "\nBubble sorted array: \n";

printArray(copiedArray);

cout << "Array sorted in " << fixed << stopTimeInSeconds << " second(s). \n";

break;

case 2: // РЕШЕНИЕ 2.2 ЗАДАНИЯ (shaker sort)

startTimer = steady\_clock::now();

shakerSort(copiedArray);

stopTimeInSeconds = stopSecondsTimer(startTimer);

cout << "\nShaker sorted array: \n";

printArray(copiedArray);

cout << "Array sorted in " << fixed << stopTimeInSeconds << " second(s). \n";

break;

case 3: // РЕШЕНИЕ 2.3 ЗАДАНИЯ (comb sort)

startTimer = steady\_clock::now();

combSort(copiedArray);

stopTimeInSeconds = stopSecondsTimer(startTimer);

cout << "\nComb sorted array: \n";

printArray(copiedArray);

cout << "Array sorted in " << fixed << stopTimeInSeconds << " second(s). \n";

break;

case 4: // РЕШЕНИЕ 2.4 ЗАДАНИЯ (insert sort)

startTimer = steady\_clock::now();

insertSort(copiedArray);

stopTimeInSeconds = stopSecondsTimer(startTimer);

cout << "\nInsert sorted array: \n";

printArray(copiedArray);

cout << "Array sorted in " << fixed << stopTimeInSeconds << " second(s). \n";

break;

case 5: // РЕШЕНИЕ 2.5 ЗАДАНИЯ (quick sort)

startTimer = steady\_clock::now();

quickSort(copiedArray, sizeOfArray - 1, 0);

stopTimeInSeconds = stopSecondsTimer(startTimer);

cout << "\nQuick sorted array: \n";

printArray(copiedArray);

cout << "Array sorted in " << fixed << stopTimeInSeconds << " second(s). \n";

break;

case 6: // РЕШЕНИЕ ДОПОЛНИТЕЛЬНОГО ЗАДАНИЯ (merge sort)

startTimer = steady\_clock::now();

MergeSort(copiedArray, 0, sizeOfArray - 1);

stopTimeInSeconds = stopSecondsTimer(startTimer);

cout << "\nMerge sorted array: \n";

printArray(copiedArray);

cout << "Array sorted in " << fixed << stopTimeInSeconds << " second(s). \n";

break;

}

} while (choiseNextAction());

int copiedArray[sizeOfArray]; // Создание отсортированного массива, с которым будет производиться работа в дальнейших заданиях

copyArray(mainArray, copiedArray);

quickSort(copiedArray, sizeOfArray - 1, 0);

do { // РЕШЕНИЕ ТРЕТЬЕГО ЗАДАНИЯ

cout << "\n\tTask 3. Searching min and max elements of sorted and unsorted arrays. \n";

startTimer = steady\_clock::now(); // Поиск в неотсортированном массиве

int max = getMaxArrayElement(mainArray);

int min = getMinArrayElement(mainArray);

stopTimeInNanoSeconds = stopNanoSecondsTimer(startTimer);

cout << "Min element of array: " << min << endl

<< "Max element of array: " << max << endl

<< "Elements founds in unsorted array for " << stopTimeInNanoSeconds << " nanoseconds. \n";

startTimer = steady\_clock::now(); // Поиск в отсортированном массиве

max = getMaxArrayElement(copiedArray);

min = getMinArrayElement(copiedArray);

stopTimeInNanoSeconds = stopNanoSecondsTimer(startTimer);

//cout << "Min: " << min << "\tMax: " << max << endl; // Отладка

cout << "Elements founds in sorted array for " << stopTimeInNanoSeconds << " nanoseconds. \n";

} while (choiseNextAction());

do { // РЕШЕНИЕ ЧЕТВЁРТОГО ЗАДАНИЯ

cout << "\n\tTask 4. Search average elements from array. \n";

int averageElement = int(abs(getMaxArrayElement(copiedArray)) - abs(getMinArrayElement(copiedArray)));

cout << "Average element: " << averageElement << endl;

bool isExists = false; // Проверка на то, существует ли вообще в массиве подходящий средний элемент

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

if (copiedArray[i] == averageElement) {

isExists = true;

break;

}

if (isExists) {

cout << "Index(es) of average elements: ";

int count = 0;

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

if (copiedArray[i] == averageElement) {

cout << i << " ";

count++;

}

cout << "\nCount of average elements in array: " << count << endl;

}

else

cout << "Element not found in array. \n";

} while (choiseNextAction());

do { // РЕШЕНИЕ ПЯТОГО ЗАДАНИЯ

cout << "\n\tTask 5. The number of smaller elements than the one entered by the user. \n"

<< "Enter the integer element...\n>> ";

int input;

cin >> input;

int count = 0;

for (int element : copiedArray)

if (element < input)

count++;

cout << "The array has " << count << " element(s) less than you entered. \n";

} while (choiseNextAction());

do { // РЕШЕНИЕ ШЕСТОГО ЗАДАНИЯ

cout << "\n\tTask 6. The number of larger elements than the one entered by the user. \n"

<< "Enter the integer element...\n>> ";

int input;

cin >> input;

int count = 0;

for (int element : copiedArray)

if (element > input)

count++;

cout << "The array has " << count << " element(s) larger than you entered. \n";

} while (choiseNextAction());

do { // РЕШЕНИЕ СЕДЬМОГО ЗАДАНИЯ

cout << "\n\tTask 7. The searching of a user-entered element in the array (traditional and binary search). \n"

<< "Enter the integer element...\n>> ";

int input;

cin >> input;

startTimer = steady\_clock::now();

bool isExists = false;

for (int element : copiedArray) // Поиск обычным перебором

if (element == input) {

isExists = true;

break;

}

stopTimeInNanoSeconds = stopNanoSecondsTimer(startTimer);

if (isExists)

cout << "The entered element is in the array. \n";

else

cout << "The entered element is not in the array. \n";

cout << "The search has been completed for " << stopTimeInNanoSeconds << " nanoseconds. \n";

startTimer = steady\_clock::now();

int searchedElement = binarySearch(copiedArray, input, 0, sizeOfArray - 1);

stopTimeInNanoSeconds = stopNanoSecondsTimer(startTimer);

/\* Отладка

if (searchedElement != -1)

cout << "Searched element (binary search): " << searchedElement << endl;

else cout << "Element not searched. \n"; \*/

cout << "The binary search has been completed for " << stopTimeInNanoSeconds << " nanoseconds. \n";

} while (choiseNextAction());

do { // РЕШЕНИЕ ВОСЬМОГО ЗАДАНИЯ

cout << "\n\tTask 8. Switching array elements. \n"

<< "Enter the first and second indexes of array elements...\n>> ";

int input1, input2;

cin >> input1 >> input2;

startTimer = steady\_clock::now();

//if (((input1 && input2) >= 0) && ((input1 && input2) < sizeOfArray)) { // Ошибка компилятора о небезопасном условии. Разобраться, почему.

if ((input1 >= 0) && (input2 >= 0) && (input1 < sizeOfArray) && (input2 < sizeOfArray)) {

int t = copiedArray[input1];

copiedArray[input1] = copiedArray[input2];

copiedArray[input2] = t;

}

else {

cout << "Wrong input! Try again...\n\n>> ";

continue;

}

stopTimeInNanoSeconds = stopNanoSecondsTimer(startTimer);

printArray(copiedArray); // Отладка

cout << "The switching has been completed for " << stopTimeInNanoSeconds << " nanoseconds. \n";

} while (choiseNextAction());

cout << "\nEnd of the second practical work. \n";

system("pause");

}

void createAndPrintRandomArray(int inputArray[]) {

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

inputArray[i] = -99 + rand() % 199; // В массив пишутся случайные целые числа диапазона -99..99 включительно. При желании данную функцию можно расширить возможностью выбирать граничные значения.

cout << inputArray[i] << "\t";

if ((i + 1) % 10 == 0) // Вывод по 10 элементов для удобочитаемости

cout << endl;

}

}

void bubbleSort(int inputArray[]) {

bool swapped;

do {

swapped = false;

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

if (inputArray[i] > inputArray[i + 1]) { // Соседние элементы массива меняются местами

int t = inputArray[i];

inputArray[i] = inputArray[i + 1];

inputArray[i + 1] = t;

swapped = true;

}

}

} while (swapped); // Если за последний проход цикла никакие элементы не поменялись местами, цикл прерывается.

}

void shakerSort(int inputArray[]) {

bool swapped;

do {

swapped = false;

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

if (inputArray[i] > inputArray[i + 1]) {

int t = inputArray[i];

inputArray[i] = inputArray[i + 1];

inputArray[i + 1] = t;

swapped = true;

}

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

if (inputArray[i] < inputArray[i - 1]) {

int t = inputArray[i];

inputArray[i] = inputArray[i - 1];

inputArray[i - 1] = t;

swapped = true;

}

} while (swapped);

}

void combSort(int inputArray[]) {

int swap;

float k = 1.247F, sortingRange = sizeOfArray - 1;

while (sortingRange >= 1)

{

for (int i = 0; i + sortingRange < sizeOfArray; i++)

{

if (inputArray[i] > inputArray[int(i + sortingRange)])

{

swap = inputArray[int(i + sortingRange)];

inputArray[int(i + sortingRange)] = inputArray[i];

inputArray[i] = swap;

}

}

sortingRange /= k;

}

bool swapped;

do {

swapped = false;

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

if (inputArray[i] > inputArray[i + 1]) { // Соседние элементы массива меняются местами

int t = inputArray[i];

inputArray[i] = inputArray[i + 1];

inputArray[i + 1] = t;

swapped = true;

}

}

} while (swapped == true); // Если за последний проход цикла никакие элементы не поменялись местами, цикл прерывается.

}

void insertSort(int inputArray[]) {

int extractedElement, currentPosition;

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

{

extractedElement = inputArray[i];

currentPosition = i;

while ((currentPosition >= 1) && (extractedElement < inputArray[currentPosition - 1]))

{

inputArray[currentPosition] = inputArray[currentPosition - 1];

currentPosition--;

}

inputArray[currentPosition] = extractedElement;

}

}

void quickSort(int inputArray[], int end, int begin) {

int middleElement;

int f = begin;

int l = end;

middleElement = inputArray[(f + l) / 2]; // Определяется опорный элемент

while (f < l) { // Массив сортируется так, что по окончании цикла будет представлять собой две части: элементы меньше либо равные опорному (левая), элементы больше либо равные опорному (правая)

while (inputArray[f] < middleElement) f++;

while (inputArray[l] > middleElement) l--;

if (f <= l)

{

int swap = inputArray[f];

inputArray[f] = inputArray[l];

inputArray[l] = swap;

f++;

l--;

}

}

if (begin < l) quickSort(inputArray, l, begin); // Где l - правая граница левой части разделённого пополам массива

if (f < end) quickSort(inputArray, end, f); // Где f - левая граница правой части разделённого пополам массива

}

void Merge(int array[], int first, int middle, int last, int temp[])

{

int idx = first;

int begin1 = first, end1 = middle;

int begin2 = middle + 1, end2 = last;

for (; begin1 <= end1 && begin2 <= end2; )

{

if (array[begin1] < array[begin2])

temp[idx++] = array[begin1++];

else

temp[idx++] = array[begin2++];

}

for (; begin1 <= end1; )

temp[idx++] = array[begin1++];

for (; begin2 <= end2; )

temp[idx++] = array[begin2++];

for (idx = first; idx <= last; idx++)

array[idx] = temp[idx];

}

void InternalMergeSort(int array[], int first, int last, int buffer[])

{

if (first < last)

{

int m = (first + last) / 2;

InternalMergeSort(array, first, m, buffer);

InternalMergeSort(array, m + 1, last, buffer);

Merge(array, first, m, last, buffer);

}

}

void MergeSort(int array[], int first, int last)

{

int \*buffer = new int[sizeOfArray];

InternalMergeSort(array, 0, sizeOfArray - 1, buffer);

delete[] buffer;

}

int getMinArrayElement(int inputArray[]) {

int minElement = INT\_MAX;

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

if (inputArray[i] < minElement)

minElement = inputArray[i];

return minElement;

}

int getMaxArrayElement(int inputArray[]) {

int maxElement = INT\_MIN;

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

if (inputArray[i] > maxElement)

maxElement = inputArray[i];

return maxElement;

}

int binarySearch(int inputArray[], int inputElement, int lowPos, int highPos) {

while (lowPos <= highPos) {

int mid = lowPos + (highPos - lowPos) / 2;

if (inputArray[mid] == inputElement)

return mid;

if (inputArray[mid] < inputElement)

lowPos = mid + 1;

else

highPos = mid - 1;

}

return -1;

}

void copyArray(int originalArray[], int resultArray[]) {

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

resultArray[i] = originalArray[i];

}

void printArray(int inputArray[]) {

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

cout << inputArray[i] << "\t";

if ((i + 1) % 10 == 0) // Вывод по 10 элементов для удобочитаемости

cout << endl;

}

cout << endl;

}

bool choiseNextAction() {

cout << "\nDo you want to repeat (y/n)?" << "\n>> ";

char input;

while (true) {

cin >> input;

if (input == 'y' || input == 'Y')

return true;

else if (input == 'n' || input == 'N')

return false;

else

cout << "Wrong input! Try again...\n>> ";

}

}

float stopSecondsTimer(time\_point<steady\_clock> startTimer) {

using fseconds = duration<float>; // Определение кастомного интервала времени для отображение дробных секунд в таймере.

time\_point<steady\_clock> endTimer = steady\_clock::now(); // Остановка времени

fseconds sortingTime = duration\_cast<fseconds>(endTimer - startTimer); // Вычисление разницы между финальным и стартовым временем

return sortingTime.count();

}

long long stopNanoSecondsTimer(time\_point<steady\_clock> startTimer) {

time\_point<steady\_clock> endTimer = steady\_clock::now();

nanoseconds sortingTime = duration\_cast<nanoseconds>(endTimer - startTimer);

return sortingTime.count();

}

void practicalWork3() {

int order;

cout << "Solution of task \"Arythmetics of pointers. Matrixes\". \n\n"

<< "Task 1. Visualisation of filling the matrix. \n"

<< "Enter the order of matrix... \n>> ";

bool inputIsCorrected = true;

do {

cin >> order;

if (order > 0)

inputIsCorrected = false;

else {

cout << "Wrong input! Try again...\n>> ";

}

} while (inputIsCorrected);

srand((unsigned)time(NULL));

int \*ptrarray = new int[order\*order]; // Объявление динамического одномерного массива размерности двумерного

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

for (int j = 0; j < order; j++) // В массив пишутся случайные числа от 1 до N\*N, где N - порядок матрицы.

\*(ptrarray + i \* order + j) = 1 + rand() % (order \* order);

do {

cout << "\nEnter the type of animation:"

<< "\n\t1) Snake animation"

<< "\n\t2) Spiral animation \n>> ";

inputIsCorrected = true;

do {

int input;

cin >> input;

switch (input) {

case 1:

snakeAnimation(ptrarray, order);

inputIsCorrected = false;

break;

case 2:

spiralAnimation(ptrarray, order);

inputIsCorrected = false;

break;

default:

cout << "Wrong input! Try again...\n\n>> ";

}

} while (inputIsCorrected);

} while (choiseNextAction());

int \*copiedArr = new int[order \* order]; // Копирование массива в другой массив для возможности повторно произвести перестановку элементов.

do {

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

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

\*(copiedArr + i \* order + j) = \*(ptrarray + i \* order + j);

cout << "\nTask 2. Permutation of matrix elements"

<< "\nEnter the type of permutation:"

<< "\n\t1) Consistent"

<< "\n\t2) Diagonal"

<< "\n\t3) Row"

<< "\n\t4) Column \n>> ";

inputIsCorrected = true;

do {

int input;

cin >> input;

switch (input) {

case 1:

consistentPermutation(copiedArr, order);

inputIsCorrected = false;

break;

case 2:

diagonalPermutation(copiedArr, order);

inputIsCorrected = false;

break;

case 3:

rowPermutation(copiedArr, order);

inputIsCorrected = false;

break;

case 4:

columnPermutation(copiedArr, order);

inputIsCorrected = false;

break;

default:

cout << "Wrong input! Try again...\n\n>> ";

}

} while (inputIsCorrected);

cout << endl; // Вывод получившегося массива.

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

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

cout << setw(4) << \*(copiedArr + i \* order + j);

cout << endl;

}

} while (choiseNextAction());

cout << "\nTask 3. Matrix sorting"

<< "\nBubble sorted array: \n";

bool swapped;

do {

swapped = false;

for (int i = 0; i < order \* order - 1; i++) {

if (\*(copiedArr + i) > \*(copiedArr + i + 1)) { // Соседние элементы массива меняются местами

int t = \*(copiedArr + i);

\*(copiedArr + i) = \*(copiedArr + i + 1);

\*(copiedArr + i + 1) = t;

swapped = true;

}

}

} while (swapped);

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

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

cout << setw(4) << \*(copiedArr + i \* order + j);

cout << endl;

}

do {

cout << "\nTask 4. Matrix operations"

<< "\nEnter the type of operation:"

<< "\n\t1) Reducing the matrix by a number"

<< "\n\t2) Increacing the matrix by a number"

<< "\n\t3) Matrix multiplication by a number"

<< "\n\t4) Dividing the matrix by a number \n>> ";

/\*

Если бы данная программа использовалась для чего-то большего, чем написание курсовой, то её однозначно стоило бы доработать на предмет возможности работы с нецелыми числами, т. к. в случае деления чисел матрицы на введённое пользователем число очень часто могут получаться нецелые числа в ячейках матрицы, а в данном случае мы будем видеть искажённый результат.

\*/

inputIsCorrected = true;

int input;

do {

cin >> input;

if (input == 1 || input == 2 || input == 3 || input == 4)

inputIsCorrected = false;

else {

cout << "Wrong input! Try again...\n\n>> ";

break;

}

} while (inputIsCorrected);

int number;

cout << "\nEnter the integer: \n>> ";

cin >> number;

switch (input) {

case 1:

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

for (int j = 0; j < order; j++) {

\*(copiedArr + i \* order + j) -= number;

cout << setw(4) << \*(copiedArr + i \* order + j);

}

cout << endl;

}

break;

case 2:

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

for (int j = 0; j < order; j++) {

\*(copiedArr + i \* order + j) += number;

cout << setw(4) << \*(copiedArr + i \* order + j);

}

cout << endl;

}

break;

case 3:

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

for (int j = 0; j < order; j++) {

\*(copiedArr + i \* order + j) \*= number;

cout << setw(4) << \*(copiedArr + i \* order + j);

}

cout << endl;

}

break;

case 4:

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

for (int j = 0; j < order; j++) {

\*(copiedArr + i \* order + j) /= number;

cout << setw(4) << \*(copiedArr + i \* order + j);

}

cout << endl;

}

break;

}

} while (choiseNextAction());

cout << "\nEnd of the third practical work. \n";

system("pause");

}

void practicalWork4() {

cout << "Solution of task \"Working with a text string\". \n\n"

<< "Task 1. Editing text. \n";

string sourceStr;

do {

cout << "\nEnter the input type: \n"

<< "1) Keyboard \n"

<< "2) File \n>> ";

int input;

bool inputIsCorrected = true;

do {

cin >> input;

if (input == 1 || input == 2)

inputIsCorrected = false;

else {

cout << "Wrong input! Try again...\n>> ";

}

} while (inputIsCorrected);

switch (input) {

case 1:

cout << "\nEnter the string for edit (only English). For ending press Enter... \n>> ";

cin.ignore(32767, '\n');

getline(cin, sourceStr);

cout << "\nYou entered: \n" << sourceStr << endl;

break;

case 2:

ifstream file;

do {

cout << "\nEnter the path to the file. \n"

<< "Only english words in the file and path! Example: C:\\anime\\flex.txt \n>> ";

string path;

cin.ignore(32767, '\n');

getline(cin, path);

file.open(path);

if (!file.is\_open()) {

cout << "Error opening file! \n";

continue;

}

getline(file, sourceStr);

} while (!file.is\_open());

cout << "\nYou entered: \n" << sourceStr << endl;

file.close();

break;

}

} while (choiseNextAction());

cout << "\nTask 2. Text cleaning. \n"

<< "Entered text without innecessary characters, spaces and case of letters: \n";

sourceStr = textCleaner(sourceStr);

cout << sourceStr << endl << endl;

system("pause");

cout << "\nTask 3 (var. 1)."

<< "\nOutput words in reverse order: \n";

reverseOutput(sourceStr);

cout << endl;

system("pause");

/\*

Данный алгоритм может показаться избыточным в реализации,

т. к. изначально он разрабатывался под вывод слов в колонках по левому выравниванию и был переработан минимальными усилиями - создан небольшой массив размера длины самого длинного слова в тексте, на вывод подаётся этот массив, т. к. cout умеет выравнивать целые слова. Таким образом не приходится задумываться над тем, как выравнивать по правому краю слова посимвольно.

Его можно переработать более оптимизированно/изящно следующими путями:

1) Поместив текст пословно в двумерный массив, где каждая строка/столбец (кому как удобно) содержит отдельно слово, на вывод соответственно будет подаваться строка/столбец целиком, форматировать в таком случае будет проще.

В текущем же алгоритме схожая реализация.

2) Без создания доп. массивов с помощью каретки выводить слова по правому выравниванию.

Этот вариант предпочтительнее, но требует больше трудозатрат.

Варианты не единственные, можно придумать более изящные.

\*/

cout << "\nTask 4 (var. 7)."

<< "\nDisplaying right aligned text in columns: \n";

int arrSize = sourceStr.size(); // Определение количества символов в тексте, очищенном от мусора

int lWord = longestWord(sourceStr); // Определение самого длинного слова для корректного вывода в ячейках далее

int pos = 0; // Текущий индекс, по которому передаётся на вывод элемент массива с текстом

char \*word = new char[lWord]; // Массив, в который будет записываться текущее слово

for (int t = 0; t < lWord; t++)

word[t] = 0;

// Согласно постановке задачи, в зависимости от чётности-нечётности текст выводится либо в 2, либо в 3 колонки.

if ((countOfWords(sourceStr) % 2) == 0) {

while (pos < arrSize) {

for (int i = 0; i < 2; i++) { // Вывод по 2 слова в строку

short j = 0; // Текущая позиция внутри массива текущего слова (word)

while ((sourceStr[pos] != ' ') && (pos < arrSize)) {

word[j] = sourceStr[pos];

pos++;

j++;

}

word[j] = 0; // Установка конца слова для корректного вывода

cout << setw(lWord + 1) << right << word;

pos++; // Пропуск пробела в тексте

for (int t = 0; t < lWord; t++)

word[t] = 0; // Заполнение нулями для очистки, чтобы можно было отобразить следующее слово

}

cout << endl;

}

} else

while (pos < arrSize) { // Ниже всё аналогично тому, что было выше.

for (int i = 0; i < 3; i++) { // Вывод по 3 слова в строку

if (!(pos < arrSize))

break;

short j = 0;

while ((sourceStr[pos] != ' ') && (pos < arrSize)) {

word[j] = sourceStr[pos];

pos++;

j++;

}

word[j] = 0;

cout << setw(lWord + 1) << right << word;

pos++;

for (int t = 0; t < lWord, word[t] != 0; t++)

word[t] = 0;

}

cout << endl;

}

cout << endl;

system("pause");

time\_point<steady\_clock> startTimer = steady\_clock::now(); // Нужно для таймера

float stopTimeInSeconds = stopSecondsTimer(startTimer);

cout << "\nTask 5 (with additional (odd) task)."

<< "\nSearching substring in the string.\n";

cout << "\nEntered text without innecessary characters, spaces and case of letters: \n";

cout << sourceStr << endl ; // Выводим ещё раз очищенный от мусора текст, чтобы пользователь понимал, где и что ищется

do {

cout << "\nEnter the subsctring (only English, only ASCII characters). For ending press Enter... \n>> ";

string substring;

cin.ignore(32767, '\n');

getline(cin, substring);

cout << "\nYou entered: \n" << substring << endl;

// Используем таймер для засечения разницы времени между разными методами поиска подстроки в строке

startTimer = steady\_clock::now();

int subStrPos = linearSubscringSearch(sourceStr, substring);

stopTimeInSeconds = stopSecondsTimer(startTimer);

if (subStrPos != -1)

cout << "Position of substring (first letter, ): " << subStrPos << endl

<< "The lineary search has been completed for " << fixed << stopTimeInSeconds << " seconds. \n";

else

cout << "Substring was not found!\n"

<< "The lineary search has NOT been completed for " << fixed << stopTimeInSeconds << " seconds! \n";

subStrPos = -1;

startTimer = steady\_clock::now();

subStrPos = bouyerMooreSubstringSearch(sourceStr, substring);

stopTimeInSeconds = stopSecondsTimer(startTimer);

if (subStrPos != -1)

cout << "The search with the Boyer-Moore alghoritm has been completed for " << stopTimeInSeconds << " seconds. \n";

else

cout << "The search with the Boyer-Moore alghoritm has NOT been completed for " << stopTimeInSeconds << " seconds! \n";

} while (choiseNextAction());

cout << "\nEnd of the fourth and last practical work."

<< "\nThanks for the attention! \n";

system("pause");

}

void snakeAnimation(int arr[], int order, short cellSize, short delay) {

/\* Функция принимает на вход:

1) Указатель на первый элемент массива-матрицы (квадратной) (required)

2) Порядок матрицы (required)

3) Размер текстовой ячейки, определяемой под одно число матрицы (минимально это количество цифр в максимальном числе + 1) (опционально)

4) Скорость вывода элементов матрицы, в миллисекундах (опционально)

Функции других анимаций работают аналогично.

\*/

cout << "\nTry to \"snake\" animation: \n";

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE); // Нужно для получения данных курсора консоли

CONSOLE\_SCREEN\_BUFFER\_INFO bi;

int column = 0;

while (column < order) {

for (int row = 0; row < order; row++) { // Вывод вниз

GetConsoleScreenBufferInfo(hConsole, &bi); // Получение координат курсора

bi.dwCursorPosition.Y += 1; // Вычисление новых координат курсора

bi.dwCursorPosition.X -= cellSize;

SetConsoleCursorPosition(hConsole, bi.dwCursorPosition); // Установка положения курсора

cout << setw(cellSize) << \*(arr + row \* order + column);

Sleep(delay); // Задержка

}

column++;

if (column < order) {

for (int row = order - 1; row >= 0; row--) { // Вывод вверх

cout << setw(cellSize) << \*(arr + row \* order + column);

Sleep(delay);

GetConsoleScreenBufferInfo(hConsole, &bi);

bi.dwCursorPosition.Y -= 1;

bi.dwCursorPosition.X -= cellSize;

SetConsoleCursorPosition(hConsole, bi.dwCursorPosition);

}

column++;

GetConsoleScreenBufferInfo(hConsole, &bi);

bi.dwCursorPosition.X += (cellSize \* 2);

SetConsoleCursorPosition(hConsole, bi.dwCursorPosition);

}

else break;

}

if (order % 2 == 0)

for (int i = 0; i < order + 1; i++)

cout << endl;

}

void spiralAnimation(int arr[], int order, short cellSize, short delay) {

cout << "\n\nTry to spiral animation: \n";

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE); // Нужно для получения данных курсора консоли

CONSOLE\_SCREEN\_BUFFER\_INFO bi;

int length = order; // Длина текущего вывода

for (int i = 0; i < length; i++) { // Вывод вправо

cout << setw(cellSize) << \*(arr + (order - length) \* order + i);

Sleep(delay);

}

length--;

int upperRightBorder = order - 1;

int lowerLeftBorder = 0;

while (length > 0)

{

for (int i = 0, t = lowerLeftBorder + 1; i < length; i++, t++) { // Вывод вниз

GetConsoleScreenBufferInfo(hConsole, &bi);

bi.dwCursorPosition.Y += 1;

bi.dwCursorPosition.X -= (cellSize);

SetConsoleCursorPosition(hConsole, bi.dwCursorPosition);

cout << setw(cellSize) << \*(arr + t \* order + upperRightBorder);

Sleep(delay);

}

if (!(length > 0))

break;

for (int i = 0, t = upperRightBorder - 1; i < length; i++, t--) { // Вывод влево

GetConsoleScreenBufferInfo(hConsole, &bi);

bi.dwCursorPosition.X -= (cellSize \* 2);

SetConsoleCursorPosition(hConsole, bi.dwCursorPosition);

cout << setw(cellSize) << \*(arr + upperRightBorder \* order + t);

Sleep(delay);

}

length--;

upperRightBorder--;

if (!(length > 0))

break;

for (int i = 0, t = upperRightBorder; i < length; i++, t--) { // Вывод вверх

GetConsoleScreenBufferInfo(hConsole, &bi);

bi.dwCursorPosition.Y -= 1;

bi.dwCursorPosition.X -= (cellSize);

SetConsoleCursorPosition(hConsole, bi.dwCursorPosition);

cout << setw(cellSize) << \*(arr + t \* order + lowerLeftBorder);

Sleep(delay);

}

lowerLeftBorder++;

if (!(length > 0))

break;

for (int i = 0, t = lowerLeftBorder; i < length; i++, t++) { // Вывод вправо

cout << setw(cellSize) << \*(arr + lowerLeftBorder \* order + t);

Sleep(delay);

}

length--;

}

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

cout << endl;

}

void consistentPermutation(int arr[], int order) {

// Определение порядка "рабочей четверти" матрицы, элементы которой (четверти) далее будут перебираться (разное поведение для матриц чётного и нечётного порядка)

int halfOrder = ceil(static\_cast<float>(order) / 2);

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

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

// Определение индекса текущего опорного элемента, далее будет производиться работа над элементами с таким же индексом, только в других четвертях матрицы

int \*p = (arr + i \* order + j);

int t = \*p;

// \*p - элемент левой верхней четверти, "рабочей четверти" матрицы

\*p = \*(p + halfOrder \* order);

// \*(p + halfOrder \* order) - элемент левой нижней четверти матрицы

\*(p + halfOrder \* order) = \*(p + halfOrder \* order + halfOrder);

// \*(p + halfOrder \* order + halfOrder) - элемент правой нижней четверти матрицы

\*(p + halfOrder \* order + halfOrder) = \*(p + halfOrder);

// элемент правой верхней четверти матрицы

\*(p + halfOrder) = t;

}

}

}

void diagonalPermutation(int arr[], int order) {

int halfOrder = ceil(static\_cast<float>(order) / 2);

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

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

int \*p = (arr + i \* order + j);

int t = \*p;

\*p = \*(p + halfOrder \* order + halfOrder);

\*(p + halfOrder \* order + halfOrder) = t;

t = \*(p + halfOrder);

\*(p + halfOrder) = \*(p + halfOrder \* order);

\*(p + halfOrder \* order) = t;

}

}

}

void rowPermutation(int arr[], int order) {

int halfOrder = ceil(static\_cast<float>(order) / 2);

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

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

int \*p = (arr + i \* order + j);

int t = \*p;

\*p = \*(p + halfOrder \* order);

\*(p + halfOrder \* order) = t;

t = \*(p + halfOrder);

\*(p + halfOrder) = \*(p + halfOrder \* order + halfOrder);

\*(p + halfOrder \* order + halfOrder) = t;

}

}

}

void columnPermutation(int arr[], int order) {

int halfOrder = ceil(static\_cast<float>(order) / 2);

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

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

int \*p = (arr + i \* order + j);

int t = \*p;

\*p = \*(p + halfOrder);

\*(p + halfOrder) = t;

t = \*(p + halfOrder \* order);

\*(p + halfOrder \* order) = \*(p + halfOrder \* order + halfOrder);

\*(p + halfOrder \* order + halfOrder) = t;

}

}

}

char \* readFile(string fileName) {

ifstream file;

file.open(fileName);

if (!file.is\_open()) {

cout << "Error opening file! \n";

cin.ignore(32767, '\n');

cin.clear();

return 0;

}

char \*pointerArr = new char[sizeof(file)];

while (!file.eof())

{

file.getline(pointerArr, sizeof(file));

}

cout << pointerArr;

return pointerArr;

}

string textCleaner(string inputStr) {

int size = inputStr.size();

bool isChanged;

do {

isChanged = false;

for (int i = 1; i < size; i++) {

if ( // Удаляем лишние символы

(isSymbol(inputStr[i])) && (isSymbol(inputStr[i - 1]))

&& (inputStr[i] != '.')

) {

inputStr = deleteChar(inputStr, size, i);

size--;

isChanged = true;

}

if ( // Редкие сочетания типа "пробел-знак препинания-пробел"? кроме тире

(i >= 2)

&& (inputStr[i] == ' ')

&& (isSymbol(inputStr[i - 1]))

&& (inputStr[i - 1] != '-')

&& (inputStr[i - 2] == ' ')

) {

inputStr = deleteChar(inputStr, size, i - 1);

size--;

isChanged = true;

}

}

for (int i = 2; i < size; i++) { // Редкие случаи с точками

if (

(inputStr[i] == '.')

&& ((isSymbol(inputStr[i - 1])) || (inputStr[i - 1] == ' '))

&& (inputStr[i - 1] != '.')

&& (inputStr[i - 1] != ')')

&& (inputStr[i - 1] != '"')

&& (inputStr[i - 1] != ']')

&& (inputStr[i - 1] != '}')

&& (inputStr[i - 1] != '>')

) {

inputStr = deleteChar(inputStr, size, i);

size--;

isChanged = true;

}

}

for (int i = 1; i < size; i++) {

if ((inputStr[i] == ' ') && (inputStr[i - 1] == ' ')) { // Удаляем лишние пробелы

inputStr = deleteChar(inputStr, size, i);

size--;

isChanged = true;

}

}

} while (isChanged);

for (int i = 1; i < size; i++) { // Исправление регистра

if (

(inputStr[i] >= 65) && (inputStr[i] <= 90)

&& (

((inputStr[i - 1] >= 97) && (inputStr[i - 1] <= 122))

|| ((inputStr[i - 1] >= 65) && (inputStr[i - 1] <= 90))

)

)

inputStr[i] = inputStr[i] + 32; // Замена прописной буквы на строчную

}

return inputStr;

}

bool isSymbol (char c) { // Если элемент является знаком препинания

if (

((c >= 33) && (c <= 47))

|| ((c >= 58) && (c <= 64))

|| ((c >= 91) && (c <= 96))

|| (c >= 123)

)

return true;

else

return false;

}

string deleteChar(string inputStr, int size, int pos) { // Удаление элемента путём смещения массива влево. В теории можно реализовать более эффективный алгоритм, смещая до ближайшего вхождения условия, но ещё нескольких вечеров на перелопачивание алгоритма нет, да и задачи на написание максимально эффективного алгоритма нет.

for (int i = pos; i < (size - 1); i++)

inputStr[i] = inputStr[i + 1];

inputStr.erase(size - 1);

return inputStr;

}

void reverseOutput(string inputStr) {

int size = inputStr.size();

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

if (inputStr[i] == ' ') {

for (int j = i + 1; (j < size) && (inputStr[j] != ' '); j++)

cout << inputStr[j];

cout << ' ';

}

else

if (i == 0)

for (int j = i; inputStr[j] != ' '; j++)

cout << inputStr[j];

}

cout << endl;

}

int countOfWords(string inputStr) {

int size = inputStr.size();

int count = 1;

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

if (inputStr[i] == ' ')

count++;

return count;

}

void wordOutput(string inputStr, int size, int pos) {

for (int i = pos; (i < size) && (inputStr[i] != ' '); i++) {

cout << inputStr[i];

}

}

int longestWord(string inputStr) {

int arrSize = inputStr.size();

int maxWordSize = 0;

int wordSize;

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

wordSize = 0;

do {

wordSize++;

if (wordSize > maxWordSize)

maxWordSize = wordSize;

i++;

} while ((inputStr[i] != ' ') && (i < arrSize));

}

return maxWordSize;

}

int linearSubscringSearch(string inputStr, string substr) {

int mainSize = inputStr.size();

int subSize = substr.size();

bool isFounded = false;

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

int pos = i;

for (int j = 0; ((inputStr[i] == substr[j]) && (j < subSize)); j++, i++) {

if (j == (subSize - 1)) {

isFounded = true;

return pos;

}

}

}

if (!isFounded)

return -1;

}

int bouyerMooreSubstringSearch(string mainStr, string subStr)

{

int mainSize = subStr.size();

int subSize = mainStr.size();

int badchar[256];

badCharHeuristic(subStr, mainSize, badchar);

int s = 0;//s is shift of the pattern with respect to text

while (s <= (subSize - mainSize))

{

int j = mainSize - 1;

while (j >= 0 && subStr[j] == mainStr[s + j])

j--;

if (j < 0)

{

return s;

}

else

s += max(1, j - badchar[mainStr[s + j]]);

}

return -1;

}

void badCharHeuristic(string str, int size, int badchar[256])

{

int i;

//Initialize all occurrences as -1

for (i = 0; i < 256; i++)

badchar[i] = -1;

//Fill the actual value of last occurrence of a character

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

badchar[(int)str[i]] = i;

}