Міністерство освіти і науки України

Національний технічний університет

«Харківський політехнічний інститут»

Факультет комп’ютерних наук

ЗВІТ

про виконання лабораторної роботи №5

Варіант 7

за дисципліною

«Алгоритми і структури даних»

Виконав:

студент групи КН-320Д

Кулдошин Є.І.

Перевірила:

старший викладач

Мошко Є.О.

Харків 2022

Мета: ознайомитися із основними способами написань алгоритмів сортировок та асимптотичні алгоритми, та особливостями їх програмної реалізації, також ознайомитись із основними алгоритмами пошуку даних. Набути практичних навичок роботи з алгоритмами пошуку даних.

Умова задачі:

1. Визначити масив, в якому буде виконуватися пошук. Використовувати масив, створений в роботі №4.

2. Розробити функції лінійного пошуку в масиві, бінарного пошуку в масиві. Для забезпечення бінарного пошуку в масиві використовувати найбільш ефективний алгоритм сортування, визначений при виконанні лабораторної роботі №4:

− ввести інформацію в масив з файлу;

− виконати лінійний пошук в масиві;

− виконати пошук з бар'єром;

− упорядкувати елементи масиву функцією сортування і виконати бінарний пошук.

3. Розробити функції для пошуку підрядка в рядку:

− виконати прямий пошук підрядка;

− виконати алгоритм Кнута-Морріса-Пратта;

− виконати алгоритм Бойера-Мура;

− виконати алгоритм Рабина – Карпа.

4. Дослідити складність алгоритмів. Провести асимптотичний аналіз алгоритмів пошуку та зробити висновки.

− для порівняння алгоритмів пошуку виконати наступні кроки;

− створити таблицю асимптотичних оцінок трудомісткості алгоритмів в кращому, середньому, гіршому випадках;

− розставити лічильники операцій у функціях пошуку;

− провести експеримент, визначити середню кількість операцій для різних алгоритмів, побудувати графіки;

− створити таблиці і представити графіки експериментальних оцінок алгоритмів.

Для всіх випадків перевірити варіанти успішного і неуспішного пошуку. Зробити висновки. Зберегти файл з текстом програми для подальших робіт.

Заборонено використовувати готові реалізації структур даних (наприклад, STL).

Опис усіх етапів розробки:

1 – Аналіз умови завдання.

2 – Повторне прочитання лекційного матеріалу, повторення методів створення алгоритмів сортировок (Порозрядна).

3 – Створення проекту і прив'язка до сервісу GitHub.

4 – Створення файлів для роботи з проектом (Main.cpp, Main.h, Sort.cpp).

5 – Програма буде працювати з роздільною компіляцією, це дозволяє коду бути більш читабельним.

6 – У файлі Main.cpp будуть описані русифікатори, сервісні змінні (для роботи з меню), цикл while де буде описано меню програми і вхід у функції.

7 – Файл Main.h створенний для підключення потрібних бібліотек (windows.h, iostream, timer.h, conio.h, fstream, string, chrono), підключення простір імен std, створення структури даних (Data) де описуємо розміра масива (size). Створив змінну search, в неї ми записуємо елемент який потрібно знайти в масиві. Створив define G, NO\_OF\_CHARS як цілочисленну константу для розміру масивів.

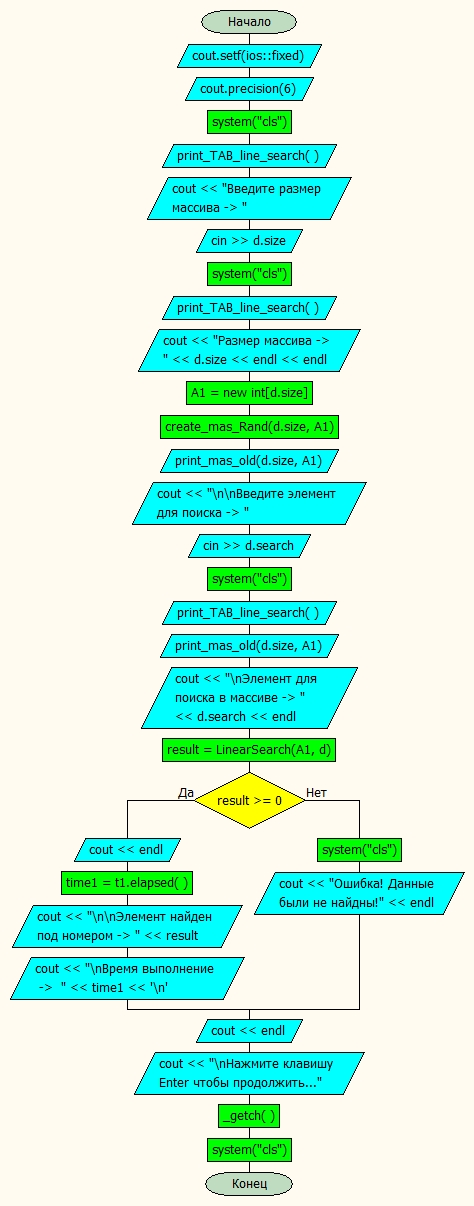
8 – У файлі Sort.cpp описана вся логіка функцій роботи зі створенням масиву, створення строк, сортування, пошуку.

Аналіз задачі: При ознайомленні та розробці коду я навчився і застосував знання зі створенням алгоритму сортування, пошуку та особливостями їх програмної реалізації.

Структура основних вхідних і вихідних даних: В якості вступних даних, система роботи з алгоритмом сортировки і пошуку в масиві використовує змінні цілочисленного типу, що передаються в алгоритм сортировок або пошуку. А система роботи з алгоритмами пошуку строк, ми використовуємо строки для відображення тексту. А вихідні дані являють собою текстові рядки і символьну інформацію, що виводиться у вигляді списку елементів.

Алгоритм розв’язку завдання:

1. Основна функція лінійного пошуку



void Fun\_line\_search() {

Data d;

int result;

cout.setf(ios::fixed);

cout.precision(6);

system("cls");

print\_TAB\_line\_search();

cout << "Введите размер массива -> ";

cin >> d.size;

system("cls");

/////////////////////////////////

print\_TAB\_line\_search();

cout << "Размер массива -> " << d.size << endl << endl;

int\* A1 = new int[d.size];

create\_mas\_Rand(d.size, A1);

print\_mas\_old(d.size, A1);

cout << "\n\nВведите элемент для поиска -> ";

cin >> d.search;

system("cls");

/////////////////////////////////

print\_TAB\_line\_search();

print\_mas\_old(d.size, A1);

cout << "\nЭлемент для поиска в массиве -> " << d.search << endl;

result = LinearSearch(A1, d);

if (result >= 0) {

Timer t1;

cout << endl;

double time1 = t1.elapsed();

cout << "\n\nЭлемент найден под номером -> " << result;

cout << "\nВремя выполнение -> " << time1 << '\n';

}

else

{

system("cls");

cout << "Ошибка! Данные были не найдны!" << endl;

}

cout << endl;

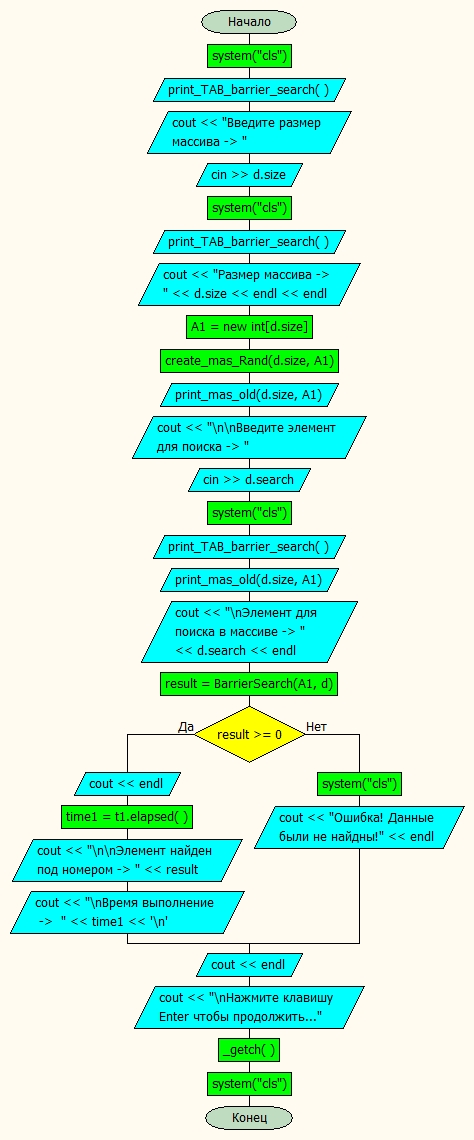
cout << "\nНажмите клавишу Enter чтобы продолжить...";

\_getch();

system("cls");

}

1. Основна функція бар'єрного пошуку



void Fun\_barrier\_search() {

Data d;

int result;

system("cls");

print\_TAB\_barrier\_search();

cout << "Введите размер массива -> ";

cin >> d.size;

system("cls");

/////////////////////////////////

print\_TAB\_barrier\_search();

cout << "Размер массива -> " << d.size << endl << endl;

int\* A1 = new int[d.size];

create\_mas\_Rand(d.size, A1);

print\_mas\_old(d.size, A1);

cout << "\n\nВведите элемент для поиска -> ";

cin >> d.search;

system("cls");

/////////////////////////////////

print\_TAB\_barrier\_search();

print\_mas\_old(d.size, A1);

cout << "\nЭлемент для поиска в массиве -> " << d.search << endl;

result = BarrierSearch(A1, d);

if (result >= 0) {

Timer t1;

cout << endl;

double time1 = t1.elapsed();

cout << "\n\nЭлемент найден под номером -> " << result;

cout << "\nВремя выполнение -> " << time1 << '\n';

}

else

{

system("cls");

cout << "Ошибка! Данные были не найдны!" << endl;

}

cout << endl;

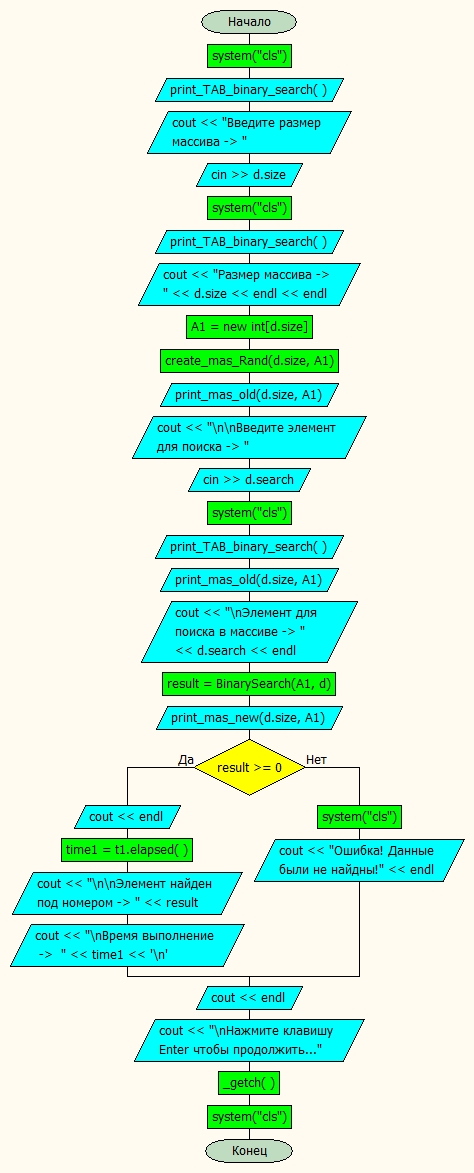
cout << "\nНажмите клавишу Enter чтобы продолжить...";

\_getch();

system("cls");

}

1. Основна функція бінарного пошуку



void Fun\_binary\_search() {

Data d;

int result;

system("cls");

print\_TAB\_binary\_search();

cout << "Введите размер массива -> ";

cin >> d.size;

system("cls");

/////////////////////////////////

print\_TAB\_binary\_search();

cout << "Размер массива -> " << d.size << endl << endl;

int\* A1 = new int[d.size];

create\_mas\_Rand(d.size, A1);

print\_mas\_old(d.size, A1);

cout << "\n\nВведите элемент для поиска -> ";

cin >> d.search;

system("cls");

/////////////////////////////////

print\_TAB\_binary\_search();

print\_mas\_old(d.size, A1);

cout << "\nЭлемент для поиска в массиве -> " << d.search << endl;

result = BinarySearch(A1, d);

print\_mas\_new(d.size, A1);

if (result >= 0) {

Timer t1;

cout << endl;

double time1 =

t1.elapsed();

cout << "\n\nЭлемент найден под номером -> " << result;

cout << "\nВремя выполнение -> " << time1 << '\n';

}

else

{

system("cls");

cout << "Ошибка! Данные были не найдны!" << endl;

}

cout << endl;

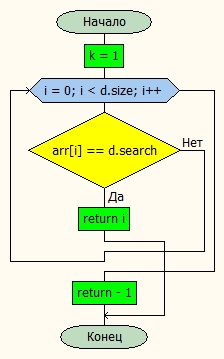
cout << "\nНажмите клавишу Enter чтобы продолжить...";

\_getch();

system("cls");

}

1. Лінійний пошук



int LinearSearch(int\* arr, Data d) {

int k = 1;

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

if (arr[i] == d.search) {

return i;

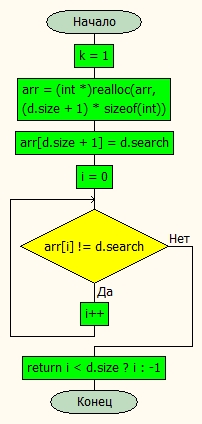
}

}

return -1;

}

1. Бар'єрний пошук



int BarrierSearch(int\* arr, Data d)

{

int k = 1;

arr = (int\*)realloc(arr, (d.size + 1) \* sizeof(int));

arr[d.size + 1] = d.search;

int i = 0;

while (arr[i] != d.search) {

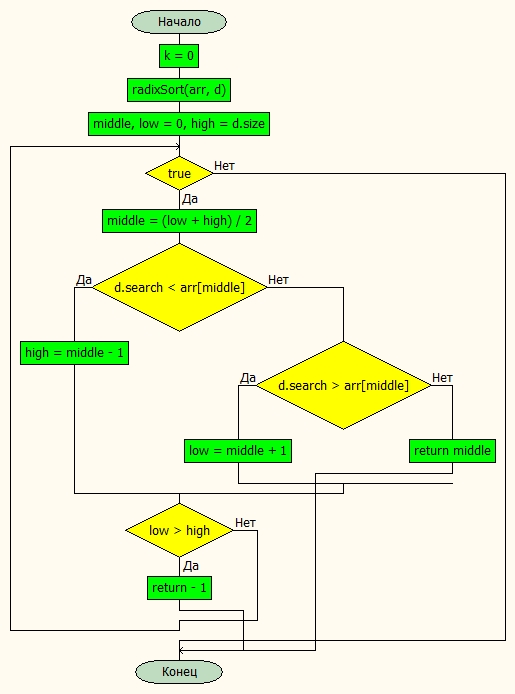
i++;

}

return i < d.size ? i : -1;

}

1. Бінарний пошук



int BinarySearch(int\* arr, Data d)

{

int k = 0;

radixSort(arr, d);

int middle,

low = 0,

high = d.size;

while (true) {

middle = (low + high) / 2;

if (d.search < arr[middle]) {

high = middle - 1;

}

else if (d.search > arr[middle]) {

low = middle + 1;

}

else {

return middle;

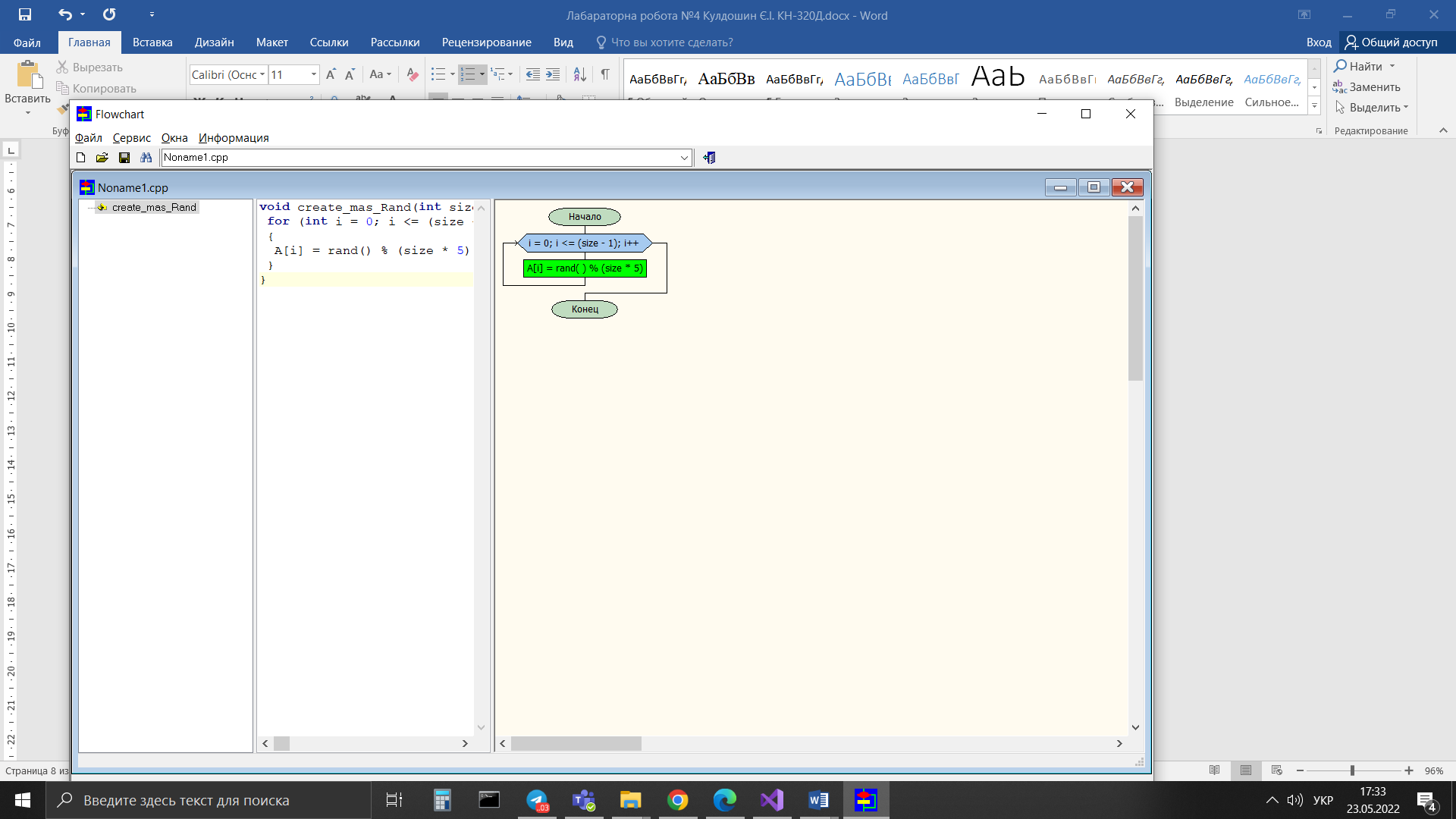
}

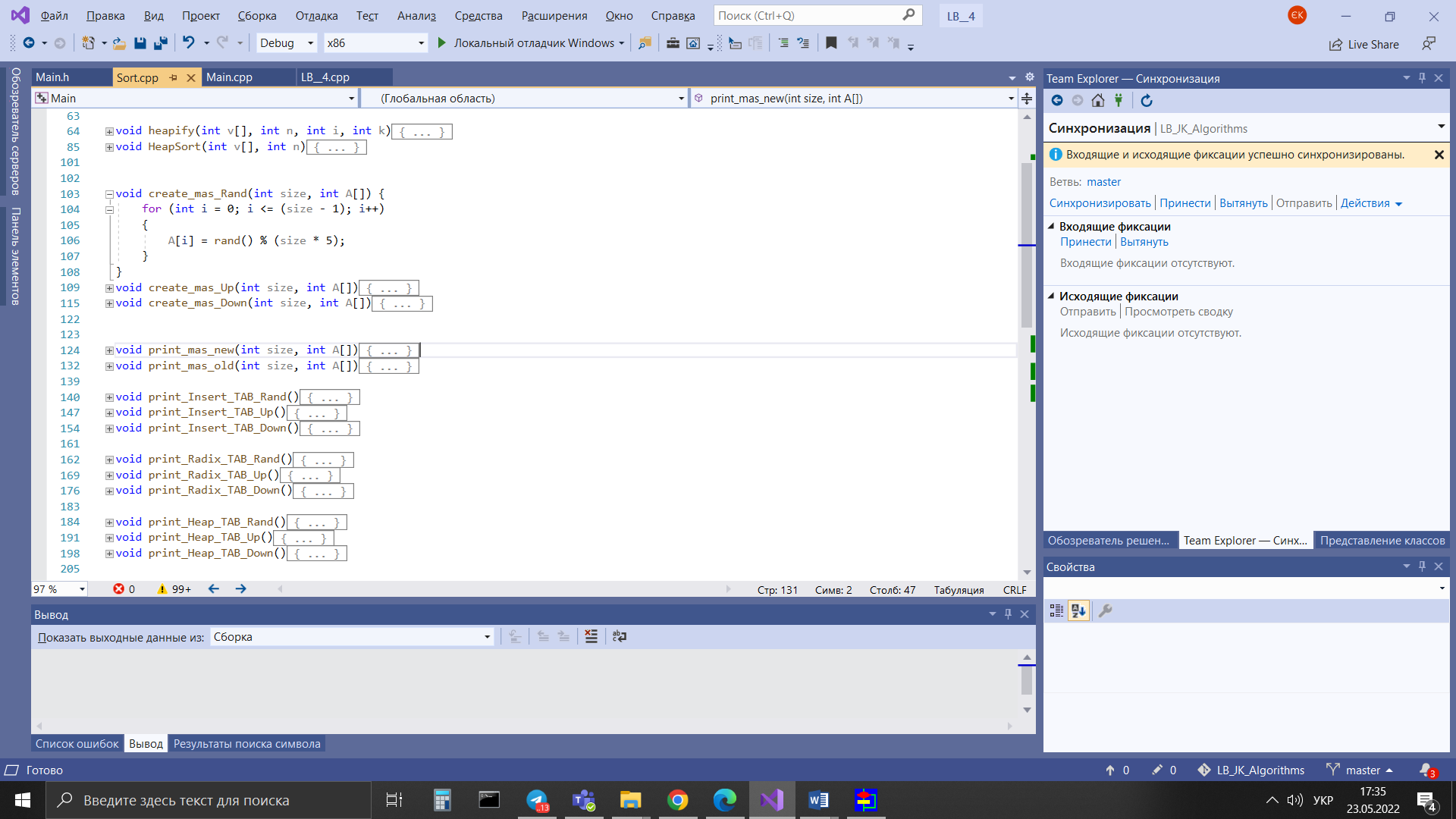
if (low > high)

return -1;

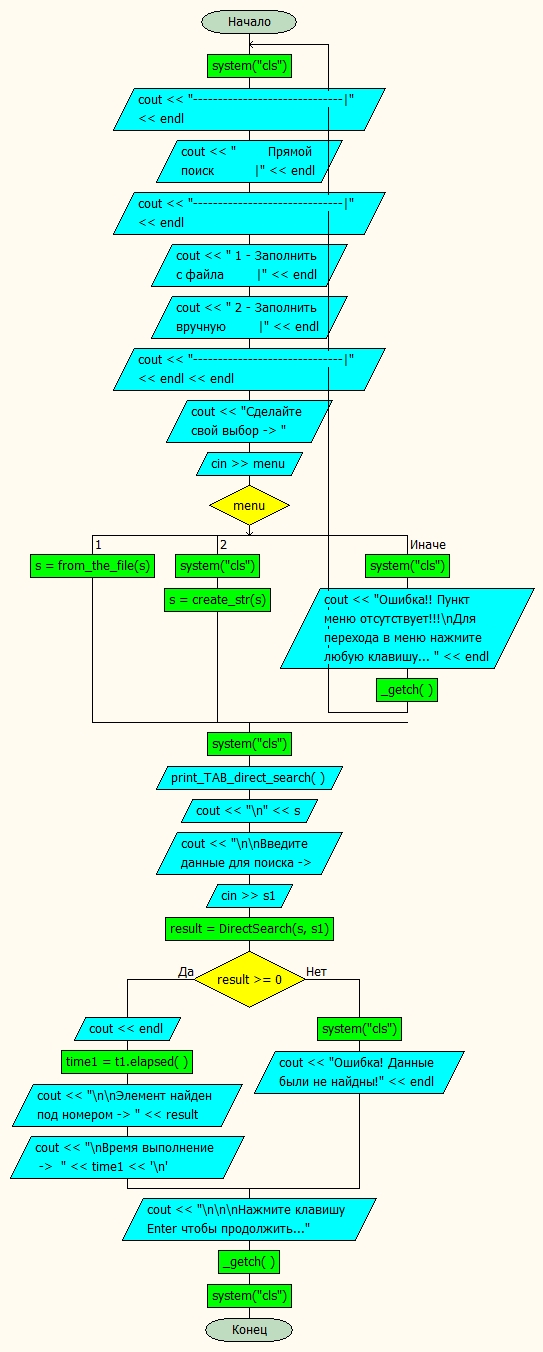
}}

1. Функція для створення рандомних елементів масива





1. Основна функція прямого пошук підрядка



void Fun\_STR\_direct\_search() {

int menu,result;

string s, s1;

link\_menu:

system("cls");

//print\_TAB\_direct\_search();

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

cout << " Прямой поиск |" << endl;

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

cout << " 1 - Заполнить с файла |" << endl;

cout << " 2 - Заполнить вручную |" << endl;

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

cout << "Сделайте свой выбор -> ";

cin >> menu;

switch (menu) {

case 1:

s = from\_the\_file(s);

break;

case 2:

system("cls");

s = create\_str(s);

break;

default:

system("cls");

cout << "Ошибка!! Пункт меню отсутствует!!!\nДля перехода в меню нажмите любую клавишу... " << endl;

\_getch();

goto link\_menu;

break;

}

system("cls");

print\_TAB\_direct\_search();

cout <<"\n"<< s;

cout << "\n\nВведите данные для поиска -> ";

cin >> s1;

result = DirectSearch(s, s1);

if (result >= 0) {

Timer t1;

cout << endl;

double time1 = t1.elapsed();

cout << "\n\nЭлемент найден под номером -> " << result;

cout << "\nВремя выполнение -> " << time1 << '\n';

}

else

{

system("cls");

cout << "Ошибка! Данные были не найдны!" << endl;

}

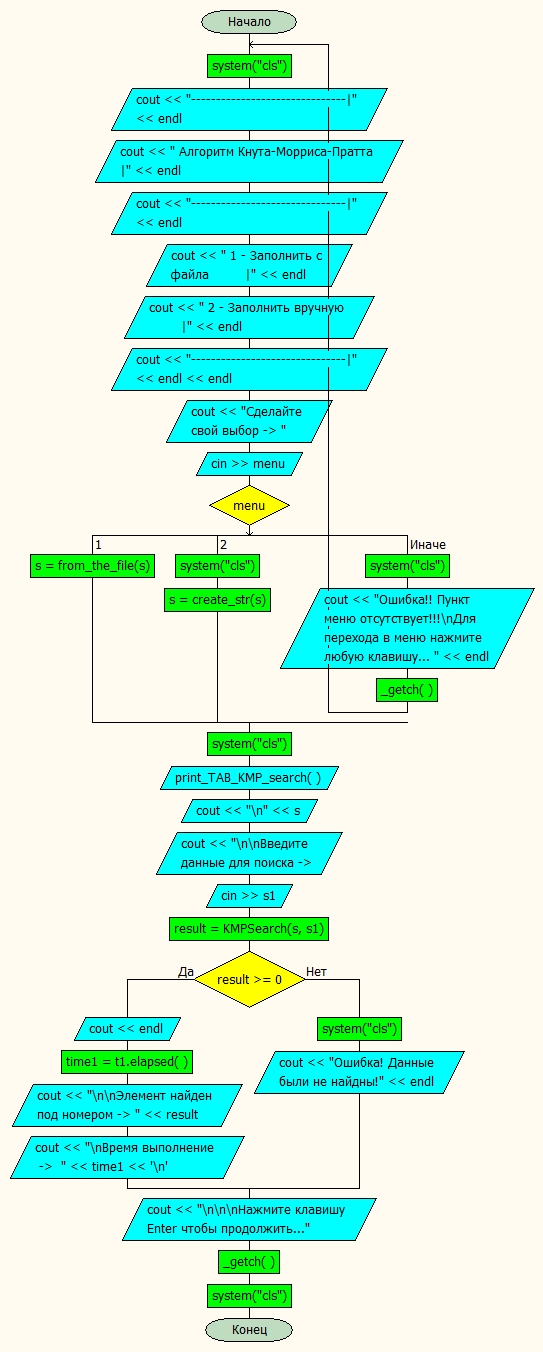
cout << "\n\n\nНажмите клавишу Enter чтобы продолжить...";

\_getch();

system("cls");

}

1. Основна функція алгоритму Кнута-Морріса-Пратта



void Fun\_STR\_KMP\_search() {

int menu, result;

string s, s1;

link\_menu:

system("cls");

//print\_TAB\_direct\_search();

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

cout << " Алгоритм Кнута-Морриса-Пратта |" << endl;

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

cout << " 1 - Заполнить с файла |" << endl;

cout << " 2 - Заполнить вручную |" << endl;

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

cout << "Сделайте свой выбор -> ";

cin >> menu;

switch (menu) {

case 1:

s = from\_the\_file(s);

break;

case 2:

system("cls");

s = create\_str(s);

break;

default:

system("cls");

cout << "Ошибка!! Пункт меню отсутствует!!!\nДля перехода в меню нажмите любую клавишу... " << endl;

\_getch();

goto link\_menu;

break;

}

system("cls");

print\_TAB\_KMP\_search();

cout << "\n" << s;

cout << "\n\nВведите данные для поиска -> ";

cin >> s1;

result = KMPSearch(s, s1);

if (result >= 0) {

Timer t1;

cout << endl;

double time1 = t1.elapsed();

cout << "\n\nЭлемент найден под номером -> " << result;

cout << "\nВремя выполнение -> " << time1 << '\n';

}

else

{

system("cls");

cout << "Ошибка! Данные были не найдны!" << endl;

}

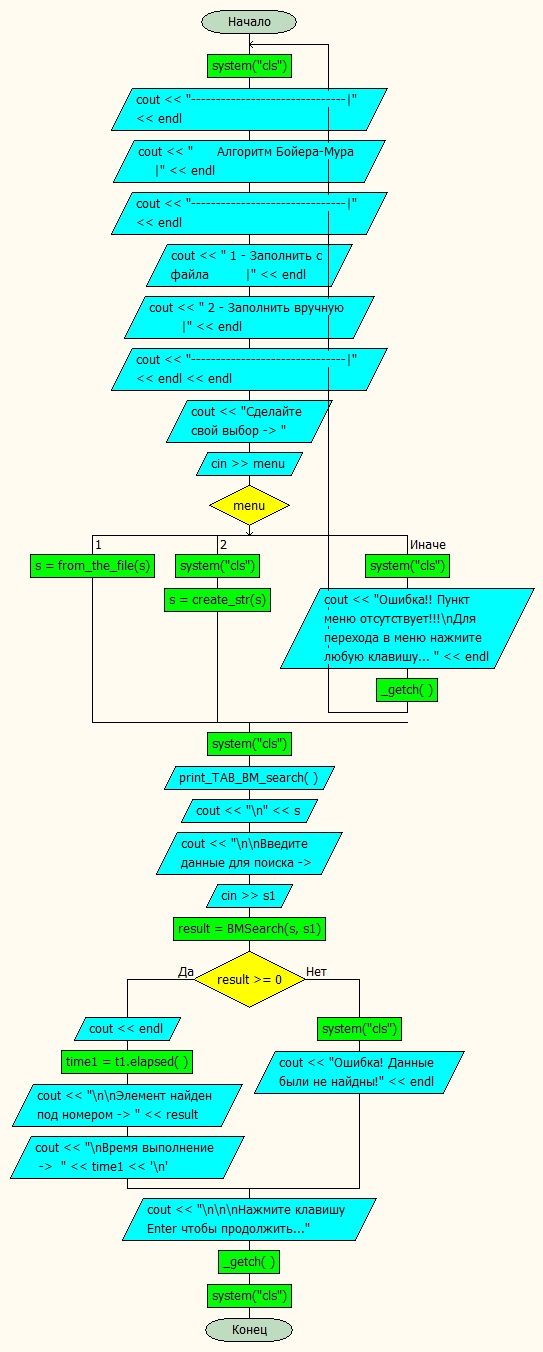
cout << "\n\n\nНажмите клавишу Enter чтобы продолжить...";

\_getch();

system("cls");

}

1. Основна функція алгоритму Бойера-Мура



void Fun\_STR\_BM\_search() {

int menu, result;

string s, s1;

link\_menu:

system("cls");

//print\_TAB\_direct\_search();

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

cout << " Алгоритм Бойера-Мура |" << endl;

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

cout << " 1 - Заполнить с файла |" << endl;

cout << " 2 - Заполнить вручную |" << endl;

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

cout << "Сделайте свой выбор -> ";

cin >> menu;

switch (menu) {

case 1:

s = from\_the\_file(s);

break;

case 2:

system("cls");

s = create\_str(s);

break;

default:

system("cls");

cout << "Ошибка!! Пункт меню отсутствует!!!\nДля перехода в меню нажмите любую клавишу... " << endl;

\_getch();

goto link\_menu;

break;

}

system("cls");

print\_TAB\_BM\_search();

cout << "\n" << s;

cout << "\n\nВведите данные для поиска -> ";

cin >> s1;

result = BMSearch(s, s1);

if (result >= 0) {

Timer t1;

cout << endl;

double time1 = t1.elapsed();

cout << "\n\nЭлемент найден под номером -> " << result;

cout << "\nВремя выполнение -> " << time1 << '\n';

}

else

{

system("cls");

cout << "Ошибка! Данные были не найдны!" << endl;

}

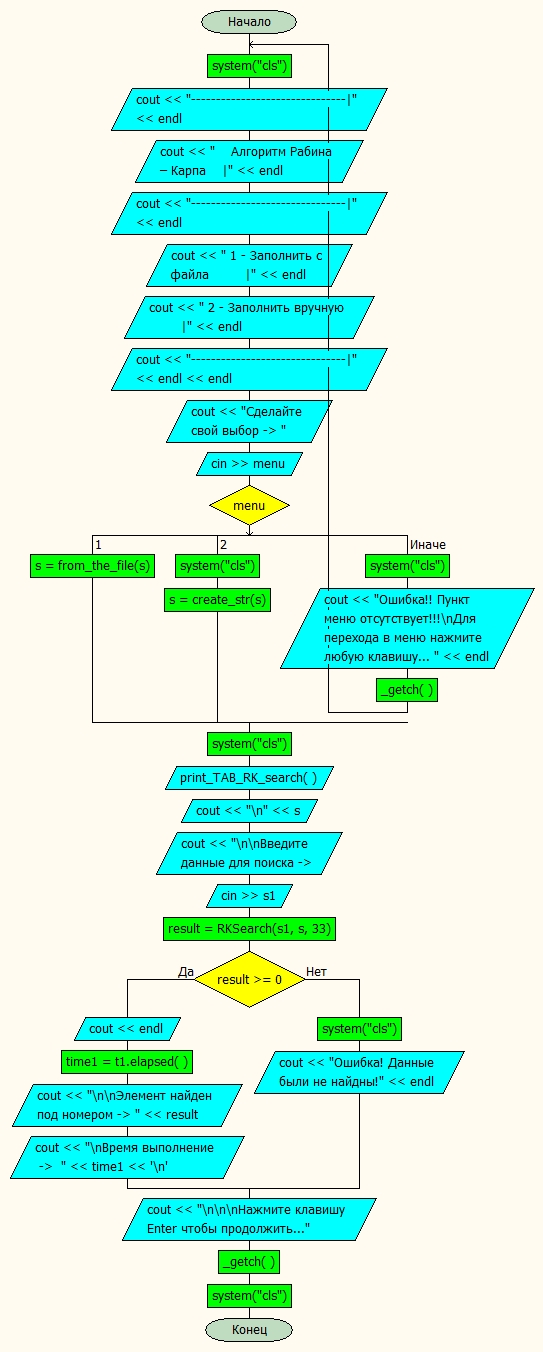
cout << "\n\n\nНажмите клавишу Enter чтобы продолжить...";

\_getch();

system("cls");

}

1. Основна функція алгоритму Рабина – Карпа



void Fun\_STR\_RK\_search() {

int menu, result;

string s, s1;

link\_menu:

system("cls");

//print\_TAB\_direct\_search();

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

cout << " Алгоритм Рабина – Карпа |" << endl;

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

cout << " 1 - Заполнить с файла |" << endl;

cout << " 2 - Заполнить вручную |" << endl;

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

cout << "Сделайте свой выбор -> ";

cin >> menu;

switch (menu) {

case 1:

s = from\_the\_file(s);

break;

case 2:

system("cls");

s = create\_str(s);

break;

default:

system("cls");

cout << "Ошибка!! Пункт меню отсутствует!!!\nДля перехода в меню нажмите любую клавишу... " << endl;

\_getch();

goto link\_menu;

break;

}

system("cls");

print\_TAB\_RK\_search();

cout << "\n" << s;

cout << "\n\nВведите данные для поиска -> ";

cin >> s1;

result = RKSearch(s1, s, 33);

if (result >= 0) {

Timer t1;

cout << endl;

double time1 = t1.elapsed();

cout << "\n\nЭлемент найден под номером -> " << result;

cout << "\nВремя выполнение -> " << time1 << '\n';

}

else

{

system("cls");

cout << "Ошибка! Данные были не найдны!" << endl;

}

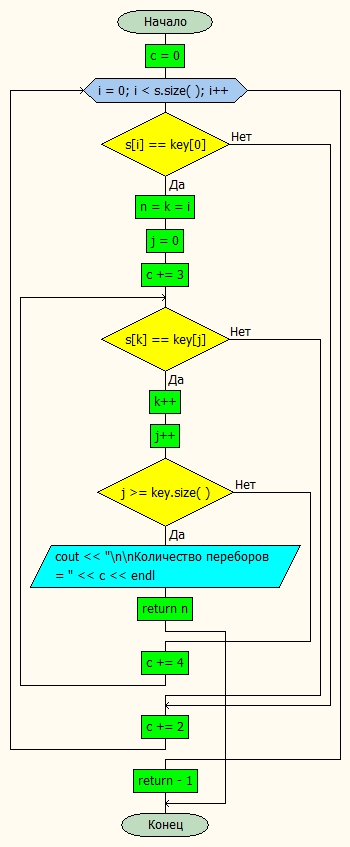
cout << "\n\n\nНажмите клавишу Enter чтобы продолжить...";

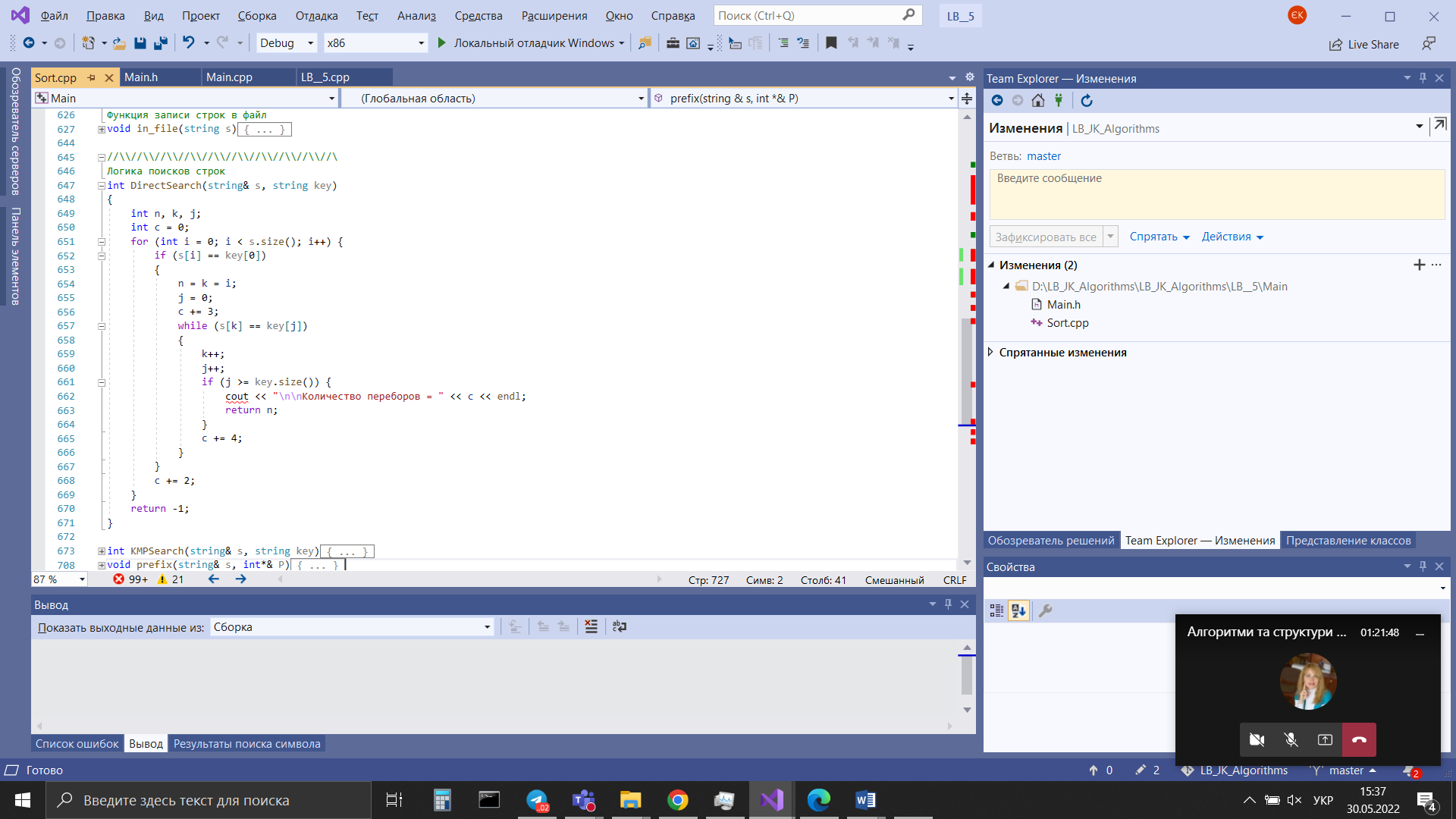
\_getch();

system("cls");

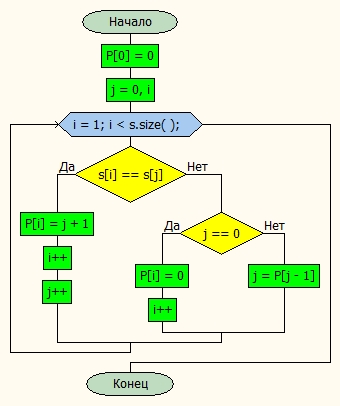
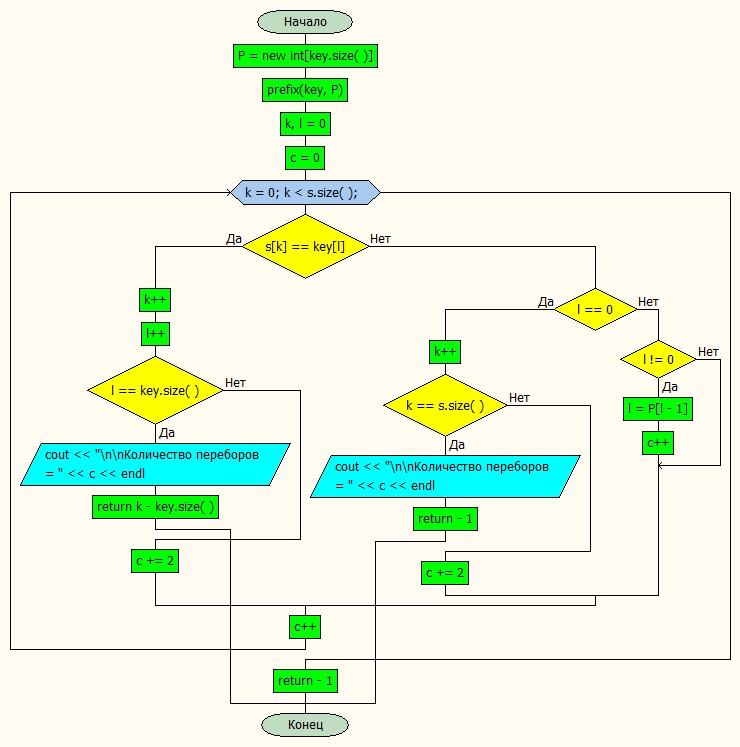
}

1. Прямий пошук підрядка





1. Алгоритм Кнута-Морріса-Пратта



int KMPSearch(string& s, string key)

{

int\* P = new int[key.size()];

prefix(key, P);

int k,

l = 0;

int c = 0;

for (k = 0; k < s.size(); ) {

if (s[k] == key[l])

{

k++;

l++;

if (l == key.size()) {

cout << "\n\nКоличество переборов = " << c << endl;

return k - key.size();

}

c += 2;

}

else if (l == 0)

{

k++;

if (k == s.size()) {

cout << "\n\nКоличество переборов = " << c << endl;

return -1;

}

c += 2;

}

else if (l != 0) {

l = P[l - 1];

c++;

}

c++;

}

return -1;

}

void prefix(string& s, int\*& P)

{

P[0] = 0;

int j = 0,

i;

for (i = 1; i < s.size();)

if (s[i] == s[j])

{

P[i] = j + 1;

i++;

j++;

}

else if (j == 0)

{

P[i] = 0;

i++;

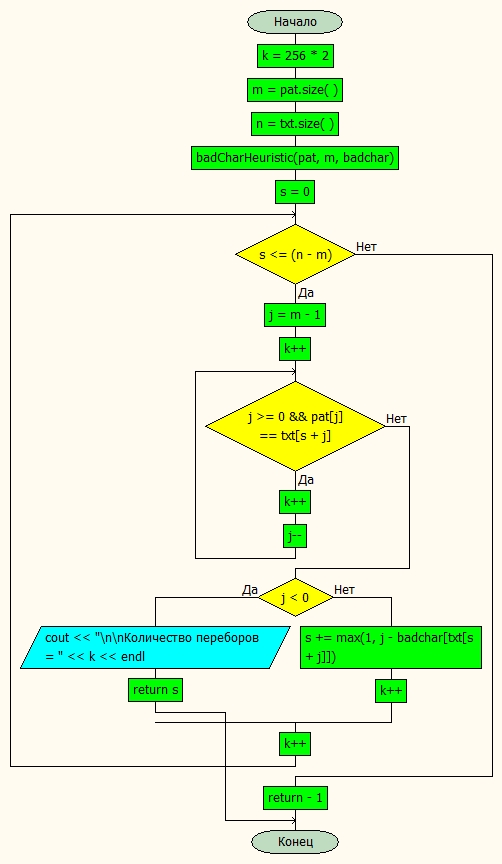
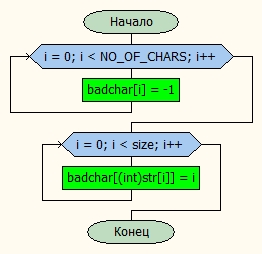
}

else

j = P[j - 1];

}

1. Алгоритм Бойера-Мура



int BMSearch(string txt, string pat)

{

int k = 256 \* 2;

int m = pat.size();

int n = txt.size();

int badchar[NO\_OF\_CHARS];

badCharHeuristic(pat, m, badchar);

int s = 0;

while (s <= (n - m))

{

int j = m - 1;

k++;

while (j >= 0 && pat[j] == txt[s + j]) {

k++;

j--;

}

if (j < 0)

{

cout << "\n\nКоличество переборов = " << k << endl;

return s;

//s += (s + m < n) ? m - badchar[txt[s + m]] : 1;

}

else {

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

k++;

}

k++;

}

return -1;

}

void badCharHeuristic(string str, int size, int badchar[NO\_OF\_CHARS])

{

int i;

for (i = 0; i < NO\_OF\_CHARS; i++)

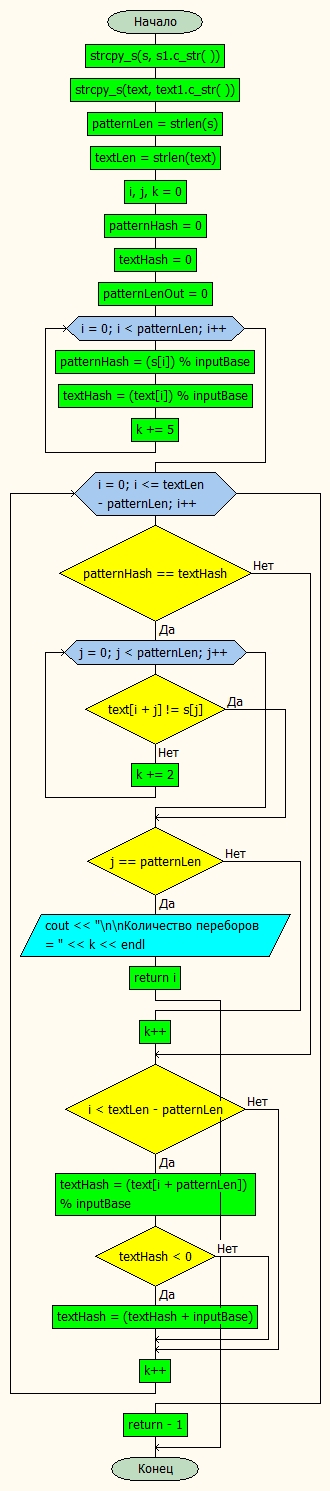
badchar[i] = -1;

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

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

}

1. Алгоритм Рабина – Карпа



int RKSearch(string s1, string text1, int inputBase) { //n, n, nm //+m

char s[10001];

strcpy\_s(s, s1.c\_str());

char text[10001];

strcpy\_s(text, text1.c\_str());

int patternLen = strlen(s);

int textLen = strlen(text);

int i, j, k = 0;

int patternHash = 0;

int textHash = 0;

int patternLenOut = 0;

for (i = 0; i < patternLen; i++) {

patternHash = (s[i]) % inputBase;

textHash = (text[i]) % inputBase;

k += 5;

}

for (i = 0; i <= textLen - patternLen; i++) {

if (patternHash == textHash) {

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

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

break;

k += 2;

}

if (j == patternLen) {

cout << "\n\nКоличество переборов = " << k << endl;

return i;

}

k++;

}

if (i < textLen - patternLen) {

textHash = (text[i + patternLen]) % inputBase;

//k += 3;

if (textHash < 0) {

textHash = (textHash + inputBase);

//k+=3;

}

//k += 2;

}

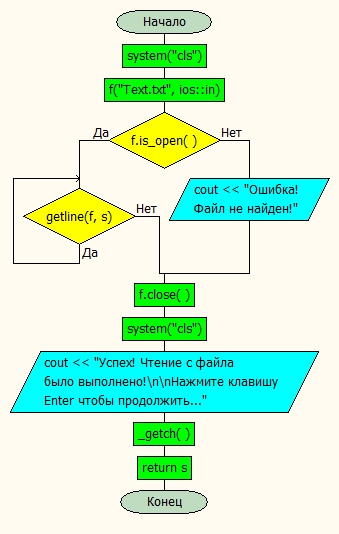
k++;

}

return -1;

}

1. Створення даних з файлу



string from\_the\_file(string s) {

system("cls");

fstream f("Text.txt", ios::in); //Переменная для работы с файлом, открыли файл

if (f.is\_open()) { //Проверка на открытие файла

while (getline(f, s))

{

f >> s;

}

}

else {

cout << "Ошибка! Файл не найден!";

}

f.close(); //Закрыли файл

system("cls");

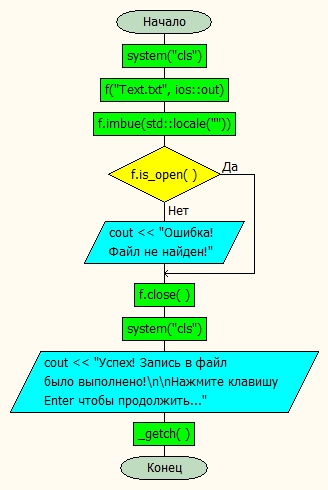
cout << "Успех! Чтение с файла было выполнено!\n\nНажмите клавишу Enter чтобы продолжить...";

\_getch();

return s;

}

1. Занесення даних до файлу



void in\_file(string s) {

system("cls");

fstream f("Text.txt", ios::out); //Переменная для работы с файлом, открыли файл

f.imbue(std::locale(""));

if (f.is\_open()) { //Проверка на открытие файла

f << s;

}

else {

cout << "Ошибка! Файл не найден!";

}

f.close(); //Закрыли файл

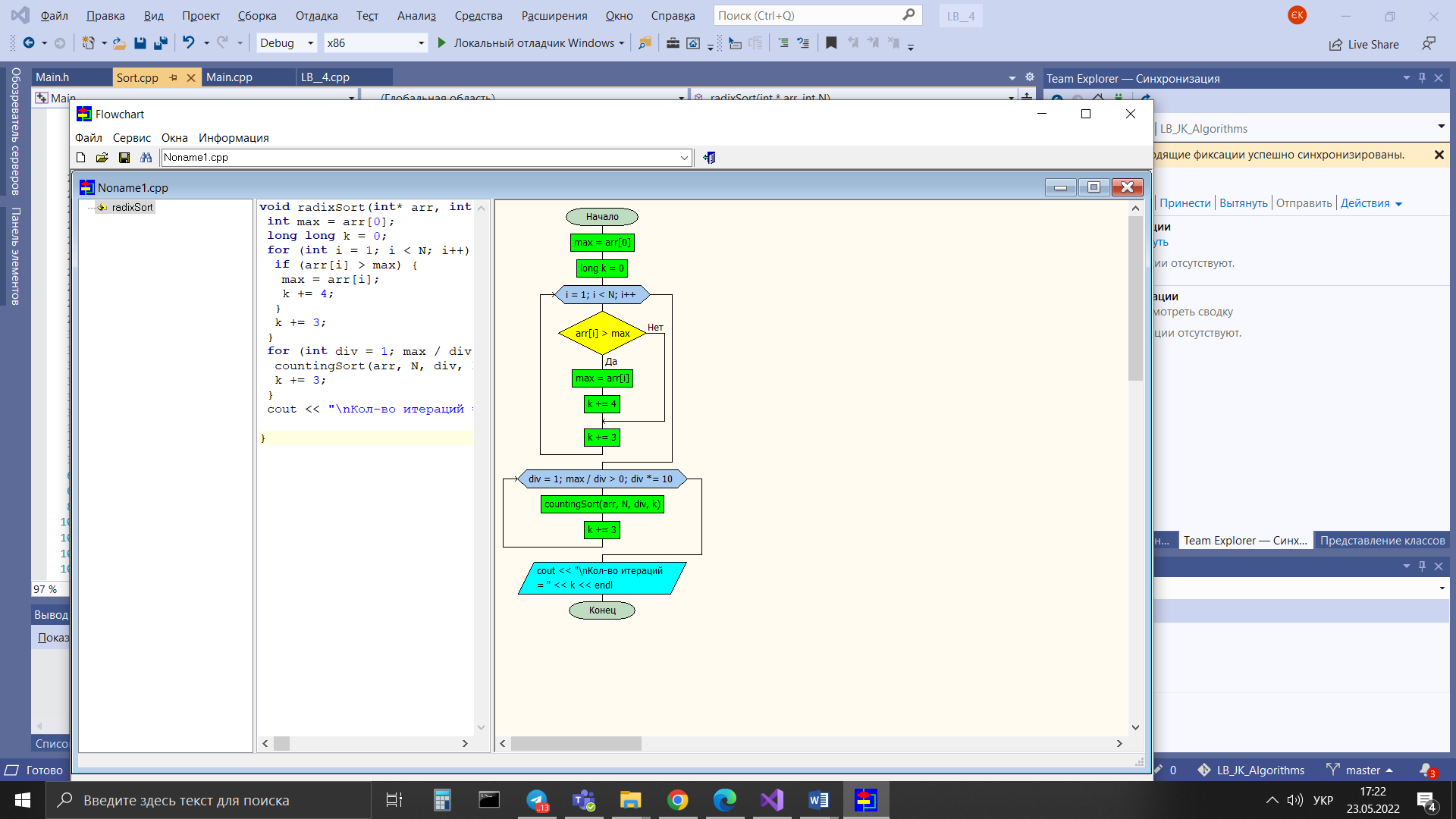
system("cls");

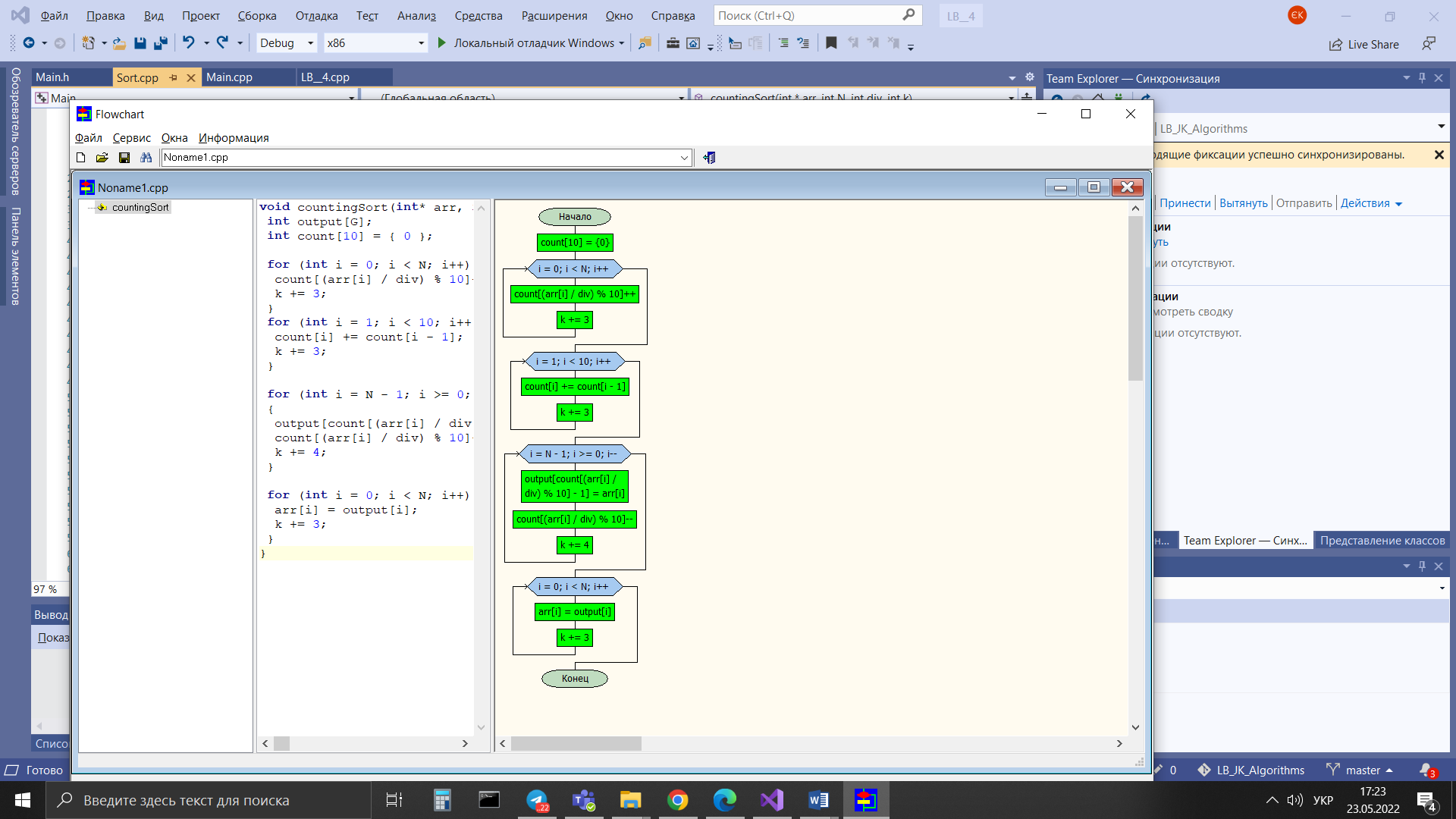
cout << "Успех! Запись в файл было выполнено!\n\nНажмите клавишу Enter чтобы продолжить...";

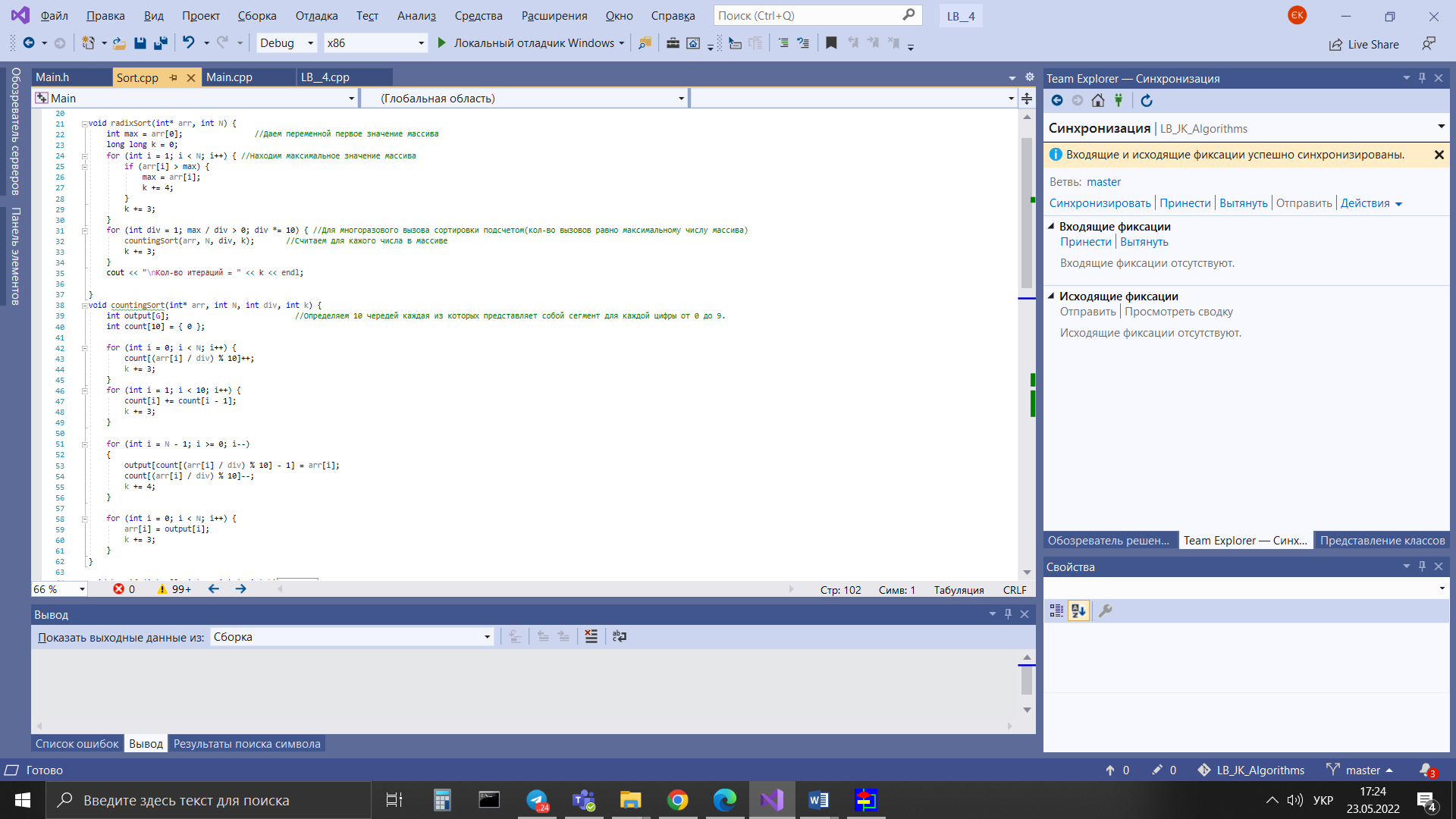
\_getch();

}

1. Алгоритм порозрядного сортування







Таблиці замірів робіт алгоритмів пошуку в масиві:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Неупорядкований масив | | | | | | |
|  | Кількість елементів в масиві | | | | | |
| 10 | 100 | 1000 | 10000 | 20000 | 30000 |
| Лінійний  пошук | 0.000043 | 0.000054 | 0.000560 | 0.000756 | 0.000529 | 0.000744 |
| Бар’єрний  Пошук | 0.000043 | 0.000043 | 0.000565 | 0.000603 | 0.000781 | 0.002814 |
| Бінарний пошук + порозрядне сортування | 0.000044 | 0.000054 | 0.000072 | 0.000072 | 0.000075 | 0.000146 |

Таблиця 1 – Неупорядкований масив для пошуку різними алгоритмами (заповнений випадковими числами).

|  |  |
| --- | --- |
| Заповнення с файлу | |
|  | Кількість символів в файлі |
| 165 |
| Прямий пошук  підрядка | 0.000122 |
| Алгоритм  Кнута – Морріса – Пратта | 0.000121 |
| Алгоритм  Бойера – Мура | 0.000044 |
| Алгоритм  Рабина – Карпа | 0.0001106 |

Таблиця 2 – Підрядок заповненний з файлу для пошуку різними алгоритмами (заповнений реченням з 165 символів).

Графіки замірів робіт алгоритмів пошуку:

Графік 1 – Алгоритми пошуку елементів масиву (заповнений випадковими числами).

Графік 2 – Алгоритми пошуку підрядка в рядку(заповнений рядок з файлу).

На діаграмах зображені результати порівнянь трьох алгоритмів, які були заповнені рандомно:

Найшвидший пошук з двох запропонованих алгоритмів це Лінійний пошук, так як витрачає саме мінімальний час роботи. При обробці 30 000 елементів, які були заповнені рандомним способом, алгоритму знадобилося лише 0.000744 секунд (Графік 1). І для порівняння візьмемо пошук Бар‘єром, для обробки точно такої ж кількості елементів, програмі доводиться витрачати 0.002814 секунди, це практично в 3.8 разів більше ніж Лінійний метод пошуку.

Бінарний пошук:

Це найшвидший пошук який перед тим як находити елемент проводить сортування для масиву який був заповнений рандомно. Використовували порозрядне сортування, бо в минулій роботі ця сортировка показала себе як найшвидша. Бінарний пошук дав результати набагато кращі ніж минулі способи пошуку (Графік 1), 30000 елементів відпрацював за 0.000146 секунд.

Алгоритми пошуку підрядка в рядку:

Найшвидший пошук з трьох запропонованих алгоритмів це алгоритм Байера – Мура, так як витрачає саме мінімальний час роботи. При обробці елементів, які були заповнені з файлу способом, алгоритму знадобилося лише 0.000044 секунд (Графік 2). І для порівняння візьмемо пошук алгоритм прямого пошуку, для обробки точно такої ж кількості елементів, програмі доводиться витрачати 0.000122 секунди, це практично в 2.7 разів більше ніж алгоритм Байера - Мура.

Висновок:

Пошук – процес знаходження конкретної інформації в раніше створеному безлічі даних. Зазвичай дані являють собою записи, кожна з яких має хоча б один ключ.

У кожного алгоритму пошуку своя тимчасова і просторова складність. Використовувати можна будь-який з представлених алгоритмів в залежності від поставлених завдань. При ознайомленні і роботі з Лінійним, Бар’єрним, Бінарним пошуком я навчився і застосував знання в створенні масивів і пошуку елементів в ньому.

Також ознайомився і пропрацював такі алгоритми пошуку як: Прямий пошук підрядка, Алгоритм Кнута – Морріса – Пратта, Алгоритм Бойера – Мура, Алгоритм Рабина – Карпа. Але на мою суб'єктивну думку найкращим алгоритмом для пошуку елементів в масиві чисел, є бінарний пошук, но для використання цього метода потрібно для початку відсортувати елементи. А для пошуку рядка використав би алгоритм Бойера – Мура, це найліпший і швидший метод. З нею зручно працювати і вона найменше витрачає часу на обробку даних.