Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа №14

По дисциплине «Основы алгоритмизации и программирования»

На тему «Хеш-таблицы с цепочками»

Выполнил(а):

Студент(ка) 1 курса 8 группы

Семёнов Даниил Вячеславович

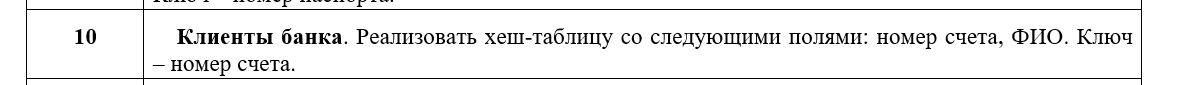
Преподаватель: асс. Андронова М.В.

Минск, 2024

5. В соответствии со своим вариантом разработать проект для условия из таблицы, представленной ниже. Построить ***хеш-таблицы с цепочками*** разного размера с коллизиями.

Для вариантов с 1 по 8 вычисление хеш-функции произвести по методу универсального хеширования. При вычислении хеш-функции для вариантов с 9 по 16 использовать алгоритм на основе исключающего ИЛИ для поля строки данных.

Исследовать время поиска информации.



Main.cpp

#include "Hash\_Chain.h"

#include <iostream>

using namespace std;

struct Client // структура клиента

{

int key;

char\* mas;

Client(int k, char\* z)

{

key = k;

mas = z;

}

Client()

{

key = 0;

mas = nullptr;

}

};

int hf(void\* d) // хэш-функция с исключающим ИЛИ

{

Client\* client = static\_cast<Client\*>(d);

int str = client->key;

int hash = 0;

hash ^= str;

return hash;

}

//-------------------------------

void Client\_print(listx::Element\* e) // вывод

{

std::cout << ((Client\*)e->data)->key << '-' << ((Client\*)e->data)->mas << " / ";

}

//-------------------------------

int main() // главная функция

{

setlocale(LC\_ALL, "rus");

int current\_size = 7;

cout << "Введите размер хеш-таблицы" << endl;

cin >> current\_size;

hashTC::Object H = hashTC::create(current\_size, hf);

int choice;

int k;

for (;;)

{

cout << "1 - вывод хеш-таблицы" << endl;

cout << "2 - добавление элемента" << endl;

cout << "3 - удаление элемента" << endl;

cout << "4 - поиск элемента" << endl;

cout << "0 - выход" << endl;

cout << "сделайте выбор" << endl;

cin >> choice;

switch (choice)

{

case 0: exit(0);

case 2: { Client\* a = new Client;

char\* str = new char[20];

cout << "Введите номер счёта: " << endl;

cin >> k;

a->key = k;

cout << "Введите ФИО" << endl;

cin >> str;

a->mas = str;

H.insert(a);

}

break;

case 1: H.Scan();

break;

case 3: { Client\* b = new Client;

cout << "Введите номер счёта: " << endl;

cin >> k;

b->key = k;

H.deleteByData(b);

}

break;

case 4: Client \* c = new Client;

cout << "Введите номер счёта: " << endl;

cin >> k;

c->key = k;

if (H.search(c) == NULL)

cout << "Клиент не найден" << endl;

else

{

cout << "Клиент с ключом: " << endl;

Client\_print(H.search(c));

cout << endl;

}

break;

}

}

return 0;

}

Hash\_Table.cpp  
  
#include "Hash\_Chain.h"

#include <iostream>

namespace hashTC

{

// функции

Object create(int size, int(\*f)(void\*))

{

return \*(new Object(size, f));

}

int Object::hashFunction(void\* data)

{

return (FunKey(data) % size);

};

bool Object::insert(void\* data)

{

return (Hash[hashFunction(data)].insert(data));

};

bool Object::deleteByData(void\* data)

{

return (Hash[hashFunction(data)].deleteByData(data));

};

listx::Element\* Object::search(void\* data)

{

return Hash[hashFunction(data)].search(data);

};

void Object::Scan()

{

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

{

Hash[i].scan();

std::cout << '\n';

}

};

}

Hash\_Chain.h  
#pragma once //Заголовочный файл Hash\_Chain.h

#include "Lists.h"

namespace hashTC

{

struct Object // структура хэш-таблицы

{

int size;

int(\*FunKey)(void\*);

listx::Object\* Hash;

Object(int siz, int(\*f)(void\*))

{

size = siz;

FunKey = f;

Hash = new listx::Object[size];

};

int hashFunction(void\* data);

bool insert(void\* data);

listx::Element\* search(void\* data);

bool deleteByData(void\* data);

void Scan();

};

Object create(int size, int(\*f)(void\*));

}

Lists.h

#pragma once //Заголовочный файл Lists.h

#define LISTNIL (Element\*)-1

namespace listx

{

struct Element // структура элемента списка

{

Element\* prev;

Element\* next;

void\* data;

Element(Element\* pr, void\* dat, Element\* nt)

{

prev = pr;

data = dat;

next = nt;

}

Element\* getNext()

{

return next;

};

Element\* getPrev()

{

return prev;

};

};

static Element\* NIL = nullptr;

struct Object // структура списка

{

Element\* head;

Object()

{

head = NIL;

};

Element\* getFirst()

{

return head;

};

Element\* getLast();

Element\* search(void\* data);

bool insert(void\* data);

bool deleteByElement(Element\* e);

bool deleteByData(void\* data);

void scan();

};

Object create();

}

#undef LISTNIL

Lists.cpp  
  
#include "Lists.h"

#include <iostream>

struct Client //элемент таблицы

{

int key;

char\* mas;

};

namespace listx

{

bool Object::insert(void\* data)

{

bool rc = NULL;

if (head == NULL)

head = new Element(NULL, data, head);

else

head = (head->prev = new Element(NULL, data, head));

return rc;

}

//-------------------------------

Element\* Object::search(void\* data)

{

Element\* rc = head;

while ((rc != NULL) && ((((Client\*)rc->data)->key) != ((Client\*)data)->key))

rc = rc->next;

return rc;

}

//-------------------------------

bool Object::deleteByElement(Element\* e)

{

bool rc = NULL;

if (rc = (e != NULL))

{

if (e->next != NULL)

e->next->prev = e->prev;

if (e->prev != NULL)

e->prev->next = e->next;

else

head = e->next;

delete e;

}

std::cout << "Элемент удален" << std::endl;

return rc;

}

//-------------------------------

bool Object::deleteByData(void\* data)

{

return deleteByElement(search(data));

}

//-------------------------------

Element\* Object::getLast()

{

listx::Element\* e = this->getFirst(), \* rc = this->getFirst();

while (e != NULL)

{

rc = e;

e = e->getNext();

};

return rc;

}

Object create()

{

return \*(new Object());

};

//-------------------------------

void Object::scan()

{

listx::Element\* e = this->getFirst();

while (e != NULL)

{

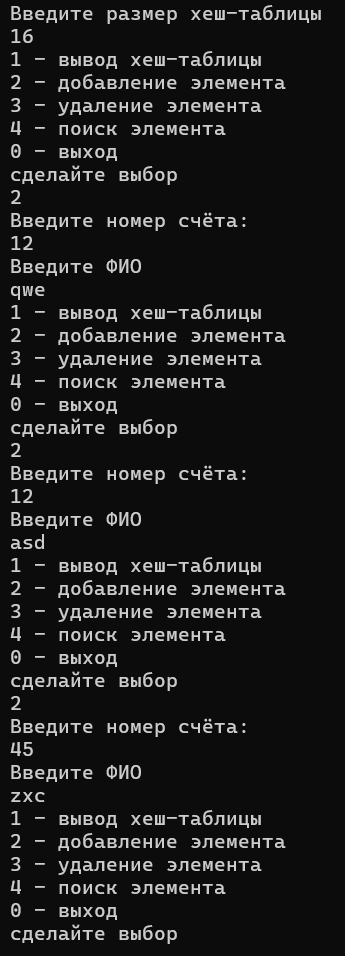
std::cout << ((Client\*)e->data)->key << '-' << ((Client\*)e->data)->mas << " / ";

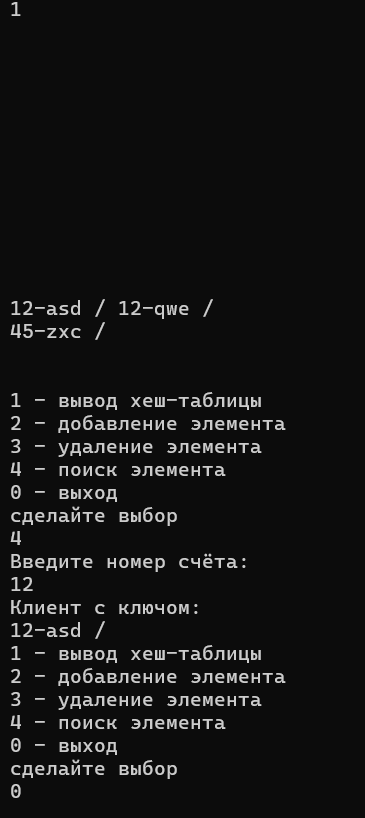
e = e->getNext();

};

}

}





Доп. Задания

Доп. 1



#include <iostream>

#include <string>

using namespace std;

class Node // класс для списка

{

public:

char key;

int value;

Node\* next;

Node(char \_key, int \_value) : key(\_key), value(\_value), next(nullptr) {}

};

class HashTable // класс хэш-таблицы

{

private:

int size;

Node\*\* table;

int hash(char key) // хэш-функция

{

return key % size;

}

public:

HashTable(int \_size) : size(\_size) // создание

{

table = new Node \* [size];

memset(table, 0, size \* sizeof(Node\*));

}

void insert(char key, int value) // вставка

{

int index = hash(key);

Node\* newNode = new Node(key, value);

newNode->next = table[index];

table[index] = newNode;

}

int search(char key) // поиск

{

int index = hash(key);

Node\* current = table[index];

while (current)

{

if (current->key == key)

{

return current->value;

}

current = current->next;

}

return -1;

}

void displayTable() // вывод

{

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

{

Node\* current = table[i];

cout << i+1 << ": ";

while (current)

{

cout << "(" << current->key << "," << current->value << ") ";

current = current->next;

}

cout << endl;

}

}

};

int main() //главная функция

{

setlocale(LC\_ALL, "rus");

string str;

cout << "Введите строку: ";

getline(cin, str);

HashTable hashTable(128); // создаём таблицу

for (char letter : str) // вводим строку

{

if (isalpha(letter))

{

hashTable.insert(letter, hashTable.search(letter) + 1);

}

}

for (;;) // меню

{

int choice;

cout << "Выберите действие:" << endl;

cout << "1. Вывести всю хеш-таблицу" << endl;

cout << "2. Поиск буквы" << endl;

cout << "3. Выход" << endl;

cin >> choice;

switch (choice) {

case 1:

hashTable.displayTable();

break;

case 2: {

char searchLetter;

cout << "Введите букву для поиска: ";

cin >> searchLetter;

int count = hashTable.search(searchLetter);

if (count != -1) {

cout << "Буква '" << searchLetter << "' встречается " << count + 1 << " раз" << endl;

}

else {

cout << "Буква '" << searchLetter << "' не найдена в строке" << endl;

}

break;

}

case 3:

return 0;

default:

cout << "Неправильный выбор." << endl;

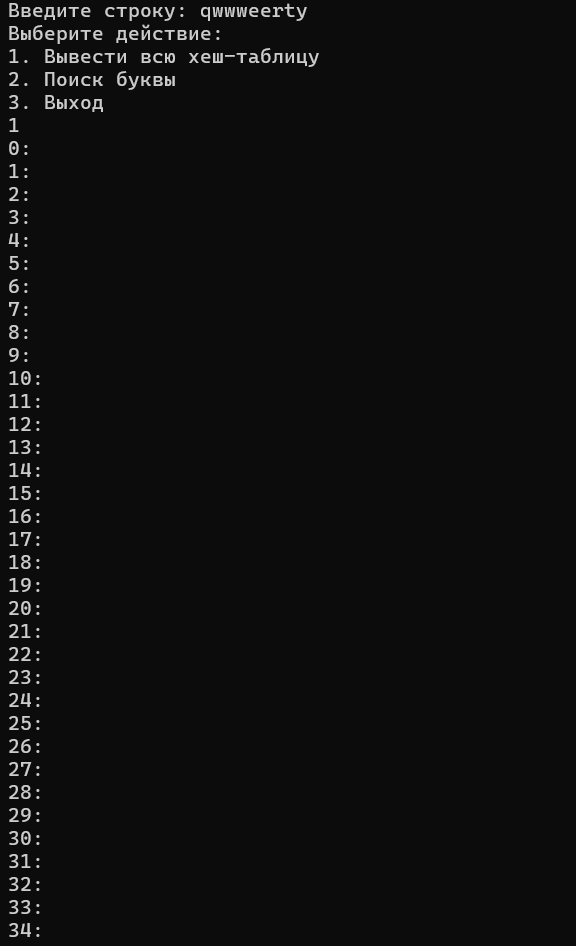
break;

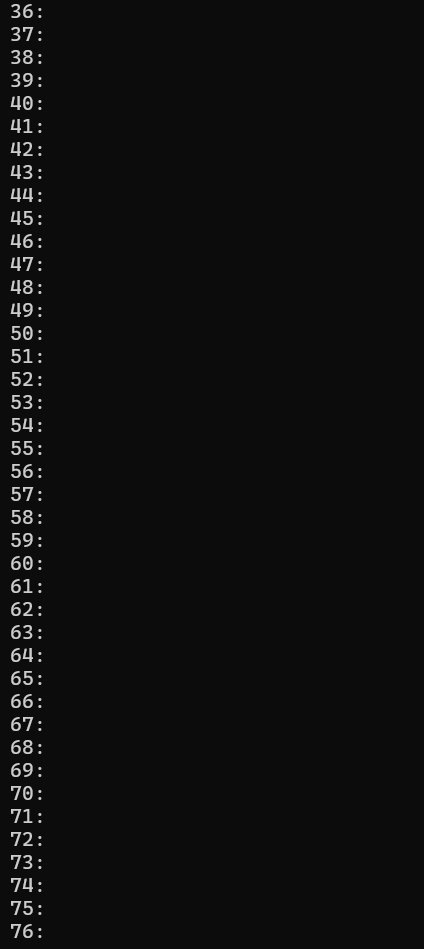
}

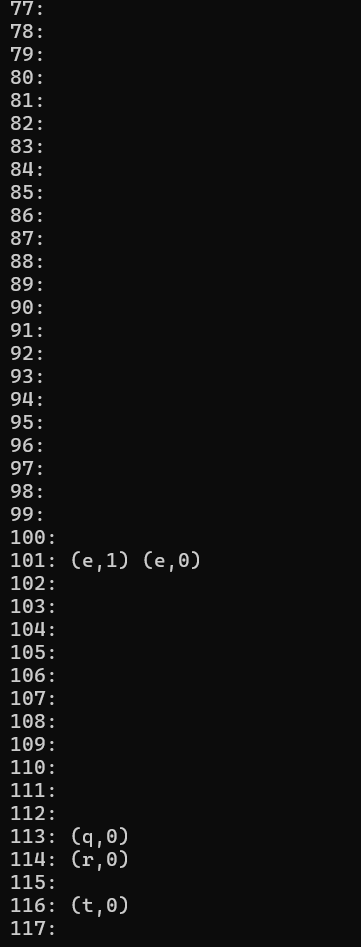
}

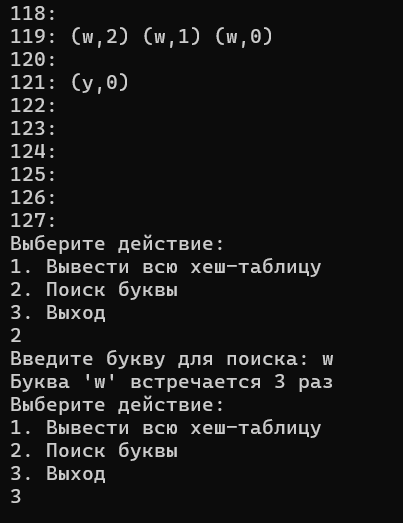
return 0;

}

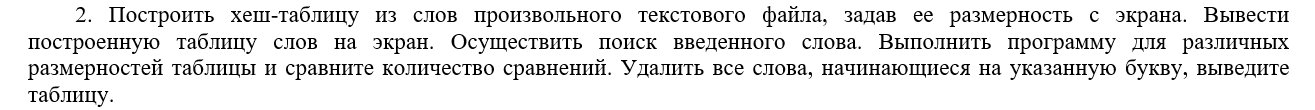








Доп. 2



#include <iostream>

#include <fstream>

#include <string>

using namespace std;

const int TABLE\_SIZE = 128;

class Node // ячейка списка

{

public:

string key;

int value;

Node\* next;

Node(const string& \_key, int \_value) : key(\_key), value(\_value), next(nullptr) {}

};

class HashTable // класс хэш-таблицы

{

private:

Node\* table[TABLE\_SIZE];

int hash(const string& key) {

int hashValue = 0;

for (char c : key) {

hashValue += static\_cast<int>(c);

}

return hashValue % TABLE\_SIZE;

}

public:

HashTable() // создание

{

for (int i = 0; i < TABLE\_SIZE; ++i) {

table[i] = nullptr;

}

}

void insert(const string& key, int value) { // вставка

int index = hash(key);

Node\* newNode = new Node(key, value);

newNode->next = table[index];

table[index] = newNode;

}

int search(const string& key) // поиск

{

int index = hash(key);

Node\* current = table[index];

while (current) {

if (current->key == key) {

return current->value;

}

current = current->next;

}

return 0;

}

void displayTable() //вывод

{

for (int i = 0; i < TABLE\_SIZE; ++i) {

Node\* current = table[i];

cout << i << ": ";

while (current) {

cout << "(" << current->key << "," << current->value << ") ";

current = current->next;

}

cout << endl;

}

}

void removeWordsStartingWith(char letter) // удаление

{

for (int i = 0; i < TABLE\_SIZE; ++i) {

Node\* current = table[i];

Node\* prev = nullptr;

while (current) {

if (current->key[0] == letter) {

// Удалить текущий узел

if (prev == nullptr) {

// Головной узел

table[i] = current->next;

}

else {

prev->next = current->next;

}

Node\* temp = current;

current = current->next;

delete temp;

}

else {

prev = current;

current = current->next;

}

}

}

}

};

int main() //главная

{

setlocale(LC\_ALL, "rus");

HashTable hashTable;

ifstream file("A.txt");

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

cout << "Ошибка открытия файла!" << endl;

return 1;

}

string word;

while (file >> word) {

hashTable.insert(word, hashTable.search(word) + 1);

}

int choice;

do {

cout << "Выберите действие:" << endl;

cout << "1. Вывести всю хеш-таблицу" << endl;

cout << "2. Поиск слова" << endl;

cout << "3. Удалить слова, начинающиеся на указанную букву" << endl;

cout << "4. Выход" << endl;

cin >> choice;

switch (choice) {

case 1:

hashTable.displayTable();

break;

case 2: {

string searchWord;

cout << "Введите слово для поиска: ";

cin >> searchWord;

int count = hashTable.search(searchWord);

cout << "Слово '" << searchWord << "' встречается " << count << " раз" << endl;

break;

}

case 3: {

char letter;

cout << "Введите букву: ";

cin >> letter;

hashTable.removeWordsStartingWith(letter);

cout << "Слова, начинающиеся на букву '" << letter << "' удалены" << endl;

break;

}

case 4:

cout << "Выход из программы" << endl;

break;

default:

cout << "Неправильный выбор." << endl;

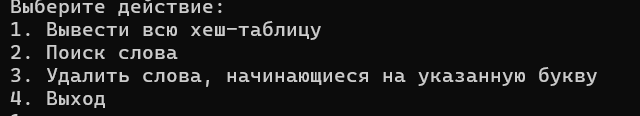
break;

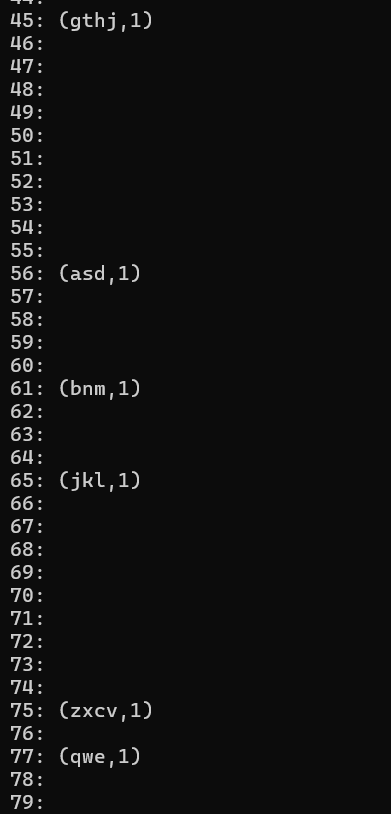
}

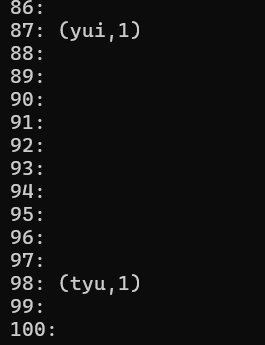
} while (choice != 4);

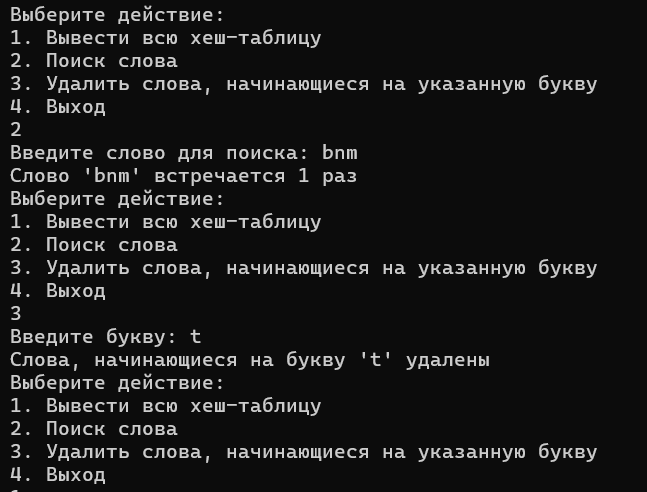
return 0;

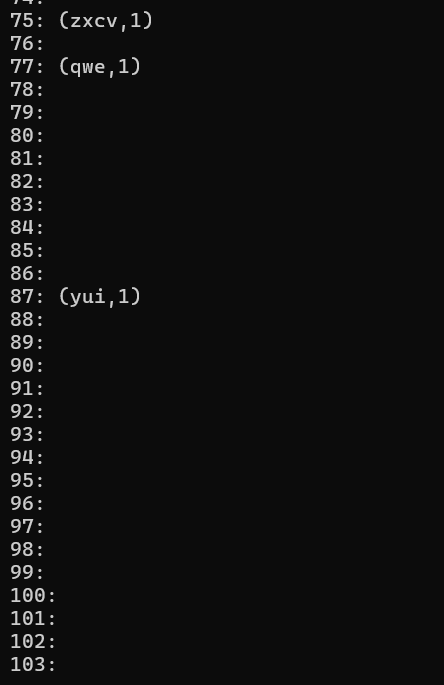
}

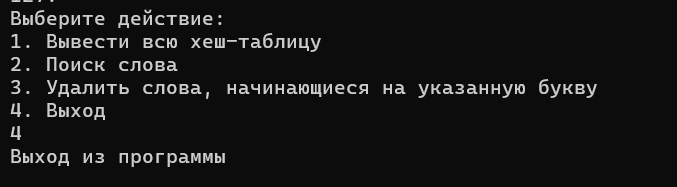


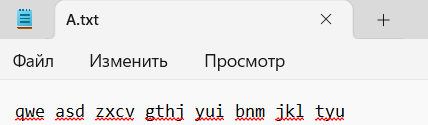




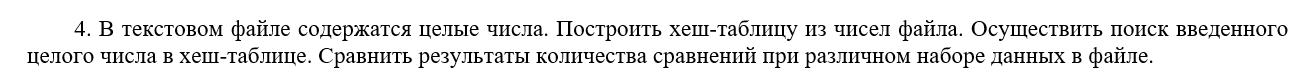






Файл A.txt  


Доп. 4



#include <iostream>

#include <fstream>

#include <string>

using namespace std;

const int TABLE\_SIZE = 128;

class Node // ячейка списка

{

public:

int key;

int value;

Node\* next;

Node(int \_key, int \_value) : key(\_key), value(\_value), next(nullptr) {}

};

class HashTable // класс хэш-таблицы

{

private:

Node\* table[TABLE\_SIZE];

int hash(int key) {

return key % TABLE\_SIZE;

}

public:

HashTable() // создание

{

for (int i = 0; i < TABLE\_SIZE; ++i) {

table[i] = nullptr;

}

}

void insert(int key, int value) { // вставка

int index = hash(key);

Node\* newNode = new Node(key, value);

newNode->next = table[index];

table[index] = newNode;

}

int search(int key) // поиск

{

int index = hash(key);

Node\* current = table[index];

int comparisons = 0;

int occurrences = 0;

while (current) {

++comparisons;

if (current->key == key) {

++occurrences;

}

current = current->next;

}

return occurrences > 0 ? comparisons : -1;

}

int main() //главная

{

setlocale(LC\_ALL, "rus");

HashTable hashTable;

ifstream file("numbers.txt");

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

cout << "Ошибка открытия файла!" << endl;

return 1;

}

int number;

while (file >> number) {

hashTable.insert(number, 1);

}

int searchNumber;

cout << "Введите число для поиска: ";

cin >> searchNumber;

int comparisons = hashTable.search(searchNumber);

if (comparisons > 0) {

cout << "Число " << searchNumber << " найдено. Количество сравнений: " << comparisons << endl;

}

else {

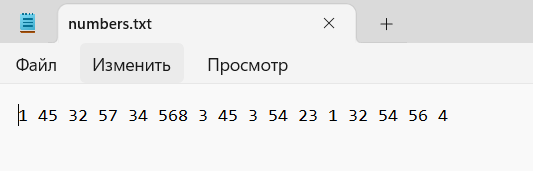
cout << "Число " << searchNumber << " не найдено." << endl;

}

return 0;

}

При следующих значения



Получаем следующие результаты

