Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа №14

По дисциплине «Основы алгоритмизации и программирования»

На тему « Хеш-таблицы c цепочками»

Выполнил:

Студент 1 курса 8 группы

Лужецкий Владислав Константинович

Преподаватель: асс. Андронова М.В.

2024, Минск

Оглавление

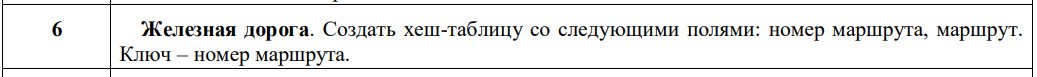
[Основное задание(вариант 6) 2](#_Toc166182090)

[Дополнительное задание 1 8](#_Toc166182091)

[Дополнительное задание 2 10](#_Toc166182092)

[Дополнительное задание 3 13](#_Toc166182093)

# Основное задание(вариант 6)



**Main.cpp**

#include "Hash\_Chain.h"

#include <iostream>

#include <Windows.h>

using namespace std;

// Структура для хранения данных

struct AAA

{

int key;

char\* mas;

AAA(int k, char\* z)

{

key = k;

mas = z;

}

AAA()

{

key = 0;

mas = nullptr;

}

};

// Хеш-функция для определения индекса в таблице

int hf(void\* d)

{

AAA\* f = (AAA\*)d;

return f->key;

}

// Функция для печати элемента AAA

void AAA\_print(listx::Element\* e)

{

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

}

int main()

{

setlocale(LC\_ALL, "rus");

SetConsoleCP(1251); //Для работы с русскими символами

SetConsoleOutputCP(1251);

int current\_size = 7;

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

cin >> current\_size;

// Создание объекта хеш-таблицы

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 по номеру маршрута" << endl;

cout << "0 - выход" << endl;

cout << "сделайте выбор" << endl;

cin >> choice;

switch (choice)

{

case 0:

exit(0);

case 2:

{

AAA\* a = new AAA;

char\* str = new char[20];

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

cin >> k;

a->key = k;

cout << "Введите маршрут: " << endl;

cin >> str;

a->mas = str;

H.insert(a);

}

break;

case 1:

H.Scan();

break;

case 3:

{

AAA\* b = new AAA;

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

cin >> k;

b->key = k;

H.deleteByData(b);

}

break;

case 4:

AAA \* c = new AAA;

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

cin >> k;

c->key = k;

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

cout << "Элемент не найден" << endl;

else

{

cout << "Первый элемент с данным ключом" << endl;

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

cout << endl;

}

break;

}

}

return 0;

}

**Hash\_Table.cpp**

#include "Hash\_Chain.h"

#include <iostream>

namespace hashTC

{

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

{

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

}

int Object::hashFunction(void\* data) //метод универсального хеширования

{

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

#include "Lists.h"

namespace hashTC

{

struct Object

{

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

// Макрос для определения значения NIL (пустого элемента)

#define LISTNIL (Element\*)-1

namespace listx

{

// Структура элемента списка

struct Element

{

Element\* prev; // Указатель на предыдущий элемент

Element\* next; // Указатель на следующий элемент

void\* data; // Указатель на данные

// Конструктор элемента

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

{

prev = pr;

data = dat;

next = nt;

}

// Метод для получения следующего элемента

Element\* getNext()

{

return next;

};

// Метод для получения предыдущего элемента

Element\* getPrev()

{

return prev;

};

};

// Пустой элемент NIL

static Element\* NIL = nullptr;

// Структура списка

struct Object

{

Element\* head; // Указатель на головной элемент списка

// Конструктор списка

Object()

{

head = NIL;

};

Element\* getFirst(); // Получение первого элемента списка

Element\* getLast(); // Получение последнего элемента списка

Element\* search(void\* data); // Поиск элемента по данным

bool insert(void\* data); // Вставка элемента в список

bool deleteByElement(Element\* e); // Удаление элемента из списка по указателю на него

bool deleteByData(void\* data); // Удаление элемента из списка по данным

void scan(); // Печать содержимого списка

};

Object create(); // Создание списка

}

// Отмена определения макроса LISTNIL

#undef LISTNIL

**Lists.cpp**

#include "Lists.h"

#include <iostream>

// Структура элемента таблицы

struct AAA

{

int key;

char\* mas;

};

namespace listx

{

// Вставка элемента в список

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;

}

// Поиск элемента по данным

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

{

Element\* rc = head;

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

rc = rc->next;

return rc;

}

// Удаление элемента из списка по указателю на него

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;

}

// Удаление элемента из списка по данным

bool Object::deleteByData(void\* data)

{

return deleteByElement(search(data));

}

// Получение последнего элемента списка

Element\* Object::getLast()

{

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

while (e != NULL)

{

rc = e;

e = e->getNext();

};

return rc;

}

// Создание списка

Object create()

{

return \*(new Object());

};

// Печать содержимого списка

void Object::scan()

{

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

while (e != NULL)

{

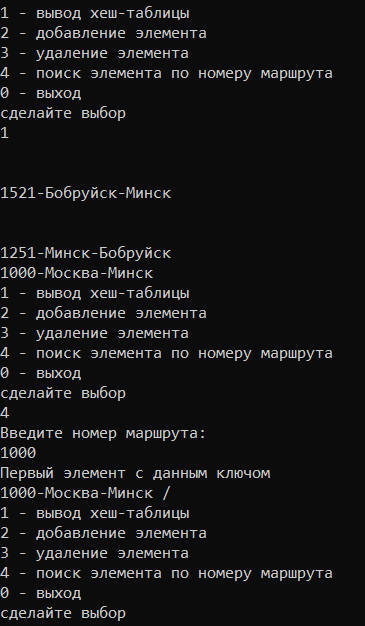
std::cout << ((AAA\*)e->data)->key << '-' << ((AAA\*)e->data)->mas << " ";

e = e->getNext();

};

}

}



# Дополнительное задание 1



#include <iostream>

#include <string>

#include <vector>

#include <Windows.h>

// Структура для хранения пары ключ-значение

struct Table {

char key;

int value;

};

class HashTable {

private:

std::vector<Table> table;

int size;

public:

HashTable(int size) : size(size) {

table.resize(size);

}

// Хеш-функция для вычисления индекса

int hash(char key) {

return key % size;

}

// Вставка элемента в хеш-таблицу

void insert(char key) {

int index = hash(key);

table[index].key = key;

table[index].value++;

}

// Поиск элемента в хеш-таблице

int search(char key) {

int index = hash(key);

return table[index].value;

}

};

int main() {

setlocale(0, "");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

std::string input;

std::cout << "Введите строку: ";

std::getline(std::cin, input);

// Создаем хеш-таблицу

HashTable frequencyTable(26); // Предполагаем, что работаем только с латинскими буквами

// Подсчитываем количество вхождений каждой буквы

for (char c : input) {

// Игнорируем пробелы и другие разделители

if (std::isspace(c)) {

continue;

}

// Увеличиваем счетчик вхождений буквы

frequencyTable.insert(std::tolower(c));

}

// Выводим хеш-таблицу на экран

std::cout << "Хеш-таблица:\n";

for (char c = 'a'; c <= 'z'; c++) {

int count = frequencyTable.search(c);

if (count > 0) {

std::cout << c << ": " << count << std::endl;

}

}

// Осуществляем поиск введенной буквы в хеш-таблице

char searchLetter;

std::cout << "Введите букву для поиска: ";

std::cin >> searchLetter;

// Проверяем наличие буквы в хеш-таблице

int count = frequencyTable.search(std::tolower(searchLetter));

if (count > 0) {

std::cout << "Буква " << searchLetter << " найдена. Количество вхождений: " << count << std::endl;

}

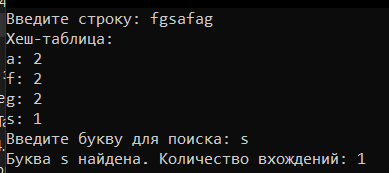
else {

std::cout << "Буква " << searchLetter << " не найдена." << std::endl;

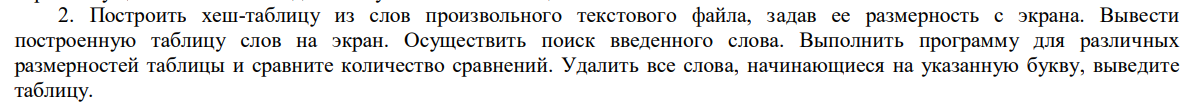
}

return 0;

}



# Дополнительное задание 2



#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <algorithm>

#include <windows.h>

// Структура для хранения пары ключ-значение (слово-количество)

struct KeyValuePair {

std::string key;

int value;

};

class HashTable {

private:

std::vector<std::vector<KeyValuePair>> table;

int size;

public:

HashTable(int size) : size(size) {

table.resize(size);

}

// Хеш-функция для вычисления индекса

int hash(const std::string& key) {

int sum = 0;

for (char c : key) {

sum += c;

}

return sum % size;

}

// Вставка элемента в хеш-таблицу

void insert(const std::string& key) {

int index = hash(key);

for (auto& entry : table[index]) {

if (entry.key == key) {

entry.value++;

return;

}

}

table[index].push\_back({ key, 1 });

}

// Поиск элемента в хеш-таблице

int search(const std::string& key) {

int index = hash(key);

for (const auto& entry : table[index]) {

if (entry.key == key) {

return entry.value;

}

}

return 0;

}

void removeByFirstLetter(char letter) {

int index = hash(std::string(1, letter));

auto& bucket = table[index];

bucket.erase(

std::remove\_if(bucket.begin(), bucket.end(),

[letter](const KeyValuePair& entry) {

return entry.key[0] == letter;

}),

bucket.end());

}

// Вывод хеш-таблицы на экран

void printTable() {

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

std::cout << "Хеш " << i << ": ";

for (const auto& entry : table[i]) {

std::cout << entry.key << "(" << entry.value << ") ";

}

std::cout << std::endl;

}

}

};

int main() {

setlocale(0, "");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

std::string filename = "file.txt";

std::ifstream file(filename);

if (!file) {

std::cout << "Не удалось открыть файл." << std::endl;

return 1;

}

int tableSize;

std::cout << "Введите размерность хеш-таблицы: ";

std::cin >> tableSize;

// Создаем хеш-таблицу

HashTable wordTable(tableSize);

// Читаем слова из файла и заполняем хеш-таблицу

std::string word;

while (file >> word) {

// Удаляем знаки препинания и приводим слово к нижнему регистру

word.erase(std::remove\_if(word.begin(), word.end(), ispunct), word.end());

std::transform(word.begin(), word.end(), word.begin(), tolower);

wordTable.insert(word);

}

// Выводим построенную таблицу слов на экран

std::cout << "Хеш-таблица слов:\n";

wordTable.printTable();

// Осуществляем поиск введенного слова в хеш-таблице

std::string searchWord;

std::cout << "Введите слово для поиска: ";

std::cin >> searchWord;

// Проверяем наличие слова в хеш-таблице

int count = wordTable.search(searchWord);

if (count > 0) {

std::cout << "Слово " << searchWord << " найдено. Количество вхождений: " << count << std::endl;

}

else {

std::cout << "Слово " << searchWord << " не найдено." << std::endl;

}

// Удаление слов, начинающихся на указанную букву

char letter;

std::cout << "Введите букву для удаления слов: ";

std::cin >> letter;

wordTable.removeByFirstLetter(letter);

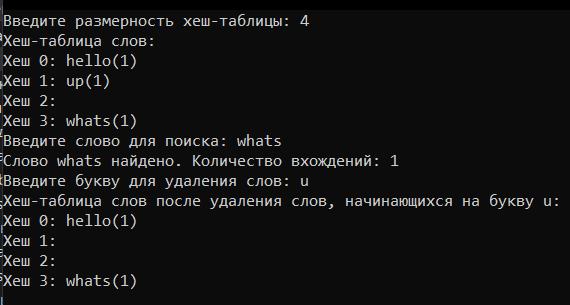
// Вывод обновленной таблицы слов на экран

std::cout << "Хеш-таблица слов после удаления слов, начинающихся на букву " << letter << ":\n";

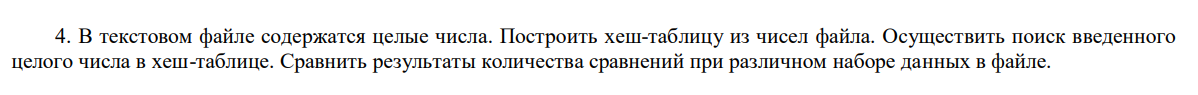
wordTable.printTable();

return 0;

}



# Дополнительное задание 3



#include <iostream>

#include <fstream>

#include <vector>

#include <string>

struct KeyValuePair {

int key;

// Дополнительные данные, связанные с ключом

};

class HashTable {

private:

std::vector<std::vector<KeyValuePair>> table;

int size;

public:

HashTable(int size) : size(size) {

table.resize(size);

}

// Хеш-функция для вычисления индекса

int hash(int key) {

return key % size;

}

// Вставка элемента в хеш-таблицу

void insert(int key) {

int index = hash(key);

for (auto& entry : table[index]) {

if (entry.key == key) {

// Ключ уже существует, выполните необходимые действия

return;

}

}

table[index].push\_back({ key });

}

// Поиск элемента в хеш-таблице

// Поиск элемента в хеш-таблице

int search(int key) {

int comparisons = 0;

for (const auto& bucket : table) {

for (const auto& entry : bucket) {

comparisons++;

if (entry.key == key) {

// Ключ найден, выполните необходимые действия

return comparisons; // Возвращаем количество сравнений

}

}

}

return comparisons; // Ключ не найден

}

};

int main() {

setlocale(0, "");

std::ifstream file("file.txt");

if (!file) {

std::cout << "Ошибка открытия файла." << std::endl;

return 1;

}

// Создание хеш-таблицы

HashTable hashtable(100); // Укажите желаемый размер таблицы

// Чтение чисел из файла и добавление их в хеш-таблицу

int num;

while (file >> num) {

hashtable.insert(num);

}

// Поиск числа в хеш-таблице

int searchNum;

std::cout << "Введите целое число для поиска: ";

std::cin >> searchNum;

int comparisons = hashtable.search(searchNum);

if (comparisons != 0) {

std::cout << "Число найдено." << std::endl;

std::cout << "Количество сравнений: " << comparisons << std::endl;

}

else {

std::cout << "Число не найдено." << std::endl;

}

file.close();

return 0;

}

