ИНИСТЕРСТВ ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЯ ОБРАЗОВАНИЯ

ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №7

по дисциплине «Программирование»

Хеширование

Выполнил студент

группы ИП-11

Лазовой Н. А.

Проверил преподаватель

Косинов Г. П.

Гомель 2020

**Цель работы**: изучить основы построения хеш-функций.

**Ход работы**

**Вариант 20**

**Задание**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 20 | Хеширование кукушки | Двойное хеширование | Метод открытой адресации |  |

Код программы:

#include <iostream>

#include "Hash.h"

#include "hashTwo.h"

using namespace std;

void Menu();

int main()

{

setlocale(LC\_ALL, "Russian");

char Key[20];

int Info;

int choose;

Hash hash1;

hashTwo hash2;

Menu();

cin >> choose;

while (choose != 9)

{

switch (choose)

{

case 1:

{

cout << "Введите ключ:" << endl;

cin.clear();

while (cin.get() != '\n');

cin.getline(Key, 19);

cout << "Введите данные:" << endl;

cin >> Info;

hash1.AddElem(Key, Info);

}

break;

case 2:

{

cout << "Введите ключ:" << endl;

cin.clear();

while (cin.get() != '\n');

cin.getline(Key, 19);

cout << "Введите данные:" << endl;

cin >> Info;

hash2.AddElem(Key, Info);

}

break;

case 3:

{

cout << "Ввыедите ключ для удаления:" << endl;

cin.clear();

while (cin.get() != '\n');

cin.getline(Key, 19);

hash1.Del(Key);

}

break;

case 4:

{

cout << "Ввыедите ключ для удаления:" << endl;

cin.clear();

while (cin.get() != '\n');

cin.getline(Key, 19);

hash2.Del(Key);

}

break;

case 5:

{

cout << "Ввыедите ключ, который нужно найти:" << endl;

cin.clear();

while (cin.get() != '\n');

cin.getline(Key, 19);

hash1.SearchElem(Key);

}

break;

case 6:

{

cout << "Ввыедите ключ, который нужно найти:" << endl;

cin.clear();

while (cin.get() != '\n');

cin.getline(Key, 19);

hash2.SearchElem(Key);

}

break;

case 7:

{

hash1.ShowHashTable();

}

break;

case 8:

{

hash2.ShowHashTable();

}

break;

default:cout << "Повторите ввод" << endl;

}

system("pause");

system("cls");

Menu();

cin >> choose;

}

}

void Menu()

{

cout << "1. Добавить эелемент в таблицу методом кукушки" << endl;

cout << "2. Добавить эелемент в таблицу методом двойного хеширования" << endl;

cout << "3. Удалить элемент из таблицы методом кукушки" << endl;

cout << "4. Удалить элемент из таблицы двойного хеширования" << endl;

cout << "5. Найти элемент в таблице методом кукушки" << endl;

cout << "6. Найти элемент в таблице методом двойного хеширования" << endl;

cout << "7. Показать таблицу методом кукушки" << endl;

cout << "8. Показать таблицу методом двойного хеширования" << endl;

cout << "9. Выход" << endl;

}

#pragma once

#include <iostream>

struct Element

{

char key[20];

int info;

};

class Hash

{

private:

Element Array[10];

public:

Hash()

{

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

{

strcpy\_s(Array[i].key, "");

}

}

int AddativeHash(char \*Key);

int OrHash(char\* Key);

void AddElem(char\* Key, int Info);

void SearchElem(char\* Key);

void Del(char\* Key);

void ShowHashTable();

};

#include <iostream>

#include "Hash.h"

int Hash::AddativeHash(char\* Key)

{

int s = 0;

while (\*Key)

s += \*Key++;

return s % 10;

}

int Hash::OrHash(char\* Key)

{

int s = 0;

while (\*Key)

s += s ^ \*Key++;

return s % 10;

}

void Hash::AddElem(char\* Key, int Info)

{

if (strcmp(Array[AddativeHash(Key)].key, "") == 0)

{

strcpy\_s(Array[AddativeHash(Key)].key, Key);

Array[AddativeHash(Key)].info = Info;

return;

}

else

{

if (strcmp(Array[OrHash(Key)].key, "") == 0)

{

strcpy\_s(Array[OrHash(Key)].key, Key);

Array[OrHash(Key)].info = Info;

return;

}

}

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

{

if (AddativeHash(Key) + i < 10 && strcmp(Array[AddativeHash(Key) + i].key, "") == 0)

{

strcpy\_s(Array[AddativeHash(Key) + i].key, Key);

Array[AddativeHash(Key) + i].info = Info;

break;

}

if (AddativeHash(Key) + i >= 10 && strcmp(Array[AddativeHash(Key) + i - 10].key, "") == 0)

{

strcpy\_s(Array[AddativeHash(Key) + i - 10].key, Key);

Array[AddativeHash(Key) + i - 10].info = Info;

break;

}

}

}

void Hash::SearchElem(char\* Key)

{

if (strcmp(Array[AddativeHash(Key)].key, Key) == 0)

{

std::cout << "Элемент найден:" << Array[AddativeHash(Key)].info << std::endl;

return;

}

else

{

if (strcmp(Array[OrHash(Key)].key, Key) == 0)

{

std::cout << "Элемент найден:" << Array[OrHash(Key)].info << std::endl;

return;

}

}

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

{

if (AddativeHash(Key) + i < 10 && strcmp(Array[AddativeHash(Key) + i].key, Key) == 0)

{

std::cout << "Элемент:" << Array[AddativeHash(Key) + i].info << std::endl;

break;

}

else

{

if (AddativeHash(Key) + i >= 10 && strcmp(Array[AddativeHash(Key) + i - 10].key, Key) == 0)

{

std::cout << "Элемент:" << Array[AddativeHash(Key) + i - 10].info << std::endl;

break;

}

}

}

}

void Hash::Del(char\* Key)

{

if (strcmp(Array[AddativeHash(Key)].key, Key) == 0)

{

strcpy\_s(Array[AddativeHash(Key)].key, "");

return;

}

else

{

if (strcmp(Array[OrHash(Key)].key, Key) == 0)

{

strcpy\_s(Array[OrHash(Key)].key, "");

return;

}

}

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

{

if (AddativeHash(Key) + i < 10 && strcmp(Array[AddativeHash(Key) + i].key, Key) == 0)

{

strcpy\_s(Array[AddativeHash(Key)].key, "");

break;

}

else

{

if (AddativeHash(Key) + i >= 10 && strcmp(Array[AddativeHash(Key) + i - 10].key, Key) == 0)

{

strcpy\_s(Array[AddativeHash(Key)].key, "");

break;

}

}

}

}

void Hash::ShowHashTable()

{

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

{

std::cout << Array[i].key << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

}

}

#pragma once

#include <iostream>

struct Element1

{

char key[20];

int info;

};

class hashTwo

{

private:

Element1 Array[10];

public:

hashTwo()

{

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

{

strcpy\_s(Array[i].key, "");

}

}

int AddativeHash(char\* Key);

int OrHash(char\* Key);

void AddElem(char\* Key, int Info);

void SearchElem(char\* Key);

void Del(char\* Key);

void ShowHashTable();

};

#include <iostream>

#include "hashTwo.h"

int hashTwo::AddativeHash(char\* Key)

{

int s = 0;

while (\*Key)

s += \*Key++;

return s % 10;

}

int hashTwo::OrHash(char\* Key)

{

int s = 0;

while (\*Key)

s += s ^ \*Key++;

return s % 10;

}

void hashTwo::AddElem(char\* Key, int Info)

{

if (strcmp(Array[AddativeHash(Key)].key, "") == 0)

{

strcpy\_s(Array[AddativeHash(Key)].key, Key);

Array[AddativeHash(Key)].info = Info;

return;

}

for (int i = 1; i < 10; i++)

{

if (strcmp(Array[(AddativeHash(Key) + i \* OrHash(Key)) % 10].key, "") == 0)

{

strcpy\_s(Array[(AddativeHash(Key) + i \* OrHash(Key)) % 10].key, Key);

Array[(AddativeHash(Key) + i \* OrHash(Key)) % 10].info = Info;

break;

}

}

}

void hashTwo::SearchElem(char\* Key)

{

if (strcmp(Array[AddativeHash(Key)].key, Key) == 0)

{

std::cout << "Элемент найден:" << Array[AddativeHash(Key)].info << std::endl;

return;

}

for (int i = 1; i < 10; i++)

{

if (strcmp(Array[(AddativeHash(Key) + i \* OrHash(Key)) % 10].key, Key) == 0)

{

std::cout << "Элемент:" << Array[(AddativeHash(Key) + i \* OrHash(Key)) % 10].info << std::endl;

break;

}

}

}

void hashTwo::Del(char\* Key)

{

if (strcmp(Array[AddativeHash(Key)].key, Key) == 0)

{

strcpy\_s(Array[AddativeHash(Key)].key, "");

return;

}

for (int i = 1; i < 10; i++)

{

if (strcmp(Array[(AddativeHash(Key) + i \* OrHash(Key)) % 10].key, Key) == 0)

{

strcpy\_s(Array[(AddativeHash(Key) + i \* OrHash(Key)) % 10].key, "");

break;

}

}

}

void hashTwo::ShowHashTable()

{

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

{

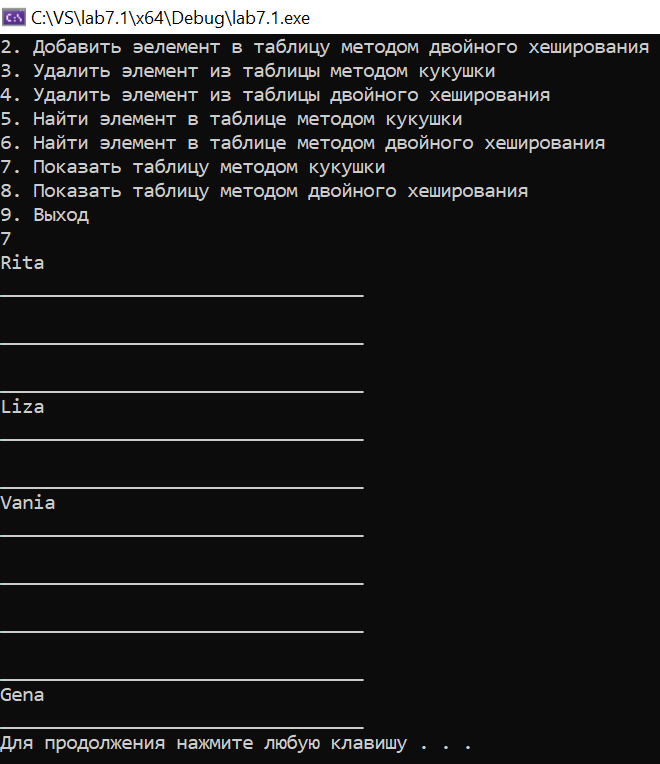
std::cout << Array[i].key << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

}

}

Результат:



**Вывод:** в ходе выполнения работы были изучены основы построения хэш-функций и методами разрешения коллизий.