Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**о работе по информатике**

Семестр: 2

На тему: хэш-таблица

Выполнил студент ИВТ-22-2б:

Бобриков Михаил Александрович

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

Пермь 2023

**Постановка задачи**

1. Создать динамический массив из записей (в соответствии с вариантом), содержащий не менее 100 элементов. Для заполнения элементов массива использовать ДСЧ.
2. Предусмотреть сохранение массива в файл и загрузку массива из файла.
3. Предусмотреть возможность добавления и удаления элементов из массива (файла).
4. Выполнить поиск элемента в массиве по ключу в соответствии с вариантом. Для поиска использовать хэш-таблицу.
5. Подсчитать количество коллизий при размере хэш-таблицы 40, 75 и 90 элементов.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Вариант** | **Данные** | **Ключ (string)** | **Хэш-функция** | **Метод рехеширования** |
| 3 | ФИО, №счета, сумма | ФИО | H(k)=k mod M | Метод открытой адресации |

**Алгоритм решения (метод открытой адресации)**

1. Заполнение массива структур: из трех предварительно заполненных массивов имен, отчеств и фамилий случайным образом формируется ФИО. С использованием ДСЧ генерируется № счета и сумма.

2. Запись массива в файл. Открывается файл в режиме записи. В цикле до конца массива каждый элемент массива записывается в файл. Файл закрывается.

3. Восстановление массива из файла. Файл открывается для чтения. Если файл не открылся, выдается сообщение «Файл не существует», работа функции завершается. Иначе, считается количество структур в файле. В цикле до конца файла считывается по одной структуре во временную переменную. Затем из этой переменной создается элемент массива.

4. Добавление элемента в массив. Пользователь вводит ФИО, № счета и сумму. В цикле в массиве ищется пустой элемент (ФИО == «»). Если он найден, то создается новый элемент.

5. Создание и печать хэш-таблицы. Хэш-таблица хранится в массиве структур, структура содержит 2 поля: хэш-код (number) и ФИО (FIO). Предварительно хэш-таблица заполняется пустыми значениями (код = -1, ФИО = «»). Затем в цикле обрабатывается существующий массив структур:

* из ФИО посредством хэш-функции формируется хэш-код;
* если в хэш-таблице элемент с индексом, равным хэш-коду, пуст (т.е. свободен: ФИО == «»), то заполняется эта строка таблицы;
* если же в хэш-таблице элемент с индексом, равным хэш-коду, заполнен (занят), то возникает коллизия (увеличивается счетчик коллизий);
* для разрешения коллизии последовательным перемещением по хэш-таблице ищется первая свободная строка, если найдена, то заполняется.

6. Поиск по ФИО в хэш-таблице:

* из введенного для поиска ФИО посредством хэш-функции формируется хэш-код;
* если в хэш-таблице у элемента с индексом, равным хэш-коду, ФИО равно заданному, то запись найдена (прямой доступ);
* если же в хэш-таблице у элемента с индексом, равным хэш-коду, ФИО не равно заданному, то последовательным перемещением по хэш-таблице ищется строка с заданным ФИО;
* если элемент найден, то выводится его данные, иначе – «человек не найден»

**Алгоритм решения (метод цепочек)**

1. Заполнение массива структур: из трех предварительно заполненных массивов имен, отчеств и фамилий случайным образом формируется ФИО. С использованием ДСЧ генерируется № счета и сумма.

2. Запись массива в файл. Открывается файл в режиме записи. В цикле до конца массива каждый элемент массива записывается в файл. Файл закрывается.

3. Восстановление массива из файла. Файл открывается для чтения. Если файл не открылся, выдается сообщение «Файл не существует», работа функции завершается. Иначе, считается количество структур в файле. В цикле до конца файла считывается по одной структуре во временную переменную. Затем из этой переменной создается элемент массива.

4. Добавление элемента в массив. Пользователь вводит ФИО, № счета и сумму. В цикле в массиве ищется пустой элемент (ФИО == «»). Если он найден, то создается новый элемент.

5. Создание и печать хэш-таблицы. Хэш-таблица хранится в массиве, структур, структура содержит 2 поля: хэш-код (number) и указатель на однонаправленный список (head). Узел списка – это структура, которая состоит из 2 полей: ФИО (FIO) и указатель на следующий элемент списка (next). Предварительно хэш-таблица заполняется пустыми значениями (ФИО = «», head = nullptr). Затем в цикле обрабатывается существующий массив структур:

* из ФИО посредством хэш-функции формируется хэш-код;
* если в хэш-таблице элемент с индексом, равным хэш-коду, пуст (т.е. свободен: head == nullptr), то создается первый узел списка;
* если же в хэш-таблице элемент с индексом, равным хэш-коду, заполнен (занят), то возникает коллизия (увеличивается счетчик коллизий);
* для разрешения коллизии создается новый узел и вставляется в начало списка.

6. Поиск по ФИО в хэш-таблице:

* из введенного для поиска ФИО посредством хэш-функции формируется хэш-код;
* в хэш-таблице для элемента с индексом, равным хэш-коду, просматривается его список – ищется узел, у которого ФИО равно заданному;
* если элемент найден, то выводится его данные, иначе – «человек не найден»

**Блок-схема (метод открытой адресации)**

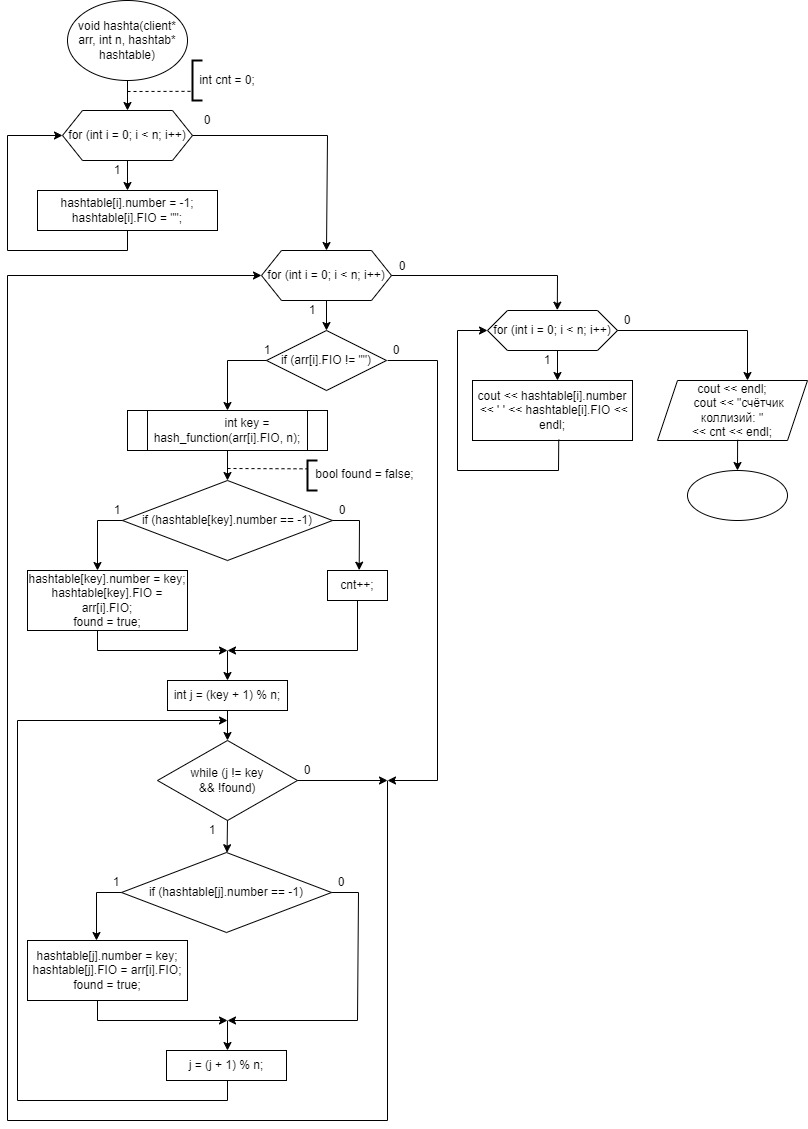
****

Рис. 1.Заполнение и печать хэш-таблицы (метод открытой адресации).

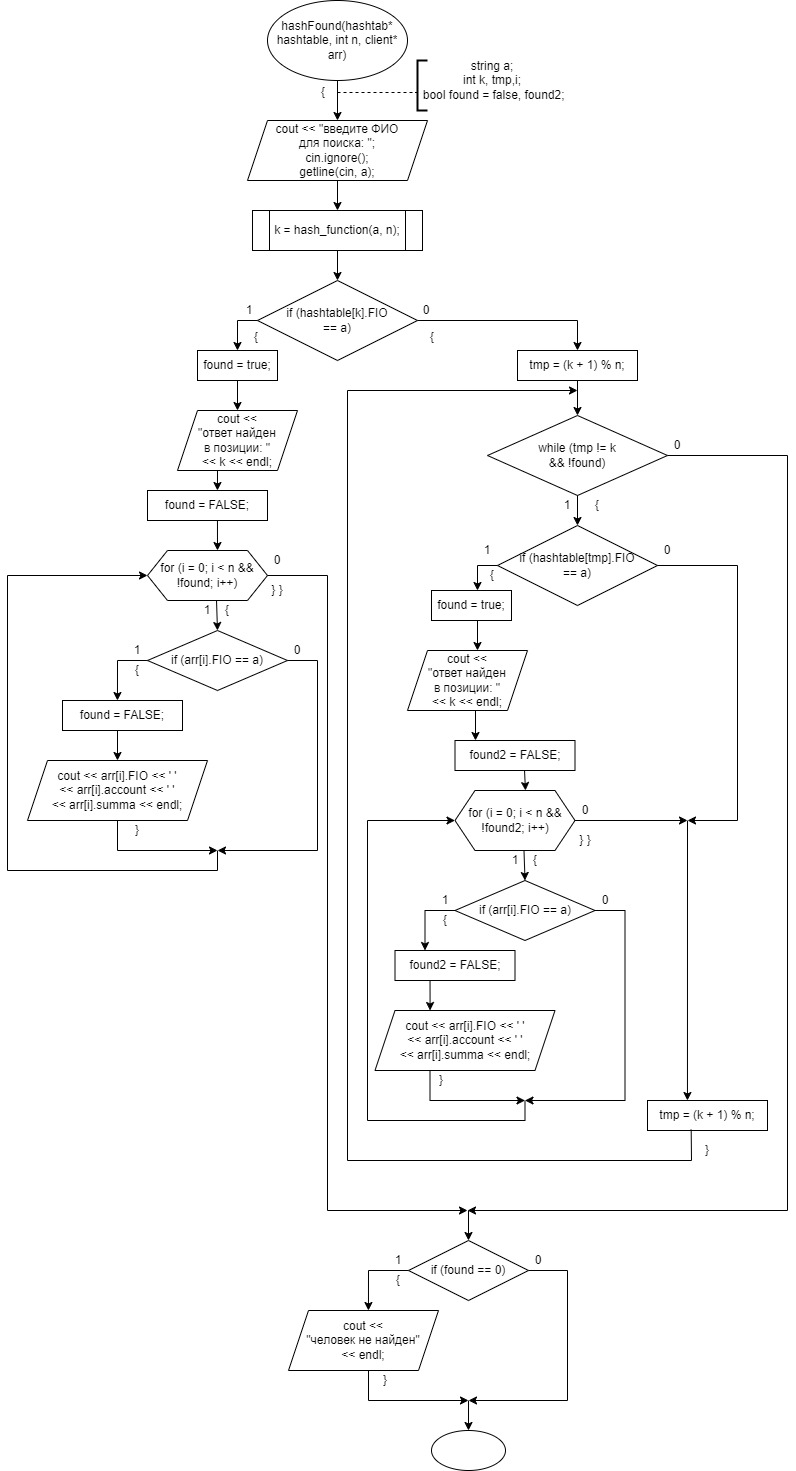


Рис. 2. Поиск в хэш-таблице (метод открытой адресации).

**Блок-схема (метод цепочек)**

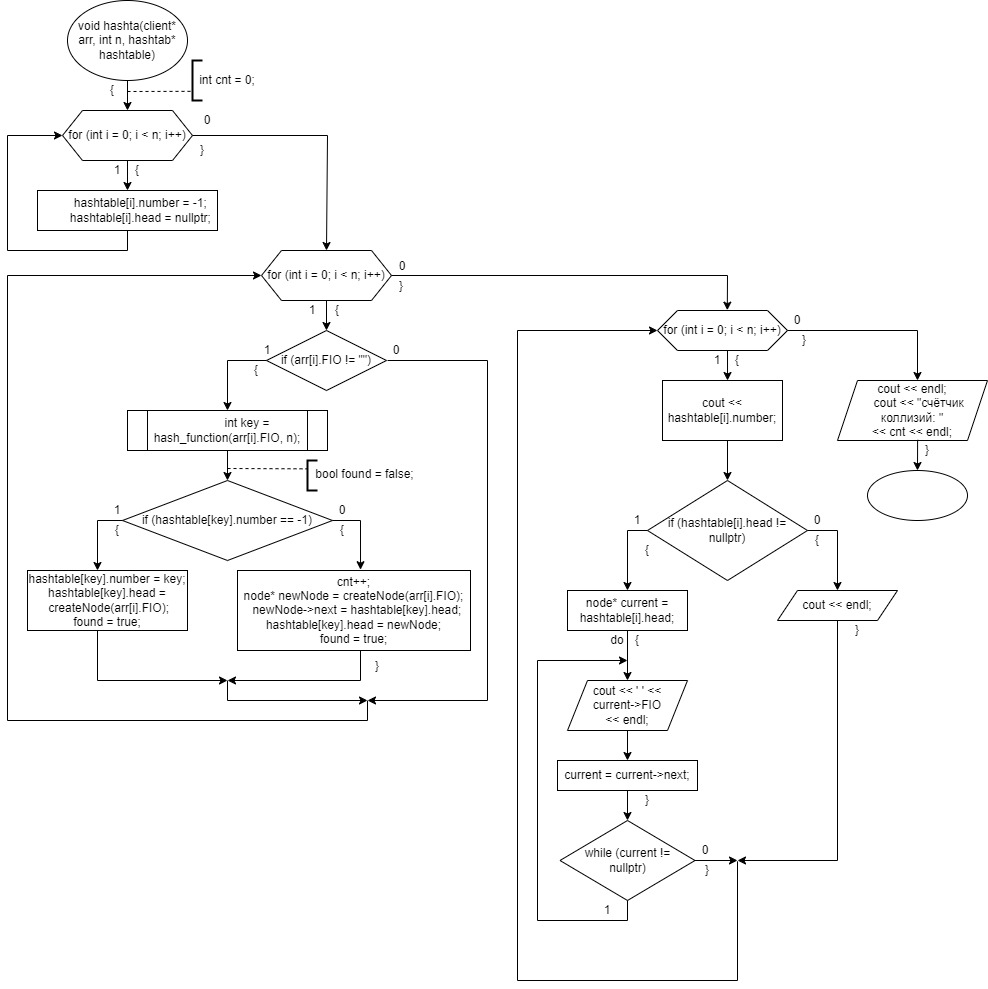


Рис. 3.Заполнение и печать хэш-таблицы (метод цепочек).

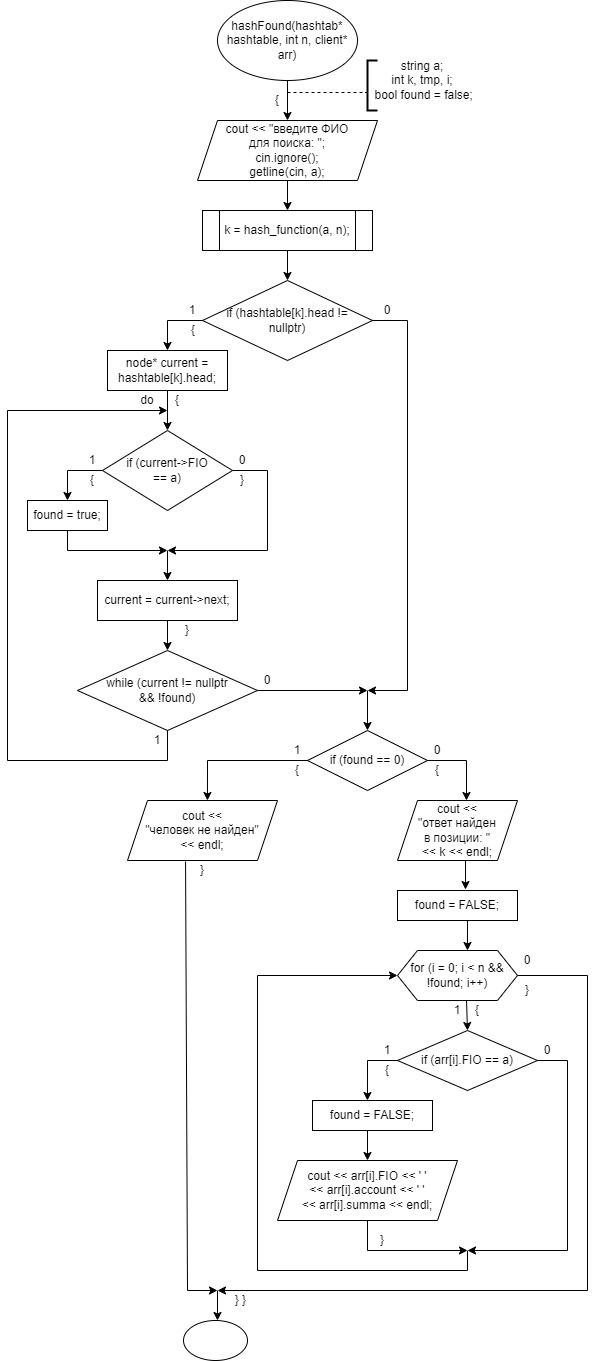


Рис. 4. Поиск в хэш-таблице (метод цепочек).

**Код программы (метод открытой адресации)**

// хэш таблица 2.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <time.h>

#include <fstream>

#include <string>

#include <Windows.h>

using namespace std;

struct hashtab {

int number = -1;

string FIO = "";

};

struct client {

char FIO[256] = "";

int account = 0;

int summa = 0;

};

// Функция для сохранения массива в файл

void SaveArr(client\* arr, int n) {

client\* current = new client;

char FName[] = "array.txt"; //Путь к файлу

ofstream f1(FName, ios\_base::out | ios\_base::binary); // открываем файл для записи

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

//current.FIO = arr[i].FIO;

strncpy(current->FIO, arr[i].FIO, 256);

current->account = arr[i].account;

current->summa = arr[i].summa;

f1.write((char\*)current, sizeof(client)); /\*записываем структуру в файл\*/

}

f1.close(); // закрываем файл

}

// Функция для чтения массива из файла

void LoadArr(client\* arr, int n) {

client current;

char FName[] = "array.txt"; //Путь к файлу

ifstream f2(FName); // открываем файл для чтения

if (!f2) {

cout << "Файл не существует\n\n";

f2.close();

}

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

strncpy(arr[i].FIO, "", 256);

arr[i].account = 0;

arr[i].summa = 0;

}

f2.seekg(0, ios::end); //функция перехода в конец файла

long NumBytes = (long)f2.tellg(); //выяснение кол-ва байтов в файле

float NumStruct = NumBytes / sizeof(client); //выяснение кол-ва структур в файле

f2.seekg(0, ios::beg); //переход в начало файла

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

f2.read((char\*)&current, sizeof(client)); // запись структуры из файла в переменную current

//cout << current2.FIO << " " << current2.account << " " << current2.summa << endl;

strncpy(arr[i].FIO, current.FIO, 256);

arr[i].account = current.account;

arr[i].summa = current.summa;

cout << arr[i].FIO << " " << arr[i].account << " " << arr[i].summa << endl;

}

f2.close();

}

int get\_rnd\_number(int max) {

return rand() % max;

}

string get\_rnd\_fio() {

string name[10] = { "Александр", "Борис", "Виктор", "Григорий", "Дмитрий", "Евгений", "Иван", "Николай", "Михаил", "Олег" };

string patronymic[10] = { "Александрович", "Борисович", "Викторович", "Григорьевич", "Дмитриевич", "Евгеньевич", "Иванович", "Николаевич", "Михайлович", "Олегович" };

string surname[10] = { "Александров", "Борисов", "Викторов", "Григорьев", "Дмитриев", "Евгеньев", "Иванов", "Николаев", "Михайлов", "Олегов" };

string fio = name[get\_rnd\_number(9)] +" " + patronymic[get\_rnd\_number(9)] + " " + surname[get\_rnd\_number(9)];

return fio;

}

void makearr\_rnd(int n, client\* arr) {

string s="";

char s2[256];

s = get\_rnd\_fio();

strcpy(s2, s.c\_str());

for (int i = 0; i < n-5; i++) {//-5 - оставляеи 5 пустых строк, чтобы было, куда добавить

s = get\_rnd\_fio();

strcpy(s2, s.c\_str());

strncpy(arr[i].FIO, s2, 256);

arr[i].account = get\_rnd\_number(100);

arr[i].summa = get\_rnd\_number(1000);

}

for (int j = 0; j < n-5; j++) {

cout << arr[j].FIO << " " << arr[j].account << " " << arr[j].summa << endl;

}

cout << endl;

}

void addEl(client\* arr, int n) {

client input;

string temp;

string temp3;

cout << "Введите ФИО: ";

cin.ignore();

getline(cin, temp);

strcpy(input.FIO, temp.c\_str());

cout << "Введите номер счета: ";

cin >> input.account;

cout << "Введите сумму: ";

cin >> input.summa;

if (arr == nullptr) {

cout << "Ошибка";

return;

}

else {

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

temp3 = arr[i].FIO;

if (temp3 == "") {

arr[i] = input;

return;

}

}

cout << "В массиве нет места для новой записи" << endl;

}

}

int hash\_function(string str, int n) {

unsigned long i = 0;

for (int j = 0; str[j]; j++) {

i += str[j];

}

return i % n;

}

void hashta(client\* arr, int n, hashtab\* hashtable) {

int cnt = 0;

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

hashtable[i].number = -1;

hashtable[i].FIO = "";

}

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

if (arr[i].FIO != "") { //строка массива не пуста

int key = hash\_function(arr[i].FIO, n);

bool found = false;

if (hashtable[key].number == -1) {

hashtable[key].number = key;

hashtable[key].FIO = arr[i].FIO;

found = true;

}

else {

cnt++;

}

int j = (key + 1) % n;

while (j != key && !found) {

if (hashtable[j].number == -1) {

hashtable[j].number = key;

hashtable[j].FIO = arr[i].FIO;

found = true;

}

j = (j + 1) % n;

}

}

}

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

cout << hashtable[i].number << ' ' << hashtable[i].FIO << endl;

}

cout << endl;

cout << "счётчик коллизий: " << cnt << endl;

}

void hashFound(hashtab\* hashtable, int n, client\* arr) {

string a;

int k, tmp, i;

bool found = false, found2;

cout << "введите ФИО для поиска: "; cin.ignore(); getline(cin, a);

k = hash\_function(a, n);

if (hashtable[k].FIO == a) {

found = true;

cout << "ответ найден в позиции: " << k << endl;

found2 = FALSE;

for (i = 0; i < n && !found2; i++) {

if (arr[i].FIO == a) {

found2 = TRUE;

cout << arr[i].FIO << ' ' << arr[i].account << ' ' << arr[i].summa << endl;

}

}

}

else {

tmp = (k + 1) % n;

while (tmp != k && !found) {

if (hashtable[tmp].FIO == a) {

found = true;

cout << "ответ найден в позиции: " << tmp << endl;

found2 = FALSE;

for (i = 0; i < n && !found2; i++) {

if (arr[i].FIO == a) {

found2 = TRUE;

cout << arr[i].FIO << ' ' << arr[i].account << ' ' << arr[i].summa << endl;

}

}

}

tmp = (tmp + 1) % n;

}

}

if (found == 0) {

cout << "человек не найден" << endl;

}

}

void DelEL(client\* arr, int n) {

string fio;

int pos;

bool found = 0;

int temp;

if (arr == nullptr) {

cout << "Ошибка";

return;

}

else {

cout << "введите ФИО для удаления: ";

cin.ignore();

getline(cin, fio);

pos = hash\_function(fio, n);

}

if (fio == arr[pos].FIO) {

found = 1;

strcpy(arr[pos].FIO, "");

arr[pos].account = 0;

arr[pos].summa = 0;

}

else {

temp = (pos + 1) % n;

while (temp != pos && !found) {

if (fio == arr[temp].FIO) {

found = 1;

strcpy(arr[temp].FIO, "");

arr[temp].account = 0;

arr[temp].summa = 0;

}

temp = (temp + 1) % n;

}

}

if (found == 0) cout << "человек не найден" << endl;

else cout << "запись удалена" << endl;

}

int main()

{

int choice = -1;

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand((int)time(NULL));

int n = 50;

client\* arr = new client[n];

hashtab\* hashtable = new hashtab[n];

while (choice != 0) {

cout << "1 заполнение массива структур" << endl;

cout << "2 сохранение в файл" << endl;

cout << "3 загрузка из файла" << endl;

cout << "4 добавление элемента в массив" << endl;

cout << "5 создание и печать хэш таблицы" << endl;

cout << "6 поиск ФИО в хэш таблице" << endl;

cout << "7 удаление заданного ФИО" << endl;

cout << "8 печать массива" << endl;

cout << "0 выход из программы" << endl;

cout << "введите пункт меню: ";

cin >> choice;

cout << endl;

switch (choice) {

case 1:

makearr\_rnd(n, arr);

break;

case 2:

SaveArr(arr, n);

break;

case 3:

LoadArr(arr, n);

break;

case 4:

addEl(arr, n);

break;

case 5:

hashta(arr, n, hashtable);

break;

case 6:

hashFound(hashtable, n, arr);

break;

case 7:

DelEL(arr, n);

break;

case 8:

for (int j = 0; j < n; j++) cout << arr[j].FIO << " " << arr[j].account << " " << arr[j].summa << endl;

break;

case 0:

return 0;

}

}

return 0;

}

**Код программы (метод цепочек)**

// хэш таблица цепочки.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <time.h>

#include <fstream>

#include <string>

#include <Windows.h>

using namespace std;

struct node {

char FIO[256] = "";

node\* next = nullptr;

};

struct hashtab {

int number = -1;

node\* head = nullptr;

};

struct client {

char FIO[256] = "";

int account = 0;

int summa = 0;

};

// Функция для создания узла списка

node\* createNode(char\* fio) {

node\* newNode = new node;

strncpy(newNode->FIO, fio, 256);

newNode->next = nullptr;

return newNode;

}

// Функция для сохранения массива в файл

void SaveArr(client\* arr, int n) {

client\* current = new client;

char FName[] = "array.txt"; //Путь к файлу

ofstream f1(FName, ios\_base::out | ios\_base::binary); // открываем файл для записи

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

//current.FIO = arr[i].FIO;

strncpy(current->FIO, arr[i].FIO, 256);

current->account = arr[i].account;

current->summa = arr[i].summa;

f1.write((char\*)current, sizeof(client)); /\*записываем структуру в файл\*/

}

f1.close(); // закрываем файл

}

// Функция для чтения массива из файла

void LoadArr(client\* arr, int n) {

client current;

char FName[] = "array.txt"; //Путь к файлу

ifstream f2(FName); // открываем файл для чтения

if (!f2) {

cout << "Файл не существует\n\n";

f2.close();

}

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

strncpy(arr[i].FIO, "", 256);

arr[i].account = 0;

arr[i].summa = 0;

}

f2.seekg(0, ios::end); //функция перехода в конец файла

long NumBytes = (long)f2.tellg(); //выяснение кол-ва байтов в файле

float NumStruct = NumBytes / sizeof(client); //выяснение кол-ва структур в файле

f2.seekg(0, ios::beg); //переход в начало файла

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

f2.read((char\*)&current, sizeof(client)); // запись структуры из файла в переменную current

//cout << current2.FIO << " " << current2.account << " " << current2.summa << endl;

strncpy(arr[i].FIO, current.FIO, 256);

arr[i].account = current.account;

arr[i].summa = current.summa;

cout << arr[i].FIO << " " << arr[i].account << " " << arr[i].summa << endl;

}

f2.close();

}

int get\_rnd\_number(int max) {

return rand() % max;

}

string get\_rnd\_fio() {

string name[10] = { "Александр", "Борис", "Виктор", "Григорий", "Дмитрий", "Евгений", "Иван", "Николай", "Михаил", "Олег" };

string patronymic[10] = { "Александрович", "Борисович", "Викторович", "Григорьевич", "Дмитриевич", "Евгеньевич", "Иванович", "Николаевич", "Михайлович", "Олегович" };

string surname[10] = { "Александров", "Борисов", "Викторов", "Григорьев", "Дмитриев", "Евгеньев", "Иванов", "Николаев", "Михайлов", "Олегов" };

string fio = name[get\_rnd\_number(9)] +" " + patronymic[get\_rnd\_number(9)] + " " + surname[get\_rnd\_number(9)];

return fio;

}

void makearr\_rnd(int n, client\* arr) {

string s = "";

char s2[256];

s = get\_rnd\_fio();

strcpy(s2, s.c\_str());

for (int i = 0; i < n-5; i++) {

s = get\_rnd\_fio();

strcpy(s2, s.c\_str());

strncpy(arr[i].FIO, s2, 256);

arr[i].account = get\_rnd\_number(100);

arr[i].summa = get\_rnd\_number(1000);

}

for (int j = 0; j < n-5; j++) {

cout << arr[j].FIO << " " << arr[j].account << " " << arr[j].summa << endl;

}

cout << endl;

}

void addEl(client\* arr, int n) {

client input;

string temp;

string temp3;

cout << "Введите ФИО: ";

cin.ignore();

getline(cin, temp);

strcpy(input.FIO, temp.c\_str());

cout << "Введите номер счета: ";

cin >> input.account;

cout << "Введите сумму: ";

cin >> input.summa;

if (arr == nullptr) {

cout << "Ошибка";

return;

}

else {

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

temp3 = arr[i].FIO;

if (temp3 == "") {

arr[i] = input;

return;

}

}

cout << "В массиве нет места для новой записи" << endl;

}

}

int hash\_function(string str, int n) {

unsigned long i = 0;

for (int j = 0; str[j]; j++) {

i += str[j];

}

return i % n;

}

void hashta(client\* arr, int n, hashtab\* hashtable) {

int cnt = 0;

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

hashtable[i].number = -1;

hashtable[i].head = nullptr;

}

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

if (arr[i].FIO != "") { //строка массива не пуста

int key = hash\_function(arr[i].FIO, n);

bool found = false;

if (hashtable[key].number == -1) { //строка хэш-таблицы свободна

hashtable[key].number = key;

hashtable[key].head = createNode(arr[i].FIO);

found = true;

}

else { // строка хэш-таблицы заполнена - это коллизия

cnt++; //счетчик коллизий ++

node\* newNode = createNode(arr[i].FIO); //добавляем новую запись в начало списка

newNode->next = hashtable[key].head;

hashtable[key].head = newNode;

found = true;

}

}

}

for (int i = 0; i < n; i++) { // печать хэш-таблицы

cout << hashtable[i].number;

if (hashtable[i].head != nullptr) {

node\* current = hashtable[i].head;

do {

cout << ' ' << current->FIO << endl;

current = current->next;

} while (current != nullptr);

}

else {

cout << endl;

}

}

cout << endl;

cout << "счётчик коллизий: " << cnt << endl;

}

void hashFound(hashtab\* hashtable, int n, client\* arr) {

string a;

int k, tmp, i;

bool found = false;

cout << "введите ФИО для поиска: "; cin.ignore(); getline(cin, a);

k = hash\_function(a, n);

if (hashtable[k].head != nullptr) { // в хэш-таблице есть запись с ключом k

node\* current = hashtable[k].head;

do {

if (current->FIO == a) {

found = true;

}

current = current->next;

} while (current != nullptr && !found);

}

if (found == 0) {

cout << "человек не найден" << endl;

}

else {

cout << "ответ найден в позиции: " << k << endl;

found = FALSE;

for (i = 0; i < n && !found; i++) {

if (arr[i].FIO == a) {

found = TRUE;

cout << arr[i].FIO << ' ' << arr[i].account << ' ' << arr[i].summa << endl;

}

}

}

}

void DelEL(client\* arr, int n) {

string fio;

int pos;

bool found = 0;

int temp;

if (arr == nullptr) {

cout << "Ошибка";

return;

}

else {

cout << "введите ФИО для удаления: ";

cin.ignore();

getline(cin, fio);

pos = hash\_function(fio, n);

}

if (fio == arr[pos].FIO) {

found = 1;

strcpy(arr[pos].FIO, "");

arr[pos].account = 0;

arr[pos].summa = 0;

}

else {

temp = (pos + 1) % n;

while (temp != pos && !found) {

if (fio == arr[temp].FIO) {

found = 1;

strcpy(arr[temp].FIO, "");

arr[temp].account = 0;

arr[temp].summa = 0;

}

temp = (temp + 1) % n;

}

}

if (found == 0) cout << "человек не найден" << endl;

else cout << "запись удалена" << endl;

}

int main()

{

int choice = -1;

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand((int)time(NULL));

int n = 50;

client\* arr = new client[n];

hashtab\* hashtable = new hashtab[n];

while (choice != 0) {

cout << "1 заполнение массива структур" << endl;

cout << "2 сохранение в файл" << endl;

cout << "3 загрузка из файла" << endl;

cout << "4 добавление элемента в массив" << endl;

cout << "5 создание и печать хэш таблицы" << endl;

cout << "6 поиск ФИО в хэш таблице" << endl;

cout << "7 удаление заданного ФИО" << endl;

cout << "8 печать массива" << endl;

cout << "0 выход из программы" << endl;

cout << "введите пункт меню: ";

cin >> choice;

cout << endl;

switch (choice) {

case 1:

makearr\_rnd(n, arr);

break;

case 2:

SaveArr(arr, n);

break;

case 3:

LoadArr(arr, n);

break;

case 4:

addEl(arr, n);

break;

case 5:

hashta(arr, n, hashtable);

break;

case 6:

hashFound(hashtable, n, arr);

break;

case 7:

DelEL(arr, n);

break;

case 8:

for (int j = 0; j < n; j++) cout << arr[j].FIO << " " << arr[j].account << " " << arr[j].summa << endl;

break;

case 0:

return 0;

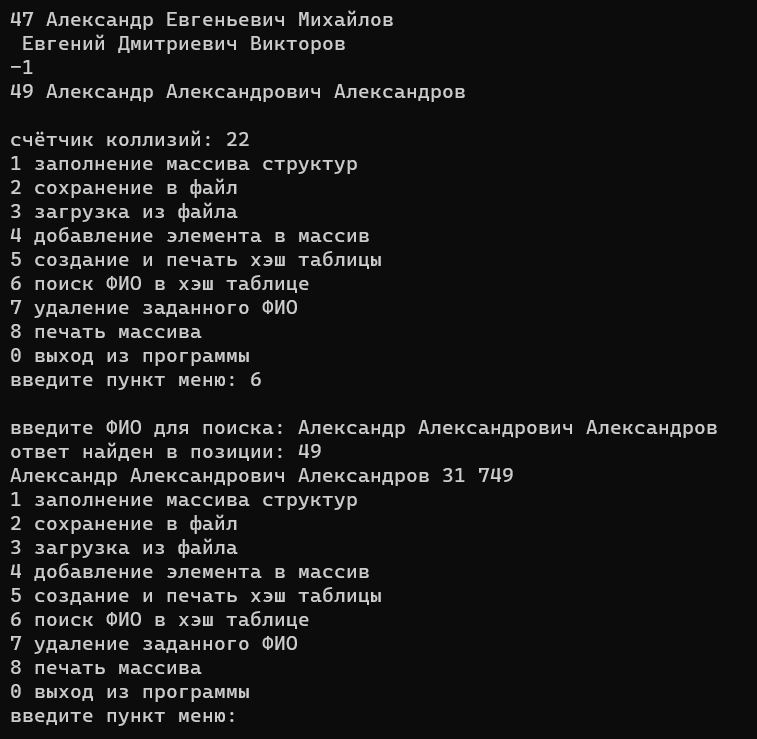
}

}

return 0;

}

**Вывод программы**

****