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

**ОТЧЁТ**

**по дисциплине «Информатика»**

Семестр: 2

На тему: «Хэш таблицы»

Выполнил студент ИВТ-22-2б:

Зайцев Павел Денисович

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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вариан т | Данные | Ключ (string) | Хэш-функция | Метод рехеширования |
| 9 | ФИО, №телефона, адрес | Адрес | H(k)=k mod M | Метод цепочек |

**Алгоритм решения (метод открытой адресации)**

1. Заполнение хэш таблицы структур: из трёх предварительно заполненных

массивов имён, отчеств и фамилий случайным образом формируется ФИО. С

использованием ДСЧ генерируется № счета и сумма.

2. Запись хэш таблицы в файл. Открывается файл в режиме записи. В цикле

до конца массива поля каждой структуры записываются в файл по очереди (ФИО, №Телефона, Адрес, ФИО, №телефона, Адрес и т.д.). В конце ставиться «!», как признак окончания файла. Файл закрывается.

1. Чтение хэш таблицы из файла. Файл открывается для чтения. Если файл не открылся, выдаётся сообщение «Error», 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 создаётся элемент массива.
2. Добавление элемента в хэш таблицу:
   1. Пользователь вводит ФИО, №телефона, адрес нового экземпляра структуры.
   2. В цикле в хэш таблице ищется пустой элемент (Адрес == «NULL»).
   3. Если он найден, то вставляется новый экземпляр структуры. Иначе, возникает коллизия, для ее разрешения нужно последовательно двигаться по массиву, пока не будет найдено свободное место.
   4. Если свободных мест не будет найдено, то выводится сообщение «В таблице нет места».
3. Поиск по ФИО в хэш-таблице:
   1. Из введенного для поиска адреса посредством хэш-функции формируется хэш-код;
   2. 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;
   3. если элемент найден, то выводится его индекс, 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, выдаётся сообщение «Файл не существует», работа функции завершается. В цикле до конца файла считывается построчно ФИО, №телефона, адрес во временный экземпляр класса. Затем из этого переменной создаётся элемент массива.

4. Добавление элемента в массив. Пользователь вводит ФИО, №телефона и

адрес. В цикле в массиве ищется пустой элемент (адрес == «NULL»). Если он

найден, то создаётся новый элемент.

5. Создание хэш-таблицы. Хэш-таблица представляет собой массив указателей на двунаправленные списки. Узел списка – это структура, которая

состоит из 5 полей: ФИО(fullname), №телефона(phone\_number), адрес(add) указатели(\*left, \*right) на другие узлы списка. Предварительно хэш-таблица заполняется пустыми значениями (ФИО=«NULL», №телефона=«NULL», адрес=«NULL», \*left=NULL, \*right=NULL).

1. Поиск по ФИО в хэш-таблице:
   1. Из введенного для поиска ФИО посредством хэш-функции формируется хэш-код;
   2. 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 индекс, иначе – «человек не найден».

**Функции**

**Метод открытой адресации**

1. Функция hash\_function() вычисляет хэш-код.
2. Функция rand\_numb() вычисляет случайное число.
3. Функция random\_human() формирует случайный экземпляр функции.
4. Функция add\_inhash() формирует случайный экземпляр структуры.
5. Функция show\_hash() выводит хэш таблицу в консоль.
6. Функция file1() записывает хэш таблицу в файл.
7. Функция file2() считывает хэш таблицу с файла.
8. Функция fillall\_hash() заполняет весь хэш таблицу случайными экземплярами структуры.
9. Функция filloneelement\_hash() заносит случайный элемент в хэш таблицу.
10. Функция find() находит индекс элемента по ключу.
11. Функция udalit1() удаляет элемент хэш таблицы по ключу.
12. Функция unich() удаляет все элементы хэш таблицы.

**Метод цепочек**

1. Функция hash\_function() вычисляет хэш-код.
2. Функция rand\_numb() вычисляет случайное число.
3. Функция random\_human() формирует случайный экземпляр функции.
4. Функция add\_inhash() формирует случайный экземпляр структуры.
5. Функция show\_hash() выводит хэш таблицу в консоль.
6. Функция mas\_random() функция заполнения массива случайными элементами.
7. Функция show\_mas() выводит массив в консоль.
8. Функция find() находит индекс элемента по ключу.
9. Функция udalit1() удаляет элемент хэш таблицы по ключу.
10. Функция del\_return() отменяет последнюю операцию удаления.
11. Функция file1() записывает массив в файл.
12. Функция file2() считывает массив с файла.
13. Функция unich\_mas() очищает массив.
14. Функция make\_hash() формирует хэш таблицу из элементов массива.

**Код программ**

**Метод открытой адресации**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <string>

#include <windows.h>

#include <ctime>

#include <stdlib.h>

#include <fstream>

using namespace std;

string name[10] = { "Владимир","Александр","Михал","Павел","Алексей","Тихон","Денис","Мирослав","Вячеслав","Богдан" };

string secname[10] = { "Дмитриевич", "Алексеевич","Олегович","Алесандрович","Купидонович","Игоревич","Валерьевич","Максимович","Богданович","Михайлович" };

string surname[10] = { "Зайцев","Коуров","Иванов","Смирнов","Сидоров","Макаров","Фокин","Голубочкин","Ципленков","Яблочков" };

string number[10] = { "88005353535","86532896565","87563569569","88954562596","87891592591","81233319663","83579512565","81533512552","87596845654","87293811937" };

string adres[10] = { "ул.Пушкина","ул.Деветаева","ул.Заречная","ул.ВасилияТатищева","ул.Ленина","ул.Поздеева","ул.Пенкина","ул.Яшина","ул.Кречкина","ул.Сталина" };

int collusions = 0;//Счетчик колизий

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

int y = 0;

for (int i = 0; n[i] != '\0'; i++) y++;

return (y % x);

}

//Стуктура объекта

struct Human {

Human() {

fullname = "NULL";

phone\_number = "NULL";

add = "NULL";

}

string fullname;

string phone\_number;

string add;

};

/\*

struct hash\_table {

Human\* table;

hash\_table(int size) {

table = new Human[size];

}

~hash\_table() {

delete[] table;

}

void add(Human temp, int size);

void find(string temp, int size);

void pop(Human temp, int size);

};

void hash\_table::find(string temp, int size) {

int index1 = hash\_function(temp, size);

int index2 = index1;

while (table[index2].add != temp && index2 < size) {

index2++;

}

if (index2 >= size) {

index2 = 0;

while (table[index2].add != temp && index2 < index1) {

index2++;

}

}

if (index2 > index1) {

cout << "\nЧеловека с таким адресом нет";

}

else {

cout << "\nЧеловека с адресом " << temp << " свстретился по индексу";

}

}

void hash\_table::add(Human temp, int size) {

int index1 = hash\_function(temp.add, size);

int index2 = index1;

int collusions = 0;

if (table[index2].fullname == "NULL") {

table[index2] = temp;

}

else {

while (index2 < size) {

if (table[index2].fullname == "NULL") {

table[index2] = temp;

}

collusions++;

index1++;

}

if (index2 >= size) {

index2 = 0;

while (table[index2].fullname != "NULL" && index2 < index1) {

if (table[index2].fullname == "NULL") {

table[index2] = temp;

}

collusions++;

index1++;

}

}

}

}

\*/

//Функция генерации случайного числа

int rand\_numb() {

return rand() % 10;;

}

//Получение заполненного экзэмпляра структуры Human

Human random\_human() {

Human buff;

buff.fullname = surname[rand\_numb()] + name[rand\_numb()] + secname[rand\_numb()];

buff.phone\_number = number[rand\_numb()];

buff.add = adres[rand\_numb()];

return buff;

}

//Функция добавления элемента в хэш

void add\_inhash(Human hash[], int size, Human ff) {

int index1 = hash\_function(ff.add, size);//Нахождение индекса по хэш функции

int index2 = index1;

if (hash[index2].add == "NULL") {//Проверка на заполненность узла хэш таблицы

hash[index2] = ff;//Занесение элемента в хэш таблицу

}

else {

while (index2 < size) {//Движение до конца таблицы

if (hash[index2].add == "NULL") {//Проверка на заполненность узла хэш таблицы

hash[index2] = ff;//Занесение элемента в хэш таблицу

return;

}

collusions++;//Счетсик колизий

index2++;//Переход на следующий индекс

}

if (index2 >= size) {

index2 = 0;//Переход в начало хэш таблицы

while (index2 < index1) {

if (hash[index2].add == "NULL") {//Проверка на заполненность узла хэш таблицы

hash[index2] = ff;//Занесение элемента в хэш таблицу

return;

}

collusions++;//Счетсик колизий

index2++;//Переход на следующий индекс

}

}

cout << "В таблице нет места";

}

}

//Функция вывода хэша в консоль

void show\_hash(Human m[], int n) {

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

cout << endl << "\nЭлемент " << i + 1 << "-ый ";

cout << "\nФИО:" << m[i].fullname;

cout << "\nНомер телефона:" << m[i].phone\_number;

cout << "\nАдрес:" << m[i].add;

}

}

//Функция для записи массива в файл

void file1(Human\* m, int n) {

if (n < 1) {

cout << "\nERROR\n";

return;

}

ofstream file("123.txt");

Human buff;//Буферный экземпляр структуры

if (!file) {//Проверка наличия файла

cout << "Произошла ошибка\n";

exit(1);

}

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

buff = m[j];//Занесение экземпляра структуры в буферную переменную

file << buff.fullname << ' ' << buff.phone\_number << ' ' << buff.add << ' ';

}

file << " !";//Маячок конца файла

file.close();

}

//Функция cчитывание массива с файла

Human\* file2(int n) {

if (n < 1) {//Проверка длинны хэш таблицы

cout << "\nERROR\n";

exit(1);

}

ifstream file("123.txt");

if (!file) {//Проверка наличия файла

cout << "Произошла ошибка\n";

exit(1);

}

string temp="";

Human buff;//Буферный экземпляр структуры

int j = 0;

Human\* mas = new Human[n];//Создание новой хэш таблицы

while (file>>temp) {

if (temp == "!") break;//Конец чтения если достигнут конец файла

buff.fullname = temp;

file >> temp;

buff.phone\_number = temp;

file >> temp;

buff.add = temp;

j++;

add\_inhash(mas, n, buff);//Занесение элемента в хэш

}

file.close();//Закрытие файла

return mas;

}

//Функция заполнения хэша случайными элементами

void fillall\_hash(Human\* m, int n) {

if (m==NULL) {

cout << "\nERROR\n";

return;

}

if (n < 1) {

cout << "\nERROR\n";

return;

}

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

Human buff = random\_human();//Создание случайного заполненного узла хэш таблицы

add\_inhash(m, n, buff);//Занесение элемента в хэш

}

cout << "\nКоличеств колизий=" << collusions << endl;

collusions = 0;

}

//Функция добавления одного элемента в хэш

void filloneelement\_hash(Human\* m, int n) {

Human buff;

if (m == NULL) {//Проверка инициализации хэш таблицы

cout << "\nERROR\n";

return;

}

if (n < 1) {//Проверка длинны хэш таблицы

cout << "\nERROR\n";

return;

}

cout << "\nФИО:"; cin.get(); getline(cin, buff.fullname);

cout << "№ТЕЛЕФОНА:"; getline(cin, buff.phone\_number);

cout << "АДРЕС:"; getline(cin, buff.add);

add\_inhash(m, n, buff);//Занесение элемента в хэш

cout << "\nКоличеств колизий=" << collusions << endl;

collusions = 0;

}

//Функция нахождения индекса элемена хэша по адресу

void find(Human m[], int size) {

if (m == NULL) {//Проверка инициализации хэш таблицы

cout << "\nERROR\n";

return;

}

if (size < 1) {//Проверка длинны хэш таблицы

cout << "\nERROR\n";

return;

}

string buff;

bool flag = 0;//Флаг

cout << "\nВведите адресс для поиска:";

cin.get();

getline(cin, buff);

int index1 = hash\_function(buff, size);//Нахождение индекса по хэш функции

int index2 = index1;

if (m[index2].add == buff) {//Проверка на совпадение полей

flag = 1;

}

while (flag == 0 && index2 < size) {

index2++;//Переход на следующий узел хэш таблицы

if (m[index2].add == buff) {//Проверка на совпадение полей

flag = 1;

}

}

if (index2 >= size) {

index2 = 0;//Перевод в начало хэш таблицы

if (m[index2].add == buff) {//Проверка на совпадение полей

flag = 1;

}

while (flag == 0 && index2 < index1) {

index2++;

if (m[index2].add == buff) {//Проверка на совпадение полей

flag = 1;

}

}

}

if (flag == 0) {

cout << "\nЧеловека с таким адресом нет";

}

else {

cout << "\nЧеловека с похожим адресом встретился по индексу:" << index2 << endl;

}

}

//Функция удаления элемена хэша по адресу

void udalit1(Human m[], int size) {

string buff;

bool flag = 0;

if (m == NULL) {//Проверка инициализации хэш таблицы

cout << "\nERROR\n";

return;

}

if (size < 1) {//Проверка длинны хэш таблицы

cout << "\nERROR\n";

return;

}

cout << "\nВведите адресс для удаления:";

cin.get();

getline(cin, buff);

int index1 = hash\_function(buff, size);//Нахождение индекса по хэш функции

int index2 = index1;

if (m[index2].add == buff) {//Проверка на совпадение полей

flag = 1;

}

while (flag == 0 && index2 < size) {

index2++;//Переход на следующий узел хэш таблицы

if (m[index2].add == buff) {//Проверка на совпадение полей

flag = 1;

}

}

if (index2 >= size) {

index2 = 0;//Перевод в начало хэш таблицы

if (m[index2].add == buff) {//Проверка на совпадение полей

flag = 1;

}

while (flag == 0 && index2 < index1) {

index2++;//Переход на следующий узел хэш таблицы

if (m[index2].add == buff) {//Проверка на совпадение полей

flag = 1;

}

}

}

if (flag == 0) {

cout << "\nЧеловека с таким адресом нет";

}

else {

cout << "\nЧеловек с похожим адресом встретился ибыл удален по индексу:" << index2 << endl;

string temp="";

m[index2].fullname = "NULL";

m[index2].phone\_number = "NULL";

m[index2].add = "NULL";

}

}

//Функция удаления всех элементов хэша по адресу

void unich(Human m[], int size) {

if (m == NULL) {//Проверка инициализации хэш таблицы

cout << "\nERROR\n";

return;

}

if (size < 1) {//Проверка длинны хэш таблицы

cout << "\nERROR\n";

return;

}

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

m[i].fullname = "NULL";

m[i].phone\_number = "NULL";

m[i].add = "NULL";

}

}

int main() {

srand(time(NULL));

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Human\* hash= NULL;//Хэш таблица

int c=0;//Переменная для работы программы

int x=0;//Переменная для работы программы

while (true) {

cout << endl << "\nВВедите № команды";

cout << endl << "1)Создать хэш таблицу";

cout << endl << "2)Вывести хэш в консоль";

cout << endl << "3)Заполнить хэш таблицу случайными узлами";

cout << endl << "4)Вставить 1 элемент в хэш таблицу";

cout << endl << "5)Найти индекс элемента по адресу";

cout << endl << "6)Удалить 1 узел по адресу";

cout << endl << "7)Очистьть хэш таблицу";

cout << endl << "8)Сохранить хэш таблицу в файл";

cout << endl << "9)Достать хэш таблицу из файла";

cout << endl << "10)Закрыть программу";

cout <<endl<< "№="; cin >> x;

switch (x) {

case 1:

cout << "\nВведите длинну хэш таблицы:";

cin >> c;

if (c < 1) {

cout << "\nНеправильный ввод";

break;

}

hash = new Human[c];

break;

case 2:show\_hash(hash, c); break;

case 3:fillall\_hash(hash, c); break;

case 4:filloneelement\_hash(hash, c); break;

case 5:find(hash, c); break;

case 6:udalit1(hash, c); break;

case 7:unich(hash, c); break;

case 8:file1(hash, c); break;

case 9:

cout << "\nВведите длинну новой хэш таблицы:";

cin >> c;

if (c < 1) {

cout << "\nНеправильный ввод";

break;

}

hash = file2(c);

break;

case 10: return 0;

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

}

}

return 0;

}

**Метод цепочек**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <string>

#include <windows.h>

#include <ctime>

#include <stdlib.h>

#include <fstream>

using namespace std;

string name[10] = { "Владимир","Александр","Михал","Павел","Алексей","Тихон","Денис","Мирослав","Вячеслав","Богдан" };

string secname[10] = { "Дмитриевич", "Алексеевич","Олегович","Алесандрович","Купидонович","Игоревич","Валерьевич","Максимович","Богданович","Михайлович" };

string surname[10] = { "Зайцев","Коуров","Иванов","Смирнов","Сидоров","Макаров","Фокин","Голубочкин","Ципленков","Яблочков" };

string number[10] = { "88005353535","86532896565","87563569569","88954562596","87891592591","81233319663","83579512565","81533512552","87596845654","87293811937" };

string adres[10] = { "ул.Пушкина","ул.Деветаева","ул.Заречная","ул.ВасилияТатищева","ул.Ленина","ул.Поздеева","ул.Пенкина","ул.Яшина","ул.Кречкина","ул.Сталина" };

int collusions = 0;//Счетчик колизий

struct Human {

Human() {

fullname = "NULL";

phone\_number = "NULL";

add = "NULL";

left = NULL;

right = NULL;

}

Human\* left;

Human\* right;

string fullname;

string phone\_number;

string add;

};

//Хэш функция

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

int y = 0;

for (int i = 0; n[i] != '\0'; i++) y++;

return (y % x);

}

//Функция генерации случайного числа

int rand\_numb() {

return rand() % 10;;

}

//Получение заполненного экзэмпляра структуры Human

Human random\_human() {

Human buff;

buff.fullname = surname[rand\_numb()] + name[rand\_numb()] + secname[rand\_numb()];

buff.phone\_number = number[rand\_numb()];

buff.add = adres[rand\_numb()];

return buff;

}

//Функция добавления элемента в хэш

void add\_inhash(Human\*\* hash, int size, Human\* ff) {

int index1 = hash\_function(ff->add, size);//Нахождение индекса по хэш функции

Human\* point=ff;

if (hash[index1] == NULL) {

hash[index1]=ff;

}

else {

hash[index1]->left = ff;

ff->right = hash[index1];

hash[index1] = ff;

collusions++;

}

}

//Функция вывода хэша

void show\_hash(Human\*\* hash, int size) {

cout << "ХЭШ Таблица:\n";

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

int x = 1;

cout << "\n(" << i << ")";

if (hash[i] == NULL) {

cout << "Ячека пуста";

}

else {

Human\* buff=hash[i];//Буферный указатель на первый узел списка

while (buff->right != NULL) {

cout << endl << x << "-ый элемент";

cout << "\nИмя:" << buff->fullname;

cout << "\nНомер телефона:" << buff->phone\_number;

cout << "\nАдресс:" << buff->add;

x++;

buff=buff->right;//Преход на следубщий узел

}

cout << endl << x << "-ый элемент";

cout << "\nИмя:" << buff->fullname;

cout << "\nНомер телефона:" << buff->phone\_number;

cout << "\nАдресс:" << buff->add;

}

}

}

//Функция заполнения массива

void mas\_random(Human\* mas, int size){

if (size < 1) {//Проверка длинны массива таблицы

cout << "\nERROR\n";

return;

}

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

mas[i] = random\_human();

}

}

//Функция для вывода массива в консоль

void show\_mas(Human\* mas, int size) {

if (size < 1) {//Проверка длинны массива таблицы

cout << "\nERROR\n";

return;

}

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

cout << endl << "\nЭлемент " << i + 1 << "-ый ";

cout << "\nФИО:" << mas[i].fullname;

cout << "\nНомер телефона:" << mas[i].phone\_number;

cout << "\nАдрес:" << mas[i].add;

}

}

//Функция нахождения индекса элемена хэша по адресу

void find(Human\*\* hash, int size) {

if (size < 1) {//Проверка длинны хэш таблицы

cout << "\nERROR\n";

return;

}

string buff1;

bool flag = 0;//Флаг

cout << "\nВведите адресс для поиска:";

cin.get();

getline(cin, buff1);

int index1 = hash\_function(buff1, size);//Нахождение индекса по хэш функции

Human\* buff2 = hash[index1];

if (buff2 == NULL) {//Проверка указателя на наличие указателя на узел списка

cout << "\nСписка не существует\n";

return;

}

else {

while (buff2->add != buff1 && buff2->right != NULL) {

buff2=buff2->right;

}

if (buff2->add == buff1) {

flag = 1;

}

}

if (flag == 0) {

cout << "\nЧеловека с таким адресом нет в хэше";

}

else {

cout << "\nЧеловека с похожим адресом встретился в списке по индексу:" << index1 << endl;

}

}

//Функция удаления одного элемента из хэша по адресу

Human\* udalit1(Human \*\* hash, int size) {

if (size < 1) {//Проверка длинны хэш таблицы

cout << "\nERROR\n";

return NULL;

}

string buff1;

bool flag = 0;

cout << "\nВведите адресс для удаления:";

cin.get();

getline(cin, buff1);

int index1 = hash\_function(buff1, size);//Нахождение индекса по хэш функции

Human\* buff2 = hash[index1];

if (buff2 == NULL) {//Проверка указателя на наличие ссылки на узел списка

cout << "\nНет человека с таким адресом\n";

return NULL;

}

else {

while (buff2->add != buff1 && buff2->right != NULL) {

buff2 = buff2->right;

}

if (buff2->add == buff1) {

Human\* cc1;

flag = 1;

if (buff2->right == NULL && buff2->left != NULL){

cc1 = buff2->left;

cc1->right = NULL;

buff2->left = NULL;

cout << "\nЧеловека с похожим адресом встретился и был удален в списке по индексу:" << index1 << endl;

return buff2;

}

if (buff2->left == NULL && buff2->right != NULL) {

cc1 = buff2->right;

cc1->left = NULL;

hash[index1] = cc1;

buff2->right = NULL;

cout << "\nЧеловека с похожим адресом встретился и был удален в списке по индексу:" << index1 << endl;

return buff2;

}

if (buff2->right == NULL && buff2->left == NULL) {

hash[index1] = NULL;

cout << "\nЧеловека с похожим адресом встретился и был удален в списке по индексу:" << index1 << endl;

return buff2;

}

if (buff2->right != NULL && buff2->left != NULL) {

cc1 = buff2->left;

Human\* cc2= buff2->right;

cc1->right = cc2;

cc2->left = cc1;

buff2->left = NULL;

buff2->right = NULL;

cout << "\nЧеловека с похожим адресом встретился и был удален в списке по индексу:" << index1 << endl;

return buff2;

}

}

else {

cout << "\nВ списке нет человека с таким адресом";

}

}

}

//Функция отмкны последней операции удаления

void del\_return(Human\*\* hash, int size, Human\* del) {

if (size < 1) {//Проверка длинны хэш таблицы

cout << "\nERROR\n";

return;

}

if (del==NULL ) {//Проверка длинны хэш таблицы

cout << "\nНет доступа к последней операции удаления\n";

return;

}

add\_inhash(hash, size, del);

del = NULL;

}

//Функция для записи массива в файл

void file1(Human\* mas, int size) {

if (size < 1) {

cout << "\nERROR\n";

return;

}

ofstream file("123.txt");

Human buff;//Буферный экземпляр структуры

if (!file) {//Проверка наличия файла

cout << "Произошла ошибка\n";

exit(1);

}

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

buff = mas[j];//Занесение экземпляра массива в буферную переменную

file << buff.fullname << " " << buff.phone\_number << ' ' << buff.add << " ";

}

file << " !";//Маячок конца файла

file.close();

}

//Функция для считывания масиива с файла

void file2(Human \* mas , int size ) {

if (size < 1) {//Проверка длинны хэш таблицы

cout << "\nERROR\n";

exit(1);

}

ifstream file("123.txt");

if (!file) {//Проверка наличия файла

cout << "Произошла ошибка\n";

exit(1);

}

string temp = "";

Human buff;//Буферный экземпляр структуры

int i = 0;

while ((file >> temp) && (i< size) ) {

if (temp == "!") break;//Конец чтения если достигнут конец файла

buff.fullname = temp;

file >> temp;

buff.phone\_number = temp;

file >> temp;

buff.add = temp;

mas[i] = buff;

i++;

}

file.close();//Закрытие файла

return;

}

//Функция очищения массива

void unich\_mas(Human\* mas, int ss) {

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

mas[i].fullname = "NULL";

mas[i].phone\_number = "NULL";

mas[i].add = "NULL";

}

}

//Функция создания хэш таблицы

void make\_hash(Human\*\* hash, int ss,int mm, Human \*mas) {

if (ss < 1) {

cout << "\nERROR\n";

return;

}

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

Human\* point = &mas[i];//Буферный указатель на узел массива

add\_inhash(hash, mm , point);

}

cout << "\nКоличесво колизий:(" << collusions << ") на длинну массива:" << ss<<endl;

}

int main() {

srand(time(NULL));

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int ss =100;//Переменнная отвечающая за длинну массива

int mm=90;//Переменнная отвечающая за длинну хэша

Human\* mas=new Human[ss];//Массив структур Human

Human\* del;//Буферный указатель на удаляемый узел

del = NULL;

Human\* bb = NULL;//Указатель на добавляемый элемент

Human\*\* hash=new Human\*[mm];//Хэш таблица

Human cc;

for (int y = 0; y < mm; y++) {

hash[y] = NULL;

}

int x;

while (true) {

cout << endl << "\nВВедите № команды";

cout << endl << "1)Заполнить массив случайными узлами";

cout << endl << "2)Вывести в консоль масcив структур";

cout << endl << "3)Очистить массив";

cout << endl << "4)Сохранить массив в файл";

cout << endl << "5)Считать массив с файла";

cout << endl << "6)Создать хэш таблицу";

cout << endl << "7)Вывести в консоль хэш таблицу";

cout << endl << "8)Удалить по адресу";

cout << endl << "9)Отмена последней операции удаления по адресу";

cout << endl << "10)Поиск в хэш таблице";

cout << endl << "11)Добавить случайный элемент в хэш таблицу";

cout << endl << "12)Выход из программы";

cout << endl << "№="; cin >> x;

switch (x) {

case 1:mas\_random(mas, ss); break;

case 2:show\_mas(mas,ss); break;

case 3:unich\_mas(mas, ss); break;

case 4:file1(mas, ss); break;

case 5:file2(mas, ss); break;

case 6:make\_hash(hash, ss, mm,mas); break;

case 7:show\_hash(hash,ss); break;

case 8: del = NULL; del = udalit1(hash, mm); break;

case 9:del\_return(hash, mm, del); del = NULL; break;

case 10: find(hash, mm); break;

case 11:

cc = random\_human();//Инициализация указателя на случайный экземпляр структуры

bb = &cc;

cout << endl <<"Новый элемент";

cout << "\nИмя:" << bb->fullname;

cout << "\nНомер телефона:" << bb->phone\_number;

cout << "\nАдресс:" << bb->add;

add\_inhash(hash, 40, bb);

break;

case 12: return 0;

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

}

}

return 0;

}

**Блок схемы**

**Метод открытой адресации**

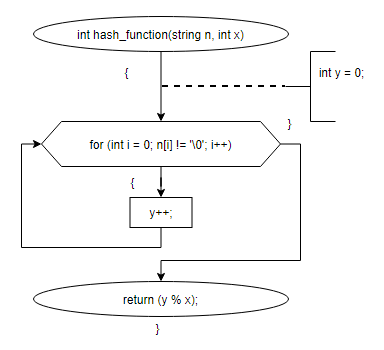


Рисунок 1-Блок схема хэш функции

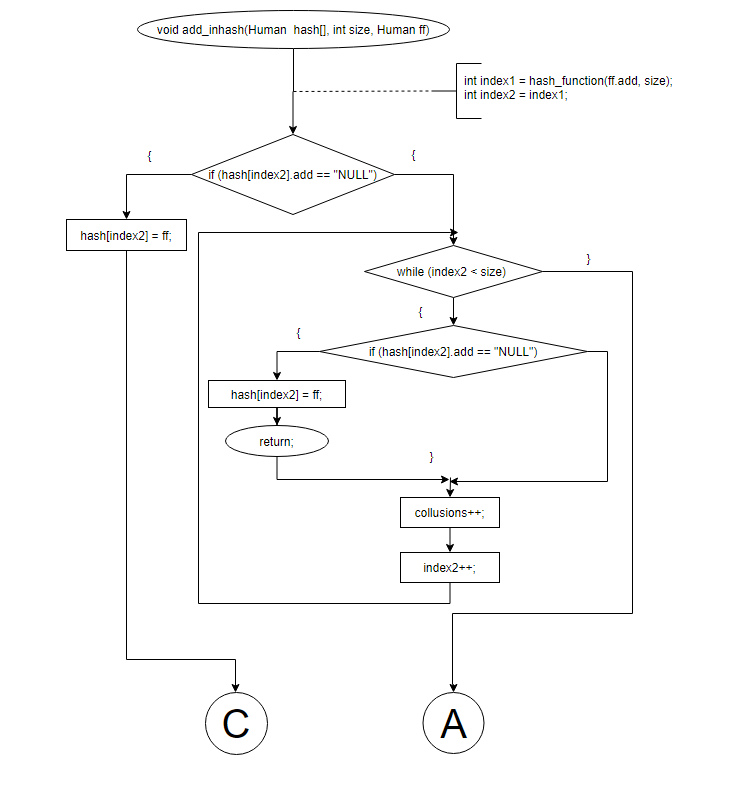


Рисунок 2-Функция занесения элемента в хэш таблицу

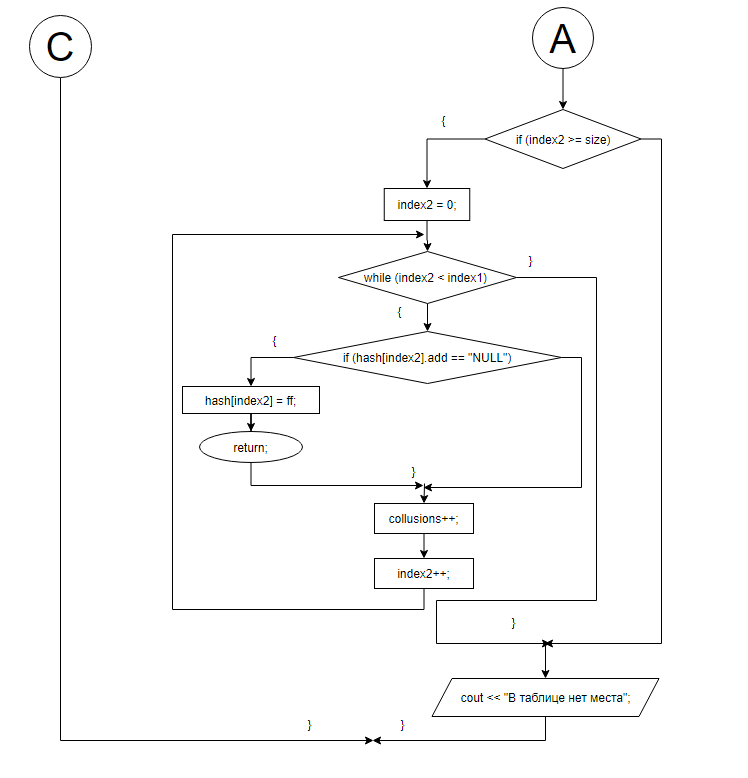
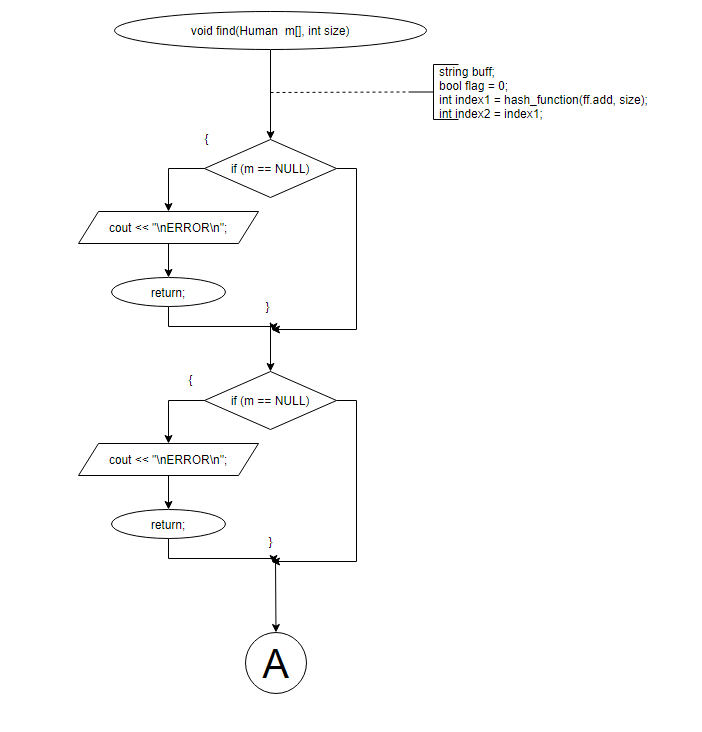


Рисунок 3-Функция занесения элемента в хэш таблицу

Рисунок 4-Функция поиска элемента в хэш таблице

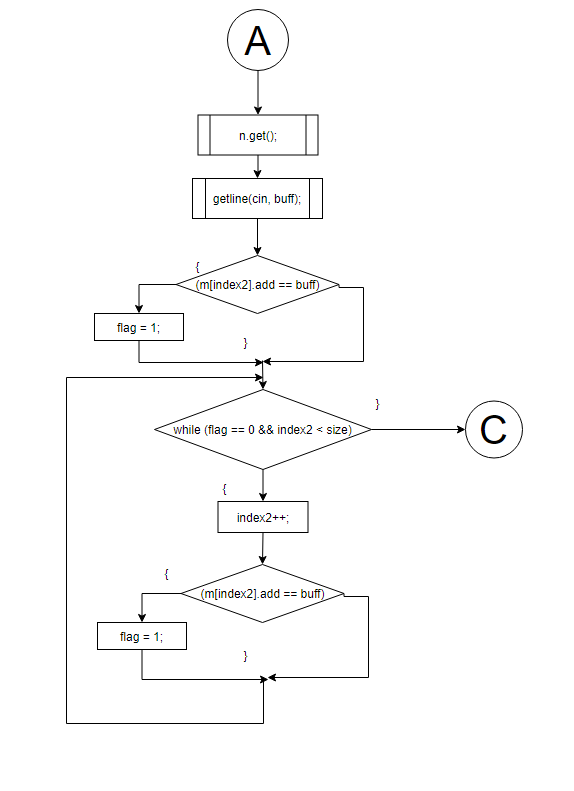


Рисунок 5-Функция поиска элемента в хэш таблице

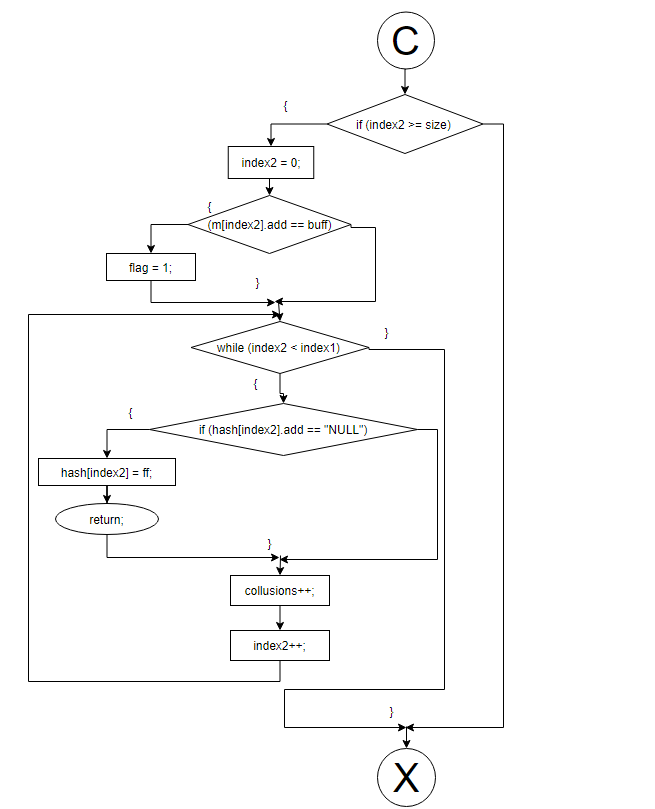


Рисунок 6-Функция поиска элемента в хэш таблице

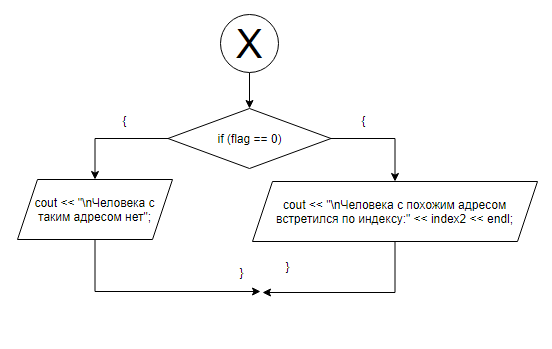


Рисунок 7-Функция поиска элемента в хэш таблице

**Метод цепочек**

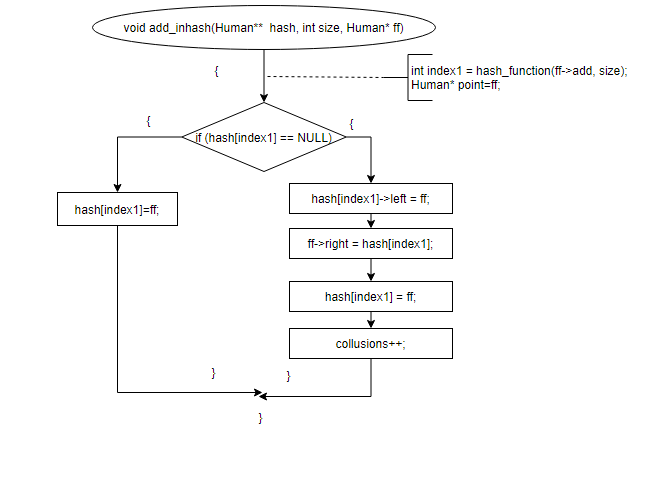


Рисунок 8-Функция занесения элемента в хэш таблицу.

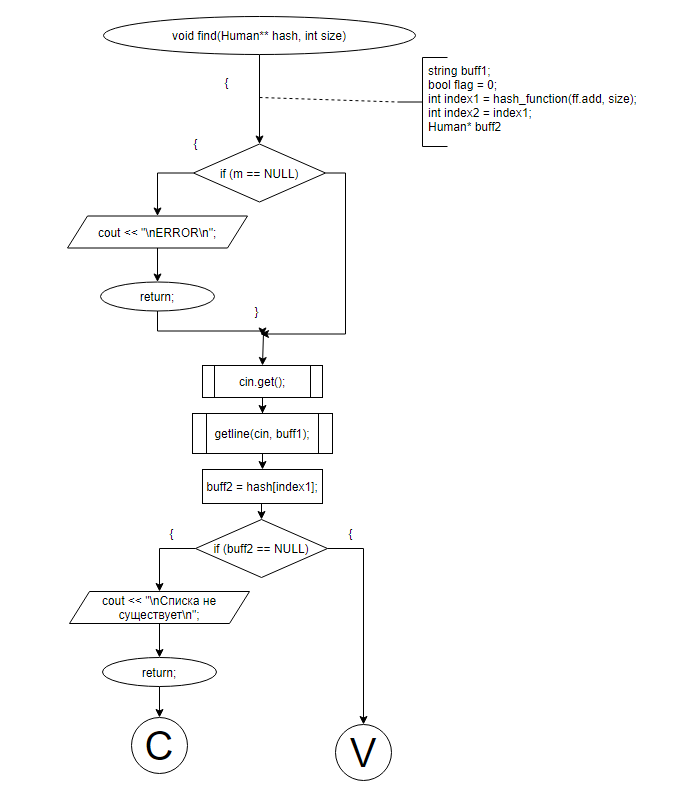


Рисунок 9-Функция поиска элемента в хэш таблице

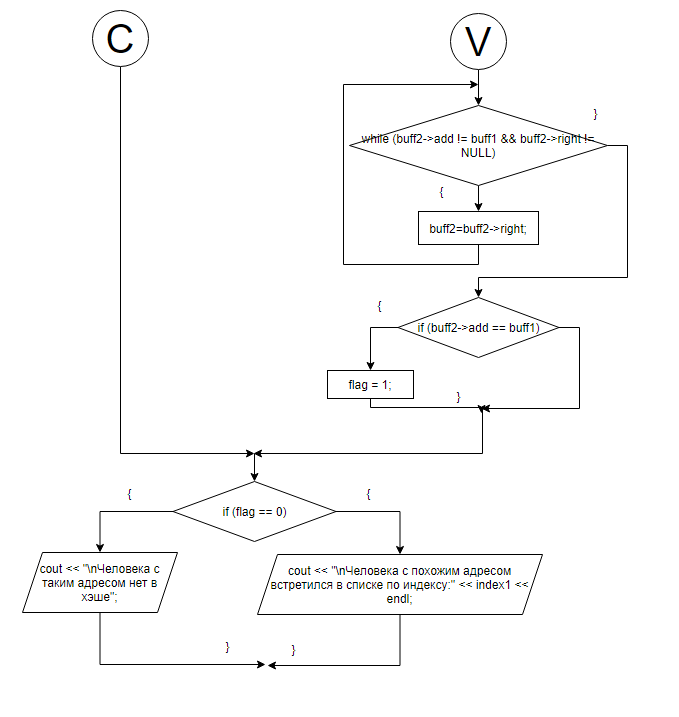


Рисунок 10-Функция поиска элемента в хэш таблице

**Выводы программ**



Рисунок 11-Вывод в консоль количества коллизий

при размере хэш таблицы 40, 75, 90 (Метод цепочек)



Рисунок 12-Вывод в консоль количества коллизий

при размере хэш таблицы 40 (Метод открытой адресации)



Рисунок 13-Вывод в консоль количества коллизий

при размере хэш таблицы 75 (Метод открытой адресации)



Рисунок 12-Вывод в консоль количества коллизий

при размере хэш таблицы 90 (Метод открытой адресации)