Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Информатика»

Тема: Лабораторная работа 11

"Работа с хеш-таблицами"

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Баженов Т.И.

Проверил

Доцент кафедры ИТАС

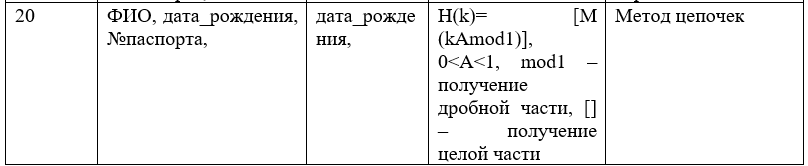
Полякова О.А.

Г. Пермь-2023

Постановка задачи

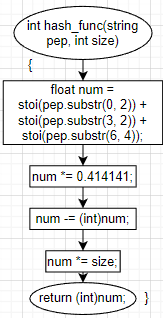
1. Создать динамический массив из записей (в соответствии с вариантом), содержащий не менее 100 элементов. Для заполнения элементов массива использовать ДСЧ.
2. Предусмотреть сохранение массива в файл и загрузку массива из файла.
3. Предусмотреть возможность добавления и удаления элементов из массива (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 20

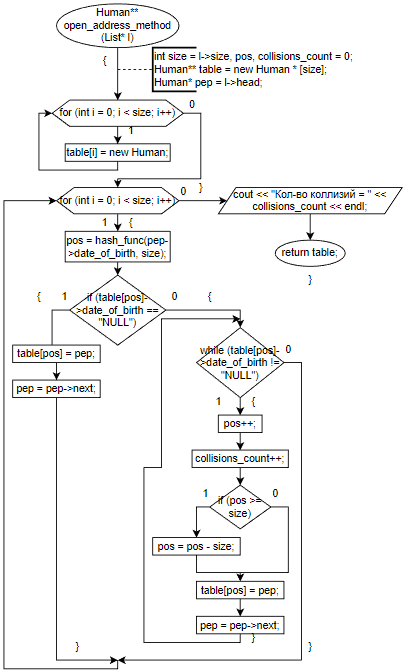


Блок-схема

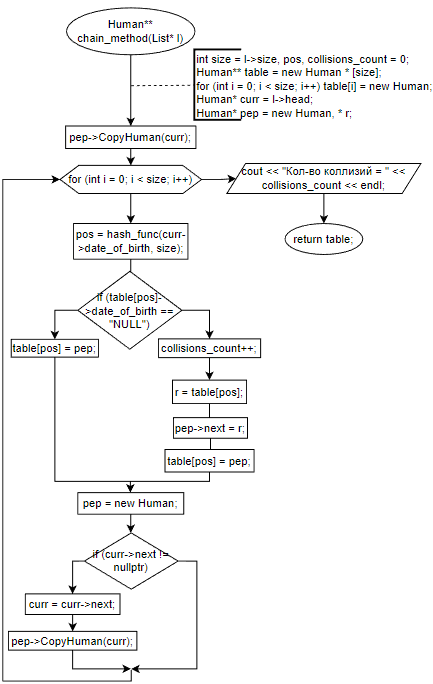
# Блок-схема



*Рисунок 1 - Функция hash\_func в блок-схеме*

**

*Рисунок 2 - Функция open\_adress\_method в блок-схеме*

**

*Рисунок 3 - Функция chain\_method в блок-схеме*

Реализация в коде

#include <iostream>

#include <string>

#include <Windows.h>

#include <ctime>

#include <fstream>

using namespace std;

string Names[10] = { "Андрей", "Иван", "Петр", "Дмитрий", "Евгений", "Антон", "Олег", "Артем", "Глеб", "Максим"};

string Patronymics[10] = { "Андреевич", "Иванович", "Петрович","Дмитриевич", "Евгеньевич", "Антонович", "Олегович", "Артемович", "Глебович", "Максимович"};

string Surnames[10] = { "Смирнов", "Маслов", "Сафронов", "Кузнецов", "Горбачёв", "Вариков", "Филатов", "Аксенов", "Боженов", "Лужков"};

string Passports[10] = { "4759\_969117", "4481\_920688", "4340\_756756", "4340\_756756", "4793\_464759", "5717\_729432", "5717\_726552", "5718\_707432", "5718\_723419", "5717\_734532"};

string Dates[10] = { "15.11.1969", "27.01.1996", "23.10.1972", "13.04.1993", "20.07.1985", "01.04.1988", "13.07.1999", "19.09.1964", "26.11.1987", "30.12.1982"};

struct Human {

    string first\_name = "NULL";

    string last\_name = "NULL";

    string patronymic = "NULL";

    string date\_of\_birth = "NULL";

    string number\_of\_passport = "NULL";

    Human\* next = nullptr;

    void SetHuman(string first\_name, string last\_name, string patronymic, string date\_of\_birth, string number\_of\_passport) {

        this->first\_name = first\_name;

        this->last\_name = last\_name;

        this->patronymic = patronymic;

        this->date\_of\_birth = date\_of\_birth;

        this->number\_of\_passport = number\_of\_passport;

    }

    void OutputHuman() {

        cout << "ФИО:\t" << this->last\_name << ' ' << this->first\_name << ' ' << this->patronymic << "\n";

        cout << "Дата рождения:\t" << this->date\_of\_birth << "\n";

        cout << "Серия и номер паспорта:\t" << this->number\_of\_passport << "\n";

    }

    void CopyHuman(Human\* copy) {

        this->first\_name = copy->first\_name;

        this->last\_name = copy->last\_name;

        this->patronymic = copy->patronymic;

        this->date\_of\_birth = copy->date\_of\_birth;

        this->number\_of\_passport = copy->number\_of\_passport;

        this->next = nullptr;

    }

};

struct List {

    int size = 0;

    Human\* head = nullptr;

    Human\* tail = nullptr;

};

bool ListIsEmpty(List\* list);

void PushBack(List\* list);

void PushFront(List\* list);

List\* CreateList(int size);

void ShowList(List\* list);

void RemoveElement(List\* list, string k);

void RemoveElement(List\* list, int n);

void AddElement(List\* list, int number);

void PrintInFile(string filename, List\* list);

void PrintFromFile(string filename, List\* list);

bool ListIsEmpty(List\* list) {

    return list == nullptr or list->size == 0;

}

void PushBack(List\* list) {

    Human\* p = new Human;

    string name = Names[rand() % 10];

    string surname = Surnames[rand() % 10];

    string patr = Patronymics[rand() % 10];

    string date = Dates[rand() % 10];

    string passport = Passports[rand() % 10];

    p->SetHuman(name, surname, patr, date, passport);

    p->next = nullptr;

    if (ListIsEmpty(list)) {

        list->head = p;

        list->tail = p;

    }

    else if (list->size == 1) {

        list->tail->next = p;

        list->tail = p;

        list->head->next = list->tail;

    }

    else {

        list->tail->next = p;

        list->tail = p;

    }

    list->size++;

}

void PushFront(List\* list) {

    Human\* p = new Human;

    string surname = Surnames[rand() % 10];

    string name = Names[rand() % 10];

    string patr = Patronymics[rand() % 10];

    string date = Dates[rand() % 10];

    string passport = Passports[rand() % 10];

    p->SetHuman(name, surname, patr, date, passport);

    if (ListIsEmpty(list)) {

        list->head = p;

        list->tail = p;

    }

    else {

        p->next = list->head;

        list->head = p;

    }

    list->size++;

}

List\* CreateList(int size) {

    List\* list = new List;

    for (int i = 0; i < size; i++)

        PushBack(list);

    return list;

}

void ShowList(List\* list) {

    if (ListIsEmpty(list)) cout << "Список пуст!\n";

    else {

        int i = 1;

        Human\* temp = list->head;

        cout << "\tСписок\n";

        while (i <= list->size){

            temp->OutputHuman();

            temp = temp->next;

            ++i;

            cout << "\n";

        }

    }

}

void RemoveElement(List\* list, string k) {

    Human\* temp = list->head;

    if (temp->date\_of\_birth == k) {

        list->head = temp->next;

        delete temp;

        list->size--;

        cout << "Элемент удалён!\n";

        return;

    }

    while (temp->next->date\_of\_birth != k) {

        temp = temp->next;

    }

    if (temp != nullptr) {

        cout << "Элемент удалён!\n";

        Human\* r = temp->next;

        temp->next = r->next;

        delete r;

        list->size--;

        r = temp->next;

        return;

    }

    else {

        cout << "Элемент не найден!\n";

    }

}

void RemoveElement(List\* list, int n) {

    Human\* temp = list->head;

    int i = 1;

    if (i == n) {

        list->head = temp->next;

        delete temp;

        list->size--;

        return;

    }

    while (i + 1 != n) {

        i++;

    }

    temp->next = temp->next->next;

    delete temp->next;

    list->size--;

    return;

}

void PrintInFile(string filename, List\* list) {

    ofstream fout;

    fout.open(filename);

    if (!fout.is\_open()) {

        cout << "Файл не открыт!\n";

    }

    else {

        cout << "Информация записана в файл!" << "\n";

        Human\* p = list->head;

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

            fout << p->last\_name << "\n";

            fout << p->first\_name << "\n";

            fout << p->patronymic << "\n";

            fout << p->date\_of\_birth << "\n";

            fout << p->number\_of\_passport << "\n";

            p = p->next;

        }

    }

    fout.close();

}

void PrintFromFile(string filename, List\* list) {

    ifstream fin;

    fin.open(filename);

    if (!fin.is\_open()) {

        cout << "Файл не открыт!\n";

    }

    else {

        cout << "Информация скопирована из файла!" << "\n";

        string word1, word2, word3, word4, word5;

        Human\* p, \* r;

        p = new (Human);

        fin >> word1 >> word2 >> word3 >> word4 >> word5;

        p->SetHuman(word1, word2, word3, word4, word5);

        p->next = nullptr;

        list->head = p;

        list->size++;

        while (!fin.eof()) {

            word1 = ""; word2 = ""; word3 = ""; word4 = ""; word5 = "";

            fin >> word1 >> word2 >> word3 >> word4 >> word5;

            r = new(Human);

            r->SetHuman(word2, word1, word3, word4, word5);

            if (word5 != "") {

                p->next = r;

                r->next = nullptr;

                p = r;

                list->size++;

            }

        }

    }

    fin.close();

}

void AddElement(List\* list, int number) {

    int i = 1;

    string key;

    Human\* temp = list->head;

    if (i == number) {

        PushFront(list);

        temp = list->head;

    }

    else {

        while (i + 1 != number) {

            temp = temp->next;

            i++;

        }

        Human\* p = new Human;

        p->next = temp->next;

        temp->next = p;

        string name = Names[rand() % 10];

        string surname = Surnames[rand() % 10];

        string patr = Patronymics[rand() % 10];

        string date = Dates[rand() % 10];

        string passport = Passports[rand() % 10];

        p->SetHuman(name, surname, patr, date, passport);

        temp = p;

        list->size++;

    }

}

int hash\_func(string pep, int size) {

    float num;

    num = stoi(pep.substr(0, 2)) + stoi(pep.substr(3, 2)) + stoi(pep.substr(6, 4));

    num \*= 0.414141;

    num -= (int)num;

    num \*= size;

    return (int)num;

}

Human\*\* open\_address\_method(List\* l) {

    int size = l->size, pos, collisions\_count = 0;

    Human\*\* table = new Human \* [size];

    for (int i = 0; i < size; i++) {

        table[i] = new Human;

    }

    Human\* pep = l->head;

    for (int i = 0; i < size; i++) {

        pos = hash\_func(pep->date\_of\_birth, size);

        if (table[pos]->date\_of\_birth == "NULL") {

            table[pos] = pep;

            pep = pep->next;

        }

        else {

            while (table[pos]->date\_of\_birth != "NULL") {

                pos++;

                collisions\_count++;

                if (pos >= size) pos = pos - size;

            }

            table[pos] = pep;

            pep = pep->next;

        }

    }

    cout << "Количество коллизий = " << collisions\_count << "\n";

    return table;

}

Human\*\* chain\_method(List\* l) {

    int size = l->size, pos, collisions\_count = 0;

    Human\*\* table = new Human \* [size];

    for (int i = 0; i < size; i++) table[i] = new Human;

    Human\* curr = l->head;

    Human\* pep = new Human, \* r;

    pep->CopyHuman(curr);

    for (int i = 0; i < size; i++) {

        pos = hash\_func(curr->date\_of\_birth, size);

        if (table[pos]->date\_of\_birth == "NULL") {

            table[pos] = pep;

        }

        else {

            collisions\_count++;

            r = table[pos];

            pep->next = r;

            table[pos] = pep;

        }

        pep = new Human;

        if (curr->next != nullptr) {

            curr = curr->next;

            pep->CopyHuman(curr);

        }

    }

    cout << "Количество коллизий = " << collisions\_count << "\n";

    return table;

}

void print\_hash(Human\*\* table, int size, int type) {

    if (type == 1) for (int i = 0; i < size; i++) {

        table[i]->OutputHuman();

    }

    else {

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

            cout << i << ":\n";

            Human\* pep = table[i];

            while (pep != nullptr){

                pep->OutputHuman();

                pep = pep->next;

                cout << "\n";

            }

        }

    }

}

void ShowMenu() {

    cout << "\t\tМеню\n";

    cout << "0 -  Завершение работы программы\n";

    cout << "1 - Создание списка\n";

    cout << "2 - Вывод списка\n";

    cout << "3 - Добавить элемент в начало списка\n";

    cout << "4 - Добавить элемент в конец списка\n";

    cout << "5 - Удалить элемент с заданым ключом\n";

    cout << "6 - Удалить элемент с заданым номером\n";

    cout << "7 - Добавить элемент с заданым номером\n";

    cout << "8 - Удалить список\n";

    cout << "9 - Записать данные в файл\n";

    cout << "10 - Записать данные из файла в список\n";

    cout << "11 - Метод открытой адрессации\n";

    cout << "12 - Метод цепочек\n";

}

void menu() {

    int a, size\_list, number, count;

    string key, file = "file.txt";

    List\* list = new List;

    Human\*\* table;

    Human\* pep;

    string date; int pos, ct = 0;

    bool flag;

    while (a != 0){

        system("pause");

        system("cls");

        ShowMenu();

        cin >> a;

        switch (a) {

        case 0:

            break;

        case 1:

            cout << "Введите размер списка: ";

            cin >> size\_list;

            list = CreateList(size\_list);

            break;

        case 2:

            ShowList(list);

            break;

        case 3:

            PushFront(list);

            break;

        case 4:

            PushBack(list);

            break;

        case 5:

            cout << "Введите ключ: ";

            getline(cin, key); getline(cin, key);

            RemoveElement(list, key);

            break;

        case 6:

            cout << "Введите номер заданного элемента: ";

            cin >> number;

            RemoveElement(list, number);

            break;

        case 7:

            cout << "Введите номер заданного элемента: ";

            cin >> number;

            AddElement(list, number);

            break;

        case 8:

            while (!ListIsEmpty(list)) {

                RemoveElement(list, 1);

            }

            cout << "Список удалён!\n";

            break;

        case 9:

            if (!ListIsEmpty(list))

                PrintInFile(file, list);

            else

                cout << "Список не создан!\n";

            break;

        case 10:

            PrintFromFile(file, list);

            break;

        case 11:

            ct = 0;

            table = open\_address\_method(list);

            print\_hash(table, list->size, 1);

            cout << "Введите дату рождения: ";

            getline(cin, date); getline(cin, date);

            while (date.length() != 10) {

                cout << "Введите дату рождения: ";

                getline(cin, date);

            }

            pos = hash\_func(date, list->size);

            if (table[pos]->date\_of\_birth == date) {

                table[pos]->OutputHuman();

            }

            else {

                while (table[pos]->date\_of\_birth != date and ct < list->size) {

                    pos++;

                    ct++;

                    if (pos >= list->size) {

                        pos = pos - list->size;

                    }

                }

                if (ct < list->size){

                    table[pos]->OutputHuman();

                }

                else{

                    cout << "Not found" << "\n";

                }

            }

            break;

        case 12:

            flag = false;

            table = chain\_method(list);

            print\_hash(table, list->size, 2);

            cout << "Введите дату рождения: ";

            getline(cin, date); getline(cin, date);

            while (date.length() != 10) {

                cout << "Введите дату рождения: ";

                getline(cin, date);

            }

            pos = hash\_func(date, list->size);

            pep = table[pos];

            while (!flag and pep != nullptr) {

                if (pep->date\_of\_birth == date) {

                    flag = true;

                }

                else {

                    pep = pep->next;

                }

            }

            if (flag){

                pep->OutputHuman();

            }

            else {

                cout << "Not found" << "\n";

            }

            break;

        }

    }

}

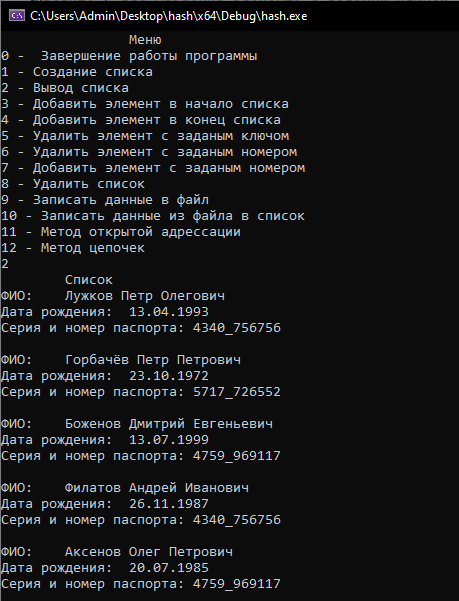
int main(){

    srand(time(NULL));

    menu();

    return 0;

}

Вывод в консоль  


Для 40 элементов

Для 75 элементов 

Для 90 элементов 