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: «Основы алгоритмизации и программирования»

Семестр 2

Тема: хеш-таблицы.

Выполнил работу

Студент группы РИС-22-1Б

Поважный В. Е.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

г. Пермь-2023

**Введение**

Для выполнения лабораторной работы требуется сделать отчет программы. Создание программ – отличный способ практики программирования.

**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 хеш-таблицу.
5. Подсчитать количество коллизий при размере хэш-таблицы 40, 75 и 90 элементов.

**Код программы**

#include <iostream>

#include <string>

#include <Windows.h>

#include <ctime>

#include <fstream>

using namespace std;

string Names[5] = { "Андрей", "Даниил", "Кирилл", "Алексей", "Тимур" };

string Surnames[5] = { "Веснин", "Летов", "Бердышев", "Казаров", "Мальцев" };

string Patronymics[5] = { "Андреевич", "Даниилович", "Кириллович", "Алексеевич", "Тимурович" };

string Dates[5] = { "30.12.2003", "08.01.2004", "05.05.2004", "14.03.2002", "15.06.2003" };

string Passports[5] = { "57\_17\_683322", "45\_23\_344453", "12\_34\_344212", "78\_45\_346213", "56\_16\_657435" };

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 << "Name:\t" << this->last\_name << ' ' << this->first\_name << ' ' << this->patronymic << endl;

cout << "Date\_of\_birth:\t" << this->date\_of\_birth << endl;

cout << "Passport:\t" << this->number\_of\_passport << endl;

}

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 || list->size == 0;

}

void PushBack(List\* list)

{

Human\* p = new Human;

string name = Names[rand() % 5];

string surname = Surnames[rand() % 5];

string patr = Patronymics[rand() % 5];

string date = Dates[rand() % 5];

string passport = Passports[rand() % 5];

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 name = Names[rand() % 5];

string surname = Surnames[rand() % 5];

string patr = Patronymics[rand() % 5];

string date = Dates[rand() % 5];

string passport = Passports[rand() % 5];

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 << "---------------------------\n";

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; //встать на удаляемый элемент

temp->next = r->next; //изменить ссылку

delete r;

list->size--;

r = temp->next; //встать на следующий

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 << "Не удалось открыть файл!\n";

else

{

cout << "Информация записана в файл!" << endl;

Human\* p = list->head;

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

{

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

fout << p->last\_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 << "Не удалось открыть файл!\n";

else

{

cout << "Информация скопирована из файла!" << endl;

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(word1, word2, 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() % 5];

string surname = Surnames[rand() % 5];

string patr = Patronymics[rand() % 5];

string date = Dates[rand() % 5];

string passport = Passports[rand() % 5];

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 << "Кол-во коллизий = " << collisions\_count << endl;

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 << "Кол-во коллизий = " << collisions\_count << endl;

return table;

}

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

{

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

{

table[i]->OutputHuman();

cout << "-----------" << endl;

}

else

{

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

{

cout << i << ":\n";

Human\* pep = table[i];

do

{

pep->OutputHuman();

pep = pep->next;

cout << endl;

} while (pep != nullptr);

cout << "-----------" << endl;

}

}

}

void menu()

{

int switch\_n, size\_list, number, count;

int pos, counter = 0;

string key, file = "MyFile.txt";

List\* list = new List;

Human\*\* table;

Human\* pep;

string date;

bool flag;

do

{

system("cls");

cout << "\t\tМеню\n";

cout << "Введите 0 для заверщения работы программы.\n";

cout << "Введите 1 для создания списка.\n";

cout << "Введите 2 для показа списка.\n";

cout << "Введите 3 для добавления элемента в начало списка.\n";

cout << "Введите 4 для добавления элемента в конец списка.\n";

cout << "Введите 5 для удаления элемента с заданым ключём.\n";

cout << "Введите 6 для удаления элемента с заданым номером.\n";

cout << "Введите 7 для добавления элемента с заданым номером.\n";

cout << "Введите 8, чтобы удалить список.\n";

cout << "Введите 9, чтобы записать данные в файл.\n";

cout << "Введите 10, чтобы записать данные из файла в список.\n";

cin >> switch\_n;

switch (switch\_n)

{

case 0:

cout << "Работа завершена!";

break;

case 1:

cout << "Введите размер списка: ";

cin >> size\_list;

while (size\_list < 1)

{

cout << "Размер списка должен быть от одного и более элементов! Введите размер: ";

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;

while (number <= 0 || number > list->size)

{

cout << "Номер должен быть больше нуля и меньше " << list->size + 1 << "!Введите номер : ";

cin >> number;

}

RemoveElement(list, number);

break;

case 7:

cout << "Введите номер заданного элемента: ";

cin >> number;

while (number <= 0 || number > list->size)

{

cout << "Номер должен быть больше нуля и меньше " << list->size + 1 << "!Введите номер : ";

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;

}

} while (switch\_n != 0);

}

int main()

{

srand(time(NULL));

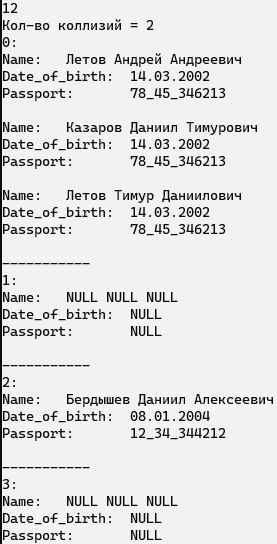
SetConsoleCP(1251); SetConsoleOutputCP(1251);

menu();

return 0;

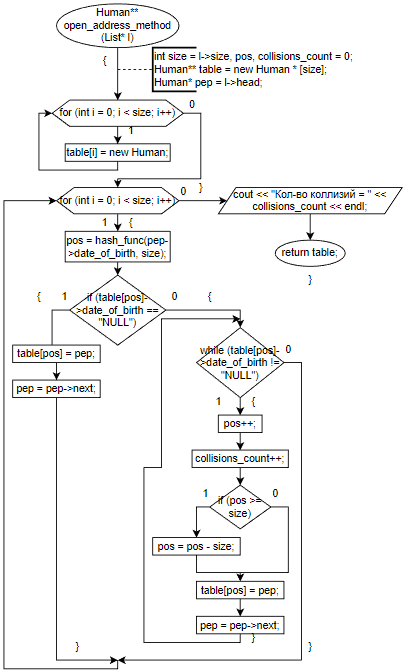
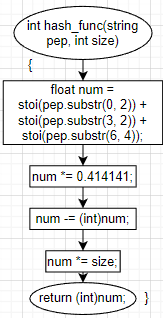
}

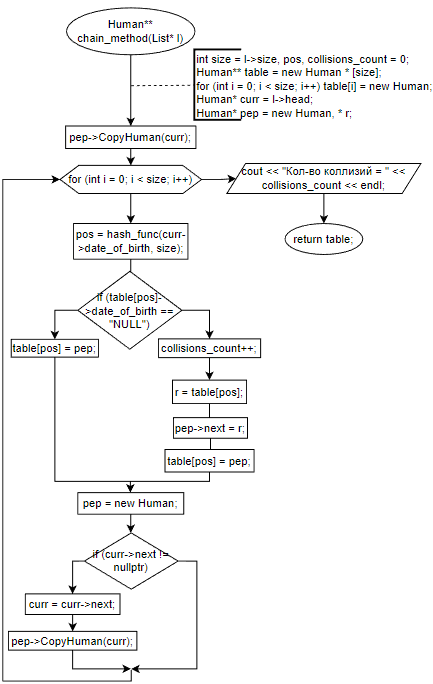
**Вывод программы**

****

***Рисунок 1 – вывод программы***

**Блок-схема**

****



***Рисунок 3 – Схема алгоритма программы***