Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное учреждение высшего образования  
«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет  
Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т  
по лабораторной работе**

Дисциплина: «Основы алгоритмизации и программирования»

Выполнил  
студент группы ИВТ 21-1Б  
Лебедев Т.А.

Проверил  
Яруллин Д.В.

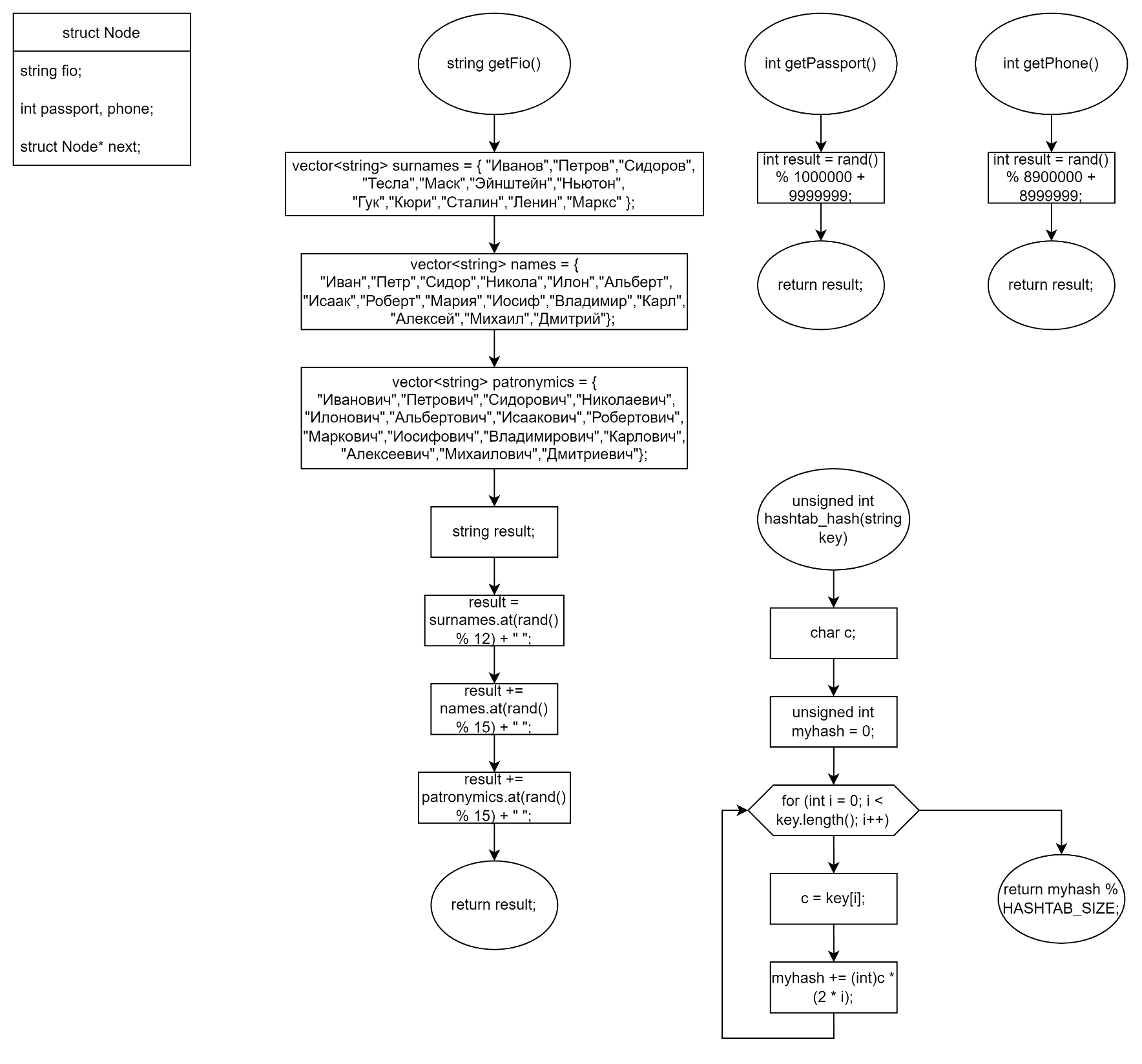
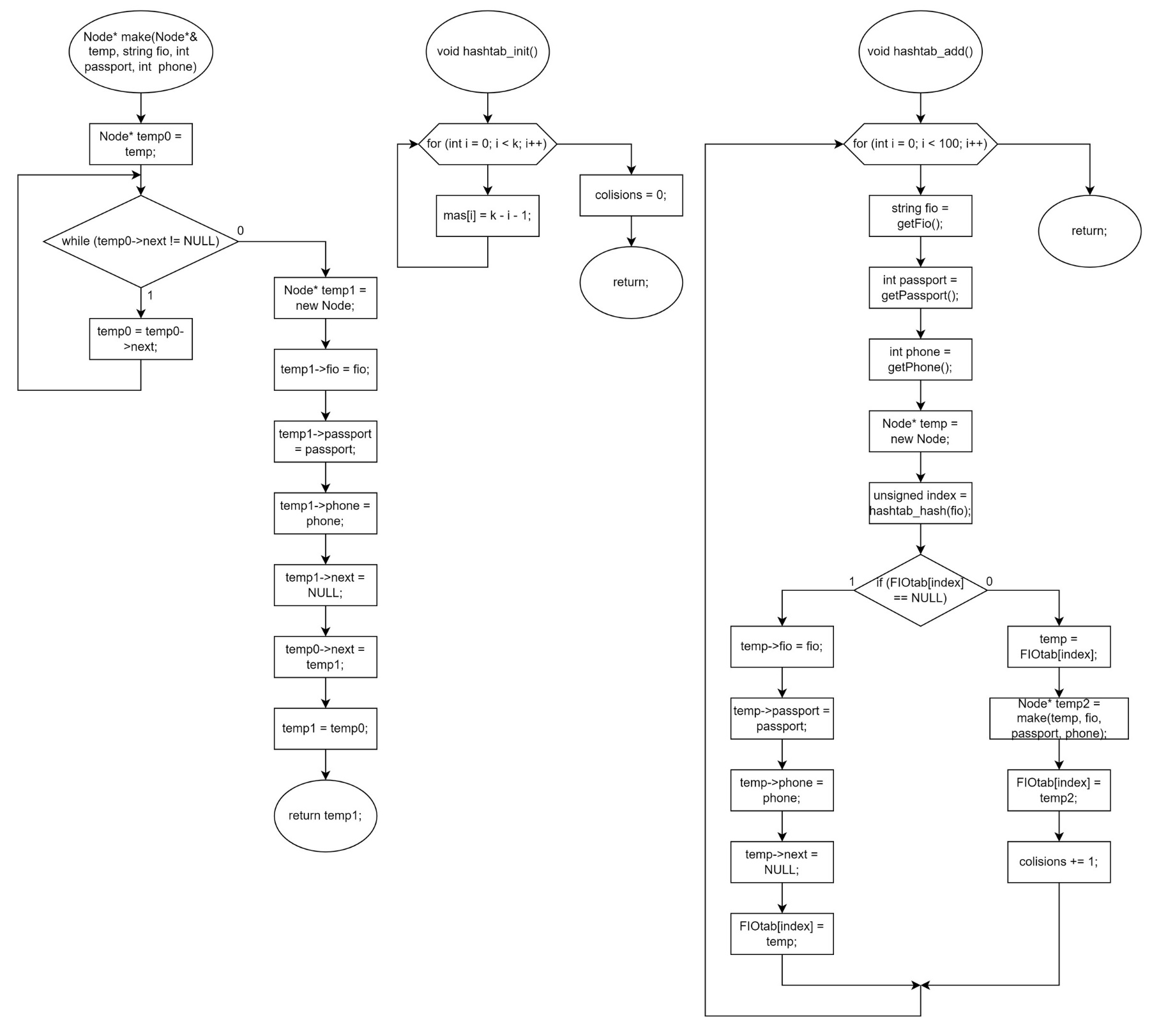
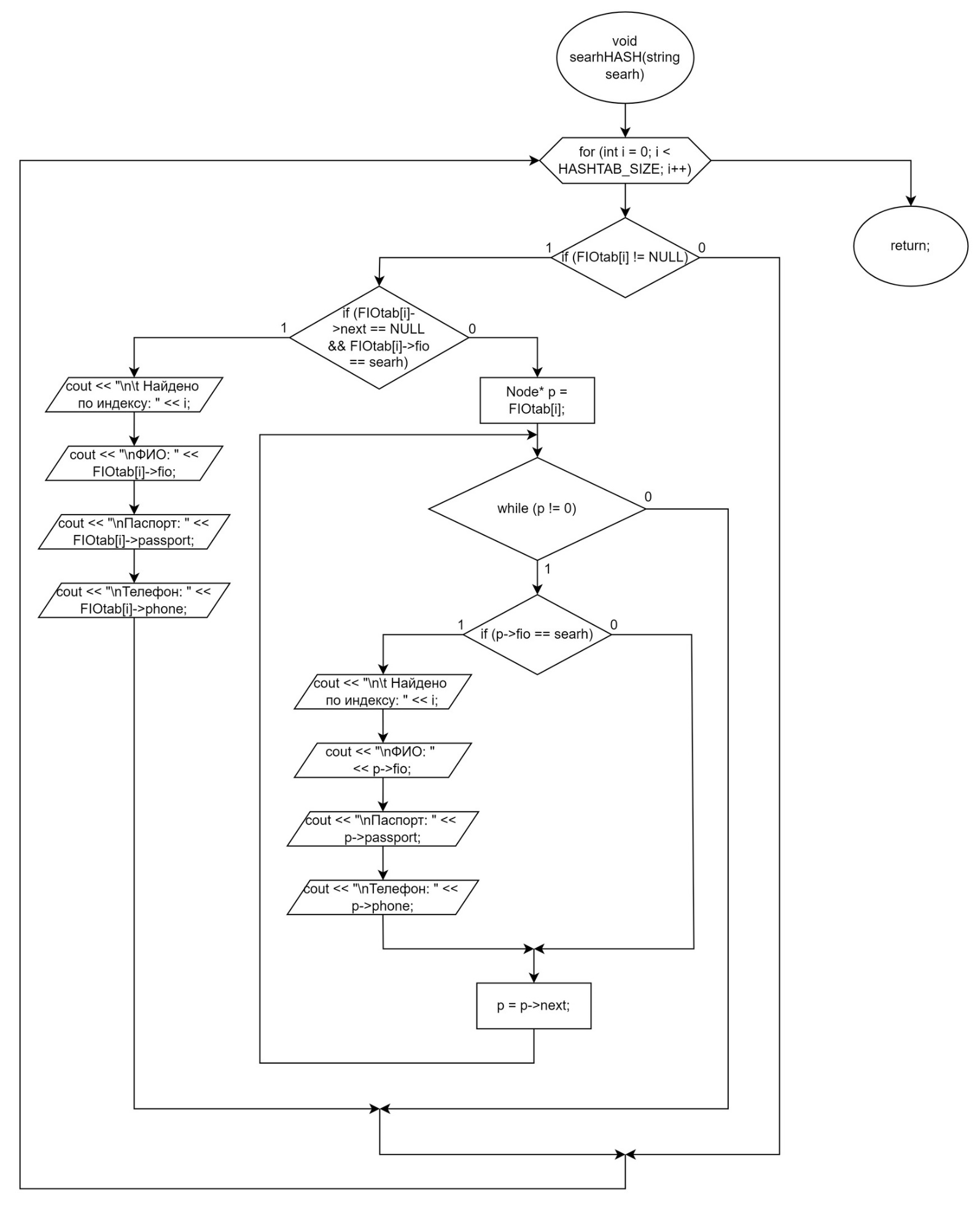
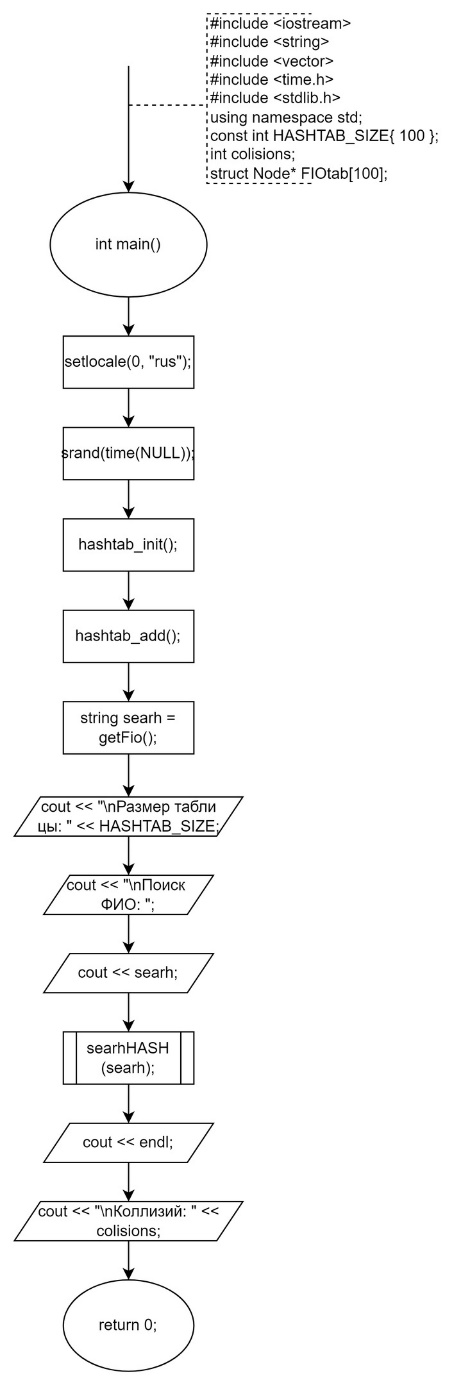
Пермь, 2022

Постановка задачи:

1. Создать динамический массив из записей (в соответствии с вариантом), содержащий не менее 100 элементов. Для заполнения элементов массива использовать ДСЧ.
2. Выполнить поиск элемента в массиве по ключу в соответствии с вариантом. Для поиска использовать хеш-таблицу.
3. Подсчитать количество коллизий при размере хеш-таблицы 40, 75 и 90 элементов.

Анализ задачи:

1. Хеширование – процесс превращения некоторых данных на входе в некоторую битовую последовательность фиксированной длины на выходе. Эта строка позволяет отличить ее от большинства других строк, так же полученных путем хеширования. Результат работы хеш-функции помогает различать одни данные от других, приводя в сравнение только значения функций хеширования, обработавших соответствующие данные, что очень удобно, так как конечные данные весят гораздо меньше информации, нежели данные, которым они соответствуют.
2. Хеш-таблица – структура данных, реализующая интерфейс ассоциативного массива. В ней хранятся пары, состоящие из ключа и хешированного ключа. Она позволяет добавлять новые пары, искать и удалять пары по ключу. Хеш-таблица формируется в определенном порядке хеш-функцией.
3. Прежде чем выполнять операции в хеш-таблице, вычисляется хеш-функция от ключа, результат которой является индексом в исходном массиве.
4. Коллизии необходимо разрешать, потому что они осложняют использование хеш-таблицы, нарушая однозначность соответствия между данными и их хешированными аналогами.
5. Для этого отводится место для хранения ключей, которые претендуют на ячейку хеш-таблицы, которая уже занята ключом, добавленным ранее. Этот механизм называется методом цепочек. Либо, если все ключи элементов известны заранее, соответственно, нет необходимости распределять элементы по ячейкам непосредственно в процессе добавления в таблицу, можно создать некоторую инъективную хеш-функцию, которая будет распределять ключи по ячейкам хеш-таблицы без коллизии. Такие хеш-таблицы, не нуждающиеся в механизме разрешения коллизий, называются хеш-таблицами с прямой или открытой адресацией.

Блок-схема:   

Код программы:

#include <iostream>

#include <string>

#include <vector>

#include <time.h>

#include <stdlib.h>

using namespace std;

const int HASHTAB\_SIZE{ 100 };

int colisions;

struct Node

{

string fio;

string passport;

string phone;

struct Node\* next;

};

struct Node\* FIOtab[100];

string getFio()

{

vector<string> surnames = { "Иванов","Петров","Сидоров","Тесла","Маск","Эйнштейн","Ньютон","Гук","Кюри","Сталин","Ленин","Маркс" };

vector<string> names = {

"Иван","Петр","Сидор","Никола","Илон","Альберт","Исаак","Роберт","Мария","Иосиф","Владимир","Карл",

"Алексей","Михаил","Дмитрий"

};

vector<string> patronymics = {

"Иванович","Петрович","Сидорович","Николаевич","Илонович","Альбертович","Исаакович","Робертович","Маркович",

"Иосифович","Владимирович","Карлович","Алексеевич","Михаилович","Дмитриевич"

};

string result;

result = surnames.at(rand() % 5) + " ";

result += names.at(rand() % 5) + " ";

result += patronymics.at(rand() % 5) + " ";

return result;

}

string make\_number()

{

string result;

vector<string>number = { "0","1", "2", "3", "4", "5", "6", "7", "8", "9" };

result = "89";

for (int i = 0; i < 9; i++)

{

result += number.at(rand() % 10);

}

return result;

}

string make\_birthday()

{

string result;

vector<string>day = { "01","02", "03", "04", "05", "06", "07", "08", "09", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31" };

vector<string>mounth = { "01","02", "03", "04", "05", "06", "07", "08", "09", "10", "11", "12", };

vector<string>year = { "1997","1998", "1999", "2000", "2001", "2002", "2003" };

result = day.at(rand() % 31) + ".";

result += mounth.at(rand() % 12) + ".";

result += year.at(rand() % 7);

return result;

}

unsigned int hashtab\_hash(string key)

{

char c;

unsigned int myhash = 0;

for (int i = 0; i < key.length(); i++)

{

c = key[i];

myhash += (int)c \* (2 \* i);

}

return myhash % HASHTAB\_SIZE;

}

Node\* make(Node\*& temp, string fio, string passport, string phone)

{

Node\* temp0 = temp;

while (temp0->next != NULL)

temp0 = temp0->next;

Node\* temp1 = new Node;

temp1->fio = fio;

temp1->passport = passport;

temp1->phone = phone;

temp1->next = NULL;

temp0->next = temp1;

temp1 = temp0;

return temp1;

}

void hashtab\_init()

{

for (int i = 0; i < 100; i++)

FIOtab[i] = NULL;

colisions = 0;

}

void hashtab\_add()

{

for (int i = 0; i < 100; i++) {

string fio = getFio();

string passport = make\_number();

string phone = make\_birthday();

Node\* temp = new Node;

unsigned index = hashtab\_hash(fio);

if (FIOtab[index] == NULL)

{

temp->fio = fio;

temp->passport = passport;

temp->phone = phone;

temp->next = NULL;

FIOtab[index] = temp;

}

else

{

temp = FIOtab[index];

Node\* temp2 = make(temp, fio, passport, phone);

FIOtab[index] = temp2;

colisions += 1;

}

}

}

void searhHASH(string searh)

{

for (int i = 0; i < HASHTAB\_SIZE; i++)

if (FIOtab[i] != NULL)

{

if (FIOtab[i]->next == NULL && FIOtab[i]->fio == searh) {

cout << "\n\t Найдено по индексу: " << i;

cout << "\nФИО: " << FIOtab[i]->fio;

cout << "\nТелефон: " << FIOtab[i]->passport;

cout << "\nДата рождения: " << FIOtab[i]->phone;

}

else

{

Node\* p = FIOtab[i];

while (p != 0)

{

if (p->fio == searh) {

cout << "\n\t Найдено по индексу: " << i;

cout << "\nФИО: " << p->fio;

cout << "\nТелефон: " << p->passport;

cout << "\nДата выхода: " << p->phone;

}

p = p->next;

}

}

}

}

int main()

{

setlocale(0, "rus");

srand(time(NULL));

hashtab\_init();

hashtab\_add();

string searh = getFio();

cout << "\nРазмер таблицы: " << HASHTAB\_SIZE;

cout << "\nПоиск ФИО: ";

cout << searh;

searhHASH(searh);

cout << endl;

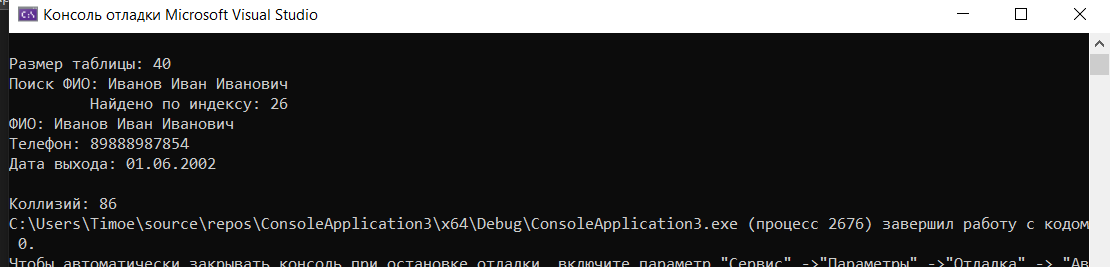
cout << "\nКоллизий: " << colisions;

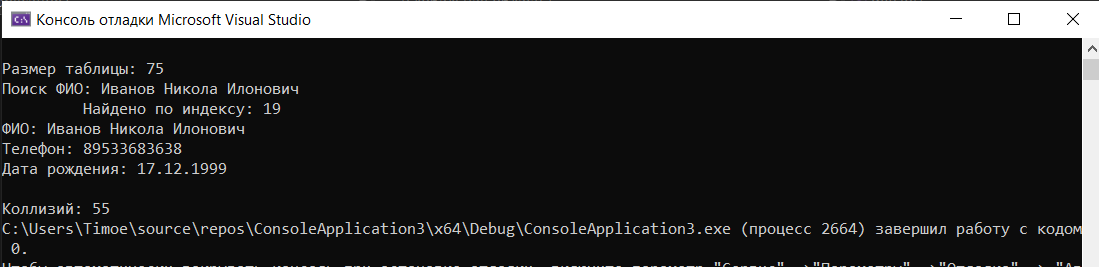
return 0;

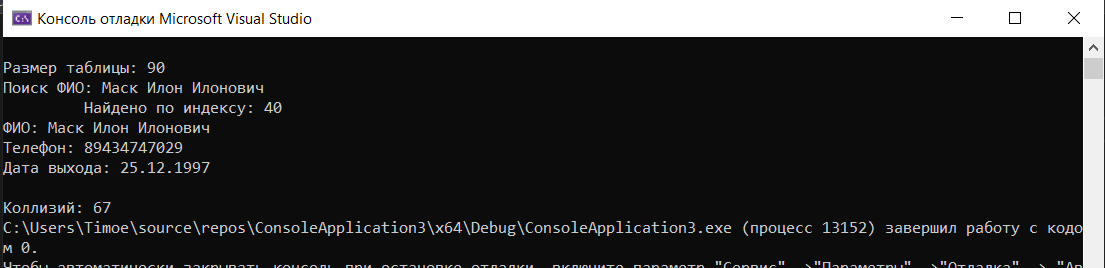
}

Скриншоты результатов:









Анализ результатов:

Программа работает правильно.