Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №14**

Дисциплина: «Основы теории алгоритмов и структуры данных»

Тема: Методы поиска: метод Кнутта-Мориса Пратта, метод Бойера-Мура

Вариант 19

Выполнила:

Студент группы ИВТ-20-2б

Ананина Арина Юрьевна

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

**Пермь, 2021**

1. 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 метод Бойера-Мура, Кнута-Морриса-Пратта.

2. Анализ задачи

1. В ходе работы были использованы следующие типы данных:
   1. Поля структуры Date – int.
   2. Поле FIO структуры Human типа string, passportNumber - int.
   3. Для хранения объектов структуры Human используется класс vector.
   4. Функции поиска, кроме Интерполяционного, возвращают тип данных int, функция Интерполяционного поиска возвращает тип данных bool.
   5. Для записи массива в файл используется класс ofstream.
   6. Функции поиска принимают в качестве параметров массив и ключ, типа Human.
2. Для решения задачи данные были представлены в следующем виде:
   1. Для хранения информации о человеке используется структура Human с 3 полями.
3. Структуры, которые использовались при решении задачи:
   1. Структура Human имеет метод заполнения объекта (GenerateData), которая заполняет поля объекта с помощью ДСЧ, для заполнения поля FIO используется 3 алфавита (имен, фамилий, отчеств). Номер паспорта - случайное число от 1 до 6000. Аналогично, заполняются поле DateOfBirth.
4. Для операций ввода и вывода использовались следующие операторы и функции:
   1. cin и cout в методе Print, в функции PrintArray.
   2. Getline, для ввода ФИО пользователем (используется в функции удаления элемента из массива по ключу)
5. Поставленные задачи будут решены следующими действиями:
   1. Функция BM преобразует исходный массив с номерами паспортов всех элементов и ключ в строки. Так же для создания массива сдвигов используется функия Reverse\_str, которая возвращает перевернутую строку. Массив сдвигов заполняется с помощью цикла for, i-ый элемент массива равен индексу i символа подстроки в перевернутой подстроке. Если же последний элемент подстроки встречается 1 раз, то в массив смещений для этого символа добавляется значение размера подстроки, иначе значение первого вхождения в эту подстроку. Для наглядности пользователю выводится исходная подстрока и массив сдвигов. Проход по элементам строк осуществляется с помощью цикла while, условием которого является проверка на выход за границы строки. Проверка символов подстроки в строке осуществляется с помощью индексов: current\_char, i, j (current\_char и j отвечают за строку, i - за подстроку). Внутри цикла проверяется сходство символом строки и подстроки, в случае совпадения, циклом while индексы i и j декрементируются, условием цикла является listOfNumbers[j] == substring[i] && i != 0. Если значение i равно нулю, то пользователю возвращается значение current\_char/8 +1, иначе проверяется наличие несовпавшего символа в алфавите подстроки. Если такой символ есть в алфавите, то current\_char сдвигается на количество равное i-ому элементу в массиве сдвигов, иначе сдвиг осуществляется на длину подстроки. Если элемента в массиве нет, то функция возвращает значение -1.

for (int i = substring.size() - 2; i >= 0; i--)

{

// Поиск равных символов в таблице

bool isEqual = false;

for (int j = 0; j < table.size() && !isEqual; j++)

{

isEqual = table[j].Symbol == substring[i];

}

if (!isEqual)

{

table.push\_back(Cell(substring[i], substring.size() - i - 1));

}

}

// Поиск

bool isElementFind = false;

for (int i = substring.size() - 1; !isElementFind && i < listOfNumbers.size(); )

{

// сравнение символов

int j = substring.size() - 1;

while (j >= 0 && listOfNumbers[i - substring.size() + 1 + j] == substring[j]) j--;

if (j < 0)

{

isElementFind = true;

}

else

{

// поиск символа в таблице

int k = 0;

while (k < table.size() && table[k].Symbol != listOfNumbers[i]) k++;

if (k == table.size())

{

i += substring.size();

}

else

{

i += table[k].Bias;

}

}

}

return isElementFind;

5.2 Функция KMP аналогично осуществляет преобразование ключа и массива в строки. Для заполнения массива префиксов реализована функция Prefix\_func, которая принимает в качестве параметра подстроку и возвращает массив префиксов. Массив префиксов заполняется с помощью цикла for, 0 индексу массива префиксов соответствует значение 0, так же важно отметить, что максимальный префикс в подстроке может увеличиваться только на 1, но изменяться на неограниченное количество (может стать нулевым). Для подсчета длины префиксов используются два указателя (i, j). J отвечает за проход по подстроке, i - за подсчет длины префикса. Заполнение осуществляется с помощью цикла while, условием которого проверяется выход за границы индексом j. В случае несовпадения i и j элементов подстроки, проверяется значения индекса i, если он нулевой, то в массив префиксов с индексом j добавляется нулевое значение, j инкрементируется. Если же i ненулевое значение, то i обнуляется. В случае совпадения i и j элементов подстроки, j индексу массива префиксов присваивается значение длины предыдущего префикса + 1, i и j инкрементируются.   
  
  
Для нахождения подстроки в строке используются два указателя на элементы строки и подстроки (k и l соответственно). Проход осуществляется с помощью цикла while, который проверят выход k за границы строки. Внутри цикла проверяется совпадение символов строки и подстроки. Если же они совпали, то индексы инкрементируются (так же проверяется значение l, если оно равно длине подстроки, то функция возвращает k/8). В ином случае проверяется значение l, если оно нулевое, то k инкрементируется, иначе l указывает на индекс подстроки равный значению массива pi с индексом [l-1]. Если элемента в массиве нет, функция возвращает -1.

bool KMP(vector<Human> humans, Human key)

{

string listOfNumbers;

string substring = "";

substring += to\_string(key.PassportNumber);

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

{

string newLine = "";

newLine += to\_string(humans[i].PassportNumber);

listOfNumbers += newLine;

}

bool result = false;

// Создание пи-таблицы

vector<int> pi(substring.size());

for (int j = 0, i = 1; i < substring.size(); i++) {

while ((j > 0) && (substring[i] != substring[j]))

j = pi[j - 1];

if (substring[i] == substring[j])

{

j++;

}

pi[i] = j;

}

// Поиск

for (int j = 0, i = 0; i < listOfNumbers.size(); ++i) {

while ((j > 0) && (substring[j] != listOfNumbers[i]))

j = pi[j - 1];

j++;

if (j == substring.size())

result = true;

}

return result;

}

3. Код программы

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

using namespace std;

struct Date

{

int Day, Month, Year;

};

struct Human

{

string FIO;

Date DateOfBirth;

long long PassportNumber;

void Print()

{

cout << endl;

cout << "ФИО: " << FIO << endl;

cout << "Дата рождения: " << DateOfBirth.Day << "." << DateOfBirth.Month << "." << DateOfBirth.Year << endl;

cout << "Номер паспорта: " << PassportNumber << endl;

}

void GenerateData()

{

string names[] = { "Иван", "Юрий", "Алексей", "Артём", "Дмитрий" };

string surnames[] = { "Иванов" , "Ананин", "Тарасов" };

string patronymic[] = { "Андреевич", "Вадимович", "Михаилович", "Кирилович", "Ильич" };

FIO = surnames[rand() % 3] + " " + names[rand() % 5] + " " + patronymic[rand() % 5];

DateOfBirth.Day = 1 + rand() % 28;

DateOfBirth.Month = 1 + rand() % 12;

DateOfBirth.Year = 1900 + rand() % 200;

PassportNumber = 5700000000 + (rand() % 100) \* 1000000 + rand() % 1000000;

}

};

vector<Human> CreateArray(int size = 100)

{

vector<Human> res;

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

{

Human newEl;

newEl.GenerateData();

res.push\_back(newEl);

}

return res;

}

void PrintArray(vector<Human> humans)

{

if (humans.size() < 1)

{

cout << "Список пуст" << endl;

return;

}

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

{

cout << "Элемент " << i << endl;

humans[i].Print();

}

}

bool KMP(vector<Human> humans, Human key)

{

string listOfNumbers;

string substring = "";

substring += to\_string(key.PassportNumber);

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

{

string newLine = "";

newLine += to\_string(humans[i].PassportNumber);

listOfNumbers += newLine;

}

bool result = false;

// Создание пи-таблицы

vector<int> pi(substring.size());

for (int j = 0, i = 1; i < substring.size(); i++) {

while ((j > 0) && (substring[i] != substring[j]))

j = pi[j - 1];

if (substring[i] == substring[j])

{

j++;

}

pi[i] = j;

}

// Поиск

for (int j = 0, i = 0; i < listOfNumbers.size(); ++i) {

while ((j > 0) && (substring[j] != listOfNumbers[i]))

j = pi[j - 1];

j++;

if (j == substring.size())

result = true;

}

return result;

}

struct Cell

{

char Symbol;

int Bias;

Cell(char c, int b)

{

Symbol = c;

Bias = b;

}

};

bool BM(vector<Human> humans, Human key)

{

string listOfNumbers;

string substring = "";

substring += to\_string(key.PassportNumber);

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

{

string newLine = "";

newLine += to\_string(humans[i].PassportNumber);

listOfNumbers += newLine;

}

//cout << substring << endl;

//cout << listOfNumbers << endl;

// Создание таблицы

vector<Cell> table;

for (int i = substring.size() - 2; i >= 0; i--)

{

// Поиск равных символов в таблице

bool isEqual = false;

for (int j = 0; j < table.size() && !isEqual; j++)

{

isEqual = table[j].Symbol == substring[i];

}

if (!isEqual)

{

table.push\_back(Cell(substring[i], substring.size() - i - 1));

}

}

// Добавление последнего элемента

reverse(table.begin(), table.end());

bool isEqual = false;

for (int j = 0; j < table.size() && !isEqual; j++)

{

isEqual = table[j].Symbol == substring[substring.size() - 1];

}

if (!isEqual)

{

table.push\_back(Cell(substring[0], substring.size()));

}

// Поиск

bool isElementFind = false;

for (int i = substring.size() - 1; !isElementFind && i < listOfNumbers.size(); )

{

// сравнение символов

int j = substring.size() - 1;

while (j >= 0 && listOfNumbers[i - substring.size() + 1 + j] == substring[j]) j--;

if (j < 0)

{

isElementFind = true;

}

else

{

// поиск символа в таблице

int k = 0;

while (k < table.size() && table[k].Symbol != listOfNumbers[i]) k++;

if (k == table.size())

{

i += substring.size();

}

else

{

i += table[k].Bias;

}

}

}

return isElementFind;

}

int main()

{

setlocale(LC\_ALL, "rus");

vector<Human> humans = CreateArray(100);

PrintArray(humans);

Human key;

cout << endl << "Введите номер паспорта, по которому необходимо искать элемент: ";

cin >> key.PassportNumber;

cout << "Бойера-Мура" << endl;

if (BM(humans, key))

{

cout << "Элемент найден" << endl;

}

else

{

cout << "Элемент не найден" << endl;

}

cout << "КМП" << endl;

if (KMP(humans, key))

{

cout << "Элемент найден" << endl;

}

else

{

cout << "Элемент не найден" << endl;

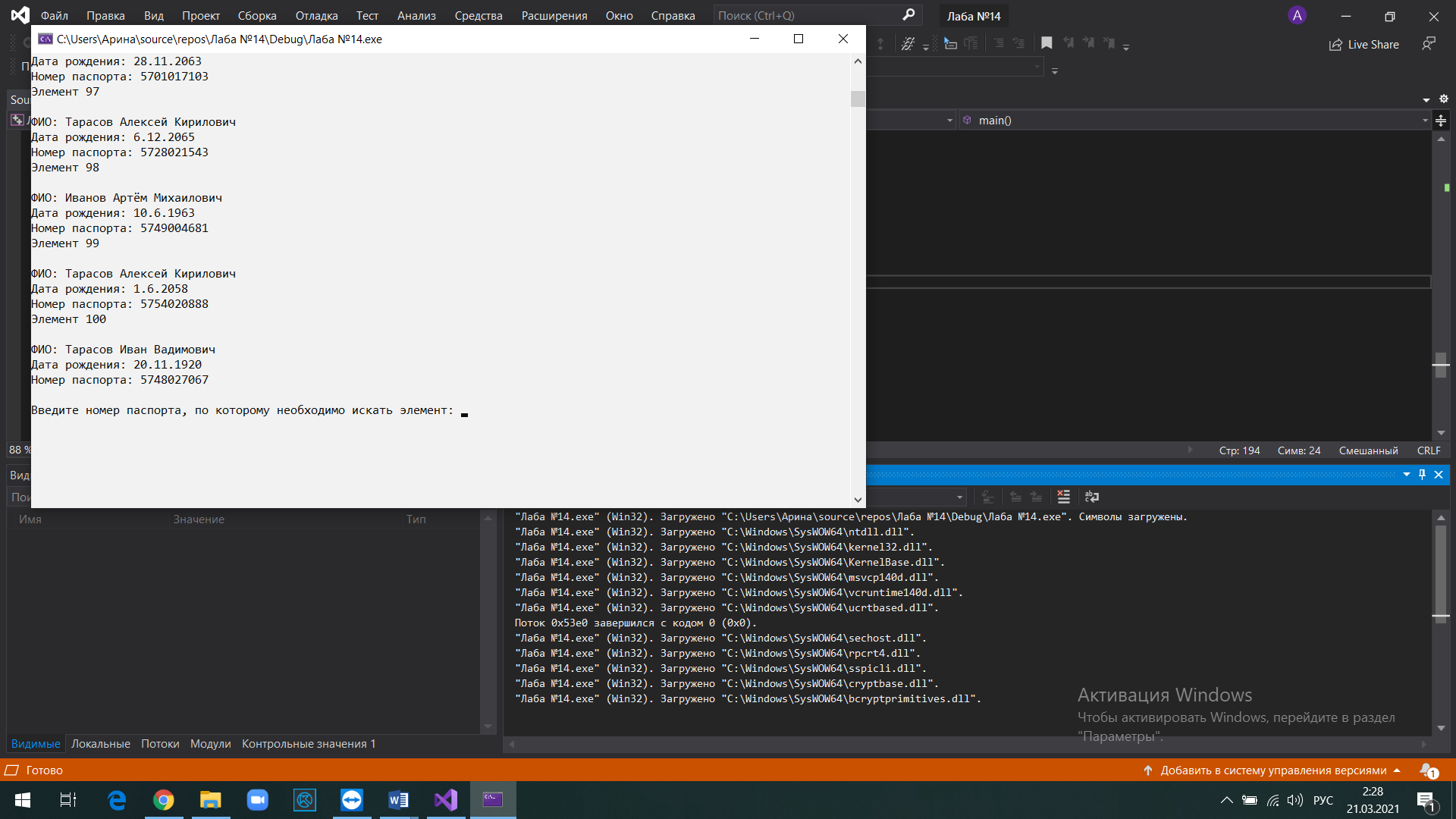
}

return 0;

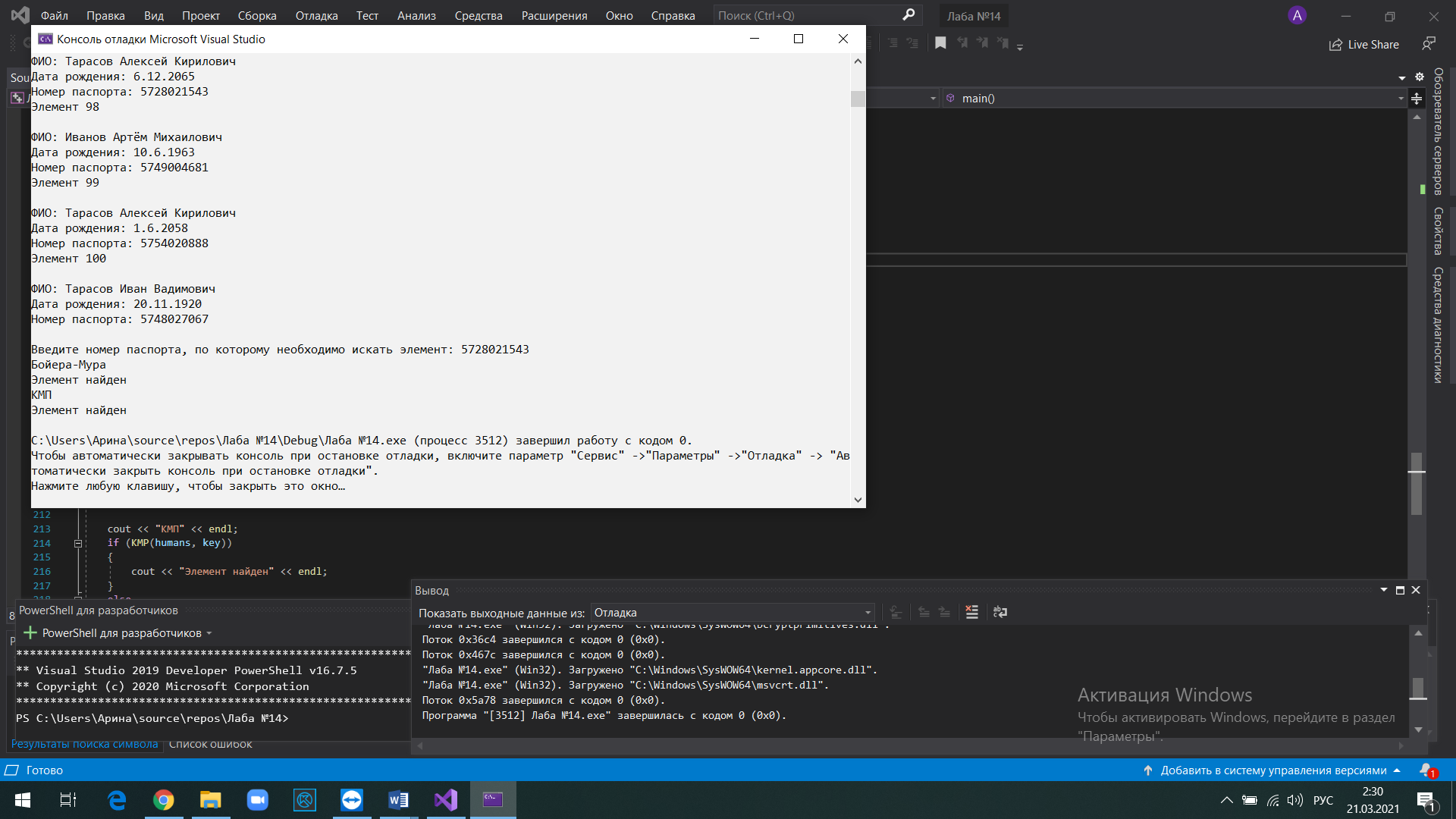
}

4. Выполнение программы.

Программа сразу же выводит на экран все 100 элементов и запрашивает ключ, по которому необходимо осуществить поиск.



При поиске элемента по заданному ключу программа выводит сообщение было ли найдено искомое значение.



5. Блок схема

