Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторным работам №12,14**

Дисциплина: основы алгоритмизации и программирования

Тема: “Методы поиска: линейный, интерполяционный, прямой поиск подстроки в строке, Бойера-Мура, Кнута-Морриса-Пратта"

**Вар.20**

Выполнил работу

студент группы ИВТ-20-2Б

Кузнецов Н.Д

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2020

**Цель задачи**

1) Получить практические навыки работы с поисками;

2) Получить практические навыки работы со списками;

3) Получить практические навыки работы с меню;  
Для достижения поставленной цели необходимо решить следующие задачи:

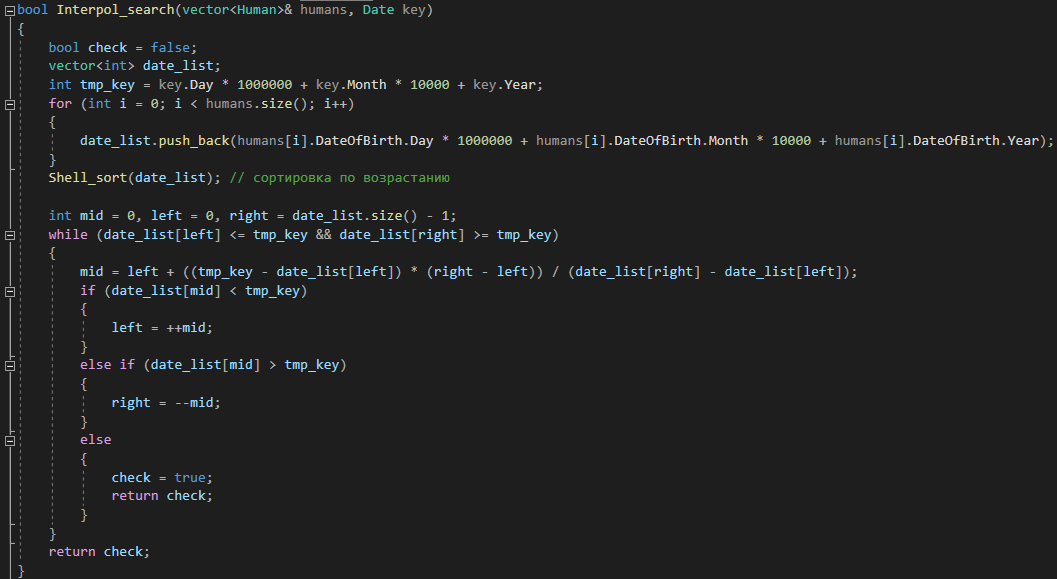
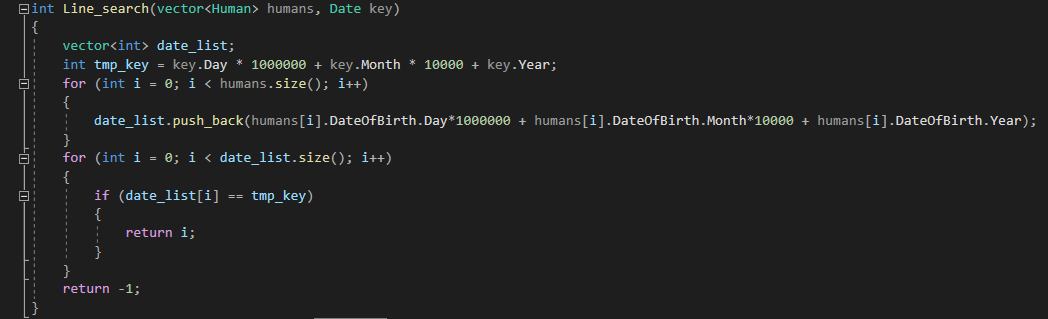
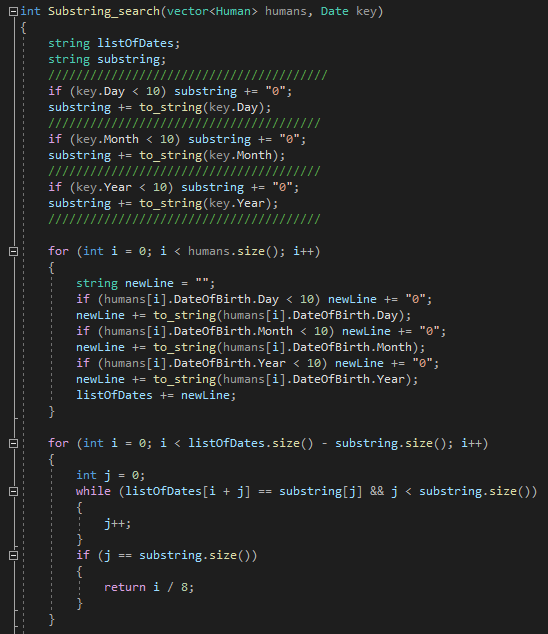
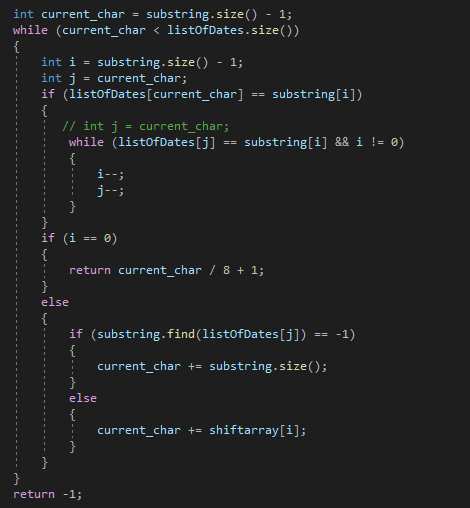
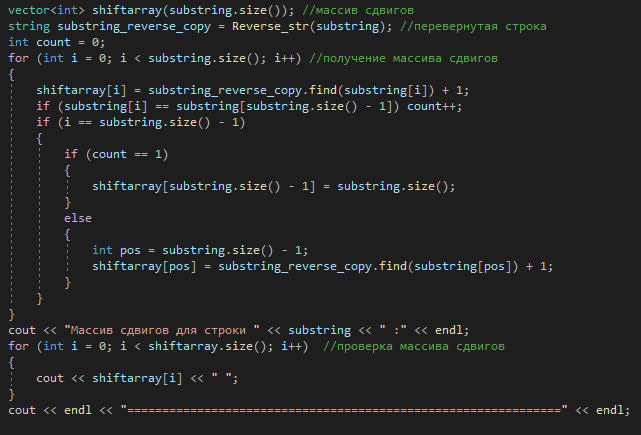
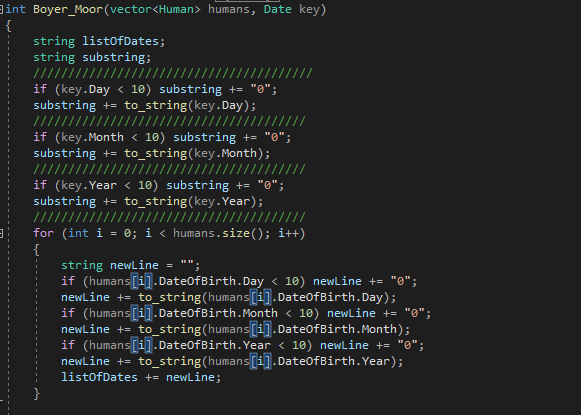
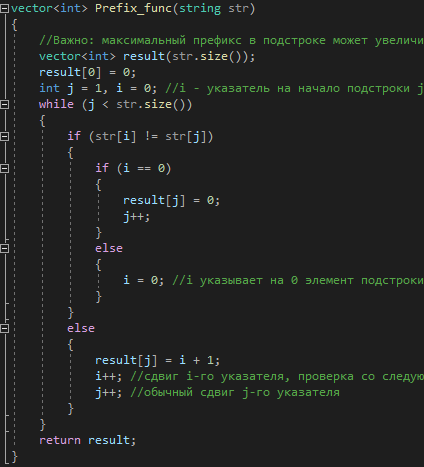
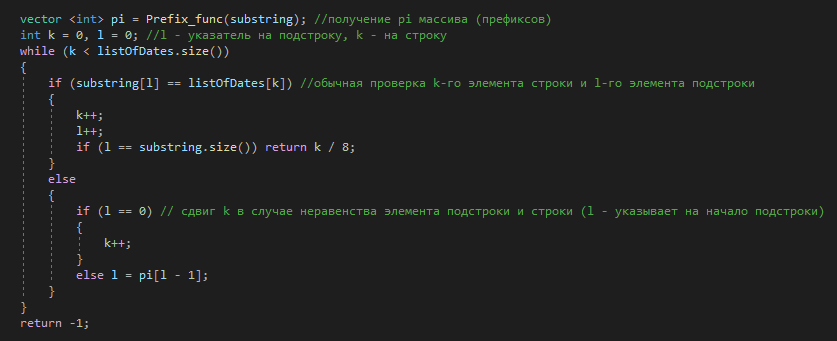
* Разработать структуру человека.
* Организовать ввод-вывод данных.
* Реализовать функцию удаления элементов из списка.
* Реализовать меню.
* Реализовать функцию отмены последнего удаления элемента.
* Разработать функцию, заполняющую массив объектов структуры случайными числами.
* Реализовать функции поиска.
* Реализовать функцию, сохраняющую измененный массив в файл.
* Разработать программу.

**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 линейного поиска, метод Прямого поиска подстроки в строке и интерполяционный метод.

ФИО, дата\_рождения, №паспорта,  
ключ: дата\_рождения,

**Анализ задачи**

1. Для решения задачи необходимо:
   1. Реализовать функцию получения корректного размера массива от пользователя.
   2. Разработать функцию меню.
   3. Разработать функцию, заполняющую массив с помощью ДСЧ.
   4. Реализовать функцию удаления элемента из массива.
   5. Разработать функции поиска по ключу.
   6. Реализовать структуру Human с полями FIO, DateOfBirth, passportNumber.
   7. Реализовать структуру Date, с полями Day, Month, Year. Поле DateOfBirth структуры Human будет типа Date.
   8. Разработать функцию записи массива в файл.
   9. Реализовать функцию сортировки массива для Интерполяционного поиска.
   10. Разработать функцию переворота строки для поиска Бойера-Мура.
   11. Разработать префикс-функцию для метода КМП.
2. 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 Date.
3. Для решения задачи данные были представлены в следующем виде:
   1. Для хранения информации о человеке используется структура Human с 3 полями (см.п.1.6 и п.1.7).
4. Структуры, которые использовались при решении задачи:
   1. Структура Human имеет метод заполнения объекта (Fill), 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.
   2. Структура Date имеет метод заполнения с помощью пользовательского ввода (Get\_date).
5. Для операций ввода и вывода использовались следующие операторы и функции:
   1. cin и cout в методе Get\_date, в функциях GetMenu, GetSize, Print\_list.
   2. Getline, для ввода ФИО пользователем (используется в функции удаления элемента из массива по ключу)
6. Поставленные задачи будут решены следующими действиями:
   1. Функция Interpol\_search преобразует массив объектов структуры в массив дат, приведенных к восьмизначному числу. Ключ преобразуется по такому же принципу. Для сортировки массива используется сортировка Шелла. С помощью цикла while, который выполняется до тех пор, пока значение левой границы меньше равно ключу и значение правой границы больше равно ключу. Внутри цикла рассчитывается индекс серединного элемента. Если значение серединного элемента меньше ключа, то левой границе присваивается инкрементированный индекс середины. Если же значение серединного элемента больше ключа, то правой границе присваивается декрементированный индекс середины. Если же значение mid совпадает с ключом, то функция возвращает true, в случае если элемента в массиве нет, то функция возвращает значение false.  
        
      
   2. Функция Line\_search преобразует ключ и массив по тому же принципу что и функция Interpol\_search (см.п.6.1). С помощью цикла for осуществляется проход по всем элементам массива, в случае удовлетворения элемента массива ключу, функция возвращает индекс i, иначе -1.  
      
   3. Функция Substring\_search преобразует массив объектов структуры Human в строку дат, ключ так же преобразовывается в строку (важно отметить, что поле даты так же приводится к восьмизнаковой строке). Проход осуществляется с помощью цикла for, который выполняется до тех пор, пока индекс не станет равен длине строки без подстроки. Внутри цикла for используется цикл while, который считает количество совпавших символов, если количество совпавших символов равно длине подстроки, то возвращается значение i/8.  
      
   4. Функция Boyer\_Moor аналогично функции Substring\_search преобразует исходный массив 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 listOfDates[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.  
      
   5. Функция KMP аналогично пунктам 6.4 и 6.3 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.  
      

**Код программы**

#include <iostream>

#include <string>

#include <vector>

#include <ctime>

#include <fstream>

using namespace std;

struct Date

{

int Day, Month, Year;

void Get\_date()

{

cout << "Введите день:\t";

cin >> Day;

cout << "Введите месяц:\t";

cin >> Month;

cout << "Введите год:\t";

cin >> Year;

}

};

struct Human

{

string FIO;

Date DateOfBirth;

unsigned int passportNumber;

void Fill()

{

srand(time(0));

vector <string> names = { "Nikolay", "Oleg", "Vadim", "Alexander", "Artem" }; //заранее подготовленные массивы фамилий

vector <string> surnames = { "Tedeev" , "Karelov", "Kuznetsov", "Galinov", "Chernykh" };

vector <string> patronymic = { "Andreevich","Alexandrovich", "Mikhailovich", "Petrovich","Dmitryevich" };

FIO = surnames[rand() % surnames.size()] + " " + names[rand() % names.size()] + " " + patronymic[rand() % patronymic.size()];

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

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

DateOfBirth.Year = rand() % 100 + 1921;

passportNumber = rand() % 6000 + 1;

}

};

int GetSize()

{

int size;

while (!(cin >> size) || (cin.peek() != '\n') || (size < 1))

{

cin.clear();

while (cin.get() != '\n')

{

cout << "Введите корректный размер списка:\t";

}

}

return size;

}

int GetMenu()

{

int menu;

cout << "Выберете действие:" << endl;

cout << "1. Создать список." << endl;

cout << "2. Вывести список." << endl;

cout << "3. Поиск элемента в списке." << endl;

cout << "4. Удалить элемент из списка." << endl;

cout << "5. Отменить удаление." << endl;

cout << "6. Сохранить изменения в файл." << endl;

cout << "7. Добавить элемент в список." << endl;

cout << "0. Выход." << endl;

cout << "Поле ввода:\t";

while (!(cin >> menu) || (cin.peek() != '\n') || (menu < 0) || (menu > 7))

{

cin.clear();

while (cin.get() != '\n')

{

cout << "Введите корректный номер функции:\t";

}

}

cout << endl;

return menu;

}

void Print\_list(vector <Human> list)

{

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

{

cout << "===============================================================" << endl;

cout << i + 1 << "-й человек в списке:" << endl;

cout << "ФИО:\t" << list[i].FIO << endl;

cout << "Дата рождения:\t" << list[i].DateOfBirth.Day << "." << list[i].DateOfBirth.Month << "." << list[i].DateOfBirth.Year << endl;

cout << "Номер паспорта:\t" << list[i].passportNumber << endl;

}

}

vector<Human> CreateArray(int size)

{

srand(time(0));

vector<Human> result;

vector <string> names = { "Nikolay", "Oleg", "Vadim", "Alexander", "Artem" }; //заранее подготовленные массивы фамилий

vector <string> surnames = { "Tedeev" , "Karelov", "Kuznetsov", "Galinov", "Chernykh" };

vector <string> patronymic = { "Andreevich","Alexandrovich", "Mikhailovich", "Petrovich","Dmitryevich" };

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

{

Human NewPerson;

NewPerson.FIO = surnames[rand() % surnames.size()] + " " + names[rand() % names.size()] + " " + patronymic[rand() % patronymic.size()];

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

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

NewPerson.DateOfBirth.Year = rand() % 100 + 1921;

NewPerson.passportNumber = rand() % 6000 + 1;

result.push\_back(NewPerson);

}

return result;

}

int Substring\_search(vector<Human> humans, Date key)

{

string listOfDates;

string substring;

////////////////////////////////////////

if (key.Day < 10) substring += "0";

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

///////////////////////////////////////

if (key.Month < 10) substring += "0";

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

///////////////////////////////////////

if (key.Year < 10) substring += "0";

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

///////////////////////////////////////

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

{

string newLine = "";

if (humans[i].DateOfBirth.Day < 10) newLine += "0";

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

if (humans[i].DateOfBirth.Month < 10) newLine += "0";

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

if (humans[i].DateOfBirth.Year < 10) newLine += "0";

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

listOfDates += newLine;

}

for (int i = 0; i < listOfDates.size() - substring.size(); i++)

{

int j = 0;

while (listOfDates[i + j] == substring[j] && j < substring.size())

{

j++;

}

if (j == substring.size())

{

return i / 8;

}

}

return -1;

}

int Line\_search(vector<Human> humans, Date key)

{

vector<int> date\_list;

int tmp\_key = key.Day \* 1000000 + key.Month \* 10000 + key.Year;

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

{

date\_list.push\_back(humans[i].DateOfBirth.Day\*1000000 + humans[i].DateOfBirth.Month\*10000 + humans[i].DateOfBirth.Year);

}

for (int i = 0; i < date\_list.size(); i++)

{

if (date\_list[i] == tmp\_key)

{

return i;

}

}

return -1;

}

void Shell\_sort(vector<int>& humans)

{

int size = humans.size();

for (int step = size / 2; size > 0; size /= 2)

{

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

{

for (int j = i; j >= step; j -= step)

{

if (humans[j] < humans[j - step])

{

int tmp = humans[j];

humans[j] = humans[j - step];

humans[j - step] = tmp;

}

}

}

}

}

bool Interpol\_search(vector<Human>& humans, Date key)

{

bool check = false;

vector<int> date\_list;

int tmp\_key = key.Day \* 1000000 + key.Month \* 10000 + key.Year;

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

{

date\_list.push\_back(humans[i].DateOfBirth.Day \* 1000000 + humans[i].DateOfBirth.Month \* 10000 + humans[i].DateOfBirth.Year);

}

Shell\_sort(date\_list); // сортировка по возрастанию

int mid = 0, left = 0, right = date\_list.size() - 1;

while (date\_list[left] <= tmp\_key && date\_list[right] >= tmp\_key)

{

mid = left + ((tmp\_key - date\_list[left]) \* (right - left)) / (date\_list[right] - date\_list[left]);

if (date\_list[mid] < tmp\_key)

{

left = ++mid;

}

else if (date\_list[mid] > tmp\_key)

{

right = --mid;

}

else

{

check = true;

return check;

}

}

return check;

}

bool DeleteHuman(vector <Human>& list, string value)

{

bool check = false;

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

{

if (list[i].FIO == value)

{

list.erase(list.begin() + i);

check = true;

}

}

if (check)

{

cout << "Человек успешно удален из списка!" << endl;

}

else

{

cout << "Такого человека в списке нет!" << endl;

}

return check;

}

string Reverse\_str(string str)

{

string result;

for (int i = str.size() - 1; i >= 0; i--)

{

result.push\_back(str[i]);

}

return result;

}

int Boyer\_Moor(vector<Human> humans, Date key)

{

string listOfDates;

string substring;

////////////////////////////////////////

if (key.Day < 10) substring += "0";

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

///////////////////////////////////////

if (key.Month < 10) substring += "0";

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

///////////////////////////////////////

if (key.Year < 10) substring += "0";

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

///////////////////////////////////////

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

{

string newLine = "";

if (humans[i].DateOfBirth.Day < 10) newLine += "0";

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

if (humans[i].DateOfBirth.Month < 10) newLine += "0";

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

if (humans[i].DateOfBirth.Year < 10) newLine += "0";

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

listOfDates += newLine;

}

vector<int> shiftarray(substring.size()); //массив сдвигов

string substring\_reverse\_copy = Reverse\_str(substring); //перевернутая строка

int count = 0;

for (int i = 0; i < substring.size(); i++) //получение массива сдвигов

{

shiftarray[i] = substring\_reverse\_copy.find(substring[i]) + 1;

if (substring[i] == substring[substring.size() - 1]) count++;

if (i == substring.size() - 1)

{

if (count == 1)

{

shiftarray[substring.size() - 1] = substring.size();

}

else

{

int pos = substring.size() - 1;

shiftarray[pos] = substring\_reverse\_copy.find(substring[pos]) + 1;

}

}

}

cout << "Массив сдвигов для строки " << substring << " :" << endl;

for (int i = 0; i < shiftarray.size(); i++) //проверка массива сдвигов

{

cout << shiftarray[i] << " ";

}

cout << endl << "==============================================================" << endl;

int current\_char = substring.size() - 1;

while (current\_char < listOfDates.size())

{

int i = substring.size() - 1;

int j = current\_char;

if (listOfDates[current\_char] == substring[i])

{

// int j = current\_char;

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

{

i--;

j--;

}

}

if (i == 0)

{

return current\_char / 8 + 1;

}

else

{

if (substring.find(listOfDates[j]) == -1)

{

current\_char += substring.size();

}

else

{

current\_char += shiftarray[i];

}

}

}

return -1;

}

vector<int> Prefix\_func(string str)

{

//Важно: максимальный префикс в подстроке может увеличиваться только на 1, но изменяться на неограниченное количество (до 0)

vector<int> result(str.size());

result[0] = 0;

int j = 1, i = 0; //i - указатель на начало подстроки j - указатель j-ый элемент подстроки

while (j < str.size())

{

if (str[i] != str[j])

{

if (i == 0)

{

result[j] = 0;

j++;

}

else

{

i = 0; //i указывает на 0 элемент подстроки

}

}

else

{

result[j] = i + 1;

i++; //сдвиг i-го указателя, проверка со следующим j-ым элементом, в случае неравенства обнуляется

j++; //обычный сдвиг j-го указателя

}

}

return result;

}

int KMP(vector<Human> humans, Date key)

{

string listOfDates; //строка

string substring; //подстрока

////////////////////////////////////////

if (key.Day < 10) substring += "0";

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

///////////////////////////////////////

if (key.Month < 10) substring += "0";

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

///////////////////////////////////////

if (key.Year < 10) substring += "0";

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

///////////////////////////////////////

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

{

string newLine = "";

if (humans[i].DateOfBirth.Day < 10) newLine += "0";

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

if (humans[i].DateOfBirth.Month < 10) newLine += "0";

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

if (humans[i].DateOfBirth.Year < 10) newLine += "0";

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

listOfDates += newLine;

}

vector <int> pi = Prefix\_func(substring); //получение pi массива (префиксов)

int k = 0, l = 0; //l - указатель на подстроку, k - на строку

while (k < listOfDates.size())

{

if (substring[l] == listOfDates[k]) //обычная проверка k-го элемента строки и l-го элемента подстроки

{

k++;

l++;

if (l == substring.size()) return k / 8;

}

else

{

if (l == 0) // сдвиг k в случае неравенства элемента подстроки и строки (l - указывает на начало подстроки)

{

k++;

}

else l = pi[l - 1];

}

}

return -1;

}

void Save\_list(vector<Human> list)

{

ofstream output("Saved.txt");

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

{

output << "===========================================================" << endl;

output << i + 1 << "-й пользователь" << endl;

output << "ФИО:\t" << list[i].FIO << endl;

string pasport = to\_string(list[i].passportNumber);

output << "Номер паспорта:\t" << pasport << endl;

string date = to\_string(list[i].DateOfBirth.Day) + "." + to\_string(list[i].DateOfBirth.Month) + "." + to\_string(list[i].DateOfBirth.Year);

output << "Дата рождения:\t" << date << endl;

}

output << "===========================================================" << endl;

output.close();

}

int main()

{

setlocale(LC\_ALL, "ru");

vector <Human> Humans;

vector <Human> reserveHumans = Humans;

bool check\_delete = false;

int menu = -1;

while (menu == -1)

{

menu = GetMenu();

switch (menu)

{

case 1: //заполнение списка

{

cout << "===============================================================" << endl;

cout << "Введите размер списка:\t";

int size = GetSize();

Humans = CreateArray(size);

cout << "Список успешно создан!" << endl;

cout << "===============================================================" << endl;

menu = -1;

break;

}

case 2://вывод списка

{

if (Humans.size() != 0)

{

cout << "Список с людьми:" << endl;

Print\_list(Humans);

cout << "===============================================================" << endl << endl;

}

else

{

cout << "Список пуст!" << endl;

cout << "===============================================================" << endl << endl;

}

menu = -1;

break;

}

case 3: // поиск элемента в списке по ключу

{

if (Humans.size() == 0)

{

cout << "Список пуст!" << endl;

cout << "===============================================================" << endl << endl;

menu = -1;

break;

}

cout << "Выберете способ поиска!" << endl;

cout << "1. Подстроки в строке." << endl;

cout << "2. Линейный." << endl;

cout << "3. Интерполяционный." << endl;

cout << "4. Бойера-Мура." << endl;

cout << "5. K-M-P method." << endl;

cout << "0. Вернуться в меню." << endl;

int search;

cout << "Поле ввода:\t";

while (!(cin >> search) || (cin.peek() != '\n') || (search < 0) || (search > 5))

{

cin.clear();

while (cin.get() != '\n')

{

cout << "Введите корректное число:\t";

}

}

switch (search)

{

case 1:

{

Date key;

key.Get\_date();

int pos = Substring\_search(Humans, key);

if (pos != -1)

{

cout << "Элемент с такой датой рождения найден. Он находится в списке под номером: " << pos + 1 << endl;

cout << "===============================================================" << endl << endl;

}

else

{

cout << "Элемента с такой датой рождения нет!" << endl;

cout << "===============================================================" << endl << endl;

}

break;

}

case 2:

{

Date key;

key.Get\_date();

int pos = Line\_search(Humans, key);

if (pos != -1)

{

cout << "Элемент с такой датой рождения найден. Он находится в списке под номером: " << pos + 1 << endl;

cout << "===============================================================" << endl << endl;

}

else

{

cout << "Элемента с такой датой рождения нет!" << endl;

cout << "===============================================================" << endl << endl;

}

break;

}

case 3:

{

Date key;

key.Get\_date();

if (Interpol\_search(Humans, key) == true)

{

cout << "Элемент с такой датой рождения найден." << endl;

cout << "===============================================================" << endl << endl;

}

else

{

cout << "Элемента с такой датой рождения нет!" << endl;

cout << "===============================================================" << endl << endl;

}

break;

}

case 4:

{

Date key;

key.Get\_date();

int pos = Boyer\_Moor(Humans, key);

if (pos == -1)

{

cout << "Такого элемента нет!" << endl;

cout << "===============================================================" << endl << endl;

}

else

{

cout << "Такой элемент находится под индексом " << pos << endl;

cout << "===============================================================" << endl << endl;

}

break;

}

case 5:

{

Date key;

key.Get\_date();

int pos = KMP(Humans, key);

if (pos == -1)

{

cout << "Такого элемента нет!" << endl;

cout << "===============================================================" << endl << endl;

}

else

{

cout << "Такой элемент находится под индексом " << pos << endl;

cout << "===============================================================" << endl << endl;

}

break;

}

case 0:

{

cout << "Выход..." << endl;

break;

}

}

menu = -1;

break;

}

case 4: //удаление элемента из списка

{

reserveHumans = Humans;

if (reserveHumans.size() == 0)

{

cout << "Список пуст!" << endl;

}

else

{

cout << "Введите ФИO человека, которого вы хотите удалить:\t";

string value;

cin.ignore();

getline(cin, value);

check\_delete = DeleteHuman(Humans, value);

Print\_list(Humans);

cout << "===============================================================" << endl << endl;

}

menu = -1;

break;

}

case 5: //отмена удаления

{

if (check\_delete)

{

cout << "Удаление элементов отменено!" << endl;

cout << "===============================================================" << endl << endl;

Humans = reserveHumans;

check\_delete = false;

}

else

{

cout << "Элементы из списка не удалялись!" << endl;

cout << "===============================================================" << endl << endl;

}

menu = -1;

break;

}

case 6: //запись в файл

{

if (Humans.size() == 0)

{

cout << "Список пуст, сохранять нечего!" << endl;

cout << "===============================================================" << endl << endl;

menu = -1;

break;

}

ofstream output("Saved.txt");

if (output.is\_open())

{

Save\_list(Humans);

cout << "Запись в файл прошла успешно!" << endl;

cout << "===============================================================" << endl << endl;

}

else

{

cout << "Не удалось открыть файл для сохранения, проверьте его на существование!" << endl;

cout << "===============================================================" << endl << endl;

}

output.close();

menu = -1;

break;

}

case 7: //добавление в список

{

int add;

cout << endl;

cout << "Введите каким способом вы хотите добавить элемент в список:" << endl;

cout << "1. Добавить в начало списка." << endl;

cout << "2. Добавить в конец списка" << endl;

cout << "3. Вставить по индексу." << endl;

cout << "0. Выйти в меню." << endl;

cout << "Поле ввода:\t";

while (!(cin >> add) || (cin.peek() != '\n') || (add < 0) || (add > 3))

{

cin.clear();

while (cin.get() != '\n')

{

cout << "Введите корректное число:\t";

}

}

switch (add)

{

case 1:

{

Human tmp;

tmp.Fill();

Humans.insert(Humans.begin(), tmp);

cout << "===============================================================" << endl << endl;

break;

}

case 2:

{

Human tmp;

tmp.Fill();

Humans.push\_back(tmp);

cout << "===============================================================" << endl << endl;

break;

}

case 3:

{

int i;

cout << "Введите позицию, куда хотите добавить элемент:\t";

while (!(cin >> i) || (cin.peek() != '\n') || (i < 0) || (i > Humans.size()))

{

cin.clear();

while (cin.get() != '\n')

{

cout << "Введите корректное число:\t";

}

}

Human tmp;

tmp.Fill();

Humans.insert(Humans.begin() + i, tmp);

cout << "===============================================================" << endl << endl;

break;

}

case 0:

{

cout << "Выход..." << endl;

break;

}

}

menu = -1;

break;

}

case 0: //выход

{

cout << "Работа завершена!" << endl;

cout << "===============================================================" << endl;

return 0;

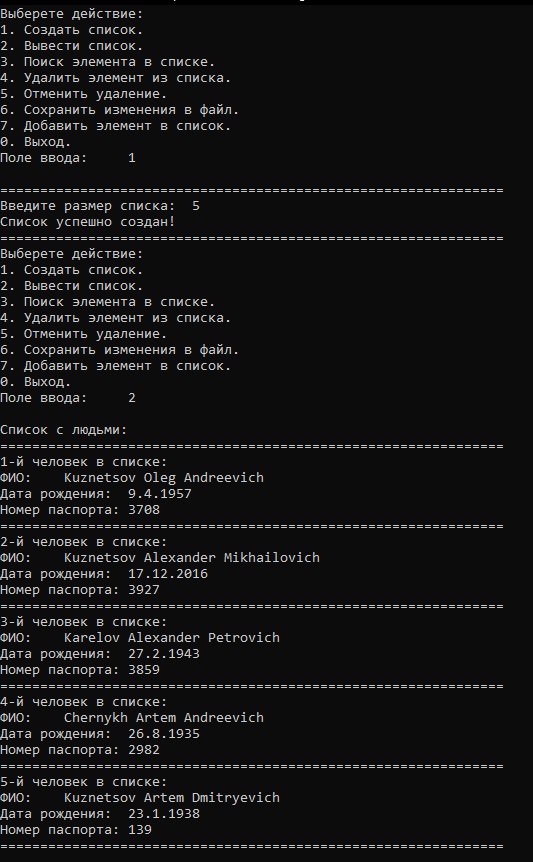
}

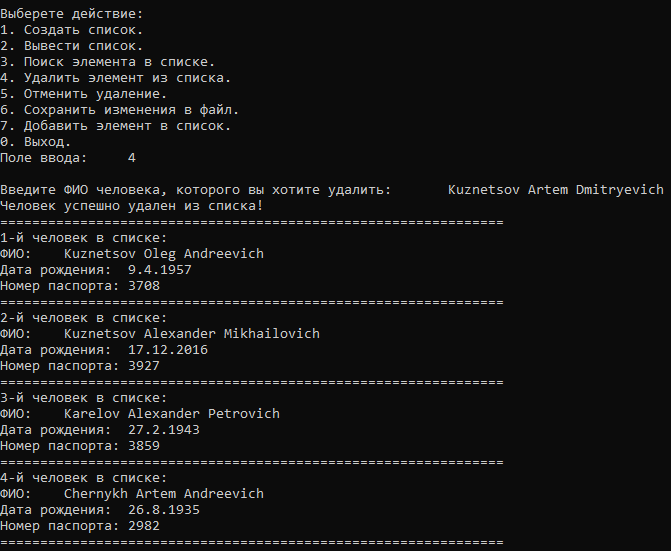
}

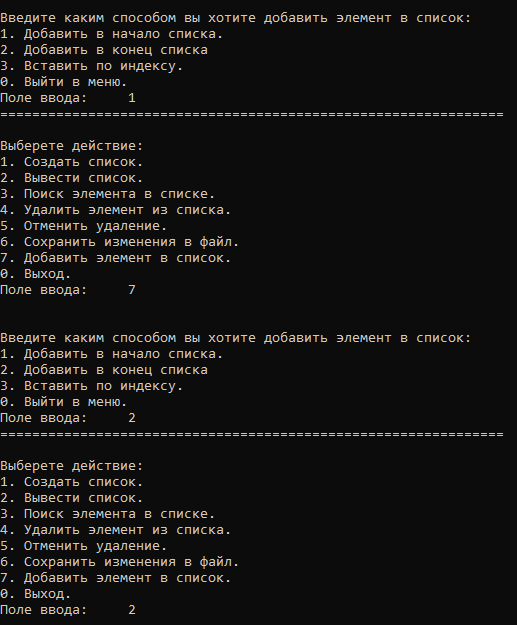
}

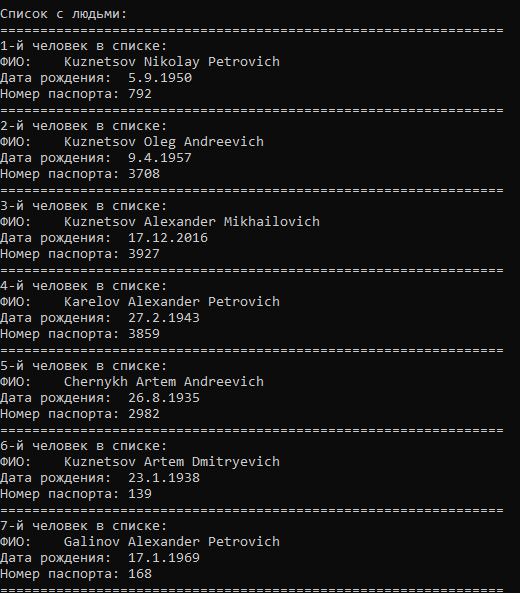
}

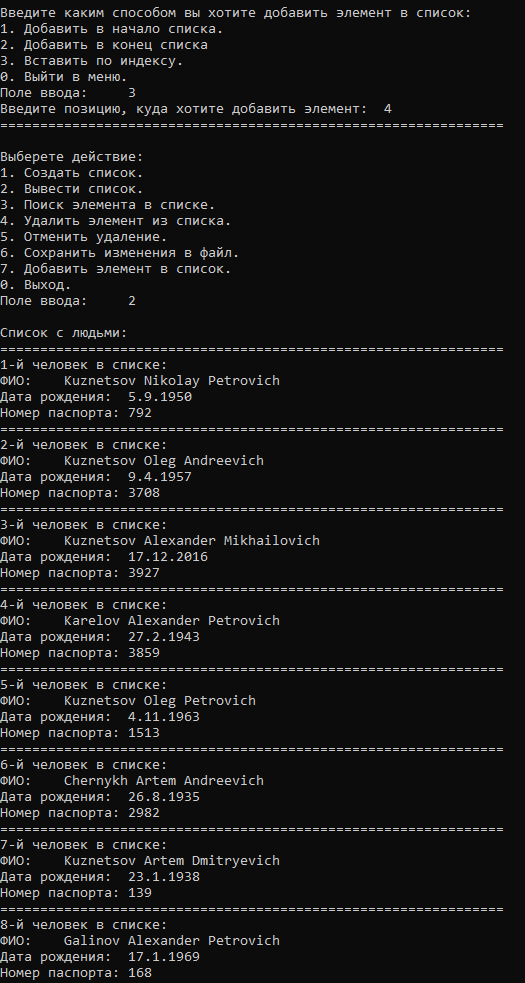
**Работа кода**

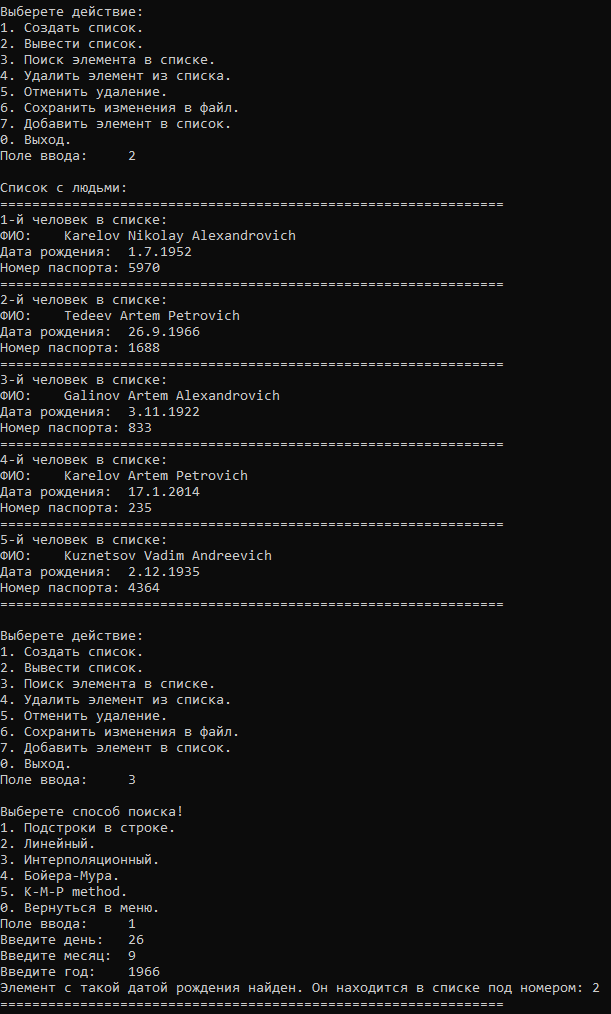


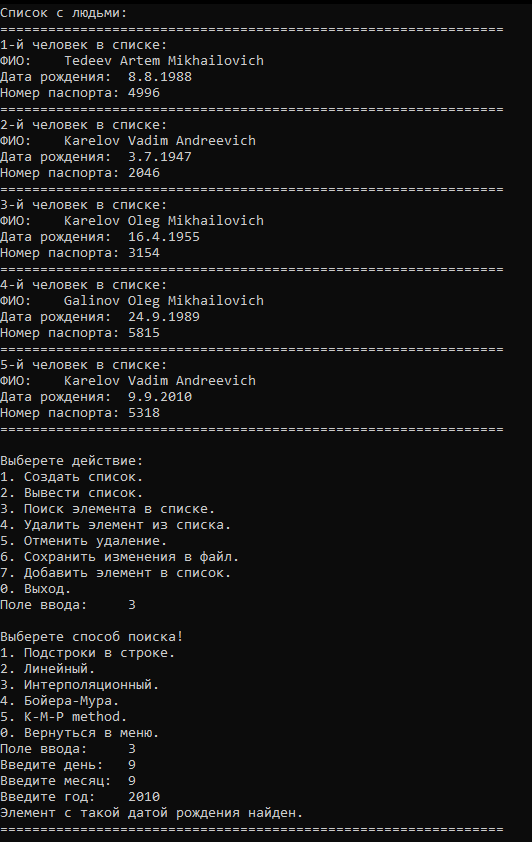
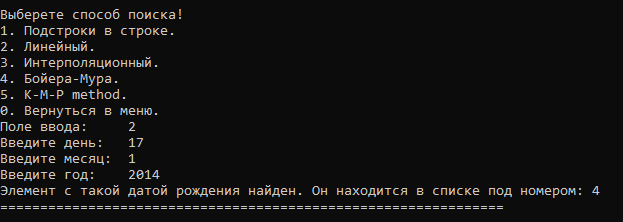


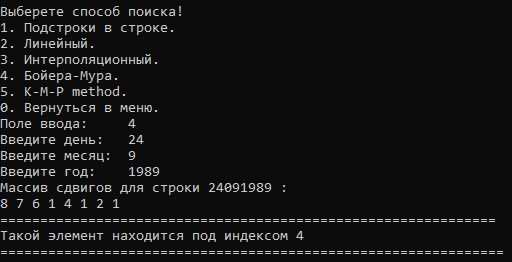


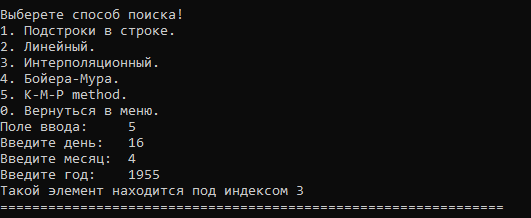


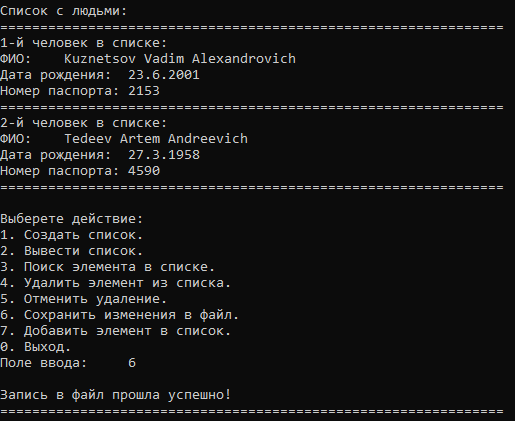


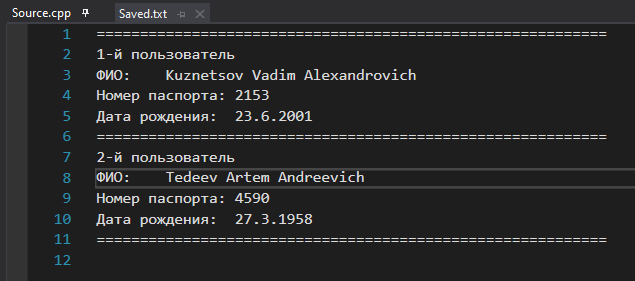












**Блок-схема**

