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 лабораторной работе №12**

Дисциплина: Основы алгоритмизации и программирования

Тема: “ Методы поиска: линейный, интерполяционный, прямой поиск подстроки в строке”

Вариант 15

Выполнил:

Студент гр. ИВТ-20-2б

Чувашев Максим Алексеевич

Проверил:

Доцент кафедры ИТАС  
Полякова О.А

Пермь, 2021

**Цель работы**

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.

**Постановка задачи**

1. Разработать структуру ФИО.
2. Организовать ввод-вывод данных.
3. Реализовать функцию удаления элементов из списка.
4. Реализовать меню.
5. Реализовать функцию отмены последнего удаления элемента.
6. Разработать функцию, заполняющую массив объектов структуры случайными числами.
7. Реализовать функции поиска.
8. Реализовать функцию, сохраняющую измененный массив в файл.
9. Разработать программу.

**Анализ задачи**

1. Для решения задачи необходимо:
   1. Организовать функцию SearchLine(human, size, str, help), которая отвечает за линейный поиск.
   2. Организовать функцию Metod2(human, size, str, help), которая отвечает за поиск подстроки в строке.
   3. Организовать функцию Interp(DATA\* human, int size, list<DATA> help), которая отвечает за интерполяционный поиск.
   4. Организовать функции void humanINIT(DATA\* human, int size), функция которая инициализирет наш список с данными
   5. Организовать функцию печати массива void PrintH(DATA\* human, int size).
   6. Организовать функцию добавления элемента в массив void add(DATA\* human, int \*size)
   7. Организовать функцию сохранения в массива в файл int saveH(DATA\* human, int size, string path\_F)
   8. Организовать функцию для ввода по какому параметру будет выполняться линейный поиск void EnterS(DATA\* human, int size, list<DATA> help)
   9. Организовать функцию для ввода по какому параметру будет выполняться линейный поиск void Kostyl(DATA\* human, int size, list<DATA> help)
   10. Организовать функцию сортировки по возрастанию void SORT(DATA\* human, int size, list<DATA> help, DATAHELP\* H)
   11. Организовать функцию изменения массива void Edit(DATA\* human, int size, list<DATA> help, DATAHELP\* H)
   12. Организовать функцию отвечающую за меню bool Choice(DATA\* human, int \*size, int foo, bool f, string path\_F, list<DATA> help)
2. В ходе работы были использованы следующие типы данных:
   1. Структура DATA, которая хранит ФИО дату рождения и индекс под которым в массиве находится объект

struct DATA

{

string Name; // поле хранящее имя

string Surname; // поле хранящее фамилию

string Middle; // поле хранящее отчество

int day; // поле хранящее день

int month; // поле хранящее месяц

int year; // поле хранящее год

int ind; // поле хранящее индекс человека

int date; // поле хранящее поле хранящее дату

};

* 1. Вспомогательная структура, которая будет хранить дату в виде одного числа и индекс элемента

struct DATAHELP // вспомогательная структура

{

int date; // поле хранящее дату

int ind; // поле хранящее индект

};

* 1. В функции saveH(DATA\* human, int size, string path\_F) используются команды из библиотеки fstream для сохранения списка в файл.

int saveH(DATA\* human, int size, string path\_F) // сохранение в документ

{

ofstream fout;

fout.open(path\_F); // открываем файл

if (!fout.is\_open()) // проверка на то, открыт ли файл

{

cout << "Ошибка открытия файла!" << endl;

return 0;

}

else

{

string str;

cout << "Файл " << path\_F << " был успешно открыт!\n";

for (int i = 0; i < size; i++) { // в цикле проходимся по всему списку и заносим все поля всех элементов в документ

fout << human[i].Name << '\n';

fout << human[i].Surname << '\n';

fout << human[i].Middle << '\n';

fout << human[i].day << "." << human[i].month << "." << human[i].year << '\n' << '\n';

}

}

fout.close(); // закрываем файл

return 0;

}

* 1. В функции bool Choice(DATA\* human, int \*size, int foo, bool f, string path\_F, list<DATA> help) используется switch, с помощью которого происходит выбор действий.

bool Choice(DATA\* human, int \*size, int foo, bool f, string path\_F, list<DATA> help) // меню

{

switch (foo)

{

case 1: humanINIT(human, \*size); break;

case 2: PrintH(human, \*size); break;

case 3: saveH(human, \*size, path\_F); break;

case 4: EnterS(human, \*size, help); break;

case 5: Kostyl(human, \*size, help); break;

case 6: Interp(human, \*size, help); break;

case 7: add(human, size); break;

case 8: f = true; break;

default:

break;

}

return f;

}

1. Для решения задачи данные были представлены в следующем виде:
   1. Для инициализации данных массива структур human используется фукнция void humanINIT(DATA\* human, int size)

void humanINIT(DATA\* human, int size) // функция заполнения массива людей

{

string name[] = { "Александр", "Алексей", "Анатолий", "Андрей", "Антон", "Аркадий", "Арсений", "Артём", "Артур", "Борис", "Вадим", "Валентин", "Валерий", "Василий", "Виктор", "Виталий", "Владимир", "Владислав", "Вячеслав", "Георгий", "Глеб", "Григорий", "Даниил", "Денис", "Дмитрий", "Евгений", "Егор", "Иван", "Игорь", "Илья", "Кирилл", "Константин", "Лев", "Леонид", "Максим", "Марк", "Матвей", "Михаил", "Никита", "Николай", "Олег", "Павел", "Пётр", "Роман", "Руслан", "Сергей", "Степан", "Тимур", "Фёдор", "Юрий", "Ярослав" };

string surname[] = { "Пантелеев", "Чувашев", "Галинов", "Ананина", "Масылюк", "Вострокнутова", "Сабуров", "Исламов", "Шамай", "Филатов", "Сафронов", "Шарпов", "Карелов", "Брейкин", "Копытов", "Солдатов", "Кузнецов", "Пажгин", "Фотин", "Бадретдинов", "Механошин", "Булдаков", "Тулинов", "Тедеев", "Колпаков", "Черных", "Нефедов", "Рябцев", "Пепеляев", "Тарасов", "Аркадьев" };

string middle[] = { "Романович", "Иосифович", "Аникитевич", "Прохорович", "Архипович", "Александрович", "Мартьянович", "Богданович", "Никифорович", "Георгиевич" , "Прокофиевич" , "Валериевич" , "Владиславович" , "Данилевич" , "Ульянович" , "Кондратиевич" , "Валериевич" , "Сократович" , "Кондратович" , "Евграфович" , "Алексеевич" , "Кондратович" , "Тихонович" , "Игнатиевич" , "Алексеевич" , "Венедиктович" , "Мартьянович" , "Родионович" , "Тимурович" , "Самуилович" , "Климентович" , "Натанович" , "Леонович" , "Игоревич" , "Захарович" , "Андронович" , "Платонович" , "Несторович" , "Модестович" , "Ерофеевич" , "Адамович" };

for (int i = 0; i < size; i++) // цикл от 0 до size

{

human[i].ind = i; // заполнение поля индекса

human[i].year = 1903 + rand() % 117; // случайное заполнение поля год

human[i].month = 1 + rand() % 12; // случайное заполнение поля месяц

int a = human[i].month; // создание временной переменной хранящей месяц

switch (a) // проверка на то сколько дней в месяце

{

case 1: human[i].day = 1 + (rand() % 31); break; // если 1 месяц и так далее

case 2: human[i].day = 1 + (rand() % 28); break;

case 3: human[i].day = 1 + (rand() % 31); break;

case 4: human[i].day = 1 + (rand() % 30); break;

case 5: human[i].day = 1 + (rand() % 31); break;

case 6: human[i].day = 1 + (rand() % 30); break;

case 7: human[i].day = 1 + (rand() % 31); break;

case 8: human[i].day = 1 + (rand() % 31); break;

case 9: human[i].day = 1 + (rand() % 30); break;

case 10: human[i].day = 1 + (rand() % 31); break;

case 11: human[i].day = 1 + (rand() % 30); break;

case 12: human[i].day = 1 + (rand() % 31); break;

default:

break;

}

int g = rand() % (sizeof(name) / sizeof(name[0])); // создание переменной которая хранит номер имени

human[i].Name = name[g]; // поле имени заполняется в соответствии с случайным числом

g = rand() % (sizeof(surname) / sizeof(surname[0])); // снова генерируется случайное число которое отвечает за выбор фамилии

human[i].Surname = surname[g]; // заполняется поле фамилии

g = rand() % (sizeof(middle) / sizeof(middle[0])); // генерируется случайное число которое отвечает за выбор отчества

human[i].Middle = middle[g]; // заполнение отчества

human[i].date = human[i].day + human[i].month \* 100 + human[i].year \* 10000; // случайное заполнение поля даты

}

}

* 1. Для хранения данных используется динамический массив размера 100 типа DATA

DATA\* human = new DATA[size];

* 1. Так же используется вспомогательный список для изменения массива.

list<DATA> help;

1. Ввод и вывод организован с помощью:
   1. Операторов cin и cout

cout << "8: Выйти из программы" << endl;

cin >> foo;

* 1. С помощью функции getline():

cin.ignore();

getline(cin, str); // пользователь вводит желаемую строку

1. Поставленные задачи будут решены следующими действиями:
   1. Функцией main(), которая воздает массив, создает вспомогательный список и вызывает функцию вывода Choice()

int main()

{

system("chcp 1251>nul");

srand(time(NULL));

int size;

size = 100;

DATA\* human = new DATA[size];

list<DATA> help;

bool f = false;

int foo = -1;

string path\_F = "DATA.txt";

while (f == false)

{

cout << endl;

cout << "Выберете действие:" << endl;

cout << "1: Инициализировать массив" << endl;

cout << "2: Напечатать массив в консоль" << endl;

cout << "3: Сохранить изменения" << endl;

cout << "4: Линейный поиск" << endl;

cout << "5: Поиск подстроки в строке" << endl;

cout << "6: Интерполяционный поиск" << endl;

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

cout << "8: Выйти из программы" << endl;

cin >> foo;

f = Choice(human, &size, foo, f, path\_F, help);

}

}

* 1. Функцией Choice() в которой пользователь выбирает, что он хочет сделать с массивом

bool Choice(DATA\* human, int \*size, int foo, bool f, string path\_F, list<DATA> help) // меню

{

switch (foo)

{

case 1: humanINIT(human, \*size); break;

case 2: PrintH(human, \*size); break;

case 3: saveH(human, \*size, path\_F); break;

case 4: EnterS(human, \*size, help); break;

case 5: Kostyl(human, \*size, help); break;

case 6: Interp(human, \*size, help); break;

case 7: add(human, size); break;

case 8: f = true; break;

default:

break;

}

return f;

}

* 1. Функция void humanINIT(DATA\* human, int size), которая инициализирует заданный массив

void humanINIT(DATA\* human, int size) // функция заполнения массива людей

{

string name[] = { "Александр", "Алексей", "Анатолий", "Андрей", "Антон", "Аркадий", "Арсений", "Артём", "Артур", "Борис", "Вадим", "Валентин", "Валерий", "Василий", "Виктор", "Виталий", "Владимир", "Владислав", "Вячеслав", "Георгий", "Глеб", "Григорий", "Даниил", "Денис", "Дмитрий", "Евгений", "Егор", "Иван", "Игорь", "Илья", "Кирилл", "Константин", "Лев", "Леонид", "Максим", "Марк", "Матвей", "Михаил", "Никита", "Николай", "Олег", "Павел", "Пётр", "Роман", "Руслан", "Сергей", "Степан", "Тимур", "Фёдор", "Юрий", "Ярослав" };

string surname[] = { "Пантелеев", "Чувашев", "Галинов", "Ананина", "Масылюк", "Вострокнутова", "Сабуров", "Исламов", "Шамай", "Филатов", "Сафронов", "Шарпов", "Карелов", "Брейкин", "Копытов", "Солдатов", "Кузнецов", "Пажгин", "Фотин", "Бадретдинов", "Механошин", "Булдаков", "Тулинов", "Тедеев", "Колпаков", "Черных", "Нефедов", "Рябцев", "Пепеляев", "Тарасов", "Аркадьев" };

string middle[] = { "Романович", "Иосифович", "Аникитевич", "Прохорович", "Архипович", "Александрович", "Мартьянович", "Богданович", "Никифорович", "Георгиевич" , "Прокофиевич" , "Валериевич" , "Владиславович" , "Данилевич" , "Ульянович" , "Кондратиевич" , "Валериевич" , "Сократович" , "Кондратович" , "Евграфович" , "Алексеевич" , "Кондратович" , "Тихонович" , "Игнатиевич" , "Алексеевич" , "Венедиктович" , "Мартьянович" , "Родионович" , "Тимурович" , "Самуилович" , "Климентович" , "Натанович" , "Леонович" , "Игоревич" , "Захарович" , "Андронович" , "Платонович" , "Несторович" , "Модестович" , "Ерофеевич" , "Адамович" };

for (int i = 0; i < size; i++) // цикл от 0 до size

{

human[i].ind = i; // заполнение поля индекса

human[i].year = 1903 + rand() % 117; // случайное заполнение поля год

human[i].month = 1 + rand() % 12; // случайное заполнение поля месяц

int a = human[i].month; // создание временной переменной хранящей месяц

switch (a) // проверка на то сколько дней в месяце

{

case 1: human[i].day = 1 + (rand() % 31); break; // если 1 месяц и так далее

case 2: human[i].day = 1 + (rand() % 28); break;

case 3: human[i].day = 1 + (rand() % 31); break;

case 4: human[i].day = 1 + (rand() % 30); break;

case 5: human[i].day = 1 + (rand() % 31); break;

case 6: human[i].day = 1 + (rand() % 30); break;

case 7: human[i].day = 1 + (rand() % 31); break;

case 8: human[i].day = 1 + (rand() % 31); break;

case 9: human[i].day = 1 + (rand() % 30); break;

case 10: human[i].day = 1 + (rand() % 31); break;

case 11: human[i].day = 1 + (rand() % 30); break;

case 12: human[i].day = 1 + (rand() % 31); break;

default:

break;

}

int g = rand() % (sizeof(name) / sizeof(name[0])); // создание переменной которая хранит номер имени

human[i].Name = name[g]; // поле имени заполняется в соответствии с случайным числом

g = rand() % (sizeof(surname) / sizeof(surname[0])); // снова генерируется случайное число которое отвечает за выбор фамилии

human[i].Surname = surname[g]; // заполняется поле фамилии

g = rand() % (sizeof(middle) / sizeof(middle[0])); // генерируется случайное число которое отвечает за выбор отчества

human[i].Middle = middle[g]; // заполнение отчества

human[i].date = human[i].day + human[i].month \* 100 + human[i].year \* 10000; // случайное заполнение поля даты

}

}

* 1. Функция void add(DATA\* human, int \*size), которая добавляет элемент в массив, подробная работа описана в комментариях к коду.

void add(DATA\* human, int \*size) // функция добавления элемента в список

{

int b;

cout << "Сколько элементов вы хотите добавить?" << endl;

cin >> b;

while (b < 1) // проверка числа на отрицательность

{

cout << "Вы можете добавить только натуральное количество элементов!" << endl;

cout << "Сколько элементов вы хотите добавить?" << endl;

cin >> b;

}

int Size;

Size = (\*size);

\*size = Size + b;

DATA\* HELP = new DATA[(\*size)]; // выделение памяти под новый элемент

int ch = 0;

bool f = false;

while (f == false)

{

switch (ch) // выбор пользователя куда добавить элемент

{

case 1: // в начало

humanINIT(HELP, b);

for (int i = b, j = 0; i < (\*size); i++, j++)

{

HELP[i] = human[j];

}

f = true;

break;

case 2: // в конец

for (int k = 0; k < Size; k++)

{

HELP[k] = human[k];

}

humanINIT(HELP, (\*size));

f = true;

break;

default:

cout << "Куда вы хотите добавить элемент?" << endl;

cout << "1: В начало списка" << endl;

cout << "2: В конец списка" << endl;

cin >> ch;

break;

}

}

PrintH(HELP, \*size); // вызов функции вывода списка

}

* 1. Функция сохранения массива в файл int saveH(DATA\* human, int size, string path\_F).

int saveH(DATA\* human, int size, string path\_F) // сохранение в документ

{

ofstream fout;

fout.open(path\_F); // открываем файл

if (!fout.is\_open()) // проверка на то, открыт ли файл

{

cout << "Ошибка открытия файла!" << endl;

return 0;

}

else

{

string str;

cout << "Файл " << path\_F << " был успешно открыт!\n";

for (int i = 0; i < size; i++) { // в цикле проходимся по всему списку и заносим все поля всех элементов в документ

fout << human[i].Name << '\n';

fout << human[i].Surname << '\n';

fout << human[i].Middle << '\n';

fout << human[i].day << "." << human[i].month << "." << human[i].year << '\n' << '\n';

}

}

fout.close(); // закрываем файл

return 0;

}

* 1. Функция линейного поиска по ФИО, функция складывает все 3 поля объекта структуры и сравнивает введенное ФИО с получившимся переменной

void SearchLine(DATA\* human, int size, string str, list<DATA> help) // функция линнейного поиска по ФИО

{

cout << endl << "Введите желаемое ФИО: ";

cin.ignore();

getline(cin, str); // пользователь вводит желаемую строку

for (int i = 0; i < size; i++) // проходимся по всем элементам списка

{

string strH = human[i].Surname + " " + human[i].Name + " " + human[i].Middle; // создаем переменную в которой хранится ФИО элемента

if (strH == str) // если введенная строка совпала, то

{

help.push\_back(human[i]); // добавляем в конец элемент

}

}

int S = help.size(); // создаем переменную, которая будет хранить размер нашего нового списка

ptr = new DATA[S]; // выделяем динамическую память размера DATA\*S

for (int i = 0; i < S; i++) // проходимся по всем элементам списка

{

ptr[i] = help.front(); // переносим элемент из изначального списка в новый

help.pop\_front(); // достаем элемент из начала старого списка

}

PrintH(ptr, S); // выводим новый список в консоль

}

* 1. Функция поиска подстроки в строке void Metod2(DATA\* human, int size, string str, list<DATA> help), в данной проходимся посимвольно по строке если все символы совпадут, то значит найдено вхождение.

void Metod2(DATA\* human, int size, string str, list<DATA> help) // фукнция для поиска подстроки в строке

{

string strH;

cout << endl << "Введите ФИО: ";

cin.ignore();

getline(cin, strH);

bool f = false;

int j = 0; // счетчик

int l = 0; // счетчик

int p = 0; // счетчик

int count = 0; // счетчик

int b = strH.length(); // переменная хранящая размер строки

for (int i = 0; i < size; i++) // цикл проходящийся по всем элементам

{

string result = human[i].Surname + " " + human[i].Name + " " + human[i].Middle; // переменная ФИО

int a = result.length(); // длинна ФИО

bool f = 0; // флаг

while (l < strH.length() && j < (a - b) && f != true) // пока не достигнем конца строки и пока не достигнется разница длин и пока флаг лож

{

p = j; // запоминаем ноомер текущего элемента

while (strH[l] == result[j] && f != true) // сравниваем две строки

{

count++; // увеличиваем счетчик

j++; // сдвигаемся по строке

l++; // сдвигаемся по строке

if (count == strH.length()) // если достигли конца строки

{

f = true; // флаг правда

help.push\_back(human[i]); // добавляем в конец списка

}

}

j = p; // возвращаемся к элемнету

count = 0; // сбрасываем счетчик

l = 0; // сбрасываем счетчик

j++; // инкрементируем счетчик

}

j = 0; // сбрасываем счетчик

l = 0; // сбрасываем счетчик

}

int SIZE; // переменная хранящая размер списка

SIZE = help.size(); // присваиваем размер списка

ptr = new DATA[SIZE]; // создаем массив размера SIZE

for (int i = 0; i < SIZE; i++) // проходимся по всему списку

{

ptr[i] = help.front(); // передаем элемент из списка в массив

help.pop\_front(); // достаем элемент спереди списка

}

PrintH(ptr, SIZE); // печатаем массив

}

* 1. Функция интерполяционного поиска. В выборе данного поиска находится середина уже отсортированного массива, с помощью формулы. Выполняется проверка на совпадение левой границы с искомым числом, если она не равна, то возвращается -1, если же совпало, то возвращаем левый элемент.

int InterSearch(DATA\* human, int size, int S) // функция интерполяционный поиск

{

int mid; // переменная хранящая стредний элемент

int left = 0; // левый элемент

int right = size - 1; // правый элемент

while (human[left].date <= S && human[right].date >= S) // пока левое значение меньше а правое польше нашего значения

{

mid = left + ((S - human[left].date) \* (right - left)) / (human[right].date - human[left].date); // ищем средний элемент по формуле

if (human[mid].date < S) // если средний элемент меньше заданного значения

{

left = mid + 1; // сдвигаем левую границу

}

else if (human[mid].date > S) // если средний элемент больше заданного значения

{

right = mid - 1; // сдвигаем правый элемент

}

else

{

return mid; // элемент найден

}

}

if (human[left].date == S) // если левая граница равна значению

{

return left; // элемент найден

}

else

{

return -1; // элемент не найден

}

}

* 1. Вспомогательная функция для интерполяционного поиска void Interp(DATA\* human, int size, list<DATA> help). В данной функции пользователем вводится дата, создается переменная, которая преобразует дату в цельное число. Создается вспомогательный массив типа данных вспомогательной структуры, заполняем поля для всех элементов. Сортируем массив по возрастанию. И вызывается функция интерполяционного поиска.

void Interp(DATA\* human, int size, list<DATA> help) // интерполяционный поиск

{

int n1; // переменная хранящая день

int n2; // переменная хранящая месяц

int n3; // переменная хранящая год

cout << "Введите Год: ";

cin >> n3;

while (n3 < 1903 || n3 > 2021) // проверяем корректность значения

{

cout << "Введите Год: ";

cin >> n3;

}

n2 = 0;

while (n2 < 1 || n2 > 12) // проверяем корректность значения

{

cout << "Введите Месяц: ";

cin >> n2;

}

n1 = 0;

bool f = false; // создаем флаг

while (f == false)

{

switch (n2)

{

case 1: case 3: case 5: case 7: case 8: case 10: case 12:

while (n1 < 1 || n1 > 31)

{

cout << "Введите День: ";

cin >> n1;

}

f = true;

break;

case 4: case 6: case 9: case 11:

while (n1 < 1 || n1 > 30)

{

cout << "Введите День: ";

cin >> n1;

}

f = true;

break;

case 2:

while (n1 < 1 || n1 > 28)

{

cout << "Введите День: ";

cin >> n1;

}

f = true;

break;

default:

cout << "Введите День: ";

cin >> n1;

break;

}

}

int S = n3 \* 10000 + n2 \* 100 + n1; // создаем переменную которая преобразует все данные в одно число

DATAHELP\* H = new DATAHELP[size]; // создаем вспомогательный массив

for (int i = 0; i < size; i++) // проходимся по всем элементам

{

H[i].date = human[i].day + human[i].month \* 100 + human[i].year \* 10000; // заполняем поле дата

H[i].ind = i; // заполняем поле индекс

}

SORT(human, size, help, H); // сортируем по возрастанию

PrintH(H, size); // выводим на экран

Edit(human, size, help, H); // функция изменения спииска

PrintH(human, size); // выводим на экран

int result = InterSearch(human, size, S); // запускаем интерполяционный поиск

if (result == -1) // если дату не нашел

{

cout << "Такой даты рождения нет!" << endl;

}

else

{

DATA\* RES = new DATA[1];

RES[0] = human[result];

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

PrintH(RES, 1);

}

}

**Полный код**

#include <iostream>

#include <string>

#include <list>

#include <ctime>

#include <fstream>

using namespace std;

struct DATA

{

string Name; // поле хранящее имя

string Surname; // поле хранящее фамилию

string Middle; // поле хранящее отчество

int day; // поле хранящее день

int month; // поле хранящее месяц

int year; // поле хранящее год

int ind; // поле хранящее индекс человека

int date; // поле хранящее поле хранящее дату

};

struct DATAHELP // вспомогательная структура

{

int date; // поле хранящее дату

int ind; // поле хранящее индект

};

DATA\* ptr;

void humanINIT(DATA\* human, int size) // функция заполнения массива людей

{

string name[] = { "Александр", "Алексей", "Анатолий", "Андрей", "Антон", "Аркадий", "Арсений", "Артём", "Артур", "Борис", "Вадим", "Валентин", "Валерий", "Василий", "Виктор", "Виталий", "Владимир", "Владислав", "Вячеслав", "Георгий", "Глеб", "Григорий", "Даниил", "Денис", "Дмитрий", "Евгений", "Егор", "Иван", "Игорь", "Илья", "Кирилл", "Константин", "Лев", "Леонид", "Максим", "Марк", "Матвей", "Михаил", "Никита", "Николай", "Олег", "Павел", "Пётр", "Роман", "Руслан", "Сергей", "Степан", "Тимур", "Фёдор", "Юрий", "Ярослав" };

string surname[] = { "Пантелеев", "Чувашев", "Галинов", "Ананина", "Масылюк", "Вострокнутова", "Сабуров", "Исламов", "Шамай", "Филатов", "Сафронов", "Шарпов", "Карелов", "Брейкин", "Копытов", "Солдатов", "Кузнецов", "Пажгин", "Фотин", "Бадретдинов", "Механошин", "Булдаков", "Тулинов", "Тедеев", "Колпаков", "Черных", "Нефедов", "Рябцев", "Пепеляев", "Тарасов", "Аркадьев" };

string middle[] = { "Романович", "Иосифович", "Аникитевич", "Прохорович", "Архипович", "Александрович", "Мартьянович", "Богданович", "Никифорович", "Георгиевич" , "Прокофиевич" , "Валериевич" , "Владиславович" , "Данилевич" , "Ульянович" , "Кондратиевич" , "Валериевич" , "Сократович" , "Кондратович" , "Евграфович" , "Алексеевич" , "Кондратович" , "Тихонович" , "Игнатиевич" , "Алексеевич" , "Венедиктович" , "Мартьянович" , "Родионович" , "Тимурович" , "Самуилович" , "Климентович" , "Натанович" , "Леонович" , "Игоревич" , "Захарович" , "Андронович" , "Платонович" , "Несторович" , "Модестович" , "Ерофеевич" , "Адамович" };

for (int i = 0; i < size; i++) // цикл от 0 до size

{

human[i].ind = i; // заполнение поля индекса

human[i].year = 1903 + rand() % 117; // случайное заполнение поля год

human[i].month = 1 + rand() % 12; // случайное заполнение поля месяц

int a = human[i].month; // создание временной переменной хранящей месяц

switch (a) // проверка на то сколько дней в месяце

{

case 1: human[i].day = 1 + (rand() % 31); break; // если 1 месяц и так далее

case 2: human[i].day = 1 + (rand() % 28); break;

case 3: human[i].day = 1 + (rand() % 31); break;

case 4: human[i].day = 1 + (rand() % 30); break;

case 5: human[i].day = 1 + (rand() % 31); break;

case 6: human[i].day = 1 + (rand() % 30); break;

case 7: human[i].day = 1 + (rand() % 31); break;

case 8: human[i].day = 1 + (rand() % 31); break;

case 9: human[i].day = 1 + (rand() % 30); break;

case 10: human[i].day = 1 + (rand() % 31); break;

case 11: human[i].day = 1 + (rand() % 30); break;

case 12: human[i].day = 1 + (rand() % 31); break;

default:

break;

}

int g = rand() % (sizeof(name) / sizeof(name[0])); // создание переменной которая хранит номер имени

human[i].Name = name[g]; // поле имени заполняется в соответствии с случайным числом

g = rand() % (sizeof(surname) / sizeof(surname[0])); // снова генерируется случайное число которое отвечает за выбор фамилии

human[i].Surname = surname[g]; // заполняется поле фамилии

g = rand() % (sizeof(middle) / sizeof(middle[0])); // генерируется случайное число которое отвечает за выбор отчества

human[i].Middle = middle[g]; // заполнение отчества

human[i].date = human[i].day + human[i].month \* 100 + human[i].year \* 10000; // случайное заполнение поля даты

}

}

void PrintH(DATA\* human, int size) // функция вывода массива

{

cout << endl;

for (int i = 0; i < size; i++) // цикл от 0 до size

{

cout << i << endl;

cout << "ind: "<< "[" << human[i].ind << "]" << endl;

cout << human[i].Surname << endl;

cout << human[i].Name << endl;

cout << human[i].Middle << endl;

cout << human[i].day << "." << human[i].month << "." << human[i].year << endl << endl;

}

}

void PrintH(DATAHELP\* H, int size) // функция вывода для вспомогательного массива

{

for (int i = 0; i < size; i++) // цикл от 0 до size

{

cout << "[" << H[i].ind << "]" << endl;

cout << H[i].date << endl;

}

}

void add(DATA\* human, int \*size) // функция добавления элемента в список

{

int b;

cout << "Сколько элементов вы хотите добавить?" << endl;

cin >> b;

while (b < 1) // проверка числа на отрицательность

{

cout << "Вы можете добавить только натуральное количество элементов!" << endl;

cout << "Сколько элементов вы хотите добавить?" << endl;

cin >> b;

}

int Size;

Size = (\*size);

\*size = Size + b;

DATA\* HELP = new DATA[(\*size)]; // выделение памяти под новый элемент

int ch = 0;

bool f = false;

while (f == false)

{

switch (ch) // выбор пользователя куда добавить элемент

{

case 1: // в начало

humanINIT(HELP, b);

for (int i = b, j = 0; i < (\*size); i++, j++)

{

HELP[i] = human[j];

}

f = true;

break;

case 2: // в конец

for (int k = 0; k < Size; k++)

{

HELP[k] = human[k];

}

humanINIT(HELP, (\*size));

f = true;

break;

default:

cout << "Куда вы хотите добавить элемент?" << endl;

cout << "1: В начало списка" << endl;

cout << "2: В конец списка" << endl;

cin >> ch;

break;

}

}

PrintH(HELP, \*size); // вызов функции вывода списка

}

int saveH(DATA\* human, int size, string path\_F) // сохранение в документ

{

ofstream fout;

fout.open(path\_F); // открываем файл

if (!fout.is\_open()) // проверка на то, открыт ли файл

{

cout << "Ошибка открытия файла!" << endl;

return 0;

}

else

{

string str;

cout << "Файл " << path\_F << " был успешно открыт!\n";

for (int i = 0; i < size; i++) { // в цикле проходимся по всему списку и заносим все поля всех элементов в документ

fout << human[i].Name << '\n';

fout << human[i].Surname << '\n';

fout << human[i].Middle << '\n';

fout << human[i].day << "." << human[i].month << "." << human[i].year << '\n' << '\n';

}

}

fout.close(); // закрываем файл

return 0;

}

void SearchLine(DATA\* human, int size, string str, list<DATA> help) // функция линнейного поиска по ФИО

{

cout << endl << "Введите желаемое ФИО: ";

cin.ignore();

getline(cin, str); // пользователь вводит желаемую строку

for (int i = 0; i < size; i++) // проходимся по всем элементам списка

{

string strH = human[i].Surname + " " + human[i].Name + " " + human[i].Middle; // создаем переменную в которой хранится ФИО элемента

if (strH == str) // если введенная строка совпала, то

{

help.push\_back(human[i]); // добавляем в конец элемент

}

}

int S = help.size(); // создаем переменную, которая будет хранить размер нашего нового списка

ptr = new DATA[S]; // выделяем динамическую память размера DATA\*S

for (int i = 0; i < S; i++) // проходимся по всем элементам списка

{

ptr[i] = help.front(); // переносим элемент из изначального списка в новый

help.pop\_front(); // достаем элемент из начала старого списка

}

PrintH(ptr, S); // выводим новый список в консоль

}

void EnterS(DATA\* human, int size, list<DATA> help) // функция ввода поиска

{

string str; // переменная хранящая строку

int foo = 0; // переменная отвечающая за выбор

int f = false;

while (f == false) {

switch (foo)

{

case 1: SearchLine(human, size, str, help); f = true; break;

default:

cout << "1: Поиск по ФИО" << endl;

cin >> foo;

break;

}

}

}

void Metod2(DATA\* human, int size, string str, list<DATA> help) // фукнция для поиска подстроки в строке

{

string strH;

cout << endl << "Введите ФИО: ";

cin.ignore();

getline(cin, strH);

bool f = false;

int j = 0; // счетчик

int l = 0; // счетчик

int p = 0; // счетчик

int count = 0; // счетчик

int b = strH.length(); // переменная хранящая размер строки

for (int i = 0; i < size; i++) // цикл проходящийся по всем элементам

{

string result = human[i].Surname + " " + human[i].Name + " " + human[i].Middle; // переменная ФИО

int a = result.length(); // длинна ФИО

bool f = 0; // флаг

while (l < strH.length() && j < (a - b) && f != true) // пока не достигнем конца строки и пока не достигнется разница длин и пока флаг лож

{

p = j; // запоминаем ноомер текущего элемента

while (strH[l] == result[j] && f != true) // сравниваем две строки

{

count++; // увеличиваем счетчик

j++; // сдвигаемся по строке

l++; // сдвигаемся по строке

if (count == strH.length()) // если достигли конца строки

{

f = true; // флаг правда

help.push\_back(human[i]); // добавляем в конец списка

}

}

j = p; // возвращаемся к элемнету

count = 0; // сбрасываем счетчик

l = 0; // сбрасываем счетчик

j++; // инкрементируем счетчик

}

j = 0; // сбрасываем счетчик

l = 0; // сбрасываем счетчик

}

int SIZE; // переменная хранящая размер списка

SIZE = help.size(); // присваиваем размер списка

ptr = new DATA[SIZE]; // создаем массив размера SIZE

for (int i = 0; i < SIZE; i++) // проходимся по всему списку

{

ptr[i] = help.front(); // передаем элемент из списка в массив

help.pop\_front(); // достаем элемент спереди списка

}

PrintH(ptr, SIZE); // печатаем массив

}

void Kostyl(DATA\* human, int size, list<DATA> help) // функция выбора поиска

{

string str; // переменная хранящая строку

int foo = 0; // переменная отвечающая за выбор

bool f = false; // флаг

while (f == false) { // пока ложь

switch (foo)

{

case 1: Metod2(human, size, str, help); f = true; break; // вызов фунции

default:

cout << "Какой поиск вы хотите выполнить?" << endl;

cout << "1: Поиск по ФИО" << endl;

cin >> foo;

break;

}

}

}

void SORT(DATA\* human, int size, list<DATA> help, DATAHELP\* H) // функция сортировки

{

int MAX = 0; // переменная хранящая максимальный элемент

DATAHELP HP; // переменная типа вспомогательной структуры

int IND; // переменная хранящая индекс

DATAHELP foo; // переменная вспомогательный структуры

bool f = false; // флаг

for (int j = size - 1; j >= 0; j--) { // проходимся по всем элементам

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

{

if (MAX < H[i].date) // если поле даты больше максимального

{

MAX = H[i].date; // присваиваем это поле нашей переменной

HP = H[i]; // запоминаем элемент

IND = i; // запоминаем индекс

f = true; // меняем значение флага

}

}

if (f == true) // если мы успешно вышли

{

foo = H[j]; // запоминаем элемент

H[j] = HP; // в массив записываем запомненный ранее элемент

H[IND] = foo; // в массив под запомненным индексом записываем запомненный элемент

f = false; // меняем флаг

MAX = 0; // обнуляем значение максимальной переменной

}

}

}

void Edit(DATA\* human, int size, list<DATA> help, DATAHELP\* H) // функция изменения списка

{

DATA Save; // переменная хранящая элемент нашей структуры

int N; // перменная хранящая индекс

list<DATA> Assistant; // вспомогательный список

for (int i = 0; i < size; i++) // проходимся по всему списку

{

N = H[i].ind; // запоминаем индекс

Assistant.push\_back(human[N]); // добавляем в конец нашего вспомогательного списка наш элемент

}

for (int i = 0; i < size; i++) // проходимся по всем элементам

{

human[i] = Assistant.front(); // добавляем в массив первый элемент вспомогательного списка

Assistant.pop\_front(); // достаем первый элемент списка

}

}

int InterSearch(DATA\* human, int size, int S) // функция интерполяционный поиск

{

int mid; // переменная хранящая стредний элемент

int left = 0; // левый элемент

int right = size - 1; // правый элемент

while (human[left].date <= S && human[right].date >= S) // пока левое значение меньше а правое польше нашего значения

{

mid = left + ((S - human[left].date) \* (right - left)) / (human[right].date - human[left].date); // ищем средний элемент по формуле

if (human[mid].date < S) // если средний элемент меньше заданного значения

{

left = mid + 1; // сдвигаем левую границу

}

else if (human[mid].date > S) // если средний элемент больше заданного значения

{

right = mid - 1; // сдвигаем правый элемент

}

else

{

return mid; // элемент найден

}

}

if (human[left].date == S) // если левая граница равна значению

{

return left; // элемент найден

}

else

{

return -1; // элемент не найден

}

}

void Interp(DATA\* human, int size, list<DATA> help) // интерполяционный поиск

{

int n1; // переменная хранящая день

int n2; // переменная хранящая месяц

int n3; // переменная хранящая год

cout << "Введите Год: ";

cin >> n3;

while (n3 < 1903 || n3 > 2021) // проверяем корректность значения

{

cout << "Введите Год: ";

cin >> n3;

}

n2 = 0;

while (n2 < 1 || n2 > 12) // проверяем корректность значения

{

cout << "Введите Месяц: ";

cin >> n2;

}

n1 = 0;

bool f = false; // создаем флаг

while (f == false)

{

switch (n2)

{

case 1: case 3: case 5: case 7: case 8: case 10: case 12:

while (n1 < 1 || n1 > 31)

{

cout << "Введите День: ";

cin >> n1;

}

f = true;

break;

case 4: case 6: case 9: case 11:

while (n1 < 1 || n1 > 30)

{

cout << "Введите День: ";

cin >> n1;

}

f = true;

break;

case 2:

while (n1 < 1 || n1 > 28)

{

cout << "Введите День: ";

cin >> n1;

}

f = true;

break;

default:

cout << "Введите День: ";

cin >> n1;

break;

}

}

int S = n3 \* 10000 + n2 \* 100 + n1; // создаем переменную которая преобразует все данные в одно число

DATAHELP\* H = new DATAHELP[size]; // создаем вспомогательный массив

for (int i = 0; i < size; i++) // проходимся по всем элементам

{

H[i].date = human[i].day + human[i].month \* 100 + human[i].year \* 10000; // заполняем поле дата

H[i].ind = i; // заполняем поле индекс

}

SORT(human, size, help, H); // сортируем по возрастанию

PrintH(H, size); // выводим на экран

Edit(human, size, help, H); // функция изменения спииска

PrintH(human, size); // выводим на экран

int result = InterSearch(human, size, S); // запускаем интерполяционный поиск

if (result == -1) // если дату не нашел

{

cout << "Такой даты рождения нет!" << endl;

}

else

{

DATA\* RES = new DATA[1];

RES[0] = human[result];

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

PrintH(RES, 1);

}

}

bool Choice(DATA\* human, int \*size, int foo, bool f, string path\_F, list<DATA> help) // меню

{

switch (foo)

{

case 1: humanINIT(human, \*size); break;

case 2: PrintH(human, \*size); break;

case 3: saveH(human, \*size, path\_F); break;

case 4: EnterS(human, \*size, help); break;

case 5: Kostyl(human, \*size, help); break;

case 6: Interp(human, \*size, help); break;

case 7: add(human, size); break;

case 8: f = true; break;

default:

break;

}

return f;

}

int main()

{

system("chcp 1251>nul");

srand(time(NULL));

int size;

size = 100;

DATA\* human = new DATA[size];

list<DATA> help;

bool f = false;

int foo = -1;

string path\_F = "DATA.txt";

while (f == false)

{

cout << endl;

cout << "Выберете действие:" << endl;

cout << "1: Инициализировать массив" << endl;

cout << "2: Напечатать массив в консоль" << endl;

cout << "3: Сохранить изменения" << endl;

cout << "4: Линейный поиск" << endl;

cout << "5: Поиск подстроки в строке" << endl;

cout << "6: Интерполяционный поиск" << endl;

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

cout << "8: Выйти из программы" << endl;

cin >> foo;

f = Choice(human, &size, foo, f, path\_F, help);

}

}

**Блок-схема**































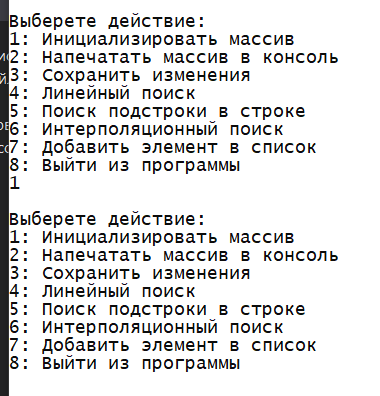
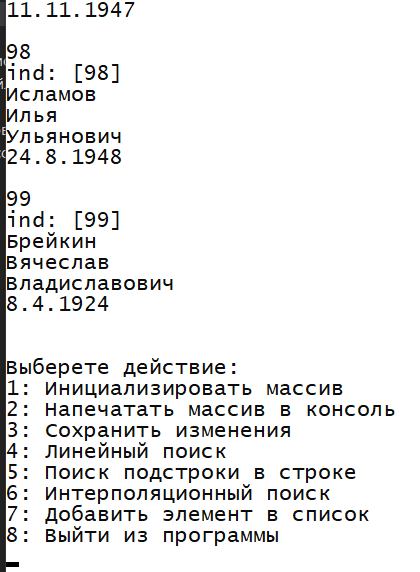
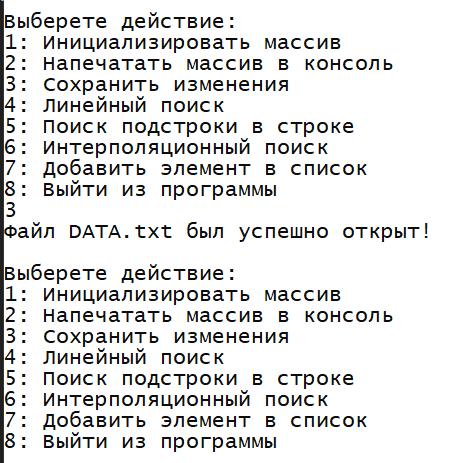
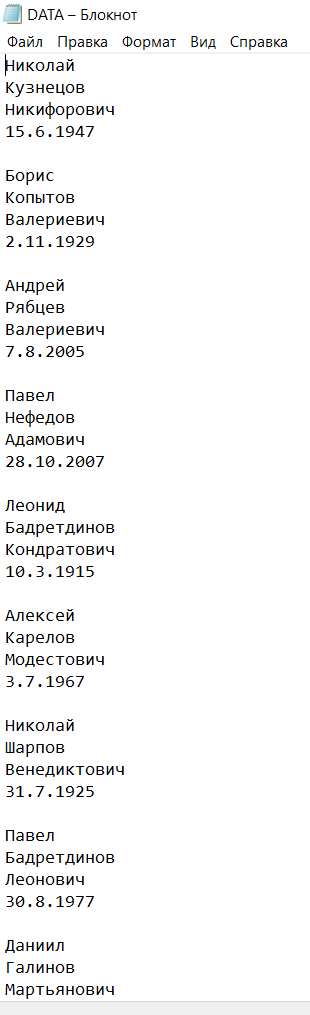
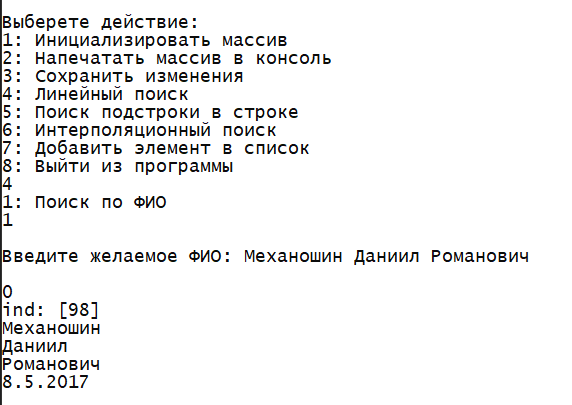
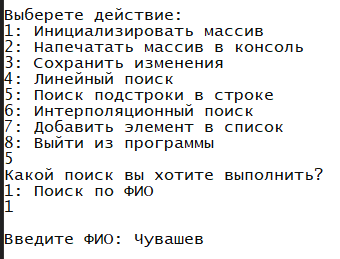
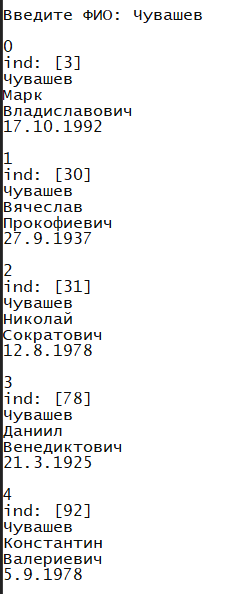
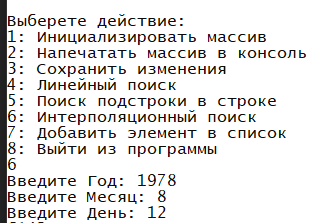
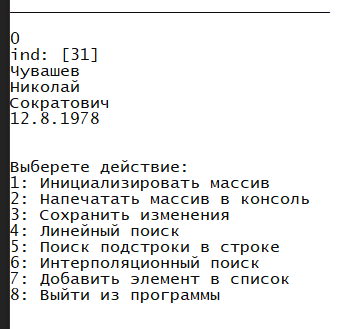
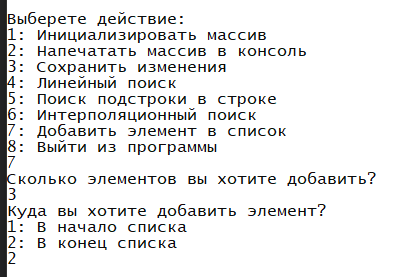








**Скриншоты результатов работы программы**

1. Нормальный ввод
   1. Линейный поиск
      1. Инициализируем массив
         * 
      2. Напечатаем сгенерированный массив в консоль
         * 
      3. Сгенерировалось 100 элементов массива
      4. Можем сохранить массив в файл
         * 
         * 
      5. Поиск по ФИО
         * 
   2. Поиск подстроки в строке
      1. Выполним поиск просто по фамилии
         * 
         * 
   3. Интерполяционный поиск
      * + 
        + 
   4. Можно добавить элемент в список
      * + 
        + 