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 лабораторным работам №12, 14**

Дисциплина: «Основы алгоритмизации и программирования»

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

Вариант 17

Выполнил:

Студент группы ИВТ-20-2б

Сафронов Владислав Владиславович

Проверила:

Доцент кафедры ИТАС

Полякова О.А.

**Пермь, 2021**

**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.
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.

Вариант 17:

Данные: ФИО, дата\_рождения, №телефона

Ключ: ФИО

**Анализ задачи**

**Какие типы данных будут использованы:**

* string
* integer
* vector
* Date
* Human
* bool

**Какие с этими данными выполняются действия:**

* integer – ввод с клавиатуры, инкремент, присваивание, сравнение;
* String – удаление, добавление символов с строку, сравнение;
* vector – удаление, добавление, изменение элементов;
* Human – генерация структуры, вывод структуры в консоль.

**В каком виде эти данные будут представлены:**

1. Структура Date с тремя целочисленными полями, хранящими дату:
2. struct Date
3. {
4. int day, mon, yr;
5. };
6. Структура Human хранит данные о одном человеке базы данных. Имеет 2 поля string для хранения имени и номера телефона, а также поле Date для хранения даты рождения. В структуре реализовано 3 метода: Print() – для вывода структуры в консоль, GeneratePhone() – для генерирования номера телефона, Generate() – для генерирования данных для экземпляра структуры.
7. struct Human
8. {
9. string name;
10. Date birth;
11. string phone;
13. void Print()
14. {
15. cout << "ФИО: " << name;
16. cout << "\nДата рождения: " << birth.day << "/" << birth.mon << "/" << birth.yr;
17. cout << "\nНомер телефона: " << phone;
18. cout << "\n--------------------------------------------------------\n";
19. }
21. string GeneratePhone()
22. {
23. string res = "+79";
24. for (int i = 0; i < 9; i++)
25. {
26. res += to\_string(rand() % 10);
27. }
28. return res;
29. }
31. void Generate()
32. {
33. name = snames[rand() % 10] + " " + names[rand() % 10] + " " + fnames[rand() % 10];
34. birth.mon = rand() % 12 + 1;
35. if (birth.mon == 2) birth.day = rand() % 28 + 1;
36. else birth.day = rand() % 30 + 1;
37. birth.yr = 1900 + (rand() % 120 + 1);
38. phone = GeneratePhone();
39. }
40. };
41. Функция CreateArr для создания массива элементов структуры Human. Элементы генерируются с помощью ДСЧ. Функция возвращает сгенерированный массив.
42. vector<Human> CreateArr(int size = 100)
43. {
44. vector<Human> lst;
45. for (int i = 0; i < size; i++)
46. {
47. Human elem;
48. elem.Generate();
49. lst.push\_back(elem);
50. }
51. return lst;
52. }
53. Функции InputFromFile и OutputToFile для ввода/вывода массива в файл
54. void InputFromFile(vector<Human>& lst)
55. {
56. ifstream input(file);
57. if (input.is\_open())
58. {
59. cout << "Чтение базы данных из файла " << file << endl;
60. lst.clear();
61. while (!input.eof())
62. {
63. Human elem = Human();
64. getline(input, elem.name);
65. input >> elem.birth.day;
66. input >> elem.birth.mon;
67. input >> elem.birth.yr;
68. input >> elem.phone;
69. input.ignore();
70. lst.push\_back(elem);
71. }
72. lst.erase(lst.end() - 1);
73. cout << "База данных из файла " << file << " загружена.\n";
74. }
75. else
76. {
77. cout << "Не удалось открыть файл " << file << endl;
78. return;
79. }
80. input.close();
81. }
83. void OutputToFile(vector<Human>& lst)
84. {
85. ofstream output(file);
86. for (int i = 0; i < lst.size(); i++)
87. {
88. output << lst[i].name << endl;
89. output << lst[i].birth.day << endl;
90. output << lst[i].birth.mon << endl;
91. output << lst[i].birth.yr << endl;
92. output << lst[i].phone << endl;
93. }
94. output.close();
95. cout << "База данных сохранена в файл " << file << endl;
96. }
97. Функция PrintArr для вывода массива элементов структуры Human в консоль.
98. void PrintArr(vector<Human> lst)
99. {
100. if (lst.empty())
101. {
102. cout << "Список пуст.\n";
103. return;
104. }
105. cout << "\n--------------------------------------------------------\n";
106. for (int i = 0; i < lst.size(); i++)
107. {
108. cout << "Номер: " << i + 1 << endl;
109. lst[i].Print();
110. }
111. }
112. Функция GetMenu для вывода меню на консоль.
113. void GetMenu()
114. {
115. cout << "\n========================================================\n";
116. cout << "| Меню управления программой\n";
117. cout << "--------------------------------------------------------\n";
118. cout << "| 1 - генерация массива\n";
119. cout << "| 2 - поиск элемента в массиве\n";
120. cout << "| 3 - добавление элемента\n";
121. cout << "| 4 - удаление элемента по номеру\n";
122. cout << "| 5 - удаление элемента по ФИО\n";
123. cout << "| 6 - восстановление последнего удаленного элемента\n";
124. cout << "| 7 - ввод из файла\n";
125. cout << "| 8 - сохранение в файл\n";
126. cout << "| 9 - вывод в консоль\n";
127. cout << "| 0 - выход\n";
128. cout << "========================================================\n";
129. }
130. Функция LineSearch реализует линейный поиск. В цикле проверяется каждое имя на соответствие ключу. В случае успеха функция выводит индекс этого элемента. В противном случае выводится -1.
131. int LineSearch(vector<Human> lst, string key)
132. {
133. for (int i = 0; i < lst.size(); i++)
134. {
135. if (lst[i].name == key) return i;
136. }
137. return -1;
138. }
139. Функция SubStringSearch реализует поиск подстроки в строке. Функция преобразует массив экземпляров структуры Human в строку имен. 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 true.
140. bool SubStringSearch(vector<Human> lst, string key)
141. {
142. string names = "";
144. for (int i = 0; i < lst.size(); i++)
145. {
146. names += lst[i].name;
147. }
149. for (int i = 0; i <= names.size() - key.size(); i++)
150. {
151. int j = 0;
152. while (names[i + j] == key[j] && j < key.size()) j++;
153. if (j == key.size()) return true;
154. }
155. return false;
156. }
157. Функция KMPSearch реализует метод поиска Кнута-Морриса-Пратта. Функция преобразует массив экземпляров структуры Human в строку имен. Затем производится поиск по методу в 2 этапа – формирование таблицы и сам поиск.
158. bool KMPSearch(vector<Human> lst, string key)
159. {
160. string names = "";
161. for (int i = 0; i < lst.size(); i++)
162. {
163. names += lst[i].name;
164. }
166. vector<int> a(key.length());
167. for (int j = 0, i = 1; i < key.length(); i++)
168. {
169. while (j > 0 && key[i] != key[j]) j = a[j - 1];
170. if (key[i] == key[j]) j++;
171. a[i] = j;
172. }
173. for (int j = 0, i = 0; i < names.length(); ++i)
174. {
175. while (j > 0 && key[j] != names[i]) j = a[j - 1];
176. j++;
177. if (j == key.length()) return true;
178. }
179. return false;
180. }
181. Функция BoyerMoorSearch реализует метод поиска Бойера-Мура. Функция преобразует массив экземпляров структуры Human в строку имен. Затем происходит формирование таблицы и сам поиск.
182. bool BoyerMoorSearch(vector<Human> lst, string key)
183. {
184. int stl, ctl;
185. string st = "";
186. for (int i = 0; i < lst.size(); i++)
187. {
188. st += lst[i].name;
189. }
190. stl = st.size();
191. ctl = key.size();
192. if (stl != 0 && ctl != 0 && stl >= ctl)
193. {
194. int i, p;
195. int b[256];
196. for (i = 0; i < 256; i++)
197. {
198. b[i] = ctl;
199. }
200. for (i = ctl - 2; i >= 0; i--)
201. {
202. if (b[int(unsigned char(key[i]))] == ctl)
203. {
204. b[int(unsigned char(key[i]))] = ctl - i - 1;
205. }
206. }
207. p = ctl - 1;
208. while (p < stl)
209. {
210. if (key[ctl - 1] != st[p])
211. {
212. p += b[int((unsigned char)st[p])];
213. }
214. else
215. {
216. for (i = ctl - 1; i >= 0; i--)
217. {
218. if (key[i] != st[p - ctl + i + 1])
219. {
220. p += b[int((unsigned char)st[p])];
221. break;
222. }
223. else if (i == 0)
224. {
225. return true;
226. }
227. }
228. }
229. }
230. }
231. return false;
232. }
233. Функция InterpolationSearch реализует интерполяционный поиск.
234. bool InterpolationSearch(vector<Human> lst, string key)
235. {
236. vector<string> names(lst.size());
237. int n;
238. for (int i = 0; i < lst.size(); i++)
239. {
240. names[i] = lst[i].name;
241. }
242. for (int i = 0; i < names.size(); i++)
243. {
244. for (int j = i; j > 0 && names[j - 1] > names[j]; j--)
245. {
246. swap(names[j - 1], names[j]);
247. }
248. }
249. int mid, left = 0, right = names.size() - 1;
250. while (names[left] < key && names[right] > key)
251. {
252. mid = left + (key[0] - names[left][0]) \* (right - left) / (names[right][0] - names[left][0]);
253. if (names[mid] < key) left = mid + 1;
254. else if (names[mid] > key) right = mid - 1;
255. }
256. if (names[left] == key)
257. {
258. n = left;
259. return true;
260. }
261. if (names[right] == key)
262. {
263. n = right;
264. return true;
265. }
266. return false;
267. }
268. Функция Search выполняет поиск ключа в массиве с помощью указанных выше функций.
269. void Search(vector<Human> lst)
270. {
271. if (lst.empty())
272. {
273. cout << "Список пуст, поиск невозможен.\n";
274. return;
275. }
277. string key;
278. bool f;
279. int res;
281. cout << "Введите ФИО для поиска: ";
282. getline(cin, key);
284. cout << "Поиск элемента с именем: " << key << endl;
286. cout << "\nМетод прямого поиска:\n";
287. res = LineSearch(lst, key);
288. if (res != -1) cout << "\tЭлемент найден. Его номер: " << res + 1 << endl;
289. else cout << "\tЭлемент не найден.\n";
291. cout << "\nМетод интерполяционного поиска:\n";
292. f = InterpolationSearch(lst, key);
293. if (f) cout << "\tЭлемент найден.\n";
294. else cout << "\tЭлемент не найден.\n";
296. cout << "\nМетод поиска подстроки в строке:\n";
297. f = SubStringSearch(lst, key);
298. if (f) cout << "\tЭлемент найден.\n";
299. else cout << "\tЭлемент не найден.\n";
301. cout << "\nМетод Боера Мура\n";
302. f = BoyerMoorSearch(lst, key);
303. if (f) cout << "\tЭлемент найден.\n";
304. else cout << "\tЭлемент не найден.\n";
306. cout << "\nМетод Кнутта-Морриса-Пратта\n";
307. f = KMPSearch(lst, key);
308. if (f) cout << "\tЭлемент найден.\n";
309. else cout << "\tЭлемент не найден.\n";
310. }
311. Функция AddByNum добавляет новый элемент в массив на заданный номер.
312. void AddByNum(vector<Human>& lst)
313. {
314. int n = 0;
316. cout << "Добавление элемента в список... \n";
317. if (!lst.empty())
318. {
319. cout << "Введите номер нового элемента (1-" << lst.size() + 1 << "): ";
320. cin >> n;
321. while (n < 1 || n > lst.size() + 1)
322. {
323. cout << "\nНомер введен некорректно.\n";
324. cout << "Введите номер нового элемента (1-" << lst.size() + 1 << "): ";
325. cin >> n;
326. }
327. cin.get();
328. n--;
329. }
330. Human elem;
331. cout << "Введите ФИО: ";
332. getline(cin, elem.name);
333. cout << "Введите дату рождения (дд мм гггг): ";
334. cin >> elem.birth.day >> elem.birth.mon >> elem.birth.yr;
335. cout << "Введите номер телефона (11 цифр, начиная с 7): ";
336. cin >> elem.phone;
337. while (elem.phone < "70000000000" || elem.phone > "79999999999")
338. {
339. cout << "\nНомер телефона введен некорректно.\n";
340. cout << "Введите номер телефона (11 цифр, начиная с 7): ";
341. cin >> elem.phone;
342. }
343. lst.insert(lst.begin() + n, elem);
344. cout << "Добавление выполнено успешно.\n";
345. }
346. Функция DelByNum для удаления элемента структуры из массива по номеру.
347. void DelByNum(vector<Human>& lst)
348. {
349. if (lst.empty())
350. {
351. cout << "Список пуст. Удаление невозможно.\n";
352. return;
353. }
355. int n;
357. cout << "Удаление элемента по номеру... \n";
358. cout << "Введите номер элемента для удаления (1-" << lst.size() << "): ";
359. cin >> n;
360. while (n < 1 || n > lst.size())
361. {
362. cout << "\nНомер введен некорректно.\n";
363. cout << "Введите номер нового элемента (1-" << lst.size() << "): ";
364. cin >> n;
365. }
366. n--;
367. DelList.push\_back(lst[n]);
368. lst.erase(lst.begin() + n);
369. cout << "Элемент успешно удален.\n";
370. }
371. Функция DelByKey для удаления элемента из массива по ФИО.
372. void DelByKey(vector<Human>& lst)
373. {
374. if (lst.empty())
375. {
376. cout << "Список пуст. Удаление невозможно.\n";
377. return;
378. }
380. string dname;
381. cout << "Введите ФИО элемента, который нужно удалить: ";
382. getline(cin, dname);
383. int n = LineSearch(lst, dname);
384. if (n == -1) cout << "Элемент с таким ФИО не найден.\n";
385. else
386. {
387. DelList.push\_back(lst[n]);
388. lst.erase(lst.begin() + n);
389. cout << "Элемент успешно удален.\n";
390. }
391. }
392. Функция RecoveryLast для восстановления последнего удаленного элемента.
393. void RecoveryLast(vector<Human>& lst)
394. {
395. if (DelList.empty())
396. {
397. cout << "Список на удаление пуст, восстановление невозможно.";
398. }
399. else
400. {
401. Human elem = DelList[DelList.size() - 1];
402. DelList.erase(DelList.end() - 1);
403. lst.push\_back(elem);
404. cout << "Последний удаленный элемент восстановлен.";
405. }
406. }

**Какими операторами будет организован ввод и вывод:**

* Ввод данных осуществляется через консоль с помощью функции cin и getline();
* Вывод данных в консоль осуществляется с помощью функции cout;
* Ввод и вывод данных из файлов реализован с помощью библиотеки <fstream>.

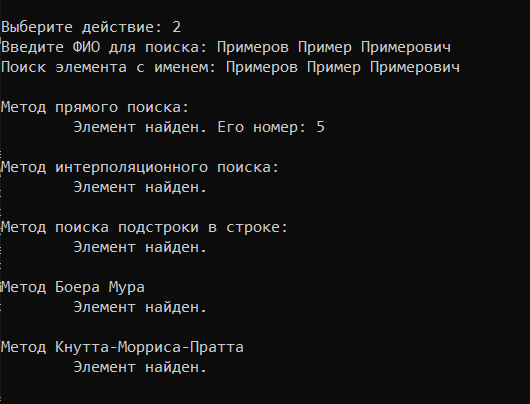
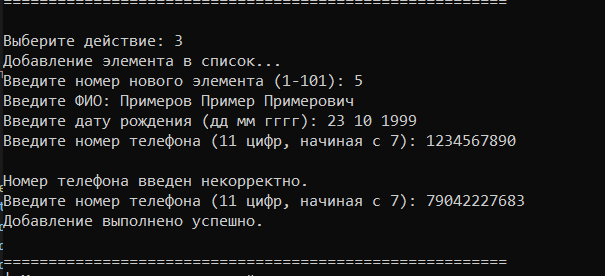
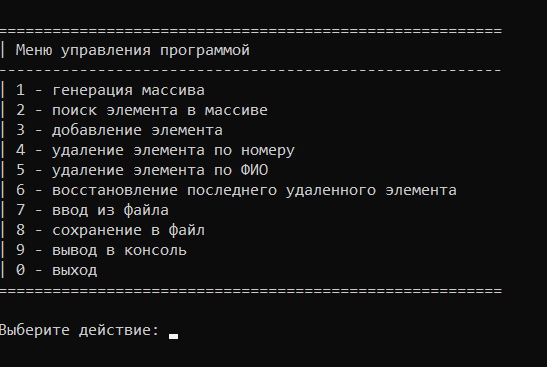
**Описание функции main():**

**Блок-схема программы**

**Код программы**

1. #include <iostream>
2. #include <vector>
3. #include <string>
4. #include <fstream>
5. #include <ctime>
6. using namespace std;
8. string file = "database.txt";
10. string names[] = { "Марк", "Егор", "Руслан", "Андрей", "Дмитрий", "Павел", "Юрий", "Сергей", "Никита", "Кирилл" };
11. string snames[] = { "Терехов", "Латышев", "Адрианов", "Титов", "Логинов", "Тихонов", "Беликов", "Поздняков", "Блинов", "Исаков" };
12. string fnames[] = { "Маркович", "Егорович", "Русланович", "Андреевич", "Дмитриевич", "Павлович", "Юрьевич", "Сергеевич", "Никитович", "Кириллович" };
14. struct Human;
16. vector<Human> DelList;
18. struct Date
19. {
20. int day, mon, yr;
21. };
23. struct Human
24. {
25. string name;
26. Date birth;
27. string phone;
29. void Print()
30. {
31. cout << "ФИО: " << name;
32. cout << "\nДата рождения: " << birth.day << "/" << birth.mon << "/" << birth.yr;
33. cout << "\nНомер телефона: " << phone;
34. cout << "\n--------------------------------------------------------\n";
35. }
37. string GeneratePhone()
38. {
39. string res = "+79";
40. for (int i = 0; i < 9; i++)
41. {
42. res += to\_string(rand() % 10);
43. }
44. return res;
45. }
47. void Generate()
48. {
49. name = snames[rand() % 10] + " " + names[rand() % 10] + " " + fnames[rand() % 10];
50. birth.mon = rand() % 12 + 1;
51. if (birth.mon == 2) birth.day = rand() % 28 + 1;
52. else birth.day = rand() % 30 + 1;
53. birth.yr = 1900 + (rand() % 120 + 1);
54. phone = GeneratePhone();
55. }
56. };
58. vector<Human> CreateArr(int size = 100)
59. {
60. vector<Human> lst;
61. for (int i = 0; i < size; i++)
62. {
63. Human elem;
64. elem.Generate();
65. lst.push\_back(elem);
66. }
67. return lst;
68. }
70. void OutputToFile(vector<Human>& lst)
71. {
72. ofstream output(file);
73. for (int i = 0; i < lst.size(); i++)
74. {
75. output << lst[i].name << endl;
76. output << lst[i].birth.day << endl;
77. output << lst[i].birth.mon << endl;
78. output << lst[i].birth.yr << endl;
79. output << lst[i].phone << endl;
80. }
81. output.close();
82. cout << "База данных сохранена в файл " << file << endl;
83. }
85. void InputFromFile(vector<Human>& lst)
86. {
87. ifstream input(file);
88. if (input.is\_open())
89. {
90. cout << "Чтение базы данных из файла " << file << endl;
91. lst.clear();
92. while (!input.eof())
93. {
94. Human elem = Human();
95. getline(input, elem.name);
96. input >> elem.birth.day;
97. input >> elem.birth.mon;
98. input >> elem.birth.yr;
99. input >> elem.phone;
100. input.ignore();
101. lst.push\_back(elem);
102. }
103. lst.erase(lst.end() - 1);
104. cout << "База данных из файла " << file << " загружена.\n";
105. }
106. else
107. {
108. cout << "Не удалось открыть файл " << file << endl;
109. return;
110. }
111. input.close();
112. }
114. void PrintArr(vector<Human> lst)
115. {
116. if (lst.empty())
117. {
118. cout << "Список пуст.\n";
119. return;
120. }
121. cout << "\n--------------------------------------------------------\n";
122. for (int i = 0; i < lst.size(); i++)
123. {
124. cout << "Номер: " << i + 1 << endl;
125. lst[i].Print();
126. }
127. }
129. void GetMenu()
130. {
131. cout << "\n========================================================\n";
132. cout << "| Меню управления программой\n";
133. cout << "--------------------------------------------------------\n";
134. cout << "| 1 - генерация массива\n";
135. cout << "| 2 - поиск элемента в массиве\n";
136. cout << "| 3 - добавление элемента\n";
137. cout << "| 4 - удаление элемента по номеру\n";
138. cout << "| 5 - удаление элемента по ФИО\n";
139. cout << "| 6 - восстановление последнего удаленного элемента\n";
140. cout << "| 7 - ввод из файла\n";
141. cout << "| 8 - сохранение в файл\n";
142. cout << "| 9 - вывод в консоль\n";
143. cout << "| 0 - выход\n";
144. cout << "========================================================\n";
145. }
147. int LineSearch(vector<Human> lst, string key)
148. {
149. for (int i = 0; i < lst.size(); i++)
150. {
151. if (lst[i].name == key) return i;
152. }
153. return -1;
154. }
156. bool SubStringSearch(vector<Human> lst, string key)
157. {
158. string names = "";
160. for (int i = 0; i < lst.size(); i++)
161. {
162. names += lst[i].name;
163. }
165. for (int i = 0; i <= names.size() - key.size(); i++)
166. {
167. int j = 0;
168. while (names[i + j] == key[j] && j < key.size()) j++;
169. if (j == key.size()) return true;
170. }
171. return false;
172. }
174. bool BoyerMoorSearch(vector<Human> lst, string key)
175. {
176. int stl, ctl;
177. string st = "";
178. for (int i = 0; i < lst.size(); i++)
179. {
180. st += lst[i].name;
181. }
182. stl = st.size();
183. ctl = key.size();
184. if (stl != 0 && ctl != 0 && stl >= ctl)
185. {
186. int i, p;
187. int b[256];
188. for (i = 0; i < 256; i++)
189. {
190. b[i] = ctl;
191. }
192. for (i = ctl - 2; i >= 0; i--)
193. {
194. if (b[int(unsigned char(key[i]))] == ctl)
195. {
196. b[int(unsigned char(key[i]))] = ctl - i - 1;
197. }
198. }
199. p = ctl - 1;
200. while (p < stl)
201. {
202. if (key[ctl - 1] != st[p])
203. {
204. p += b[int((unsigned char)st[p])];
205. }
206. else
207. {
208. for (i = ctl - 1; i >= 0; i--)
209. {
210. if (key[i] != st[p - ctl + i + 1])
211. {
212. p += b[int((unsigned char)st[p])];
213. break;
214. }
215. else if (i == 0)
216. {
217. return true;
218. }
219. }
220. }
221. }
222. }
223. return false;
224. }
226. bool KMPSearch(vector<Human> lst, string key)
227. {
228. string names = "";
229. for (int i = 0; i < lst.size(); i++)
230. {
231. names += lst[i].name;
232. }
234. vector<int> a(key.length());
235. for (int j = 0, i = 1; i < key.length(); i++)
236. {
237. while (j > 0 && key[i] != key[j]) j = a[j - 1];
238. if (key[i] == key[j]) j++;
239. a[i] = j;
240. }
241. for (int j = 0, i = 0; i < names.length(); ++i)
242. {
243. while (j > 0 && key[j] != names[i]) j = a[j - 1];
244. j++;
245. if (j == key.length()) return true;
246. }
247. return false;
248. }
250. bool InterpolationSearch(vector<Human> lst, string key)
251. {
252. vector<string> names(lst.size());
253. int n;
254. for (int i = 0; i < lst.size(); i++)
255. {
256. names[i] = lst[i].name;
257. }
258. for (int i = 0; i < names.size(); i++)
259. {
260. for (int j = i; j > 0 && names[j - 1] > names[j]; j--)
261. {
262. swap(names[j - 1], names[j]);
263. }
264. }
265. int mid, left = 0, right = names.size() - 1;
266. while (names[left] < key && names[right] > key)
267. {
268. mid = left + (key[0] - names[left][0]) \* (right - left) / (names[right][0] - names[left][0]);
269. if (names[mid] < key) left = mid + 1;
270. else if (names[mid] > key) right = mid - 1;
271. }
272. if (names[left] == key)
273. {
274. n = left;
275. return true;
276. }
277. if (names[right] == key)
278. {
279. n = right;
280. return true;
281. }
282. return false;
283. }
285. void Search(vector<Human> lst)
286. {
287. if (lst.empty())
288. {
289. cout << "Список пуст, поиск невозможен.\n";
290. return;
291. }
293. string key;
294. bool f;
295. int res;
297. cout << "Введите ФИО для поиска: ";
298. getline(cin, key);
300. cout << "Поиск элемента с именем: " << key << endl;
302. cout << "\nМетод прямого поиска:\n";
303. res = LineSearch(lst, key);
304. if (res != -1) cout << "\tЭлемент найден. Его номер: " << res + 1 << endl;
305. else cout << "\tЭлемент не найден.\n";
307. cout << "\nМетод интерполяционного поиска:\n";
308. f = InterpolationSearch(lst, key);
309. if (f) cout << "\tЭлемент найден.\n";
310. else cout << "\tЭлемент не найден.\n";
312. cout << "\nМетод поиска подстроки в строке:\n";
313. f = SubStringSearch(lst, key);
314. if (f) cout << "\tЭлемент найден.\n";
315. else cout << "\tЭлемент не найден.\n";
317. cout << "\nМетод Боера Мура\n";
318. f = BoyerMoorSearch(lst, key);
319. if (f) cout << "\tЭлемент найден.\n";
320. else cout << "\tЭлемент не найден.\n";
322. cout << "\nМетод Кнутта-Морриса-Пратта\n";
323. f = KMPSearch(lst, key);
324. if (f) cout << "\tЭлемент найден.\n";
325. else cout << "\tЭлемент не найден.\n";
326. }
328. void AddByNum(vector<Human>& lst)
329. {
330. int n = 0;
332. cout << "Добавление элемента в список... \n";
333. if (!lst.empty())
334. {
335. cout << "Введите номер нового элемента (1-" << lst.size() + 1 << "): ";
336. cin >> n;
337. while (n < 1 || n > lst.size() + 1)
338. {
339. cout << "\nНомер введен некорректно.\n";
340. cout << "Введите номер нового элемента (1-" << lst.size() + 1 << "): ";
341. cin >> n;
342. }
343. cin.get();
344. n--;
345. }
346. Human elem;
347. cout << "Введите ФИО: ";
348. getline(cin, elem.name);
349. cout << "Введите дату рождения (дд мм гггг): ";
350. cin >> elem.birth.day >> elem.birth.mon >> elem.birth.yr;
351. cout << "Введите номер телефона (11 цифр, начиная с 7): ";
352. cin >> elem.phone;
353. while (elem.phone < "70000000000" || elem.phone > "79999999999")
354. {
355. cout << "\nНомер телефона введен некорректно.\n";
356. cout << "Введите номер телефона (11 цифр, начиная с 7): ";
357. cin >> elem.phone;
358. }
359. lst.insert(lst.begin() + n, elem);
360. cout << "Добавление выполнено успешно.\n";
361. }
363. void DelByNum(vector<Human>& lst)
364. {
365. if (lst.empty())
366. {
367. cout << "Список пуст. Удаление невозможно.\n";
368. return;
369. }
371. int n;
373. cout << "Удаление элемента по номеру... \n";
374. cout << "Введите номер элемента для удаления (1-" << lst.size() << "): ";
375. cin >> n;
376. while (n < 1 || n > lst.size())
377. {
378. cout << "\nНомер введен некорректно.\n";
379. cout << "Введите номер нового элемента (1-" << lst.size() << "): ";
380. cin >> n;
381. }
382. n--;
383. DelList.push\_back(lst[n]);
384. lst.erase(lst.begin() + n);
385. cout << "Элемент успешно удален.\n";
386. }
388. void DelByKey(vector<Human>& lst)
389. {
390. if (lst.empty())
391. {
392. cout << "Список пуст. Удаление невозможно.\n";
393. return;
394. }
396. string dname;
397. cout << "Введите ФИО элемента, который нужно удалить: ";
398. getline(cin, dname);
399. int n = LineSearch(lst, dname);
400. if (n == -1) cout << "Элемент с таким ФИО не найден.\n";
401. else
402. {
403. DelList.push\_back(lst[n]);
404. lst.erase(lst.begin() + n);
405. cout << "Элемент успешно удален.\n";
406. }
407. }
409. void RecoveryLast(vector<Human>& lst)
410. {
411. if (DelList.empty())
412. {
413. cout << "Список на удаление пуст, восстановление невозможно.";
414. }
415. else
416. {
417. Human elem = DelList[DelList.size() - 1];
418. DelList.erase(DelList.end() - 1);
419. lst.push\_back(elem);
420. cout << "Последний удаленный элемент восстановлен.";
421. }
422. }
424. int main()
425. {
426. setlocale(LC\_ALL, "Russian");
427. system("chcp 1251>nul");
428. srand(time(NULL));
430. vector<Human> data;
431. int input;
433. do {
434. GetMenu();
435. cout << "\nВыберите действие: ";
436. cin >> input;
437. cin.get();
439. switch (input)
440. {
441. case 1: data = CreateArr(); cout << "\nМассив сгенерирован.\n"; break;
442. case 2: Search(data); break;
443. case 3: AddByNum(data); break;
444. case 4: DelByNum(data); break;
445. case 5: DelByKey(data); break;
446. case 6: RecoveryLast(data); break;
447. case 7: InputFromFile(data); break;
448. case 8: OutputToFile(data); break;
449. case 9: PrintArr(data); break;
450. case 0: break;
451. default: cout << "Ввод некорректен.\n";
452. }
453. } while (input != 0);
454. cout << "Завершеие работы.\n";
455. return 0;
456. }

**Скриншоты результатов работы программы**

****