Выполнил

студент группы КТбо1-2 А. С. Мумладзе

Принял

доцент кафедры САиТ В. С. Лапшин

Таганрог 2024

Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное

учреждение высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

(ФГАОУ ВО «ЮФУ»)

Институт компьютерных технологий и информационной безопасности

Кафедра системного анализа и телекоммуникаций

ЛАБОРАТОРНАЯ РАБОТА №3

по дисциплине «Основы алгоритмизации и программирования»

**«РАБОТА СО СТРУКТУРАМИ»**

Вариант 9

# Содержание

Содержание 2

Техническое задание 3

Цель задания 3

Задача 3

Ход работы 4

Алгоритм 4

Написание программы 4

Пример работы программы 6

Вывод 7

Листинг 8

main.cpp 8

common.h 15

# Техническое задание

## Цель задания

Цель данной лабораторной работы состоит в том, чтобы научить студентов работать со сложными типами данных, каковыми являются структуры данных.

Для успешного выполнения лабораторной работы студенту необходимо освоить способы задания и методы обработки данных структурного типа. Для этого необходимо по конспекту лекций освоить понятие структур, способы ее задания, способы выделения памяти под нее. Дополнительно необходимо знать инициализацию и присваивание структур, доступ к элементам структур, понятие массива структур, указателей на структуры и операции над указателями. Кроме того, необходимо уметь работать со структурами и функциями.

## Задача

Задание. Написать программу, которая записывает с клавиатуры в файл структуру согласно выданному варианту задания. В качестве разделителя полей структуры использовать символ табуляции. В программе реализовать:

1. Дополнение существующего массива структур новыми структурами;
2. Поиск структуры с заданным значением выбранного элемента;
3. Вывод на экран содержимого массива структур;
4. Упорядочение массива структур по заданному полю (элементу), например государство по численности.

Заданием моего варианта являлся массив описания владельцев автомобилей, содержащий следующие поля: имя, номер автомобиля, номер техпаспорта, дата рождения, телефон, отделение регистрации ГИБДД.

# Ход работы

## Алгоритм

Алгоритм моей программы следующий:

1. Выделить память под пустой массив.
2. В цикле while (true) печатается список операций над массивом, и ожидается ввод пользователя (int), по которому будут вызываться необходимые случаи.
   1. При вводе пользователя «1» вызывается функция add\_node, которая принимает у пользователя ввод – новую запись в массив.
   2. При вводе пользователя «2» вызывается функция find\_node, которая принимает у пользователя ввод – искомую запись, и совершает поиск по всему массиву. Функция выводит в консоль все совпадающие записи и их индексы по массиву.
   3. При вводе пользователя «3» вызывается функция print\_list, которая выводит пользователю на экран все записи в массиве.
   4. При вводе пользователя «4» вызывается функция sort\_list, которая принимает у пользователя число от 1 до 6 – номер поля структур в массиве, и после этого совершает сортировку по указанному полю (в алфавитном порядке).
   5. При любом ином вводе пользователя программа печатает предупреждение, что ввод не соответствует инструкции.
3. Для завершения программы пользователь вводит «0». После этого вызывается функция close\_program, которая записывает массив в файл и очищает память, выделенную под массив.

## Написание программы

Для массива создается структура CarsList, содержащая указатель на массив структур CarNode и длину массива. Указатель на массив – это выделенная память под массив при помощи calloc().

Выбор пользователем нужной функции осуществлен при помощи switch case и нумерованного списка enum UserTask.

Каждая из функций операций над массивом принимает указатель на структуру CarsList, и совершает операции над нам через указатели.

## Пример работы программы

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

# Вывод

Итогом работы стала программа, работающая с массивом структур. Программа может добавлять в массив запись, сортировать массив по нужному полю, искать запись и выводить весь массив в консоль.

В ходе работы я ознакомился с возможностями создания массива структур и над операциями над ним.

# Листинг

## main.cpp

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include "common.h"
5. int main() {
6. CarsList all\_cars = {(CarNode\*\*) calloc(100, sizeof(CarNode)), 0};
7. printf("List of commands:\n0 - Close program\n1 - Add node\n2 - Find node\n3 - Print all nodes\n4 - Sort\n");
8. while (true) {
9. UserTask user\_task = (UserTask) get\_user\_task();
10. switch (user\_task) {
11. case CLOSE\_PROGRAM:
12. close\_program(&all\_cars);
13. break;
14. case ADD\_NODE:
15. add\_node(&all\_cars);
16. break;
17. case FIND\_NODE:
18. find\_node(&all\_cars);
19. break;
20. case PRINT\_LIST:
21. print\_list(&all\_cars);
22. break;
23. case SORT\_LIST:
24. sort\_list(&all\_cars);
25. break;
26. default:
27. printf("Your input doesn't match with any of command. Input again\n");
28. break;
29. }
30. }
31. }
32. void clean\_input() {
33. while (getchar() != '\n');
34. }
35. int get\_user\_task() {
36. int user\_task = 0;
37. printf("Choose command: ");
38. scanf("%d", &user\_task);
39. clean\_input();
40. return user\_task;
41. }
42. void close\_program(CarsList\* all\_cars) {
43. char\* output\_string = (char\*) calloc(all\_cars->len, sizeof(CarNode));
44. for (int i = 0; i < all\_cars->len; i++) {
45. CarNode\* node = all\_cars->list[i];
46. char\* temp = (char\*) calloc(1, sizeof(CarNode));
47. sprintf(temp, "%s; %s; %s; %s; %s; %s\n", node->p\_name, node->p\_birthday, node->p\_phone,
48. node->c\_sign, node->c\_passport, node->c\_police\_code);
49. strcat(output\_string, temp);
50. free(temp);
51. }
52. FILE\* file = fopen("list.txt", "w");
53. fputs(output\_string, file);
54. fclose(file);
55. free(output\_string);
56. printf("The program is corrently terminated\n");
57. exit(0);
58. }
59. void add\_node(CarsList\* all\_cars) {
60. printf("\nEnter data using next template:\n"
61. "Name; Birthday; Phone; Car\_Sign; Car\_Passport; Code\_of\_transport\_police\n");
62. CarNode\* car = (CarNode\*) calloc(1, sizeof(CarNode));
63. scanf("%[^;]; %[^;]; %[^;]; %[^;]; %[^;]; %[^\n]", car->p\_name, car->p\_birthday,
64. car->p\_phone, car->c\_sign, car->c\_passport, car->c\_police\_code);
65. clean\_input();
66. all\_cars->list[all\_cars->len] = car;
67. all\_cars->len++;
68. printf("Node has added\n\n");
69. }
70. void find\_node(CarsList\* all\_cars) {
71. printf("\nChoose info to search:\n1 - Name\n2 - Birthday\n3 - Phone\n4 - Car Sign\n"
72. "5 - Car Passport\n6 - Transport Police Code\n");
73. NodeField field;
74. scanf("%d", (int\*) &field);
75. clean\_input();
76. if (field < 0 || field > 5) {
77. printf("Entered wrong field code\n");
78. return;
79. }
80. printf("Enter searcing data: ");
81. char\* user\_input = (char\*) calloc(21, sizeof(char));
82. scanf("%[^\n]", user\_input);
83. clean\_input();
84. printf("Founded data:\n");
85. for (int i = 0; i < all\_cars->len; i++) {
86. CarNode\* node = all\_cars->list[i];
87. int diff = 21;
88. switch (field) {
89. case NAME:
90. diff = strcmp(node->p\_name, user\_input);
91. break;
92. case BIRTHDAY:
93. diff = strcmp(node->p\_birthday, user\_input);
94. break;
95. case PHONE:
96. diff = strcmp(node->p\_phone, user\_input);
97. break;
98. case SIGN:
99. diff = strcmp(node->c\_sign, user\_input);
100. break;
101. case PASSPORT:
102. diff = strcmp(node->c\_passport, user\_input);
103. break;
104. case POLICE\_NUM:
105. diff = strcmp(node->c\_police\_code, user\_input);
106. break;
107. default:
108. printf("Error");
109. break;
110. }
111. if (diff == 0) {
112. printf("%d: %s; %s; %s; %s; %s; %s\n", i, node->p\_name, node->p\_birthday, node->p\_phone,
113. node->c\_sign, node->c\_passport, node->c\_police\_code);
114. }
115. }
116. }
117. void print\_list(CarsList\* all\_cars) {
118. printf("\n");
119. for (int i = 0; i < all\_cars->len; i++) {
120. CarNode\* node = all\_cars->list[i];
121. printf("%d: %s; %s; %s; %s; %s; %s\n", i, node->p\_name, node->p\_birthday, node->p\_phone,
122. node->c\_sign, node->c\_passport, node->c\_police\_code);
123. }
124. }
125. void sort\_list(CarsList\* all\_cars) {
126. printf("\nChoose info to search:\n1 - Name\n2 - Birthday\n3 - Phone\n4 - Car Sign\n"
127. "5 - Car Passport\n6 - Transport Police Code\n");
128. NodeField field;
129. scanf("%d", (int\*) &field);
130. clean\_input();
131. if (field < 0 || field > 5) {
132. printf("Entered wrong field code\n");
133. return;
134. }
135. bool is\_sorted = false;
136. CarNode\* node\_1 = nullptr;
137. CarNode\* node\_2 = nullptr;
138. int diff = 0;
139. while (!is\_sorted) {
140. is\_sorted = true;
141. for (int i = 0; i < all\_cars->len - 1; i++) {
142. node\_1 = all\_cars->list[i];
143. node\_2 = all\_cars->list[i+1];
144. switch (field) {
145. case NAME:
146. diff = strcmp(node\_1->p\_name, node\_2->p\_name);
147. break;
148. case BIRTHDAY:
149. diff = strcmp(node\_1->p\_birthday, node\_2->p\_birthday);
150. break;
151. case PHONE:
152. diff = strcmp(node\_1->p\_phone, node\_2->p\_phone);
153. break;
154. case SIGN:
155. diff = strcmp(node\_1->c\_sign, node\_2->c\_sign);
156. break;
157. case PASSPORT:
158. diff = strcmp(node\_1->c\_passport, node\_2->c\_passport);
159. break;
160. case POLICE\_NUM:
161. diff = strcmp(node\_1->c\_police\_code, node\_2->c\_police\_code);
162. break;
163. default:
164. printf("Ërror");
165. break;
166. }
167. if (diff > 0) {
168. is\_sorted = false;
169. all\_cars->list[i] = node\_2;
170. all\_cars->list[i+1] = node\_1;
171. }
172. }
173. }
174. }

## common.h

1. struct CarNode {
2. char p\_name[21];
3. char p\_birthday[11];
4. char p\_phone[15];
5. char c\_sign[11];
6. char c\_passport[11];
7. char c\_police\_code[8];
8. };
9. struct CarsList {
10. CarNode\*\* list = nullptr;
11. int len = 0;
12. };
13. enum NodeField {
14. NAME = 0,
15. BIRTHDAY = 1,
16. PHONE = 2,
17. SIGN = 3,
18. PASSPORT = 4,
19. POLICE\_NUM = 5,
20. };
21. enum UserTask {
22. CLOSE\_PROGRAM = 0,
23. ADD\_NODE = 1,
24. FIND\_NODE = 2,
25. PRINT\_LIST = 3,
26. SORT\_LIST = 4,
27. };
28. void clean\_input();
29. int get\_user\_task();
30. void close\_program(CarsList\* all\_cars);
31. void add\_node(CarsList\* all\_cars);
32. void find\_node(CarsList\* all\_cars);
33. void print\_list(CarsList\* all\_cars);
34. void sort\_list(CarsList\* all\_cars);