**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

**КУРСОВАЯ РАБОТА**

**по дисциплине «Программирование»**

Тема: Линейные списки

|  |  |  |
| --- | --- | --- |
| Студент гр. 6303 |  | Архипцев Е.Д. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2016

**Аннотация**

В данной работе была создана программа на языке программирования C, которая позволяет работать с набором функций, отвечающих засписок музыкальных композиций. Для функционирования списка были созданы и описаны необходимые функции, позволяющие добавлять, удалять, сортировать и выводить элементы списка и их количество в консоль, также описана структура элемента списка. Помимо этого, была проведена работа над оптимизацией исходного кода программы для ускорения ее быстродействия и оптимального использования памяти и ресурсов клиента. Приведены примеры работы программы, а также полное описание исходного кода.

Оглавление

[Введение 1](#_Toc470016622)

[Функции для работы с двунаправленным линейным списком 2](#_Toc470016623)

[1.1. Функция для создания элемента списка. 2](#_Toc470016624)

[1.2. Функция для создания списка музыкальных композиций. 2](#_Toc470016625)

[1.3. Функция для добавления нового элемента в конец списка. 3](#_Toc470016626)

[1.4. Функция для удаления элементов списка 3](#_Toc470016627)

[1.5. Функция для подсчёта количества элементов списка. 4](#_Toc470016628)

[1.6. Функция для выведения названия композиций. 4](#_Toc470016629)

[1.7. Функция для сортировки списка по названию. 4](#_Toc470016630)

[Заключение 5](#_Toc470016632)

[Приложение А 6](#_Toc470016634)

# Введение

Необходимо создать двунаправленный список музыкальных композиций MusicalComposition и **api** (***a****pplication****p****rogramming****i****nterface - в данном случае набор функций*) для работы со списком.

Структура элемента списка (тип - MusicalComposition)

* name - строка неизвестной длины (гарантируется, что длина не может быть больше 80 символов), название композиции.
* author - строка неизвестной длины (гарантируется, что длина не может быть больше 80 символов), автор композиции/музыкальная группа.
* year - целое число, год создания.

Функции, необходимые для работы программы

* Создание элемента типа MusicalComposition
* Создание двунаправленного списка связанных элементов типа MusicalComposition
* Добавление элемента в конец списка
* Удаление определенного элемента (или элементов) из списка
* Подсчет количества элементов в списке
* Вывод элементов списка
* Сортировка элементов списка

# Функции для работы с двунаправленным линейным списком

## Функция для создания элемента списка.

1. MusicalComposition\* createMusicalComposition(char\* name, char\* author, int year)
2. {
3. MusicalComposition\* ptr = (MusicalComposition\*)malloc(sizeof(MusicalComposition));
4. ptr->name = name;
5. ptr->author = author;
6. ptr->year = year;
7. ptr->next = NULL;
8. ptr->prev = NULL;
9. return ptr;
10. }

Данной функции передают три параметра: название (char\* name), имя автора (char\* author) и год создания композиции (int year); которые записываются в элемент после выделения памяти для структуры MusicalComposition. Функция возвращает указатель на созданный элемент списка.

## Функция для создания списка музыкальных композиций.

1. MusicalComposition\* createMusicalCompositionList(char\*\* array\_names, char\*\* array\_authors, int\* array\_years, int n)
2. {
3. MusicalComposition \*head = createMusicalComposition(array\_names[0], array\_authors[0], array\_years[0]);
4. MusicalComposition \*prev = head;
5. MusicalComposition \*ptr;
6. int i;
7. for (i = 1; i < n; i++)
8. {
9. ptr = createMusicalComposition(array\_names[i], array\_authors[i], array\_years[i]);
10. ptr->prev = prev;
11. prev->next = ptr;
12. prev = ptr;
13. }
14. return head;
15. }

Функция получает количество существующих композиций и указатели на три массива: массив названий (char\*\* array\_names), массив авторов (char\*\* array\_authors) и массив годов созданий (int\* array\_years); и формирует линейный двунаправленный список.

## Функция для добавления нового элемента в конец списка.

1. void push(MusicalComposition\* head, MusicalComposition\* element)
2. {
3. MusicalComposition \*ptr = head;
4. while (ptr->next)
5. ptr = ptr->next;
6. ptr->next = element;
7. element->prev = ptr;
8. }

Данной функции передается указатель на начало линейного списка (MusicalComposition\* head) и элемент, который необходимо добавить (MusicalComposition\* element). Функция перемещается по списку, пока не доходит до последнего элемента, а затем соединяет последний и новый элементы списка посредством указателей.

## Функция для удаления элементов списка

1. void removeEl(MusicalComposition\* head, char\* name\_for\_remove)
2. {
3. MusicalComposition\* ptr = head;
4. while (ptr!= NULL)
5. {
6. if (strcmp(ptr->name, name\_for\_remove) == 0)
7. {
8. if (ptr == head)
9. {
10. \*head = \*head->next;
11. head->prev = NULL;
12. if (head->next != NULL) {
13. head->next->prev = head;
14. }
15. } else if (ptr->next == NULL){
16. ptr->prev->next = NULL;
17. } else{
18. ptr->next->prev = ptr->prev;
19. ptr->prev->next = ptr->next;
20. }
21. }
22. ptr = ptr->next;
23. }
24. }

Функция получает указатель на начало списка (MusicalComposition\* head) и название композиции (char\* name\_for\_remove), которую нужно удалить. Перемещаясь по списку, она сверяет название композиции с переданной строкой. При совпадении значений функция связывает предыдущий и следующий элементы посредством указателей, тем самым удаляя найденный элемент из списка.

## Функция для подсчёта количества элементов списка.

1. int count(MusicalComposition\* head)
2. {
3. MusicalComposition\* ptr = head;
4. int count = 0;
5. while (ptr->next)
6. {
7. ++count;
8. ptr = ptr->next;
9. }
10. ++count;
11. return count;
12. }

Функция получает указатель на первый элемент списка (MusicalComposition\* head), и пробегая весь список, увеличивает счётчик counter на каждом элементе списка.

## Функция для выведения названия композиций.

1. void print\_names(MusicalComposition\* head)
2. {
3. MusicalComposition\* ptr = head;
4. while (ptr)
5. {
6. printf("%s\n", ptr->name);
7. ptr = ptr->next;
8. }
9. }

Данной функции передается указатель на первый элемент списка (MusicalComposition\* head). Перебирая все элементы, функция печатает названия каждой композиции.

## Функция для сортировки списка по названию.

1. void Sort(MusicalComposition\* head)
2. {
3. for (MusicalComposition \*i = head; i; i = i->next) {
4. for (MusicalComposition \*j = head; j; j = j->next) {
5. if (i->name > j->name)
6. {
7. MusicalComposition r = \*i;
8. i->name = j->name;
9. j->name = r.name;
10. i->author = j->author;
11. j->author = r.author;
12. i->year = j->year;
13. j->year = r.year;
14. }
15. }
16. }
17. }

Функция получается указатель на первый элемент массива (MusicalComposition\* head) и, пользуясь методом пузырька, сортирует список композиций по названию, при этом меняя местами не элементы списка, а лишь их содержимое.

# Заключение

В ходе работы был создан двунаправленный линейный список музыкальных композиций MusicalComposition и api для работы со с ним. Также для работы программы были созданы и описаны все необходимые функции и структуры.

# Приложение А

**Исходный код программы**

1. #include <stdlib.h>
2. #include <stdio.h>
3. #include <string.h>
4. typedef struct MusicalComposition
5. {
6. char\* name;
7. char\* author;
8. int year;
9. struct MusicalComposition \*next;
10. struct MusicalComposition \*prev;
11. }MusicalComposition;
12. MusicalComposition\* createMusicalComposition(char\* name, char\* author, int year);
13. MusicalComposition\* createMusicalCompositionList(char\*\* array\_names, char\*\* array\_authors, int\* array\_years, int n);
14. void push(MusicalComposition\* head, MusicalComposition\* element);
15. void removeEl(MusicalComposition\* head, char\* name\_for\_remove);
16. int count(MusicalComposition\* head);
17. void print\_names(MusicalComposition\* head);
18. void Sort(MusicalComposition\* head);
19. int main(){
20. int length, i;
21. scanf("%d\n", &length);
22. char\*\* names = (char\*\*)malloc(sizeof(char\*)\*length);
23. char\*\* authors = (char\*\*)malloc(sizeof(char\*)\*length);
24. int\* years = (int\*)malloc(sizeof(int)\*length);
25. for (i=0; i<length; i++)
26. {
27. char name[80];
28. char author[80];
29. fgets(name, 80, stdin);
30. fgets(author, 80, stdin);
31. fscanf(stdin, "%d\n", &years[i]);
32. (\*strstr(name,"\n"))=0;
33. (\*strstr(author,"\n"))=0;
34. names[i] = (char\*)malloc(sizeof(char\*) \* (strlen(name)+1));
35. authors[i] = (char\*)malloc(sizeof(char\*) \* (strlen(author)+1));
36. strcpy(names[i], name);
37. strcpy(authors[i], author);
38. }
39. MusicalComposition\* head = createMusicalCompositionList(names, authors, years, length);
40. char name\_for\_push[80];
41. char author\_for\_push[80];
42. int year\_for\_push;
43. char name\_for\_remove[80];
44. fgets(name\_for\_push, 80, stdin);
45. fgets(author\_for\_push, 80, stdin);
46. fscanf(stdin, "%d\n", &year\_for\_push);
47. (\*strstr(name\_for\_push, "\n")) = 0;
48. (\*strstr(author\_for\_push, "\n")) = 0;
49. MusicalComposition\* element\_for\_push = createMusicalComposition(name\_for\_push, author\_for\_push, year\_for\_push);
50. fgets(name\_for\_remove, 80, stdin);
51. (\*strstr(name\_for\_remove, "\n")) = 0;
52. printf("%s %s %d\n", head->name, head->author, head->year);
53. int k = count(head);
54. printf("%d\n", k);
55. push(head, element\_for\_push);
56. k = count(head);
57. printf("%d\n", k);
58. removeEl(head, name\_for\_remove);
59. print\_names(head);
60. k = count(head);
61. printf("%d\n", k);
62. Sort(head);
63. print\_names(head);
65. return 0;
66. }
67. MusicalComposition\* createMusicalComposition(char\* name, char\* author, int year)
68. {
69. MusicalComposition\* ptr = (MusicalComposition\*)malloc(sizeof(MusicalComposition));
70. ptr->name = name;
71. ptr->author = author;
72. ptr->year = year;
73. ptr->next = NULL;
74. ptr->prev = NULL;
75. return ptr;
76. }
77. MusicalComposition\* createMusicalCompositionList(char\*\* array\_names, char\*\* array\_authors, int\* array\_years, int n)
78. {
79. MusicalComposition \*head = createMusicalComposition(array\_names[0], array\_authors[0], array\_years[0]);
80. MusicalComposition \*prev = head;
81. MusicalComposition \*ptr;
82. int i;
83. for (i = 1; i < n; i++)
84. {
85. ptr = createMusicalComposition(array\_names[i], array\_authors[i], array\_years[i]);
86. ptr->prev = prev;
87. prev->next = ptr;
88. prev = ptr;
89. }
90. return head;
91. }
92. void push(MusicalComposition\* head, MusicalComposition\* element)
93. {
94. MusicalComposition \*ptr = head;
95. while (ptr->next)
96. ptr = ptr->next;
97. ptr->next = element;
98. element->prev = ptr;
99. }
100. void removeEl(MusicalComposition\* head, char\* name\_for\_remove)
101. {
102. MusicalComposition\* ptr = head;
103. while (ptr!= NULL)
104. {
105. if (strcmp(ptr->name, name\_for\_remove) == 0)
106. {
107. if (ptr == head)
108. {
109. \*head = \*head->next;
110. head->prev = NULL;
111. if (head->next != NULL) {
112. head->next->prev = head;
113. }
114. } else if (ptr->next == NULL){
115. ptr->prev->next = NULL;
116. } else{
117. ptr->next->prev = ptr->prev;
118. ptr->prev->next = ptr->next;
119. }
120. }
121. ptr = ptr->next;
122. }
123. }
124. int count(MusicalComposition\* head)
125. {
126. MusicalComposition\* ptr = head;
127. int count = 0;
128. while (ptr->next)
129. {
130. ++count;
131. ptr = ptr->next;
132. }
133. ++count;
134. return count;
135. }
136. void print\_names(MusicalComposition\* head)
137. {
138. MusicalComposition\* ptr = head;
139. while (ptr)
140. {
141. printf("%s\n", ptr->name);
142. ptr = ptr->next;
143. }
144. }
145. void Sort(MusicalComposition\* head)
146. {
147. for (MusicalComposition \*i = head; i; i = i->next) {
148. for (MusicalComposition \*j = head; j; j = j->next) {
149. if (i->name > j->name)
150. {
151. MusicalComposition r = \*i;
152. i->name = j->name;
153. j->name = r.name;
154. i->author = j->author;
155. j->author = r.author;
156. i->year = j->year;
157. j->year = r.year;
158. }
159. }
160. }
161. }