**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

Курсовая РАБОТА

**по дисциплине «Программирование»**

Тема: Линейные списки

|  |  |  |
| --- | --- | --- |
| Студент гр. 6303 |  | Сапрыга В.В. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2016

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Доброхвалов М.О. | | |
| Группа 6303 | | |
| Тема работы: Линейные списки | | |
|  | | |
| Содержание пояснительной записки:   * Содержание * Введение * Описание функций, необходимых для работы с двунаправленным линейным списком * Примеры работы программы * Заключение * Список использованных источников * Приложение А. Исходный код программы | | |
| Предполагаемый объем пояснительной записки:  Не менее 10 страниц. | | |
| Дата выдачи задания: 02.12.2016 | | |
| Дата сдачи реферата: 29.12.2016 | | |
| Дата защиты реферата: 29.12.2016 | | |
| Студент |  | Сапрыга В.В. |
| Преподаватель |  | Берленко Т.А. |

**Аннотация**

В данной работе была разработана 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 по имени, выводить элементы списка и их количество в консоль, а 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. Приведено полное описание исходного кода.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 5 |
| 1. | Описание функций, необходимых для работы с двунаправленным линейным списком | 6 |
| 1.1. | Созданиеэлемента | 6 |
| 1.2. | Создание двунаправленного списка связанных элементов | 6 |
|  |  | 6 |
| 1.3. | Удаление элементов из списка | 7 |
| 1.4. | Подсчет количества элементов в списке | 8 |
| 1.5. | Вывод элементов списка в консоль | 8 |
|  | Примеры работы программы | 9 |
|  | Заключение | 11 |
|  | Список использованных источников | 12 |
|  | Приложение А. Исходный код программы | 13 |

**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.

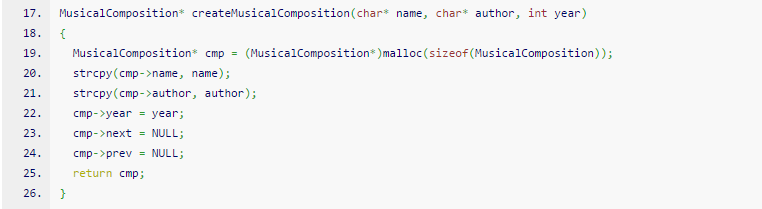
Функции, необходимые для работы программы

* Создание элемента типа MusicalComposition
* Создание двунаправленного списка связанных элементов типа MusicalComposition
* Добавление элемента в начало, середину и конец списка
* Удаление определенного элемента (или элементов) из списка
* Подсчет количества элементов в списке
* Вывод элементов списка в консоль

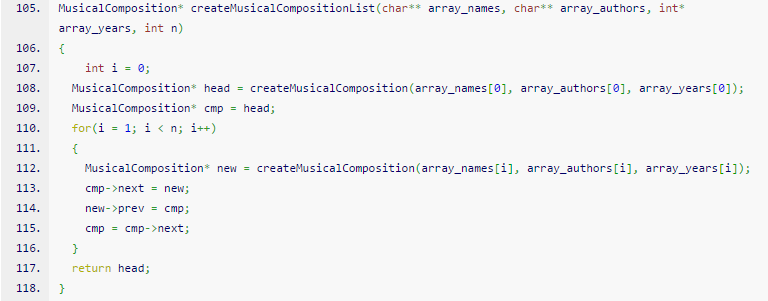
**1. функции для работы с двунаправленным линейным списком**

* 1. **Создание элемента**

Функция принимает в качестве аргументов указатели на название композиции (**char**\* name) и ее автора (**char**\* author), а также год написания (**int** year). Происходит выделение памяти для структуры типа MusicalComposition и заполняются ее переменные.

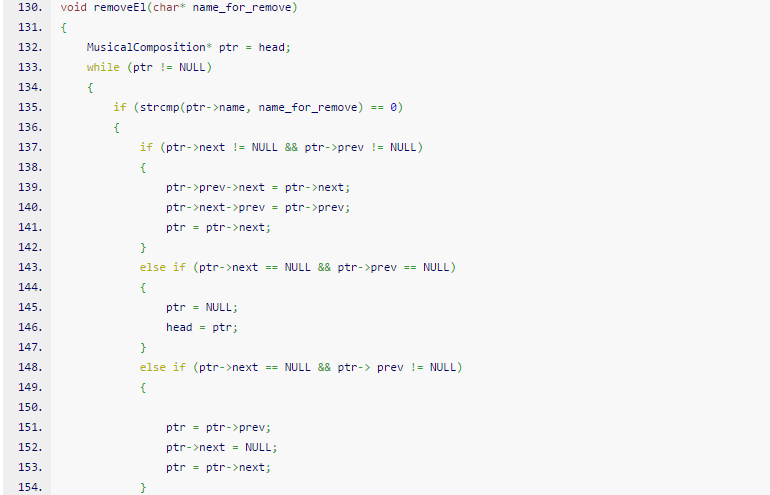


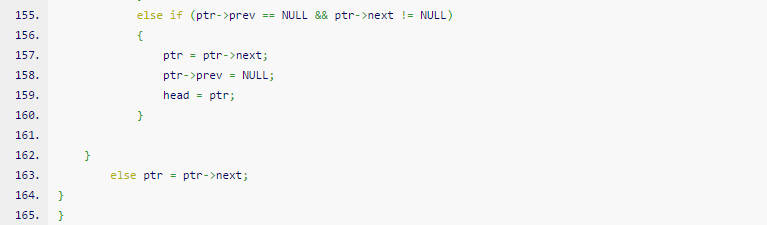
* 1. **Создание двунаправленного списка связанных элементов**

****Функция принимает в качестве аргументов указатели на указатель на массив названий композиций (**char**\*\* array\_names) и их авторов (**char**\*\* array\_authors), указатель на массив лет написания (**int\*** array\_years), размер массива (**int** n). Сначала создается “голова” списка и происходит заполнение ее переменных данными, затем аналогичные операции происходят для всех последующих n-1 элементов списка

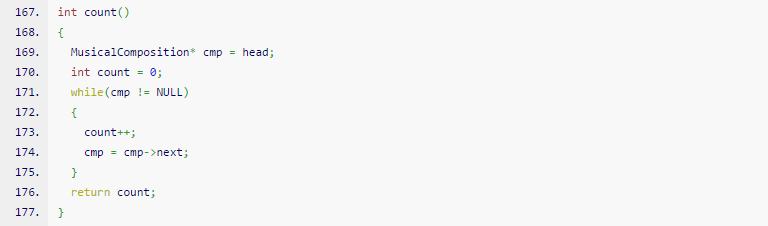
* 1. **Удаление элементов из списка**

Функция принимает в качестве аргумента название композиций (**char**\* name\_for\_remove), которые нужно удалить из списка. Для каждого его элемента происходит сравнение названия композиции с именем произведений, которые нужно удалить.





* 1. **Подсчет количества элементов в списке**



Пока следующий элемент списка существует, происходит увеличение счетчика элементов и перевод на следующий элемент.

* 1. **Вывод элементов списка в консоль**

В функции происходит вывод названия и даты написания композиции в списке, пока следующий элемент существует.

****

**примеры работы программы**

Для демонстрации всех возможностей программы используется следующий формат ввода:

* Количество элементов в списке
* Элементы списка
* Элементы для добавления в список
* Название композиции, которую нужно удалить

И вывода:

* Название, автор и год написания первого элемента списка
* Количество элементов до добавления нового
* Количество элементов после добавления
* Названия всех композиций после удаления
* Количество элементов после удаления

|  |  |  |
| --- | --- | --- |
| № | ВВОд | вЫВОД |
| 1 | 5  The One  Deuce  2012  Heathens  Twenty One Pilots  2015  Sucker For Pain  Imagine Dragons  2016  Hall of Fame  The script  2013  Hear Me Now  Hollywood Undead  2010  Lonely Day  System Of a Down  2007  Hall of fame | The One Deuce 2011 //первый элемент списка  5 //количество элементов до добавления  6 //количество элементов после добавления  The One / /список элементов  Heathens  Sucker For Pain  Hear Me Now  Lonely Day  5 //количество элементов после удаления |
| 2 | 5  The One  Deuce  2012  Heathens  Twenty One Pilots  2015  Sucker For Pain  Imagine Dragons  2016  Hall of Fame  The script  2013  Hear Me Now  Hollywood Undead  2010  Lonely Day  System Of a Down  2007  The One | Heathens Twenty One Pilots 2015  //первый элемент списка  5 //количество элементов до добавления  6 //количество элементов после добавления  The One / /список элементов  Heathens  Sucker For Pain  Hall of Fame  Hear Me Now  Lonely Day  5 //количество элементов после удаления |
| 3 | 5  The One  Deuce  2012  The One  Deuce  2012  The One  Deuce  2012  The One  Deuce  2012  The One  Deuce  2012  The One  Deuce  2012  The One | The One Deuce 2012 //первый элемент списка  5 //количество элементов до добавления  6 //количество элементов после добавления  0 //количество элементов после удаления |

**заключение**

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

**список использованных источников**

1. Язык программирования СИ / Керниган Б., Ритчи Д. СПб.: Издательство "Невский Диалект", 2001. 352 с.
2. UNIX.  Программное окружение / Керниган Б., Пайк Р. СПб.: Символ Плюс, 2003. 416 с.

**приложение А**

**Исходный код программы**

1. #include <stdlib.h>
2. #include <stdio.h>
3. #include <string.h>

6. typedef struct MusicalComposition
7. {
8. char name[80];
9. char author[80];
10. int year;
11. struct MusicalComposition\* next;
12. struct MusicalComposition\* prev;
13. } MusicalComposition;
15. MusicalComposition\* head;
17. MusicalComposition\* createMusicalComposition(char\* name, char\* author, int year)
18. {
19. MusicalComposition\* cmp = (MusicalComposition\*)[malloc](http://www.opengroup.org/onlinepubs/009695399/functions/malloc.html)(sizeof(MusicalComposition));
20. [strcpy](http://www.opengroup.org/onlinepubs/009695399/functions/strcpy.html)(cmp->name, name);
21. [strcpy](http://www.opengroup.org/onlinepubs/009695399/functions/strcpy.html)(cmp->author, author);
22. cmp->year = year;
23. cmp->next = NULL;
24. cmp->prev = NULL;
25. return cmp;
26. }

29. MusicalComposition\* createMusicalCompositionList(char\*\* array\_names, char\*\* array\_authors, int\* array\_years, int n);
31. void push(MusicalComposition\* [**element**](http://ideone.com/L4tOGX));
33. void removeEl(char\* name\_for\_remove);
35. int count();
37. void print\_names();
39. int main()
40. {
41. int k = 0;
42. int i = 0;
43. int length;
44. [scanf](http://www.opengroup.org/onlinepubs/009695399/functions/scanf.html)("%d**\n**", &length);
46. char\*\* names = (char\*\*)[malloc](http://www.opengroup.org/onlinepubs/009695399/functions/malloc.html)(sizeof(char\*) \* length);
47. char\*\* authors = (char\*\*)[malloc](http://www.opengroup.org/onlinepubs/009695399/functions/malloc.html)(sizeof(char\*) \* length);
48. int\* years = (int\*)[malloc](http://www.opengroup.org/onlinepubs/009695399/functions/malloc.html)(sizeof(int) \* length);
50. for(i = 0; i < length; i++) {
51. char name[80];
52. char author[80];
54. [fgets](http://www.opengroup.org/onlinepubs/009695399/functions/fgets.html)(name, 80, stdin);
55. [fgets](http://www.opengroup.org/onlinepubs/009695399/functions/fgets.html)(author, 80, stdin);
56. [fscanf](http://www.opengroup.org/onlinepubs/009695399/functions/fscanf.html)(stdin, "%d**\n**", &years[i]);
58. (\*[strstr](http://www.opengroup.org/onlinepubs/009695399/functions/strstr.html)(name, "**\n**")) = 0;
59. (\*[strstr](http://www.opengroup.org/onlinepubs/009695399/functions/strstr.html)(author, "**\n**")) = 0;
61. names[i] = (char\*)[malloc](http://www.opengroup.org/onlinepubs/009695399/functions/malloc.html)(sizeof(char\*) \* ([strlen](http://www.opengroup.org/onlinepubs/009695399/functions/strlen.html)(name) + 1));
62. authors[i] = (char\*)[malloc](http://www.opengroup.org/onlinepubs/009695399/functions/malloc.html)(sizeof(char\*) \* ([strlen](http://www.opengroup.org/onlinepubs/009695399/functions/strlen.html)(author) + 1));
64. [strcpy](http://www.opengroup.org/onlinepubs/009695399/functions/strcpy.html)(names[i], name);
65. [strcpy](http://www.opengroup.org/onlinepubs/009695399/functions/strcpy.html)(authors[i], author);
66. }
67. head = createMusicalCompositionList(names, authors, years, length);
68. char name\_for\_push[80];
69. char author\_for\_push[80];
70. int year\_for\_push;
72. char name\_for\_remove[80];
74. [fgets](http://www.opengroup.org/onlinepubs/009695399/functions/fgets.html)(name\_for\_push, 80, stdin);
75. [fgets](http://www.opengroup.org/onlinepubs/009695399/functions/fgets.html)(author\_for\_push, 80, stdin);
76. [fscanf](http://www.opengroup.org/onlinepubs/009695399/functions/fscanf.html)(stdin, "%d**\n**", &year\_for\_push);
77. (\*[strstr](http://www.opengroup.org/onlinepubs/009695399/functions/strstr.html)(name\_for\_push, "**\n**")) = 0;
78. (\*[strstr](http://www.opengroup.org/onlinepubs/009695399/functions/strstr.html)(author\_for\_push, "**\n**")) = 0;
80. MusicalComposition\* element\_for\_push = createMusicalComposition(name\_for\_push, author\_for\_push, year\_for\_push);
82. [fgets](http://www.opengroup.org/onlinepubs/009695399/functions/fgets.html)(name\_for\_remove, 80, stdin);
83. (\*[strstr](http://www.opengroup.org/onlinepubs/009695399/functions/strstr.html)(name\_for\_remove, "**\n**")) = 0;
85. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("%s %s %d**\n**", head->name, head->author, head->year);
86. k = count(head);
88. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("%d**\n**", k);
89. push(element\_for\_push);
91. k = count();
92. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("%d**\n**", k);
94. removeEl(name\_for\_remove);
95. print\_names();
97. k = count();
98. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("%d**\n**", k);
100. return 0;
101. }


105. MusicalComposition\* createMusicalCompositionList(char\*\* array\_names, char\*\* array\_authors, int\* array\_years, int n)
106. {
107. int i = 0;
108. MusicalComposition\* head = createMusicalComposition(array\_names[0], array\_authors[0], array\_years[0]);
109. MusicalComposition\* cmp = head;
110. for(i = 1; i < n; i++)
111. {
112. MusicalComposition\* new = createMusicalComposition(array\_names[i], array\_authors[i], array\_years[i]);
113. cmp->next = new;
114. new->prev = cmp;
115. cmp = cmp->next;
116. }
117. return head;
118. }
120. void push(MusicalComposition\* [**element**](http://ideone.com/L4tOGX))
121. {
122. MusicalComposition\* cmp = head;
123. MusicalComposition\* pushed = createMusicalComposition(element->name, [**element**](http://ideone.com/L4tOGX)->author, element->year);
124. while(cmp->next != NULL)
125. cmp = cmp->next;
126. cmp->next = pushed;
127. pushed->prev = cmp;
128. }
130. void removeEl(char\* name\_for\_remove)
131. {
132. MusicalComposition\* ptr = head;
133. while (ptr != NULL)
134. {
135. if ([strcmp](http://www.opengroup.org/onlinepubs/009695399/functions/strcmp.html)(ptr->name, name\_for\_remove) == 0)
136. {
137. if (ptr->next != NULL && ptr->prev != NULL)
138. {
139. ptr->prev->next = ptr->next;
140. ptr->next->prev = ptr->prev;
141. ptr = ptr->next;
142. }
143. else if (ptr->next == NULL && ptr->prev == NULL)
144. {
145. ptr = NULL;
146. head = ptr;
147. }
148. else if (ptr->next == NULL && ptr-> prev != NULL)
149. {
151. ptr = ptr->prev;
152. ptr->next = NULL;
153. ptr = ptr->next;
154. }
155. else if (ptr->prev == NULL && ptr->next != NULL)
156. {
157. ptr = ptr->next;
158. ptr->prev = NULL;
159. head = ptr;
160. }
162. }
163. else ptr = ptr->next;
164. }
165. }
167. int count()
168. {
169. MusicalComposition\* cmp = head;
170. int count = 0;
171. while(cmp != NULL)
172. {
173. count++;
174. cmp = cmp->next;
175. }
176. return count;
177. }
179. void print\_names()
180. {
181. MusicalComposition\* cmp = head;
182. while(cmp != NULL)
183. {
184. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("%s**\n**", cmp->name);
185. cmp = cmp->next;
186. }