**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 |  | Доброхвалов М.О. |
| 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: 21.12.2016 | | |
| Дата защиты реферата: 21.12.2016 | | |
| Студент |  | Доброхвалов М.О. |
| 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 | 4 |
| 1. | Описание функций, необходимых для работы с двунаправленным линейным списком | 6 |
| 1.1. | Созданиеэлемента | 6 |
| 1.2. | Создание двунаправленного списка связанных элементов | 6 |
| 1.3. | Добавление элементов в начало, середину и конец списка | 6 |
| 1.4. | Удаление элементов из списка | 7 |
| 1.5. | Подсчет количества элементов в списке | 8 |
| 1.6. | 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**

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

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 структура типа MusicalComposition и заполняются ее переменные.

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

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

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. **Добавление элемента в начало, середину и конец списка**

1. void push\_course(MusicalComposition \*element\_top, MusicalComposition \*element\_middle, MusicalComposition \*element\_end)
2. {
3. MusicalComposition \*tmp = head;
4. element\_top->next = tmp;
5. tmp->previous = element\_top;
6. head = element\_top;
7. int i=1,n=Count();
8. while (i < n/2)
9. {
10. tmp = tmp->next;
11. i++;
12. }
13. element\_middle->next = tmp->next;
14. element\_middle->previous = tmp;
15. tmp->next->previous = element\_middle;
16. tmp->next = element\_middle;
17. while (tmp->next != NULL)
18. {
19. tmp = tmp->next;
20. }
21. tmp->next = element\_end;
22. element\_end->next = NULL;
23. element\_end->previous = tmp;
24. }

Функция принимает в качестве аргументов указатель на “голову” списка (MusicalComposition\* head) и элементы (MusicalComposition \*element\_top, MusicalComposition \*element\_middle, MusicalComposition \*element\_end), который нужно добавить в начало, середину и конец списка соответственно. В первую очередь добавляем элемент(element\_top) в начало списка и меняем указатель на голову. Затем доходим до середины списка с помощью функции for и добавляем элемент(element\_middle) в середину списка. Далее, используя цикл while доходим до конца списка и вписываем элемент(element\_end).

* 1. **Удаление элементов из списка**

1. void removeEl(char\* name\_for\_remove)
2. {
3. MusicalComposition \*tmp = head;
4. while(tmp != NULL)
5. {
6. if (strcmp(tmp->name, name\_for\_remove) == 0)
7. {
8. if ( (tmp->previous == NULL) && (tmp->next != NULL) )
9. {
10. tmp = tmp->next;
11. tmp->previous = NULL;
12. head = tmp;
13. }
14. else
15. {
16. if ( (tmp->previous != NULL) && (tmp->next == NULL) )
17. {
18. tmp = tmp->previous;
19. tmp->next = NULL;
20. tmp = tmp->next;
21. }
22. else
23. {
24. if ( (tmp->previous != NULL) && (tmp->next != NULL) )
25. {
26. tmp->next->previous = tmp->previous;
27. tmp->previous->next = tmp->next;
28. tmp = tmp->next;
29. }
30. else
31. {
32. if ( (tmp->next == NULL) && (tmp->previous == NULL) )
33. {
34. tmp = NULL;
35. head = tmp;
36. }
37. }
38. }
39. }
40. }
41. else
42. {
43. tmp = tmp->next;
44. }
45. }
46. }

Функция принимает в качестве аргументов указатель на “голову” списка (MusicalComposition\* head) и 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**

1. int Count()
2. {
3. MusicalComposition \*tmp = head;
4. int counter = 0;
5. while (tmp != NULL)
6. {
7. counter++;
8. tmp = tmp->next;
9. }
10. return counter;
11. }

Функция принимает в качестве аргументов указатель на “голову” списка (MusicalComposition\* head). Далее происходит подсчет элементов в списке, пока следующий элемент существует

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

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

Функция принимает в качестве аргументов указатель на “голову” списка (MusicalComposition\* head). Далее 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 | 4  Skin Deep  Memory of a Melody  2011  Little Pills  Tech N9ne  2009  Monster  Skillet  2009  In the End  Linkin Park  2000  High Hopes  Pink Floyd  1994  Gone Forever  Thirty Seconds to Mars  2005  Country Girl  Black Sabbath  1981  Skin Deep | Skin Deep Memory of a Melody 2011 //первый элемент списка  4 //количество элементов до добавления  7 //количество элементов после добавления  High Hopes / /список элементов  Little Pills  Gone Forever  Monster  In the End  Country girl  6 //количество элементов после удаления |
| 2 | 4  Skin Deep  Memory of a Melody  2011  Little Pills  Tech N9ne  2009  Monster  Skillet  2009  In the End  Linkin Park  2000  High Hopes  Pink Floyd  1994  Gone Forever  Thirty Seconds to Mars  2005  Country Girl  Black Sabbath  1981  High Hopes | Skin Deep Memory of a Melody 2011 //первый элемент списка  4 //количество элементов до добавления  7 //количество элементов после добавления  Skin Deep / /список элементов  Little Pills  Gone Forever  Monster  In the End  Country girl  6 //количество элементов после удаления |
| 3 | 4  Skin Deep  Memory of a Melody  2011  Little Pills  Tech N9ne  2009  Monster  Skillet  2009  In the End  Linkin Park  2000  High Hopes  Pink Floyd  1994  Gone Forever  Thirty Seconds to Mars  2005  Country Girl  Black Sabbath  1981  Country Girl | Skin Deep Memory of a Melody 2011 //первый элемент списка  4 //количество элементов до добавления  7 //количество элементов после добавления  High Hopes/ /список элементов  Skin Deep  Little Pills  Gone Forever  Monster  In the End  6 //количество элементов после удаления |
| 4 | 4  Monster  Skillet  2009  Monster  Skillet  2009  Monster  Skillet  2009  Monster  Skillet  2009  Monster  Skillet  2009  Monster  Skillet  2009  Monster  Skillet  2009  Monster | Monster Skillet 2009 //первый элемент списка  4 //количество элементов до добавления  7 //количество элементов после добавления  Empty list  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>
4. typedef struct MusicalComposition
5. {
6. char \*name;
7. char \*author;
8. int year;
9. struct MusicalComposition \*next;
10. struct MusicalComposition \*previous;
11. }MusicalComposition;
12. MusicalComposition \*head;
13. MusicalComposition \*createMusicalComposition(char \*name, char \*author, int year)
14. {
15. MusicalComposition \*composition = (MusicalComposition\*)malloc(sizeof(MusicalComposition));
16. composition->name = name;
17. composition->author = author;
18. composition->year = year;
19. composition->next = NULL;
20. composition->previous = NULL;
21. return composition;
22. }
23. MusicalComposition \*createMusicalCompositionList(char \*\*array\_names, char \*\*array\_authors, int \*array\_years, int n)
24. {
25. head = createMusicalComposition(array\_names[0], array\_authors[0], array\_years[0]);
26. MusicalComposition \*tmp = head;
27. for (int i = 1; i < n; i++)
28. {
29. tmp->next = createMusicalComposition(array\_names[i], array\_authors[i], array\_years[i]);
30. tmp->next->previous = tmp;
31. tmp = tmp->next;
32. }
33. return head;
34. }
35. int Count()
36. {
37. MusicalComposition \*tmp = head;
38. int counter = 0;
39. while (tmp != NULL)
40. {
41. counter++;
42. tmp = tmp->next;
43. }
44. return counter;
45. }
46. void push\_course(MusicalComposition \*element\_top, MusicalComposition \*element\_middle, MusicalComposition \*element\_end)
47. {
48. MusicalComposition \*tmp = head;
49. element\_top->next = tmp;
50. tmp->previous = element\_top;
51. head = element\_top;
52. int i = 1, n = Count();
53. while (i < n / 2)
54. {
55. tmp = tmp->next;
56. i++;
57. }
58. element\_middle->next = tmp->next;
59. element\_middle->previous = tmp;
60. tmp->next->previous = element\_middle;
61. tmp->next = element\_middle;
62. while (tmp->next != NULL)
63. {
64. tmp = tmp->next;
65. }
66. tmp->next = element\_end;
67. element\_end->next = NULL;
68. element\_end->previous = tmp;
69. }
70. void removeEl(char\* name\_for\_remove)
71. {
72. MusicalComposition \*tmp = head;
73. while(tmp != NULL)
74. {
75. if (strcmp(tmp->name, name\_for\_remove) == 0)
76. {
77. if ( (tmp->previous == NULL) && (tmp->next != NULL) )
78. {
79. tmp = tmp->next;
80. tmp->previous = NULL;
81. head = tmp;
82. }
83. else
84. {
85. if ( (tmp->previous != NULL) && (tmp->next == NULL) )
86. {
87. tmp = tmp->previous;
88. tmp->next = NULL;
89. tmp = tmp->next;
90. }
91. else
92. {
93. if ( (tmp->previous != NULL) && (tmp->next != NULL) )
94. {
95. tmp->next->previous = tmp->previous;
96. tmp->previous->next = tmp->next;
97. tmp = tmp->next;
98. }
99. else
100. {
101. if ( (tmp->next == NULL) && (tmp->previous == NULL) )
102. {
103. tmp = NULL;
104. head = tmp;
105. }
106. }
107. }
108. }
109. }
110. else
111. {
112. tmp = tmp->next;
113. }
114. }
115. }
116. void print\_names()
117. {
118. MusicalComposition \*tmp = head;
119. if (tmp == NULL)
120. {
121. printf("Empty list\n");
122. }
123. else
124. {
125. while (tmp != NULL)
126. {
127. printf("%s\n", tmp->name);
128. tmp = tmp->next;
129. }
130. }
131. }
132. int main()
133. {
134. int length;
135. scanf("%d\n", &length);
136. char \*\*names = (char\*\*)malloc(sizeof(char\*)\*length);
137. char \*\*authors = (char\*\*)malloc(sizeof(char\*)\*length);
138. int \*years = (int\*)malloc(sizeof(int)\*length);
139. for (int i = 0; i < length; i++)
140. {
141. char name[80];
142. char author[80];
143. fgets(name, 80, stdin);
144. fgets(author, 80, stdin);
145. fscanf(stdin, "%d\n", &years[i]);
146. (\*strstr(name, "\n")) = 0;
147. (\*strstr(author, "\n")) = 0;
148. names[i] = (char\*)malloc(sizeof(char\*)\*(strlen(name) + 1));
149. authors[i] = (char\*)malloc(sizeof(char\*)\*(strlen(author) + 1));
150. strcpy(names[i], name);
151. strcpy(authors[i], author);
152. }
153. head = createMusicalCompositionList(names, authors, years, length);
154. char name\_for\_push\_top[80];
155. char author\_for\_push\_top[80];
156. int year\_for\_push\_top;
157. fgets(name\_for\_push\_top, 80, stdin);
158. fgets(author\_for\_push\_top, 80, stdin);
159. fscanf(stdin, "%d\n", &year\_for\_push\_top);
160. (\*strstr(name\_for\_push\_top, "\n")) = 0;
161. (\*strstr(author\_for\_push\_top, "\n")) = 0;
162. MusicalComposition \*element\_for\_push\_top = createMusicalComposition(name\_for\_push\_top, author\_for\_push\_top, year\_for\_push\_top);
163. char name\_for\_push\_middle[80];
164. char author\_for\_push\_middle[80];
165. int year\_for\_push\_middle;
166. fgets(name\_for\_push\_middle, 80, stdin);
167. fgets(author\_for\_push\_middle, 80, stdin);
168. fscanf(stdin, "%d\n", &year\_for\_push\_middle);
169. (\*strstr(name\_for\_push\_middle, "\n")) = 0;
170. (\*strstr(author\_for\_push\_middle, "\n")) = 0;
171. MusicalComposition \* element\_for\_push\_middle = createMusicalComposition(name\_for\_push\_middle, author\_for\_push\_middle, year\_for\_push\_middle);
172. char name\_for\_push\_end[80];
173. char author\_for\_push\_end[80];
174. int year\_for\_push\_end;
175. fgets(name\_for\_push\_end, 80, stdin);
176. fgets(author\_for\_push\_end, 80, stdin);
177. fscanf(stdin, "%d\n", &year\_for\_push\_end);
178. (\*strstr(name\_for\_push\_end, "\n")) = 0;
179. (\*strstr(author\_for\_push\_end, "\n")) = 0;
180. MusicalComposition \*element\_for\_push\_end = createMusicalComposition(name\_for\_push\_end, author\_for\_push\_end, year\_for\_push\_end);
181. char name\_for\_remove[80];
182. fgets(name\_for\_remove, 80, stdin);
183. (\*strstr(name\_for\_remove, "\n")) = 0;
184. printf("%s %s %d\n", head->name, head->author, head->year);
185. int k = Count();
186. printf("%d\n", k);
187. push\_course(element\_for\_push\_top, element\_for\_push\_middle, element\_for\_push\_end);
188. k = Count();
189. printf("%d\n", k);
190. removeEl(name\_for\_remove);
191. print\_names();
192. k = Count();
193. printf("%d\n", k);
194. return 0;
195. }