**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 |  | Ильяшук Д.И. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2016

**ЗАДАНИЕ на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Ильяшук Д.И. | | |
| Группа 6303 | | |
| Тема работы: Линейные списки | | |
|  | | |
| Содержание пояснительной записки:   * Содержание * Введение * Описание функций, необходимых для работы с двунаправленным линейным списком * Описание функции перестановки элементов * 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 |  | Ильяшук Д.И. |
| 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, сортировать и выводить элементы списка и их количество в консоль, а также описана структура элемента списка. 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 |
| 1.3. | Добавление элемента в конец списка | 6 |
| 1.4. | Удаление элементов из списка | 7 |
| 1.5. | Подсчет количества элементов в списке | 7 |
| 1.6. | Вывод элементов списка в консоль | 7 |
| 2. | Функция перестановки элементов | 8 |
| 2.1. | Перестановка местами 1 и n, 2 и n-1 и т.д. элементов списка | 8 |
| 2.2. | Изменения в функции main | 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 конец списка
* Удаление определенного элемента (или элементов) из списка
* Подсчет количества элементов в списке
* Вывод элементов списка в консоль
* Перестановка элементов (Поменять местами 1й и nй, 2й и n-1 и т.д. элементы списка)

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

* 1. **Создание элемента**

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

Функция принимает в качестве аргументов указатели на название композиции (char\* name), автора (char\* author) и год написания (int year). Происходит выделение памяти для структуры типа MusicalComposition и заполнение её переменных.

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

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

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

* 1. **Добавление элемента в конец списка**

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

Функция принимает в качестве аргументов указатель на «голову» списка (MusicalComposition\* head) и указатель на элемент (MusicalComposition\* element), который нужно добавить в конец списка. Сначала происходит переход к последнему элементу списка, а затем новый элемент вставляется после текущего.

* 1. **Удаление элементов из списка**

1. void removeEl(MusicalComposition\* head, char\* name\_for\_remove)
2. {
3. MusicalComposition\* current = head;
4. while(current)
5. {
6. if(strcmp(current->name, name\_for\_remove) == 0)
7. {
8. if(current->next != NULL && current->prev != NULL)
9. {
10. current->prev->next = current->next;
11. current->next->prev = current->prev;
12. }
13. else if(current->next == NULL)
14. {
15. current->prev->next = NULL;
16. }
17. else if(current == head)
18. {
19. \*head = \*head->next;
20. head->prev = NULL;
21. if(head->next != NULL)
22. head->next->prev = head;
23. current = head;
24. }
25. }
26. current = current->next;
27. }
28. }

Функция принимает в качестве аргументов указатель на «голову» списка (MusicalComposition\* head) и название композиций (**char**\* name\_for\_remove), которые нужно удалить. Для каждого элемента списка происходит сравнение названия композиции с именем произведений, которые нужно удалить, и, если они совпадают, происходит удаление элемента.

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

1. int count(MusicalComposition\* head)
2. {
3. MusicalComposition\* current = head;
4. int count = 0;
5. while (current)
6. {
7. ++count;
8. current = current->next;
9. }
10. return count;
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(MusicalComposition\* head)
2. {
3. MusicalComposition\* current = head;
4. while (current)
5. {
6. printf("%s\n", current->name);
7. current = current->next;
8. }
9. }

Функция принимает в качестве аргументов указатель на «голову» списка (MusicalComposition\* head). Далее происходит вывод названия композиции в списке, пока следующий элемент существует.

1. **функция перестановки элементов**
   1. **Перестановка местами 1 и n, 2 и n-1 и т.д. элементов списка**
2. MusicalComposition\* swap(MusicalComposition\* head)
3. {
4. MusicalComposition\* current = head;
5. MusicalComposition\* buffer = current;
6. while (current)
7. {
8. buffer = current->next;
9. current->next = current->prev;
10. current->prev = buffer;
11. buffer = current;
12. current = current->prev;
13. }
14. return buffer;
15. }

Функция принимает в качестве аргумента указатель на «голову» списка (MusicalComposition\* head). Затем последовательно у каждого элемента меняются местами указатели на следующий и предыдущий элементы. Функция возвращает указатель на последний измененный элемент, который должен стать новой «головой» перевернутого списка.

* 1. **Изменения в функции main**

В функцию main был добавлен вызов функции перестановки и дополнительный вывод названий музыкальных композиций на экран до перестановки.

**примеры работы программы**

Для демонстрации всех возможностей программы используется следующий формат ввода:

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

И вывода:

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

|  |  |  |
| --- | --- | --- |
| № | ВВОд | вЫВОД |
| 1 | 7  Fields of Gold  Sting  1993  Summertime  Bon Jovi  2007  Time of Dying  Three Days Grace  2006  Been to Hell  Hollywood Undead  2011  Engel  Rammstein  1997  Hells Bells  AC/DC  1993  Coming Undone  Korn  2005  Sonne  Rammstein  2001  Fields of Gold | Fields of Gold Sting 1993 //первая композиция  7 //количество элементов до добавления  8 //количество элементов после добавления  Summertime //список композиций до перестановки  Time of Dying  Been to Hell  Engel  Hells Bells  Coming Undone  Sonne  7 //количество элементов после удаления  Sonne //список композиций после перестановки  Coming Undone  Hells Bells  Engel  Been to Hell  Time of Dying  Summertime |
| 2 | 7  Fields of Gold  Sting  1993  Summertime  Bon Jovi  2007  Time of Dying  Three Days Grace  2006  Been to Hell  Hollywood Undead  2011  Engel  Rammstein  1997  Hells Bells  AC/DC  1993  Coming Undone  Korn  2005  Sonne  Rammstein  2001  Sonne | Fields of Gold Sting 1993 //первая композиция  7 //количество элементов до добавления  8 //количество элементов после добавления  Fields of Gold //список композиций до перестановки  Summertime  Time of Dying  Been to Hell  Engel  Hells Bells  Coming Undone  7 //количество элементов после удаления  Coming Undone //список композиций после перестановки  Hells Bells  Engel  Been to Hell  Time of Dying  Summertime  Fields of Gold |
| 3 | 7  Fields of Gold  Sting  1993  Summertime  Bon Jovi  2007  Time of Dying  Three Days Grace  2006  Hells Bells  AC/DC  1993  Hells Bells  AC/DC  1993  Hells Bells  AC/DC  1993  Coming Undone  Korn  2005  Sonne  Rammstein  2001  Hells Bells | Fields of Gold Sting 1993 //первая композиция  7 //количество элементов до добавления  8 //количество элементов после добавления  Fields of Gold //список композиций до перестановки  Summertime  Time of Dying  Coming Undone  Sonne  5 //количество элементов после удаления  Sonne //список композиций после перестановки  Coming Undone  Time of Dying  Summertime  Fields of Gold |
| 4 | 7  Fields of Gold  Sting  1993  Summertime  Bon Jovi  2007  Time of Dying  Three Days Grace  2006  Hells Bells  AC/DC  1993  Higthway to Hell  AC/DC  1979  Rock or Bust  AC/DC  2014  Coming Undone  Korn  2005  Sonne  Rammstein  2001  name | Fields of Gold Sting 1993 //первая композиция  7 //количество элементов до добавления  8 //количество элементов после добавления  Fields of Gold //список композиций до перестановки  Summertime  Time of Dying  Hells Bells  Higthway to Hell  Rock or Bust  Coming Undone  Sonne  8 //количество элементов после удаления  Sonne //список композиций после перестановки  Coming Undone  Rock or Bust  Higthway to Hell  Hells Bells  Time of Dying  Summertime  Fields of Gold |

**заключение**

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. Язык программирования C. Лекции и упражнения/ Стивен Прата. СПб.: Издательство «Вильямс», 2015. 928 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 \*prev;
11. }MusicalComposition;
12. MusicalComposition\* createMusicalComposition(char\* name, char\* author, int year)
13. {
14. MusicalComposition\* MusCom = (MusicalComposition\*)malloc(sizeof(MusicalComposition));
15. MusCom->name = name;
16. MusCom->author = author;
17. MusCom->year = year;
18. MusCom->next = NULL;
19. MusCom->prev = NULL;
20. return MusCom;
21. }
22. MusicalComposition\* createMusicalCompositionList(char\*\* array\_names, char\*\* array\_authors, int\* array\_years, int n)
23. {
24. MusicalComposition \*head = createMusicalComposition(array\_names[0], array\_authors[0], array\_years[0]);
25. MusicalComposition \*prev = head;
26. MusicalComposition\* current;
27. int i;
28. for(i = 1; i < n; i++)
29. {
30. current = createMusicalComposition(array\_names[i], array\_authors[i], array\_years[i]);
31. current->prev = prev;
32. prev->next = current;
33. prev = current;
34. }
35. return head;
36. }
37. void push(MusicalComposition\* head, MusicalComposition\* element)
38. {
39. MusicalComposition\* current = head;
40. while(current->next)
41. current = current->next;
42. current->next = element;
43. element->prev = current;
44. }
45. void removeEl(MusicalComposition\* head, char\* name\_for\_remove)
46. {
47. MusicalComposition\* current = head;
48. while(current)
49. {
50. if(strcmp(current->name, name\_for\_remove) == 0)
51. {
52. if(current->next != NULL && current->prev != NULL)
53. {
54. current->prev->next = current->next;
55. current->next->prev = current->prev;
56. }
57. else if(current->next == NULL)
58. {
59. current->prev->next = NULL;
60. }
61. else if(current == head)
62. {
63. \*head = \*head->next;
64. head->prev = NULL;
65. if(head->next != NULL)
66. head->next->prev = head;
67. current = head;
68. }
69. }
70. current = current->next;
71. }
72. }
73. int count(MusicalComposition\* head)
74. {
75. MusicalComposition\* current = head;
76. int count = 0;
77. while (current->next)
78. {
79. ++count;
80. current = current->next;
81. }
82. ++count;
83. return count;
84. }
85. void print\_names(MusicalComposition\* head)
86. {
87. MusicalComposition\* current = head;
88. while (current->next)
89. {
90. printf("%s\n", current->name);
91. current = current->next;
92. }
93. printf("%s\n", current->name);
94. }
95. MusicalComposition\* swap(MusicalComposition\* head)
96. {
97. MusicalComposition\* current = head;
98. MusicalComposition\* buffer = current;
99. while (current)
100. {
101. buffer = current->next;
102. current->next = current->prev;
103. current->prev = buffer;
104. buffer = current;
105. current = current->prev;
106. }
107. return buffer;
108. }
109. int main(){
110. int length, i;
111. scanf("%d\n", &length);
112. char\*\* names = (char\*\*)malloc(sizeof(char\*)\*length);
113. char\*\* authors = (char\*\*)malloc(sizeof(char\*)\*length);
114. int\* years = (int\*)malloc(sizeof(int)\*length);
115. for (i=0; i<length; i++)
116. {
117. char name[80];
118. char author[80];
119. fgets(name, 80, stdin);
120. fgets(author, 80, stdin);
121. fscanf(stdin, "%d\n", &years[i]);
122. (\*strstr(name,"\n"))=0;
123. (\*strstr(author,"\n"))=0;
124. names[i] = (char\*)malloc(sizeof(char\*) \* (strlen(name)+1));
125. authors[i] = (char\*)malloc(sizeof(char\*) \* (strlen(author)+1));
126. strcpy(names[i], name);
127. strcpy(authors[i], author);
128. }
129. MusicalComposition\* head = createMusicalCompositionList(names, authors, years, length);
130. char name\_for\_push[80];
131. char author\_for\_push[80];
132. int year\_for\_push;
133. char name\_for\_remove[80];
134. fgets(name\_for\_push, 80, stdin);
135. fgets(author\_for\_push, 80, stdin);
136. fscanf(stdin, "%d\n", &year\_for\_push);
137. (\*strstr(name\_for\_push,"\n"))=0;
138. (\*strstr(author\_for\_push,"\n"))=0;
139. MusicalComposition\* element\_for\_push = createMusicalComposition(name\_for\_push, author\_for\_push, year\_for\_push);
140. fgets(name\_for\_remove, 80, stdin);
141. (\*strstr(name\_for\_remove,"\n"))=0;
142. printf("%s %s %d\n", head->name, head->author, head->year);
143. int k = count(head);
144. printf("%d\n", k);
145. push(head, element\_for\_push);
146. k = count(head);
147. printf("%d\n", k);
148. removeEl(head, name\_for\_remove);
149. print\_names(head);
150. k = count(head);
151. printf("%d\n", k);
152. printf("\n");
153. head = swap(head);
154. print\_names(head);
155. return 0;
156. }