**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**КУРСОВАЯ РАБОТА**

**по дисциплине «Программирование»**

**Тема: Линейные списки**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6303 |  | Шабалин М.В. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2016

**ЗАДАНИЕ**

**НА КУРСОВУЮ РАБОТУ**

|  |  |  |
| --- | --- | --- |
| Студент Шабалин М.В. | | |
| Группа 6303 | | |
| Тема работы: Линейные списки | | |
|  | | |
| Содержание пояснительной записки:   * Содержание * Введение * Описание функций, необходимых для работы с двунаправленным линейным списком * Описание функций сортировки списка * Примеры работы программы * Заключение * Список использованных источников * Приложение А. Исходный код программы | | |
| Предполагаемый объем пояснительной записки:  Не менее 10 страниц. | | |
| Дата выдачи задания: | | |
| Дата сдачи реферата: | | |
| Дата защиты реферата: | | |
| Студент |  | Шабалин М.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, сортировать и 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 |
| 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 | 8 |
| 2. | Описание функции сортировки списка | 8 |
| 3. | Примеры работы программы | 10 |
|  | Заключение | 12 |
|  | Список использованных источников | 13 |
|  | Приложение А. Исходный код программы | 14 |
|  |  |  |
|  |  |  |

**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
* Сортировка первой половины элементов в списке по убыванию, а второй по возрастанию

**1. ФУНКЦИИ ДЛЯ РАБОТЫ С ДВУНАПРАВЛЕННЫМ ЛИНЕЙНЫМ СПИСКОМ**

* 1. **Создание элемента**

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

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.

**1.2. Создание двунаправленного списка связанных элементов**

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

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 конец списка**

1. **void** **push**(MusicalComposition\* head, MusicalComposition\* element) {
2. MusicalComposition \*tmp = (MusicalComposition\*)malloc(sizeof(MusicalComposition));
3. tmp = head;
4. while (tmp -> next != NULL) {
5. tmp = tmp -> next;
6. }
7. tmp->next = element;
8. element->prev = tmp;
9. }

Функция принимает в качестве аргументов указатель на “голову” списка (MusicalComposition\* head) и элемент (MusicalComposition\* element), который нужно добавить в конец списка. В первую очередь происходит поиск последнего элемента списка, далее новый элемент вставляется после текущего.

* 1. **Удаление элементов из списка**

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

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

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. MusicalComposition \*tmp = (MusicalComposition\*)malloc(sizeof(MusicalComposition));
3. tmp = head;
4. while (tmp -> next != NULL) {
5. printf("%s %d\n", tmp -> name, tmp->year);
6. tmp = tmp->next;
7. }
8. printf("%s %d\n", tmp -> name, tmp->year);
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.

1. **ФУНКЦИЯ СОРТИРОВКИ СПИСКА**

**Сортировка элементов списка по году написания композиции**

1. void sort(MusicalComposition\* head)
2. {
3. int cnt = count(head);
4. int cen = cnt/2;
5. MusicalComposition \*tmp;
6. MusicalComposition \*a;
7. int t=0;
8. char \*n;
9. char \*s;
10. tmp=head;
11. a=tmp->next;
12. for (int i=0; i<cen-1; i++){
13. for (int j = 0; j<cen-1; j++){
14. if((tmp->year)>(a->year)){
15. n = tmp->name;
16. tmp->name = a->name;
17. a->name = n;
18. s = tmp->author;
19. tmp->author = a->author;
20. a->author = s;
21. t=tmp->year;
22. tmp->year=a->year;
23. a->year=t;
24. }
25. if ((tmp->next)&&(a->next)){
26. tmp=tmp->next;
27. a=a->next;}
28. }
29. tmp=head;
30. a=tmp->next;
31. }
32. MusicalComposition \*newhead = head;
33. a=newhead->next;
34. for (int i=0; i<cen; i++) newhead=newhead->next;
35. tmp=newhead;
36. a=tmp->next;
37. for (int i=0; i<cen-1; i++){
38. for (int j = 0; j<cen-1; j++){
39. if((tmp->year)<(a->year)){
40. n = tmp->name;
41. tmp->name = a->name;
42. a->name = n;
43. s = tmp->author;
44. tmp->author = a->author;
45. a->author = s;
46. t=tmp->year;
47. tmp->year=a->year;
48. a->year=t;
49. }
50. if ((tmp->next)&&(a->next)){
51. tmp=tmp->next;
52. a=a->next;}
53. }
54. tmp=newhead;
55. a=tmp->next;
56. }

Функция принимает в качестве аргумента указатель на “голову” списка (MusicalComposition\* head). Затем происходит сортировка первой половины композиций по убыванию относительно года написания композиции. Далее происходит определение “новой головы” для второй половины списка и последующая его сортировка по возрастанию относительно года написания композиции.

**3 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 нового
* Количество элементов после добавления
* Названия всех композиций и год написания после удаления
* Количество элементов после удаления
* Названия всех композиций и год написания после сортировки

|  |  |  |
| --- | --- | --- |
| № | ВВОД | ВЫВОД |
| 1 | 6  Radio Ga Ga  Electric six  2004  Karma Police  Radiohead  1997  Heartbreaker  Metronomy  2008  Another Brick In The Wall  Pink Floyd  1979  Dance Dance  Moe Shop  2016  Firestarter  Prodigy  1997  Feelings  Harmful logic  2016  Feelings | Radio Ga Ga Electric six 2004 //первый элемент списка  6 //количество элементов до добавления  7 //количество элементов после добавления  Radio Ga Ga 2004 //список композиций  Karma Police 1997  Heartbreaker 2008  Another Brick In The Wall 1979  Dance Dance 2016  Firestarter 1997  6 //количество элементов после удаления  After sorting: //отсортированный список  Karma Police 1997  Radio Ga Ga 2004  Heartbreaker 2008  Dance Dance 2016  Firestarter 1997  Another Brick In The Wall 1979 |
| 2 | 6  Radio Ga Ga  Electric six  2004  Karma Police  Radiohead  1997  Heartbreaker  Metronomy  2008  Another Brick In The Wall  Pink Floyd  1979  Dance Dance  Moe Shop  2016  Firestarter  Prodigy  1997  Feelings  Harmful logic  2016  Firestarter | Radio Ga Ga Electric six 2004 //первый элемент списка  6 //количество элементов до добавления  7 //количество элементов после добавления  Radio Ga Ga 2004 //список композиций  Karma Police 1997  Heartbreaker 2008  Another Brick In The Wall 1979  Dance Dance 2016  Feelings 2016  6 //количество элементов после удаления  After sorting: //отсортированный список  Karma Police 1997  Radio Ga Ga 2004  Heartbreaker 2008  Dance Dance 2016  Feelings 2016  Another Brick In The Wall 1979 |
| 3 | 6  Radio Ga Ga  Electric six  2004  Karma Police  Radiohead  1997  Heartbreaker  Metronomy  2008  Another Brick In The Wall  Pink Floyd  1979  Dance Dance  Moe Shop  2016  Firestarter  Prodigy  1997  Feelings  Harmful logic  2016  Dance Dance | Radio Ga Ga Electric six 2004 //первый элемент списка  6 //количество элементов до добавления  7 //количество элементов после добавления  Radio Ga Ga 2004 //список композиций  Karma Police 1997  Heartbreaker 2008  Another Brick In The Wall 1979  Firestarter 1997  Feelings 2016  6 //количество элементов после удаления  After sorting: //отсортированный список  Karma Police 1997  Radio Ga Ga 2004  Heartbreaker 2008  Feelings 2016  Firestarter 1997  Another Brick In The Wall 1979 |

**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. UNIX.  Программное окружение / Керниган Б., Пайк Р. СПб.: Символ Плюс, 2003. 416 с.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. // Описание структуры MusicalComposition
5. typedef struct MusicalComposition {
6. char\* name;
7. char\* author;
8. int year;
9. struct MusicalComposition \*next;
10. struct MusicalComposition \*prev;
11. } MusicalComposition;
12. // Создание структуры MusicalComposition
13. MusicalComposition\* createMusicalComposition(char\* name, char\* author, int year) {
14. MusicalComposition\* song = (MusicalComposition\*)malloc(sizeof(MusicalComposition));
15. song -> name = name;
16. song -> author = author;
17. song -> year = year;
18. song -> next = NULL;
19. song -> prev = NULL;
20. return song;
21. }
22. // Функции для работы со списком MusicalComposition
23. MusicalComposition \*createMusicalCompositionList(char \*\*array\_names, char \*\*array\_authors, int \*array\_years, int n) {
24. MusicalComposition \*head = createMusicalComposition(array\_names[0], array\_authors[0], array\_years[0]);
25. MusicalComposition \*prev = head;
26. MusicalComposition \*current;
27. for (int i = 1; i < n; i++) {
28. current = createMusicalComposition(array\_names[i], array\_authors[i], array\_years[i]);
29. current -> prev = prev;
30. prev -> next = current;
31. prev = current;
32. }
33. return head;
34. }
35. void push(MusicalComposition\* head, MusicalComposition\* element) {
36. MusicalComposition \*tmp = (MusicalComposition\*)malloc(sizeof(MusicalComposition));
37. tmp = head;
38. while (tmp -> next != NULL) {
    1. tmp = tmp -> next;
39. }
40. tmp -> next = element;
41. element -> prev = tmp;
42. }
43. void removeEl(MusicalComposition\* head, char\* name\_for\_remove) {
44. MusicalComposition \*tmp = (MusicalComposition\*)malloc(sizeof(MusicalComposition));
45. tmp = head;
46. while (strcmp(tmp -> name, name\_for\_remove)) {
47. tmp = tmp -> next;
48. }
49. if (tmp -> prev == NULL) {
50. tmp = tmp -> next;
51. tmp -> prev = NULL;
52. \*head = \*tmp;
53. }
54. else if (tmp -> next == NULL) {
55. tmp = tmp -> prev;
56. tmp -> next = NULL;
57. } else {
58. tmp -> prev -> next = tmp -> next;
59. tmp -> next -> prev = tmp -> prev;
60. }
61. }
62. int count(MusicalComposition\* head) {
63. MusicalComposition \*tmp = (MusicalComposition\*)malloc(sizeof(MusicalComposition));
64. tmp = head;
65. int count = 1;
66. while (tmp -> next != NULL) {
67. tmp = tmp -> next;
68. count++;
69. }
70. return count;
71. }
72. void print\_names(MusicalComposition \* head) {
73. MusicalComposition \*tmp = (MusicalComposition\*)malloc(sizeof(MusicalComposition));
74. tmp = head;
75. while (tmp -> next != NULL) {
76. printf("%s\n", tmp -> name);
77. tmp = tmp -> next;
78. }
79. printf("%s\n", tmp -> name);
80. }
81. void sort(MusicalComposition\* head)
82. {
83. int cnt = count(head);
84. int cen = cnt/2;
85. MusicalComposition \*tmp;
86. MusicalComposition \*a;
87. int t=0;
88. char \*n;
89. char \*s;
90. tmp=head;
91. a=tmp->next;
92. for (int i=0; i<cen-1; i++){
93. for (int j = 0; j<cen-1; j++){
94. if((tmp->year)>(a->year)){
95. n = tmp->name;
96. tmp->name = a->name;
97. a->name = n;
98. s = tmp->author;
99. tmp->author = a->author;
100. a->author = s;
101. t=tmp->year;
102. tmp->year=a->year;
103. a->year=t;
104. }
105. if ((tmp->next)&&(a->next)){
106. tmp=tmp->next;
107. a=a->next;}
108. }
109. tmp=head;
110. a=tmp->next;
111. }
112. MusicalComposition \*newhead = head;
113. a=newhead->next;
114. for (int i=0; i<cen; i++) newhead=newhead->next;
115. tmp=newhead;
116. a=tmp->next;
117. for (int i=0; i<cen-1; i++){
118. for (int j = 0; j<cen-1; j++){
119. if((tmp->year)<(a->year)){
120. n = tmp->name;
121. tmp->name = a->name;
122. a->name = n;
123. s = tmp->author;
124. tmp->author = a->author;
125. a->author = s;
126. t=tmp->year;
127. tmp->year=a->year;
128. a->year=t;
129. }
130. if ((tmp->next)&&(a->next)){
131. tmp=tmp->next;
132. a=a->next;}
133. }
134. tmp=newhead;
135. a=tmp->next;
136. }
137. }
138. int main() {
139. int length;
140. scanf("%d\n", &length);
141. char\*\* names = (char\*\*)malloc(sizeof(char\*)\*length);
142. char\*\* authors = (char\*\*)malloc(sizeof(char\*)\*length);
143. int\* years = (int\*)malloc(sizeof(int)\*length);
144. for (int i = 0; i < length; i++) {
145. char name[80];
146. char author[80];
147. fgets(name, 80, stdin);
148. fgets(author, 80, stdin);
149. fscanf(stdin, "%d\n", &years[i]);
150. (\*strstr(name,"\n")) = 0;
151. (\*strstr(author,"\n")) = 0;
152. names[i] = (char\*)malloc(sizeof(char\*) \* (strlen(name)+1));
153. authors[i] = (char\*)malloc(sizeof(char\*) \* (strlen(author)+1));
154. strcpy(names[i], name);
155. strcpy(authors[i], author);
156. }
157. MusicalComposition\* head = createMusicalCompositionList(names, authors, years, length);
158. char name\_for\_push[80];
159. char author\_for\_push[80];
160. int year\_for\_push;
161. char name\_for\_remove[80];
162. fgets(name\_for\_push, 80, stdin);
163. fgets(author\_for\_push, 80, stdin);
164. fscanf(stdin, "%d\n", &year\_for\_push);
165. (\*strstr(name\_for\_push,"\n"))=0;
166. (\*strstr(author\_for\_push,"\n"))=0;
167. MusicalComposition\* element\_for\_push = createMusicalComposition(name\_for\_push, author\_for\_push, year\_for\_push);
168. fgets(name\_for\_remove, 80, stdin);
169. (\*strstr(name\_for\_remove,"\n")) = 0;
170. printf("%s %s %d\n", head -> name, head -> author, head -> year);
171. int k = count(head);
172. printf("%d\n", k);
173. push(head, element\_for\_push);
174. k = count(head);
175. printf("%d\n", k);
176. removeEl(head, name\_for\_remove);
177. print\_names(head);
178. k = count(head);
179. printf("%d\n", k);
180. printf("After sorting:\n" );
181. sort(head);
182. print\_names(head);
183. free(names);
184. free(authors);
185. free(years);
186. return 0;
187. }