**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 |  | Сергеенков М.Ю. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2016

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Сергеенков М.Ю. | | |
| Группа 6303 | | |
| Тема работы: Линейные списки | | |
|  | | |
| Содержание пояснительной записки:   * Содержание * Введение * Описание функций для работы с двунаправленным линейным списком * Описание функции сортировки двунаправленного линейного списка * 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 |  | Сергеенков М.Ю. |
| Преподаватель |  | Берленко Т.А. |

**Аннотация**

В ходе работы была реализована программа на языке программирования С, с помощью которой создается двунаправленный линейный список, содержащий сведения о музыкальных композициях, а также функции для работы с ним, позволяющие добавлять, удалять, сортировать и выводить элементы списка и их количество. Также описана структура элемента списка. Помимо этого, была проведена работа по оптимизации исходного кода программы для ускорения ее быстродействия и максимально экономичного использования памяти и ресурсов клиента. Приведено полное описание исходного кода.

**содержание**

|  |
| --- |
| [**введение** 5](#_Toc470082407)  [**1.** **функции для работы с двунаправленным линейным списком** 6](#_Toc470082408)  [1.1. Создание элемента 6](#_Toc470082409)  [Создание двунаправленного списка связанных элементов 6](#_Toc470082410)  [1.2. Добавление элемента в конец списка 7](#_Toc470082411)  [1.3. Функция для определения, есть ли в списке заданный элемент 7](#_Toc470082412)  [1.4. Удаление элементов из списка 8](#_Toc470082413)  [1.5. Подсчет количества элементов в списке 8](#_Toc470082414)  [1.6. Вывод названий музыкальных композиций в консоль 9](#_Toc470082415)  [**1.7.** **Вывод авторов музыкальных композиций в консоль** 9](#_Toc470082416)  [**2.** **функции СОРТИРОВКИ СПИСКА** 10](#_Toc470082417)  [2.1. Сортировка элементов списка по автору в порядке, обратном алфавитному 10](#_Toc470082418)  [2.2. Вывод перестановки двух элементов списка 10](#_Toc470082419)  [**3.** **примеры работы программы** 12](#_Toc470082420)  [**заключение** 14](#_Toc470082421)  [**список использованных источников** 15](#_Toc470082422)  [**приложение А** 16](#_Toc470082423)  [**Исходный код программы** 16](#_Toc470082424) |
|  |
|  |
|  |

# **введение**

В ходе работы 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 (элементов) из списка
* Подсчет количества элементов в списке
* Вывод элементов списка в консоль
* Сортировка элементов в списке по автору в порядке, обратном алфавитному
* Перестановка двух элементов списка

# **функции для работы с двунаправленным линейным списком**

## Создание элемента

MusicalComposition\* createMusicalComposition(char\* name, char\* author, int year){

MusicalComposition \*tmp;

tmp=(MusicalComposition\*)malloc(sizeof(MusicalComposition));

tmp->name=name;

tmp->author=author;

tmp->year=year;

return tmp;

}

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

## Создание двунаправленного списка связанных элементов

MusicalComposition\* createMusicalCompositionList(char\*\* array\_names, char\*\* array\_authors, int\* array\_years, int n){

MusicalComposition \*ptr;

head=(MusicalComposition\*)malloc(sizeof(MusicalComposition));

ptr=head;

head->name=array\_names[0];

head->author=array\_authors[0];

head->year=array\_years[0];

for (int i = 1; i < n; ++i)

{

MusicalComposition\* temp;

temp=(MusicalComposition\*)malloc(sizeof(MusicalComposition));

temp->prev=ptr;

temp->next=NULL;

temp->name=array\_names[i];

temp->author=array\_authors[i];

temp->year=array\_years[i];

ptr->next=temp;

ptr=ptr->next;

}

return head;

}

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 (первый элемент) и происходит заполнение ее переменных данными, затем аналогичные операции происходят для всех последующих n-1 элементов списка

## Добавление элемента в конец списка

void push(MusicalComposition\* element){

MusicalComposition \*ptr,\*temp;

if (head==NULL){

head=element;

}

else{

ptr=head;

//ищем хвостовой элемент списка

while(ptr->next!=NULL){

ptr=ptr->next;

}

if (ptr==head){

head=element;

}

else{

element->next=NULL;

element->prev=ptr;

ptr->next=element;

}

}

}

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

## Функция для определения, есть ли в списке заданный элемент

Функция принимает в качестве аргумента указатель на название музыкальной композиции, которую необходимо удалить, и, проходя по списку, сравнивает каждый элемент с ним. Функция возвращает 1, если такой элемент есть в списке, и 0 при его отсутствии.

int elementInList(char\* element){

MusicalComposition \*ptr;

ptr=head;

int flag=0;

while (ptr!=NULL) {

if (strcmp(ptr->name, element)==0) {

flag=1;

break;

}

else{

ptr=ptr->next;

}

}

if (flag==1) {

return 1;

}

else{

return 0;

}

}

## Удаление элементов из списка

Функция принимает в качестве аргумента указатель на название композиции (**char**\* name\_for\_remove) и производит удаление из списка всех элементов с таким названием, вызывая вышеописанную функцию elementInList для определения наличия в списке элементов для удаления.

void removeEl( char\* name\_for\_remove){

MusicalComposition \*ptr;

ptr=head;

MusicalComposition\* next;

MusicalComposition\* prev;

while (elementInList(name\_for\_remove)==1) {

ptr=head;

while (\*ptr->name!=\*name\_for\_remove) {

ptr=ptr->next;

}

if (ptr==head) {

head=head->next;

}

else if(ptr->next==NULL) {

prev=ptr->prev;

prev->next=NULL;

free(ptr);

}

else{

next=ptr->next;

prev=ptr->prev;

next->prev=prev;

prev->next=next;

free(ptr);

}

}

}

## Подсчет количества элементов в списке

Функция производит подсчет элементов в списке, пока следующий элемент существует. Возвращает количество элементов списка.

int count(){

MusicalComposition \*ptr;

ptr=head;

int count=0;

while (ptr!=NULL){

++count;

ptr=ptr->next;

}

return count;

}

## Вывод названий музыкальных композиций в консоль

Функция производит вывод на консоль названий музыкальных композиций, пока следующий элемент списка существует.

void print\_names(){

MusicalComposition \*ptr;

ptr=head;

while (ptr!=NULL) {

for (int i = 0; i < strlen(ptr->name); ++i)

{

putchar(ptr->name[i]);

}

putchar('\n');

ptr = ptr->next;

}

}

* 1. **Вывод авторов музыкальных композиций в консоль**

Функция производит вывод на консоль авторов музыкальных композиций, пока следующий элемент списка существует.

void print\_authors(){

MusicalComposition \*ptr;

ptr=head;

while (ptr!=NULL) {

for (int i = 0; i < strlen(ptr->author); ++i)

{

putchar(ptr->author[i]);

}

putchar('\n');

ptr = ptr->next;

}

}

# **функции СОРТИРОВКИ СПИСКА**

## Сортировка элементов списка по автору в порядке, обратном алфавитному

Функция сортировки сравнивает строку, содержащую имя автора текущей композиции, со следующей аналогичной строкой. Сравнение происходит с помощью функции strcmp, описанной в заголовочном файле string.h, которая возвращает отрицательное число, если код символа из первой строки окажется меньше кода символа второй строки. Затем с помощью функции swap, описанной ниже, происходить перестановка двух элементов списка. Таким образом, реализуется алгоритм сортировки «пузырьком». Стоит отметить, что функция будет выполняться лишь в том случае, если «голова» списка не ссылается на NULL.

void sortByAuthor(){

MusicalComposition\* current;

MusicalComposition\* next;

if (head!=NULL){

current=head;

next=current->next;

int len=count();

for(int i=0; i<len; ++i) {

current=head;

next=current->next;

for(int j=0; j<len-i-1; ++j) {

if (strcmp(current->author, next->author)<0)

swap(current, next);

current=current->next;

next=next->next;

}

}

}

}

## Вывод перестановки двух элементов списка

Функция принимает в качестве аргументов указатели на два соседних элементов списка (MusicalComposition\* a) и (MusicalComposition\* b) , которые необходимо поменять местами. В случае, если один из них является головой, происходит обмен данными между элементами, в остальных случаях перенаправляются указатели (это используется для экономии памяти).

void swap(MusicalComposition\* a, MusicalComposition\* b) {

if(a == head) {

MusicalComposition\* temp = (MusicalComposition\*)malloc(sizeof(MusicalComposition));

\*temp = \*head;

a->name = b->name;

a->author = b->author;

a->year = b->year;

b->name = temp->name;

b->author = temp->author;

b->year = temp->year;

return;

}

a->next = b->next;

b->prev = a->prev;

if(a->next != NULL) {

a->next->prev = a;

}

if(b->prev != NULL) {

b->prev->next = b;

}

b->next = a;

a->prev = b;

return;

}

1. **примеры работы программы**

Для демонстрации всех возможностей разработанной программы используется следующий формат ввода:

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

Формат вывода представлен ниже:

* 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
* Авторы всех композиций после сортировки в порядке, обратном алфавитному

|  |  |  |
| --- | --- | --- |
| № | ВВОд | вЫВОД |
| 1 | 6  Fields of Gold  Sting  1993  Pretty Fly  The Offspring  1998  American Idiot  Green Day  2004  Going to the run  Golden Earring  1991  Fear of the dark  Iron Maiden  1994  The Unforgiven  Metallica  1991  Nothing Else Matters  Metallica  1991  Pretty Fly | Fields of Gold Sting 1993 //первый элемент списка  6 //количество элементов до добавления  7 //количество элементов после добавления  Fields of Gold 1993 //список композиций  Pretty Fly 1998  American Idiot 2004  Going to the run 1991  Fear of the dark 1994  The Unforgiven 1991  Nothing Else Matters 1991  6 //количество элементов после удаления  //отсортированный список  Sting  Metallica  Metallica  Iron Maiden  Green Day  Golden Earring |
| 2 | 6  Fields of Gold  Sting  1993  Pretty Fly  The Offspring  1998  American Idiot  Green Day  2004  Going to the run  Golden Earring  1991  Fear of the dark  Iron Maiden  1994  The Unforgiven  Metallica  1991  Nothing Else Matters  Metallica  1991  Fields of Gold | Fields of Gold Sting 1993 //первый элемент списка  6 //количество элементов до добавления  7 //количество элементов после добавления  Pretty Fly 1998 //список композиций  American Idiot 2004  Going to the run 1991  Fear of the dark 1994  The Unforgiven 1991  Nothing Else Matters 1991  6 //количество элементов после удаления  //отсортированный список  Metallica  Metallica  Iron Maiden  Green Day  Golden Earring |
| 3 | 3  Fear of the dark  Iron Maiden  1994  Fear of the dark  Iron Maiden  1994  Fear of the dark  Iron Maiden  1994  The Unforgiven  Metallica  1991  Fear of the dark | Fear of the dark 1994 //первый элемент списка  3 //количество элементов до добавления  4 //количество элементов после добавления  The Unforgiven  1 //количество элементов после удаления  //отсортированный список  Metallica |

# **заключение**

В ходе работы была реализована программа на языке Си для работы с двунаправленным линейным списком музыкальных композиций. В частности, был создан и описан сам двунаправленный список музыкальных композиций MusicalComposition и разработан 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. Моли Б. UNIX/Linux: теория и практика программирования. М.: КУДИЦ-ОБРАЗ, 2004. 576 с.

# **приложение А**

# **Исходный код программы**

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

typedef struct MusicalComposition MusicalComposition;

struct MusicalComposition {

char \*name;

char \*author;

int year;

MusicalComposition\* next;

MusicalComposition\* prev;

};

MusicalComposition\* head;

MusicalComposition\* createMusicalComposition(char\* name, char\* author, int year);

// Функции для работы со списком MusicalComposition

MusicalComposition\* createMusicalCompositionList(char\*\* array\_names, char\*\* array\_authors, int\* array\_years, int n);

MusicalComposition \* deleteHead(MusicalComposition\* head);

int elementInList(char\* element);

void push(MusicalComposition\* element);

void removeEl( char\* name\_for\_remove);

int count();

void print\_names();

void sortByAuthor();

void swap(MusicalComposition\* a, MusicalComposition\* b);

void print\_authors();

int main(){

int length;

scanf("%d\n", &length);

char\*\* names = (char\*\*)malloc(sizeof(char\*)\*length);

char\*\* authors = (char\*\*)malloc(sizeof(char\*)\*length);

int\* years = (int\*)malloc(sizeof(int)\*length);

for (int i=0; i<length; i++)

{

char name[80];

char author[80];

fgets(name, 80, stdin);

fgets(author, 80, stdin);

fscanf(stdin, "%d\n", &years[i]);

(\*strstr(name,"\n"))=0;

(\*strstr(author,"\n"))=0;

names[i] = (char\*)malloc(sizeof(char\*) \* (strlen(name)+1));

authors[i] = (char\*)malloc(sizeof(char\*) \* (strlen(author)+1));

strcpy(names[i], name);

strcpy(authors[i], author);

}

head = createMusicalCompositionList(names, authors, years, length);

char name\_for\_push[80];

char author\_for\_push[80];

int year\_for\_push;

char name\_for\_remove[80];

fgets(name\_for\_push, 80, stdin);

fgets(author\_for\_push, 80, stdin);

fscanf(stdin, "%d\n", &year\_for\_push);

(\*strstr(name\_for\_push,"\n"))=0;

(\*strstr(author\_for\_push,"\n"))=0;

MusicalComposition\* element\_for\_push = createMusicalComposition(name\_for\_push, author\_for\_push, year\_for\_push);

fgets(name\_for\_remove, 80, stdin);

(\*strstr(name\_for\_remove,"\n"))=0;

printf("%s %s %d\n", head->name, head->author, head->year);

int k = count();

///printf("%d\n", k);

push(element\_for\_push);

k = count();

printf("%d\n", k);

removeEl(name\_for\_remove);

print\_names();

k = count();

printf("%d\n", k);

//sortByAuthor();

// print\_authors();

return 0;

}

MusicalComposition\* createMusicalComposition(char\* name, char\* author, int year){

MusicalComposition \*tmp;

tmp=(MusicalComposition\*)malloc(sizeof(MusicalComposition));

tmp->name=name;

tmp->author=author;

tmp->year=year;

return tmp;

}

MusicalComposition\* createMusicalCompositionList(char\*\* array\_names, char\*\* array\_authors, int\* array\_years, int n){

MusicalComposition \*ptr;

head=(MusicalComposition\*)malloc(sizeof(MusicalComposition));

ptr=head;

head->name=array\_names[0];

head->author=array\_authors[0];

head->year=array\_years[0];

for (int i = 1; i < n; ++i)

{

MusicalComposition\* temp;

temp=(MusicalComposition\*)malloc(sizeof(MusicalComposition));

temp->prev=ptr;

temp->next=NULL;

temp->name=array\_names[i];

temp->author=array\_authors[i];

temp->year=array\_years[i];

ptr->next=temp;

ptr=ptr->next;

}

return head;

}

void push(MusicalComposition\* element){

MusicalComposition \*ptr,\*temp;

if (head==NULL){

head=element;

}

else{

ptr=head;

//ищем хвостовой элемент списка

while(ptr->next!=NULL){

ptr=ptr->next;

}

if (ptr==head){

head=element;

}

else{

element->next=NULL;

element->prev=ptr;

ptr->next=element;

}

}

}

void removeEl( char\* name\_for\_remove){

MusicalComposition \*ptr;

ptr=head;

MusicalComposition\* next;

MusicalComposition\* prev;

while (elementInList(name\_for\_remove)==1) {

ptr=head;

while (\*ptr->name!=\*name\_for\_remove) {

ptr=ptr->next;

}

if (ptr==head) {

head=head->next;

}

else if(ptr->next==NULL) {

prev=ptr->prev;

prev->next=NULL;

free(ptr);

}

else{

next=ptr->next;

prev=ptr->prev;

next->prev=prev;

prev->next=next;

free(ptr);

}

}

}

int count(){

MusicalComposition \*ptr;

ptr=head;

int count=0;

while (ptr!=NULL){

++count;

ptr=ptr->next;

}

return count;

}

void print\_names(){

MusicalComposition \*ptr;

ptr=head;

while (ptr!=NULL) {

for (int i = 0; i < strlen(ptr->name); ++i)

{

putchar(ptr->name[i]);

}

putchar('\n');

ptr = ptr->next;

}

}

int elementInList(char\* element){

MusicalComposition \*ptr;

ptr=head;

int flag=0;

while (ptr!=NULL) {

if (strcmp(ptr->name, element)==0) {

flag=1;

break;

}

else{

ptr=ptr->next;

}

}

if (flag==1) {

return 1;

}

else{

return 0;

}

}

void sortByAuthor(){

MusicalComposition\* current;

MusicalComposition\* next;

if (head!=NULL){

current=head;

next=current->next;

int len=count();

for(int i=0; i<len; ++i) {

current=head;

next=current->next;

for(int j=0; j<len-i-1; ++j) {

if (strcmp(current->author, next->author)<0)

swap(current, next);

current=current->next;

next=next->next;

}

}

}

}

void swap(MusicalComposition\* a, MusicalComposition\* b) {

if (a->prev==NULL) {

MusicalComposition\* temp=(MusicalComposition\*)malloc(sizeof(MusicalComposition));

temp->name=b->name;

temp->author=b->author;

temp->year=b->year;

b->name=a->name;

b->author=a->author;

b->year=a->year;

a->name=temp->name;

a->author=temp->author;

a->year=temp->year;

}

else{

a->next = b->next;

b->prev = a->prev;

if(a->next != NULL) {

a->next->prev = a;

}

if(b->prev != NULL) {

b->prev->next = b;

}

b->next = a;

a->prev = b;

}

return;

}

void print\_authors(){

MusicalComposition \*ptr;

ptr=head;

while (ptr!=NULL) {

for (int i = 0; i < strlen(ptr->author); ++i)

{

putchar(ptr->author[i]);

}

putchar('\n');

ptr = ptr->next;

}

}