**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

Курсовая РАБОТА

**по дисциплине «Программирование»**

Тема: Двунаправленные линейные списки

|  |  |  |
| --- | --- | --- |
| Студентка гр. 6304 |  | Смотрова А. А |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2016

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студентка Смотрова А.А. | | |
| Группа 6304 | | |
| Тема работы: Двунаправленные линейные списки | | |
| Исходные данные: Требуется создать двунаправленный линейный список и API для работы с ним | | |
| Содержание пояснительной записки:  Требуются разделы : Аннотация, Цель работы, Формулировка задачи, Решение задачи (описание всех функций в программе), Работа с GitHub, Makefile, Примеры работы программы, Приложение с исходным кодом программы. | | |
| Дата выдачи задания: 07.12.2016 | | |
| Дата сдачи реферата: 22.12.2016 | | |
| Дата защиты реферата: 22.12.2016 | | |
| Студентка |  | Смотрова А. А. |
| Преподаватель |  | Берленко Т.А. |

# **Аннотация**

Курсовая работа посвящена созданию двунаправленного линейного списка музыкальных композиций и API с использованием указателей, работы с динамической памятью. В работе показано выполнение стандартных операция, таких как удаление элементов, поиск по ключевым словам, создание списка элементов, подсчет количества элементов и списке и прочие. В работе использовались файлы исходного кода(\*.c), заголовочные файлы(\*.h), makefile. Также продемонстрированы основные этапы работы с Git.

**Содержание**

[**Аннотация** 3](#_Toc470117357)

[**Введение** 5](#_Toc470117358)

[*Цель работы:* 5](#_Toc470117359)

[*Формулировка задачи:* 5](#_Toc470117360)

[Индивидуальное задание в дополнение к общему: 6](#_Toc470117361)

[**Решение задачи** 6](#_Toc470117362)

[*Создание структуры данных:* 6](#_Toc470117363)

[*Функция создания нового экземпляра данного типа:* 6](#_Toc470117364)

[*Функция создания списка из элементов:* 7](#_Toc470117365)

[*Функция добавления элемента в конец списка:* 8](#_Toc470117366)

[*Функция удаления элемента*: 8](#_Toc470117367)

[*Функция подсчета элементов списка:* 9](#_Toc470117368)

[*Функция печать имен:* 9](#_Toc470117369)

[*Функция добавления 3 произвольных элементов после каждого нечетного* 10](#_Toc470117370)

[*Функция удаления всех элементов с данной подстрокой в поле имени:* 11](#_Toc470117371)

[*Makefile:* 12](#_Toc470117372)

[*Работа с GitHub:* 13](#_Toc470117373)

[**Примеры работы программы** 14](#_Toc470117374)

[**Приложение 1 Код программы** 16](#_Toc470117375)

[*Spisok\_Menu.cpp:* 16](#_Toc470117376)

[*Spisok\_Functions.cpp:* 19](#_Toc470117377)

[*Spisok\_AddFunctions.cpp:* 19](#_Toc470117378)

[*Spisok\_RemoveFunctions.cpp:* 21](#_Toc470117379)

[*MusicalComposition\_API.h:* 23](#_Toc470117380)

[**Заключение** 25](#_Toc470117381)

# **Введение**

## *Цель работы:*

Выработка навыков применения типа struct в языке СИ. Повторение темы указателей, в том числе указателей на указатели. Проверка умения создать и собрать единый проект.

*Формулировка задачи:*

Создайте двунаправленный список музыкальных композиций MusicalComposition и **api** (application***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.

Функция для создания элемента списка (тип элемента MusicalComposition)

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

Функции для работы со списком:

* MusicalComposition\* createMusicalCompositionList(char\*\* array\_names, char\*\* array\_authors, int\* array\_years, int n); // создает список музыкальных композиций MusicalCompositionList, в котором:
* ***n****- длина массивов* ***array\_names****,* ***array\_authors****,* ***array\_years****.*
* поле **name** первого элемента списка соответствует первому элементу списка array\_names (**array\_names[0]**).
* поле **author** первого элемента списка соответствует первому элементу списка array\_authors (**array\_authors[0]**).
* поле **year** первого элемента списка соответствует первому элементу списка array\_authors (**array\_years [0]**).
* void push (MusicalComposition\* head, MusicalComposition\* element); // добавляет **element** в конец списка **musical\_composition\_list**
* void removeEl (MusicalComposition\*  head, char\* name\_for\_remove); // удаляет элемент **element** списка, у которого значение **name** равно значению name**\_for\_remove**
* int count(MusicalComposition\*  head); //возвращает количество элементов списка
* void print\_names (MusicalComposition\* head); //Выводит названия композиций

Индивидуальное задание:

1. Функция удалить все элементы, в имени автора которых присутствует подстрока s.
2. Функция добавления 3 элементов после каждого четного элемента в исходном списке.

**Решение задачи**

*Создание структуры данных:*

Создадим описание структуры MusicalComposition:

**struct** MusicalComposition { //объявили структуру

**char** name[**80**]; //строка не более 80 символов

**char** author[**80**];

**int** year;

**struct** MusicalComposition \* next;//следующий элемент списка

**struct** MusicalComposition \* previous; //предыдущий элемент списка

};

Определим новый тип MusicalComposition, чтобы впоследствии не использовать struct.

**typedef** **struct** MusicalComposition MusicalComposition;

Функция создания экземпляра структуры MusicalComposition

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

{

MusicalComposition \*mc = MusicalComposition\*) malloc (**sizeof**(MusicalComposition));//выделяем память по размеру структуры

strcpy(mc->name, name); //скопируем в поле «Имя» введенное имя

strcpy(mc->author, author); //в поле автор – введенного автора

mc->year = year;

mc->previous = NULL; //нет ссылок на предыдущий

mc->next = NULL; //и следующий элемент

**return** mc;

};

Функция создания списка:

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

{

MusicalComposition \*head = createMusicalComposition(array\_names[**0**], array\_authors[**0**], array\_years[**0**]); //вводим шапку таблицы

MusicalComposition \*previous = head; //все поля previous ==полям head

MusicalComposition \*current;

**for** (**int** i = **1**; i < n; ++i) //начинаем заполнять текущий узел

{

current = createMusicalComposition(array\_names[i], array\_authors[i], array\_years[i]);

current->previous = previous; //для текущего узла предыдущий узел – это previous

previous->next = current; //для предыдущего узла следующий – это текущий

previous = current; //передвигаем узел на один

}

**return** head;

}

MusicalComposition\* **added\_str**() {

system("cls"); //очищаем консольное окно

**int** len = **0**; //количество строчек в списке

**while** (len<=**0**) //пока оно отрицальное, повторяем вопрос

{

printf("Number of elements you want to add**\n**");

scanf("%d", &len); //принимает число на ввод

**if** (len <= **0**) {

printf("You should enter a positive number!"); //предупреждение

**continue**;//новая итерация

}

Else //иначе – выходим из цикла

**break**;

}

**char** \*\*names = (**char**\*\*)malloc(**sizeof**(**char**\*)\*len);//выделяем память

**char** \*\*authors = (**char**\*\*)malloc(**sizeof**(**char**\*)\*len);//под массивы названий,

**int** \*years = (**int**\*)malloc(**sizeof**(**int**)\*len); //авторов, лет

**int** i=**0**;

**for** (i = **0**; i < len; ++i)

{//заполняем ячейки списка

printf("Enter the element:name [space] author [space] year:**\n**");

**char** name[**80**]; //строка названия

**char** author[**80**]; //строка имени автора

scanf("%s %s %d", name, author, &years[i]); //считаем

printf("You entered element %d**\n**", i+**1**);

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

//выделяем память под массив

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

//выделяем память под массив

strcpy(names[i], name); //записываем название в массив names

strcpy(authors[i], author); //записываем автора в массив authors

}

**return** createMusicalCompositionList(names, authors, years, len);

}

Функция для добавления элемента в конец списка:

**void** **push**(**struct** MusicalComposition\* head)

{

**char** name\_for\_push[**80**]; //название

**char** author\_for\_push[**80**]; //имя автора

**int** year\_for\_push;//год

printf("Enter the name of the song, author and year using spaces**\n**");

scanf("%s %s %d", name\_for\_push, author\_for\_push, &year\_for\_push);

//ввод значений

MusicalComposition\* element= createMusicalComposition(name\_for\_push, author\_for\_push, year\_for\_push);//создаем элемент из введенных значений

**while** (head->next) //пока есть следующий элемент,

head = head->next;//перемещаемся по списку

head->next = element; //последний элемент ссылается на созданный

element->previous = head;// ссылка на предыдущий элемент

element->next = NULL; //следующего эл-та нет ◊этот эл-т последний

printf("Done!");

}

функция для удаления элемента по названию песни.

**void** **removeEl**(**struct** MusicalComposition\*head, **char**\*name\_for\_remove)

{

MusicalComposition \*current = head;

**while** (current->next) //пока есть следующий элемент

{

**if** (strcmp(current->name, name\_for\_remove) == **0**) //если имя и строка для удаления совпали

**if** ((current->previous) == **0**) //если это первый элемент

{

MusicalComposition \*newcurrent = current->next; //снимаем связь со следующим элементом (перенаправляем на newcurrent)

strcpy(current->name, newcurrent->name);//копируем

strcpy(current->author, newcurrent->author); //инф-цию

current->year = newcurrent->year;//в newcurrent

newcurrent->next->previous = current;

current->next = newcurrent->next;

free(newcurrent);//сбросили newcurrent

}

Else //если это не первый элемент

{

current->previous->next = current->next; //меняет значение поля next у предыдущего элемента

current->next->previous = current->previous; //меняет значение поля previous у следующего

free(current); //убираем узел

}

current = current->next;//сдвигаемся на один узел дальше

}

**if** ((strcmp(current->name, name\_for\_remove) == **0**) && (current->next == **0**)) //если это последний элемент

{

current->previous->next = NULL; //указатель на след. Эл-т предыдущего делаем NULL

free(current);

}

}

Следующая функция считает количество элементов в списке.

**int** **count**(MusicalComposition \*head) { //возвращает число элементов списка

**int** c=**0**;

**while** (head) {

c++;

head=head->next;

}

**return** c;

}

Последняя функция реализует печать всех названий песен из списка.

**void** **print\_names**(MusicalComposition \*head){

MusicalComposition \*current = head;

**while** (current->next){ //пока следующий элемент существует

printf("%s**\n**", current->name); //печатай название

current = current->next;//двигаемся к след. узлу

}

printf("%s**\n**", current->name); //печатаем последний узел

}

## *Функция добавления 3 произвольных элементов после каждого четного элемента.*

Найти четный элемент 🡪 в цикле трижды вызвать функцию push(head). В представленном варианте пользователь сам может ввести данные.

**void** **threeEl**(**struct** MusicalComposition\*head)

{

**int** c = **0**;

**int** i;

**while** (head)

{

c++;

**struct** MusicalComposition \* el = head->next;

**if** (c % **2** == **1**)

{

head = head->next;

}

**else**

{

head->next = NULL; //найденный элемент делаем последним

**for** (i = **0**; i < **3**; i++)//трижды повторяем ввод нового элемента

push(head);

**while** (head->next) //идем до последнего введенного элемента

head = head->next;

head->next = el; //ссылаемся на сохраненный элемент, тем самым восстанавливая связь в списке

head = el; //переходим на следующий элемент

}

}

}

## *Функция удаления всех элементов с данной подстрокой в поле автора:*

**void** **rem\_author**(**struct** MusicalComposition\*head, **char**\*name\_for\_remove)

{

**while** (head) { //пока список есть (не закончились узлы)

**if**(strcmp(head->author, name\_for\_remove)==**0**) { //если поле «автор» и искомая подстрока совпали:

**if** (head->previous==NULL) {//если это первый элемент

head->next->previous=NULL;//у следующего элемента обнуляем поле previous

\*head=\*head->next;//переходим на следующий элемент

**return**;//вышли из ветвления

}

**else** **if** (head->next==NULL) {//если элемент последний

head->previous->next=NULL;//обнуляем поле next у предыдущего (т.е. делаем предыдущий элемент последним)

**return**;

}

**else** {//если не в начале и не в конце

head->previous->next=head->next;//переводим поле next предыдущего элемента

head->next->previous=head->previous; //и поле previous следующего так, как будто искомого элемента и не было

}

}

head=head->next; //перемещаемся по списку

}

}

*Makefile:*

*Работа с GitHub:*

**Примеры работы программы**

|  |  |
| --- | --- |
| Ввод | Вывод |

# **Приложение 1 Код программы**

# **Заключение**

Выполнив данную курсовую работу, мы освоили и закрепили на практике написание программы с применением структур в качестве элементов сложных типов представления данных таких, как одно- и дву- направленные линейные списки, а также API для работы с ними, в частности, функции вставки, удаления, подсчета и вывода элементов списка. Повторили работу с проектом в целом: разбиение его на заголовки (header files) и файлы кода (source files), а также написание makefile для сборки программы, так же темы: указатели, динамическая память, массивы и функции стандартных библиотек.