

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МОЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №4 по**  
**дисциплине «Программирование»**  
**ТЕМА: Линейные списки**

Студент гр. 7304

Юсковец А.В.

Преподаватель

Берленко Т.А.

Санкт-Петербург

2017

Оглавление

Цель работы.....3

Ход работы.....4

    Директивы препроцессора.....4

    Определение структуры.....4

    createMusicalComposition.....5

    createMusicalCompositionList.....5

    push.....6

    removeEl.....6

    count.....7

    print\_names.....7

Вывод.....8

## Цель работы

Создайте двунаправленный список музыкальных композиций `MusicalComposition` и **api** (*application programming interface* - 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_years` (`array_years[0]`).

*Аналогично для второго, третьего, ...**n-1**-го элемента массива.*

*! длина массивов `array_names`, `array_authors`, `array_years` одинаковая и равна **n**, это проверять не требуется.*

*Функция возвращает указатель на первый элемент списка.*

- `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

## Ход работы

### ▪ Директивы препроцессора

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

Подключаются соответствующие библиотеки для использования функций: работы со строками, динамической памятью и стандартным вводом-выводом.

### ▪ Определение структуры

```
typedef struct MusicalComposition {
    char* name;
    char* author;
    int year;
    struct MusicalComposition* prev;
    struct MusicalComposition* next;
} MusicalComposition;
```

name – название композиции.

author – имя автора.

year – год создания.

prev – указатель на предыдущий элемент списка.

next - указатель на следующий элемент списка.

## ▪ createMusicalComposition

```
MusicalComposition* createMusicalComposition(char* name, char* author, int year) {
    MusicalComposition* ret = (MusicalComposition*)malloc(sizeof(MusicalComposition));

    ret->name = name;
    ret->author = author;
    ret->year = year;
    ret->prev = NULL;
    ret->next = NULL;

    return ret;
}
```

Динамически выделяется память по композиции и соответствующим полям присваиваются переданные значения. Указатели на prev и next зануляются.

## ▪ createMusicalCompositionList

```
MusicalComposition* createMusicalCompositionList(char** array_names, char** array_authors,
int* array_years, int n) {
    MusicalComposition* head = (MusicalComposition*)malloc(sizeof(MusicalComposition));
    head = createMusicalComposition(array_names[0], array_authors[0], array_years[0]);

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

    MusicalComposition* cur = (MusicalComposition*)malloc(sizeof(MusicalComposition));
    for (int i = 1; i != n; ++i) {
        cur = createMusicalComposition(array_names[i], array_authors[i], array_years[i]);
        cur->prev = prev;
        cur->prev->next = cur;
        cur->next = NULL;
        prev = cur;
    }

    return head;
}
```

head – голова списка. Соответствующая композиция инициализируется первыми элементами массивов.

prev – указатель на предыдущий элемент списка, инициализируется head

cur – текущий, создающийся на данной итерации, элемент списка

В цикле `cur` присваивается указатель на соответствующую музыкальную композицию.

Далее указатель на предыдущий элемент списка становится `prev`, указатель на следующий элемент `prev` становится `cur`, указатель на следующий за `cur` элемент зануляется. `prev` сдвигается на `cur` и цикл повторяется.

Функция возвращает голову списка.

## ▪ **push**

---

```
void push(MusicalComposition* head, MusicalComposition* element) {  
    if (head == NULL) { head = element; return; }  
  
    MusicalComposition* cur = head;  
  
    while (cur->next) { cur = cur->next; }  
    element->prev = cur;  
    cur->next = element;  
    element->next = NULL;  
}
```

---

В случае если список пустой, то голове просто присваивается значение элемента, и функция завершается.

Создается указатель на композицию `cur` и инициализируется `head`.

Далее в цикле доходим до конца списка.

Присваиваем указателю на предыдущий элемента списка элемента `element` последний элемента списка, следующему за “последним” присваиваем указатель `element`, и с помощью `element->next = NULL` показываем, что `element` это конце списка.

## ▪ removeEl

```
void removeEl(MusicalComposition* head, char* name_for_remove) {
    if (head == NULL) { return; }

    MusicalComposition* cur = head;

    if (strcmp(head->name, name_for_remove) == 0) {
        head->next->prev = NULL;
        free(head);
        return;
    }

    while (cur->next) {
        if (strcmp(cur->name, name_for_remove) == 0) {
            cur->prev->next = cur->next;
            cur->next->prev = cur->prev;
            free(cur);
            return;
        }

        cur = cur->next;
    }

    if (strcmp(cur->name, name_for_remove) == 0) {
        cur->prev->next = NULL;
        free(cur);
    }
}
```

В случае если список пустой функция завершается.

Указатель на композицию cur инициализируется head.

Отдельно обрабатывается случай удаления головы и хвоста.

В цикле поле name каждого элемента сравнивается с name\_for\_remove, в случае совпадения этот элемент удаляется: указателю на следующий элемент элемента перед тем, который удаляется присваивается указатель на следующий элемент затем, который удаляется, аналогично передвигается и указатель на предыдущий элемент, следующего за тем, который удаляется.

## ▪ count

```
int count(MusicalComposition* head) {  
    if (head == NULL) { return 0; }  
  
    MusicalComposition* cur = head;  
    int ret = 1;  
    while (cur = cur->next) { ++ret; }  
  
    return ret;  
}
```

В случае, если список пустой функция возвращает 0.  
Указатель на композицию `cur` инициализируется `head`.  
`ret` — значение, возвращаемое функцией.  
Пока цикл не дойдет до конца списка `ret` будет увеличиваться на единицу.

## ▪ print\_names

```
void print_names(MusicalComposition* head) {  
    if (head == NULL) { return ; }  
  
    MusicalComposition* cur = head;  
    do {  
        printf("%s\n", cur->name);  
    } while (cur = cur->next);  
}
```

В случае, если список пустой функция завершается.  
Указатель на композицию `cur` инициализируется `head`.  
Пока цикл не дойдет до конца списка, будет выводиться поле `name` каждого элемента.

## Вывод

Было написано несколько функций, реализующих двунаправленный линейный список.  
При компилировании, сборке и выполнении на предоставленных тестах исполняемого файла не возникает ошибок и предупреждений.