

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

**КУРСОВАЯ РАБОТА**  
**по дисциплине «программирование»**

**Тема: Линейные списки**

Студент гр. 7381

Минуллин М. А.

Преподаватель

Берленко Т. А.

Санкт-Петербург

2017

## ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Минуллин М. А.

Группа 7381

Тема работы: Линейные списки

Содержание пояснительной записки:

Содержание, введение, описание разработки, заметки о работе программы, тестирование, вывод, приложения, список использованных источников.

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 23.11.2017

Дата сдачи реферата: 15.12.2017

Дата защиты реферата: 15.12.2017

Студент

Минуллин М. А.

Преподаватель

Берленко Т. А.

## **АННОТАЦИЯ**

В данной работе был создан проект на языке программирования С, который позволяет работать с набором функций, отвечающих за список музыкальных композиций. Для функционирования списка были созданы и описаны необходимые функции, позволяющие добавлять, удалять, и выводить элементы списка и их количество на консоль, а также описана структура элемента списка. Помимо этого, была проведена работа над оптимизацией исходного кода программы для ускорения ее быстрогодействия и оптимального использования памяти и ресурсов. Приведено полное описание исходного кода.

## СОДЕРЖАНИЕ

|   |    |
|---|----|
| ВВЕДЕНИЕ  | 5  |
| 1. ОПИСАНИЕ РАЗРАБОТКИ                            | 6  |
| 1. 1. Описание структур.                          | 6  |
| 1. 1. 1. Структура MCNode.                        | 6  |
| 1. 1. 2. Структура MCList.                        | 6  |
| 1. 2. Описание функций.                           | 6  |
| 1. 2. 1. Функция createMCNode.                    | 6  |
| 1. 2. 2. Функция freeMCNode.                      | 7  |
| 1. 2. 3. Функция printMCNode.                     | 7  |
| 1. 2. 4. Функция createMCList.                    | 7  |
| 1. 2. 5. Функция freeMCList.                      | 7  |
| 1. 2. 6. Функция insert_head.                     | 7  |
| 1. 2. 7. Функция insert_midd.                     | 8  |
| 1. 2. 8. Функция insert_tail.                     | 8  |
| 1. 2. 9. Функция remove_head.                     | 8  |
| 1. 2. 10. Функция remove_tail.                    | 8  |
| 1. 2. 11. Функция nodes_count.                    | 9  |
| 1. 2. 12. Функция printMCList.                    | 9  |
| 2. ЗАМЕТКИ О РАБОТЕ ПРОГРАММЫ                     | 9  |
| 2. 1. Передача указателей на NULL.                | 9  |
| 2. 2. Передача указателей на чужую память.        | 9  |
| 2. 3. Конструктор MCNode.                         | 10 |
| 2. 4. Добавление музыкальных композиций в список. | 10 |
| 3. ТЕСТИРОВАНИЕ.                                  | 11 |
| 3. 1. Тест №1.                                    | 11 |
| 3. 2. Тест №2.                                    | 11 |
| 3. 3. Тест №3.                                    | 12 |
| 3. 4. Тест №4.                                    | 13 |
| ВЫВОД   | 14 |
| ПРИЛОЖЕНИЯ  | 15 |
| СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ                  | 24 |

## **ВВЕДЕНИЕ**

Цель курсовой работы – изучить понятие структуры в языке программирования C, изучить структуру данных, именуемую списком (одно- и двусвязный): что это такое, для чего, доступные операции для неё, их асимптотическая сложность. Результатом выполнения курсовой работы должен быть проект, содержащий в себе один или несколько заголовочных файлов с описаниями структуры, её членов и методов для работы с ней, дополнительный заголовочный файл, содержащий в себе дополнительную функцию для работы со списком, выданную преподавателем, мейк-файл, необходимый для сборки проекта, а также код для демонстрации работы списка.

## 1. ОПИСАНИЕ РАЗРАБОТКИ.

### 1. 1. Описание структур.

#### 1. 1. 1. Структура MCNode.

**struct MCNode** – структура, экземпляры которой являются элементами двусвязного списка. Содержащая в себе поля музыкальной композиции, а также указатели на соседние MCNode.

Таблица 1.

| Тип                   | Имя    | Описание                                       |
|-----------------------|--------|--|
| <b>char*</b>          | name   | Имя музыкальной композиции                     |
| <b>char*</b>          | author | Автор музыкальной композиции                   |
| <b>Int</b>            | year   | Год публикации музыкальной композиции          |
| <b>struct MCNode*</b> | prev   | Указатель на предыдущую музыкальную композицию |
| <b>struct MCNode*</b> | next   | Указатель на следующую музыкальную композицию  |

#### 1. 1. 2. Структура MCList.

**struct MCList** – структура, являющаяся обёрткой для двусвязного списка. Содержит в себе указатели на элементы двусвязного списка в его начале и конце.

Таблица 2.

| Тип            | Имя  | Описание  |
|----------------|------|---|
| <b>MCNode*</b> | head | Указатель на музыкальную композицию в начале списка |
| <b>MCNode*</b> | tail | Указатель на музыкальную композицию в конце списка  |

### 1. 2. Описание функций.

#### 1. 2. 1. Функция createMCNode.

Назначение:

Конструктор структуры **MCNode**.

Аргументы:

**char\*** name – строка, имя конструируемой музыкальной композиции.

**char\*** author – строка, автор конструируемой музыкальной композиции.

**int** year – целое число, год конструируемой музыкальной композиции.

Возвращаемое значение:

**MCNode\*** – указатель на сконструированный экземпляр музыкальной композиции.

### 1. 2. 2. Функция freeMCNode.

Назначение:

Деструктор структуры **MCNode**, освобождает память, выделенную под музыкальную композицию.

Аргументы:

**MCNode\*** node – указатель на музыкальную композицию.

Возвращаемое значение:

**void** – функция ничего не возвращает.

### 1. 2. 3. Функция printMCNode.

Назначение:

Вывод в консоль информации о музыкальной композиции.

Аргументы:

**MCNode\*** node – указатель на музыкальную композицию.

Возвращаемое значение:

**void** – функция ничего не возвращает.

### 1. 2. 4. Функция createMCList.

Назначение:

Конструктор структуры **MCList**, создаёт пустой список музыкальных композиций.

Аргументы:

Функция не принимает никаких аргументов.

Возвращаемое значение:

**MCList\*** – указатель на сконструированный экземпляр списка музыкальных композиций.

### 1. 2. 5. Функция freeMCList.

Назначение:

Деструктор структуры **MCList**, освобождает память выделенную под список музыкальных композиций и каждую музыкальную композицию отдельно.

Аргументы:

**MCList\*** list – указатель на список музыкальных композиций.

Возвращаемое значение:

**void** – функция ничего не возвращает.

### 1. 2. 6. Функция insert\_head.

Назначение:

Вставка музыкальной композиции в начало списка музыкальных композиций.

Аргументы:

**MCList\*** list – указатель на список музыкальных композиций.

**MCNode\*** node – указатель на вставляемую музыкальную композицию.

Возвращаемое значение:

**MCNode\*** – указатель на музыкальную композицию. Если вставка произведена успешно – указатель на вставляемую музыкальную композицию, иначе – NULL.

#### 1. 2. 7. Функция insert\_midd.

Назначение:

Вставка в середину списка музыкальных композиций трёх музыкальных композиций.

Аргументы:

**MCList\*** list – указатель на список музыкальных композиций.

**MCNode\*** node\_1 – указатель на первую вставляемую музыкальную композицию.

**MCNode\*** node\_2 – указатель на вторую вставляемую музыкальную композицию.

**MCNode\*** node\_3 – указатель на третью вставляемую музыкальную композицию.

Возвращаемое значение:

**MCNode\*** – указатель на музыкальную композицию. Если вставка произведена успешно – указатель на первую вставляемую музыкальную композицию, иначе – NULL.

#### 1. 2. 8. Функция insert\_tail.

Назначение:

Вставка музыкальной композиции в конец списка музыкальных композиций.

Аргументы:

**MCList\*** list – указатель на список музыкальных композиций.

**MCNode\*** node – указатель на вставляемую музыкальную композицию.

Возвращаемое значение:

**MCNode\*** – указатель на музыкальную композицию. Если вставка произведена успешно – указатель на вставляемую музыкальную композицию, иначе – NULL.

#### 1. 2. 9. Функция remove\_head.

Назначение:

Удаление музыкальной композиции из начала списка музыкальных композиций.

Аргументы:

**MCList\*** list – указатель на список музыкальных композиций.

Возвращаемое значение:

**MCNode\*** – указатель на начало списка музыкальных композиций после удаления. Если список был или оказался пустым после удаления музыкальной композиции – NULL.



### 1. 2. 10. Функция `remove_tail`.

Назначение:

Удаление музыкальной композиции из конца списка музыкальных композиций.

Аргументы:

`MCList*` `list` – указатель на список музыкальных композиций.

Возвращаемое значение:

`MCNode*` – указатель на конец списка музыкальных композиций после удаления. Если список был или оказался пустым после удаления музыкальной композиции – `NULL`.

### 1. 2. 11. Функция `nodes_count`.

Назначение:

Подсчёт количества музыкальных композиций в списке музыкальных композиций.

Аргументы:

`MCList*` `list` – указатель на список музыкальных композиций.

Возвращаемое значение:

`int` – целое число, количество музыкальных композиций в списке.

### 1. 2. 12. Функция `printMCList`.

Назначение:

Вывод в консоль информации обо всех музыкальных композициях, содержащихся в списке.

Аргументы:

`MCList*` `list` – указатель на список музыкальных композиций.

Возвращаемое значение:

`void` – функция ничего не возвращает.

## 2. ЗАМЕТКИ О ЛОГИКЕ РАБОТЫ ПРОГРАММЫ

### 2. 1. Передача указателей на `NULL`.

В функции вставки музыкальной композиции в список является допустимым передача указателей на `NULL`. В данном случае никаких действий произведено не будет, на этапе проверки переданного указателя произойдёт выход из функции. Функция также вернёт указатель на `NULL`.

Освобождение памяти списка или отдельной музыкальной композиции в случае, если будет передан `NULL`, ошибок не вызывает, поскольку происходит выход из функции на этапе проверки входных параметров.

### 2. 2. Передача указателей на чужую память.

В случае, если указатель не NULL, но память по данному адресу используется другим приложением, то функции освобождения памяти для списка или отдельной музыкальной композиции вызовут segmentation fault. Следовательно, после освобождения памяти, указатели необходимо обнулять вручную, чтобы при их передаче не происходило ошибок.

Такое же ограничение накладывается на вставку музыкальных композиций в список.

Это же ограничение распространяется на передачу недействительного адреса списка музыкальных композиций абсолютно во всех функциях, где он используется.

### 2. 3. Конструктор MCNode.

Конструктор музыкальной композиции принимает указатели на символ (строки). В конструкторе выделяется память, а затем копируются именно эти строки, а не указатели на них. Поэтому переданные строки в дальнейшем можно изменять, данные в музыкальной композиции при этом изменяться не будут.

### 2. 4. Добавление музыкальных композиций в список.

В отличие от конструктора MCNode, копируются именно указатели на музыкальные композиции. Это сделано в связи с тем, что список в данном проекте является обёрткой для музыкальных композиций. Формально, тот же список можно построить и на 2 переменных-указателях на музыкальную композицию. При вставке в список музыкальной композиции рекомендуется обнулять значение переданного указателя на музыкальную композицию, чтобы избежать изменения данных в списке извне, если в этом нет реальной необходимости. Освободить при этом память нельзя – этим должен заниматься сам список. Освобождение памяти вставляемой композиции приведёт к segmentation fault при освобождении списка.

### 3. ТЕСТИРОВАНИЕ.

#### 3. 1. Тест №1.



Рис 1. — демонстрация выполнения 1-ого теста.

Попытка удаления музыкальной композиции из начала или конца пустого списка не приводит к завершению работы программы. Вывод пустого списка не производится. Количество элементов в пустом списке определено верно и равно нулю. Освобождение памяти для пустого списка при выходе из программы производится корректно, не происходит ошибки segmentation fault.

#### 3. 2. Тест №2.

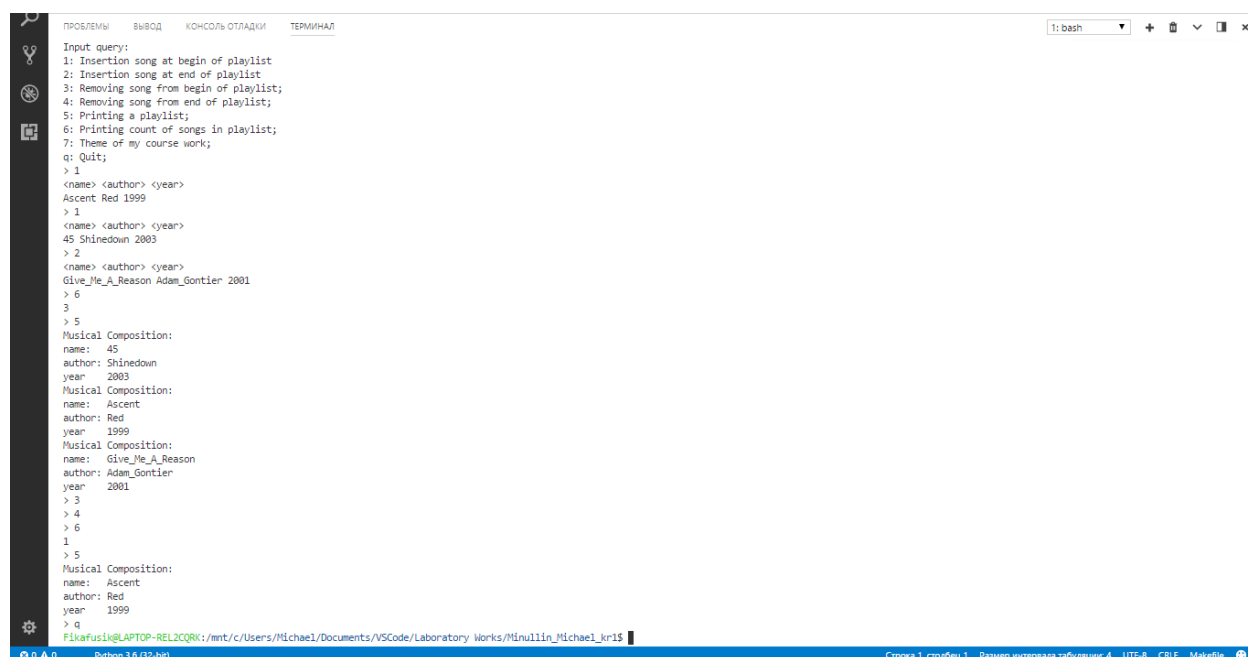
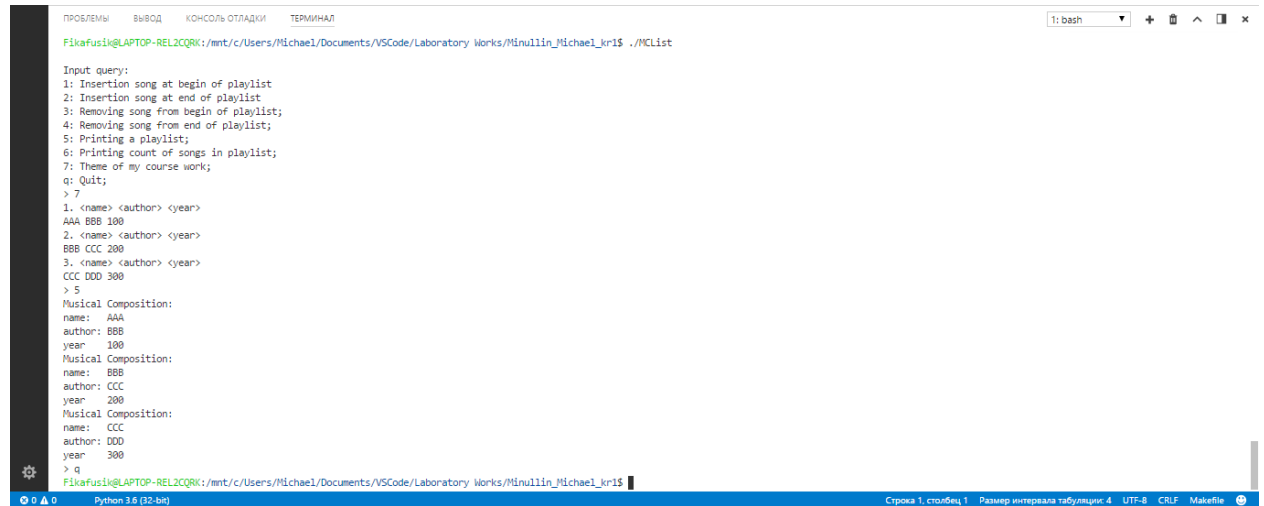


Рис 2. — демонстрация выполнения 2-ого теста.

Добавление двух музыкальных композиций в начало списка и один в конец. Функция определения количества музыкальных композиций в списке отработала корректно (три композиции). Вывод информации обо всех музыкальных композициях также произведён в правильном порядке, согласно указанному порядку вставки с учётом вставки в начало-конец списка. Удаление одной композиции из начала списка и ещё одной из конца. Вывод количества оставшихся композиций в списке (одна), а также информации об этой композиции (остался центральный элемент списка). Завершение работы приложения,

освобождение памяти из-под оставшейся музыкальной композиции. Segmentation fault отсутствует.

### 3. 3. Тест № 3.



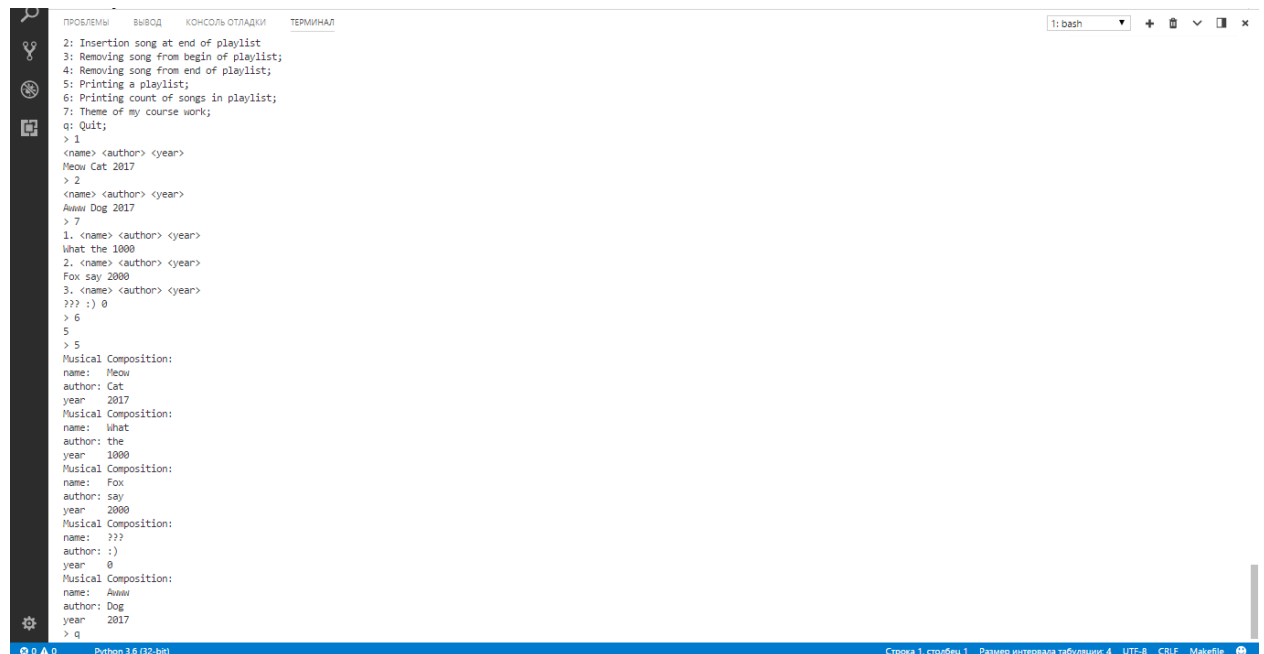
```
PROБЛЕМЫ Вывод КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ
FikaFusik@LAPTOP-REL2CQRK:/mnt/c/Users/Michael/Documents/VSCode/Laboratory Works/Minullin_Michael_kr15 ./MCTest

Input query:
1: Insertion song at begin of playlist
2: Insertion song at end of playlist
3: Removing song from begin of playlist;
4: Removing song from end of playlist;
5: Printing a playlist;
6: Printing count of songs in playlist;
7: Theme of my course work;
q: Quit;
> 7
1. <name> <author> <year>
AAA BBB 100
2. <name> <author> <year>
BBB CCC 200
3. <name> <author> <year>
CCC DDD 300
> 5
Musical Composition:
name: AAA
author: BBB
year: 100
Musical Composition:
name: BBB
author: CCC
year: 200
Musical Composition:
name: CCC
author: DDD
year: 300
> q
FikaFusik@LAPTOP-REL2CQRK:/mnt/c/Users/Michael/Documents/VSCode/Laboratory Works/Minullin_Michael_kr15
```

Рис 3. — демонстрация выполнения 3-ого теста.

Использование функции вставки трёх музыкальных композиций в середину пустого списка музыкальных композиций, вывод списка, освобождение памяти. Segmentation fault отсутствует.

### 3. 4. Тест №4.



```
PROБЛЕМЫ Вывод КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ
FikaFusik@LAPTOP-REL2CQRK:/mnt/c/Users/Michael/Documents/VSCode/Laboratory Works/Minullin_Michael_kr15 ./MCTest

2: Insertion song at end of playlist
3: Removing song from begin of playlist;
4: Removing song from end of playlist;
5: Printing a playlist;
6: Printing count of songs in playlist;
7: Theme of my course work;
q: Quit;
> 1
<name> <author> <year>
Meow Cat 2017
> 2
<name> <author> <year>
Aww Dog 2017
> 7
1. <name> <author> <year>
What the 1000
2. <name> <author> <year>
Fox say 2000
3. <name> <author> <year>
??? :) 0
> 6
5
> 5
Musical Composition:
name: Meow
author: Cat
year: 2017
Musical Composition:
name: What
author: the
year: 1000
Musical Composition:
name: Fox
author: say
year: 2000
Musical Composition:
name: ???
author: :)
year: 0
Musical Composition:
name: Aww
author: Dog
year: 2017
> q
FikaFusik@LAPTOP-REL2CQRK:/mnt/c/Users/Michael/Documents/VSCode/Laboratory Works/Minullin_Michael_kr15
```

Рис 1. — демонстрация выполнения 4-ого теста.

Вставка в начало и конец списка музыкальных композиций по одной музыкальной композиции. Использование функции вставки трёх музыкальных композиций в непустой список музыкальных композиций. Вывод количества композиций в списке, а также

информации обо всех музыкальных композициях для проверки работоспособности функций вставки. Освобождение памяти также без segmentation fault.

## ВЫВОД

В ходе выполнения курсовой работы, были закреплены навыки программирования на языке C, были изучены понятие структуры `struct` в языке C, структура данных двусвязный список, её поддерживаемый операции (вставка, удаление, поиск), их асимптотика (вставка и удаление элемента в списке производится за константное время, однако, оно может быть и линейным, если точно не известен адрес удаляемого элемента или элемента, возле которого необходимо произвести вставку. Связано это с тем, что поиск в двусвязном списке осуществляется за линейное время). Были закреплены навыки работы с утилитой `make`, а также системой контроля версий `github`.

## ПРИЛОЖЕНИЯ

### Приложение А.

#### Файл "Makefile"

```
obj = main.o MCNode.o MCList.o
exe = MCList

all: $(obj)
    gcc -o $(exe) $(obj)

test.o: main.c MCNode.h MCList.h
    gcc -c main.c

MCNode.o: MCNode.c MCNode.h
    gcc -c MCNode.c

MCList.o: MCList.c MCList.h MCNode.h
    gcc -c MCList.c

clean:
    rm $(obj) $(exe)
```

### Приложение Б.

#### Файл "MCNode.h"

```
#ifndef __MCNode_H__
#define __MCNode_H__

extern const int NAME_MAX_LENGTH;
extern const int AUTHOR_MAX_LENGTH;

struct MCNode {
    char* name;
    char* author;
    int year;
    struct MCNode* prev;
    struct MCNode* next;
};

typedef struct MCNode MCNode;
```

```
MCNode* createMCNode(char*, char*, int);
```

```
void freeMCNode(MCNode*);
```

```
void printMCNode(MCNode*);
```

```
#endif
```

Приложение В.

Файл "MCNode.c"

```
#include "MCNode.h"
```

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

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

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

```
const int NAME_MAX_LENGTH      = 80;
```

```
const int AUTHOR_MAX_LENGTH    = 80;
```

```
MCNode* createMCNode(char* name, char* author, int year) {  
    MCNode* node = (MCNode*)malloc(sizeof(MCNode));  
    node->name = (char*)malloc((NAME_MAX_LENGTH + 1) * sizeof(char));  
    strcpy(node->name, name);  
    node->author = (char*)malloc((AUTHOR_MAX_LENGTH + 1) * sizeof(char));  
    strcpy(node->author, author);  
    node->year = year;  
    node->next = NULL;  
    node->prev = NULL;  
    return node;  
}
```

```
void freeMCNode(MCNode* node) {  
    if (!node)  
        return;  
    free(node->name);  
    free(node->author);  
    free(node);  
}
```



```

void printMCNode(MCNode* node) {
    if (!node)
        return;
    printf("Musical Composition:\nname:\t%s\nauthor:\t%s\neyear\t%d\n", node-
>name, node->author, node->year);
}

```

Приложение Г.

Файл "MCList.h"

```

#ifndef __MCLIST_H__
#define __MCLIST_H__

struct MCList {
    MCNode* head;
    MCNode* tail;
};

typedef struct MCList MCList;

MCList* createMCList();
void freeMCList(MCList*);

MCNode* insert_head(MCList*, MCNode*);
MCNode* insert_midd(MCList*, MCNode*, MCNode*, MCNode*);
MCNode* insert_tail(MCList*, MCNode*);

MCNode* remove_head(MCList*);
MCNode* remove_tail(MCList*);

int nodes_count(MCList*);

void printMCList(MCList*);

#endif

```

Приложение Д.

Файл "MCList.c"

```

#include "MCNode.h"
#include "MCList.h"

```

```

#include <stdlib.h>

MCList* createMCList() {
    MCList* list = (MCList*)malloc(sizeof(MCList));
    list->head = NULL;
    list->tail = NULL;
    return list;
}

void freeMCList(MCList* list) {
    if (!list)
        return;
    if (!list->head && !list->tail) {
        free(list);
        return;
    }
    if (list->head == list->tail)
        freeMCNode(list->head);
    else {
        MCNode* iter;
        while (list->head) {
            iter = list->head;
            list->head = list->head->next;
            freeMCNode(iter);
        }
    }
    free(list);
}

MCNode* insert_head(MCList* list, MCNode* node) {
    if (!list || !node)
        return NULL;
    if (!list->head && !list->tail) {
        list->head = list->tail = node;
        list->head->next = list->tail;
        list->tail->prev = list->head;
        return list->head;
    }
}

```

```

    if (list->head == list->tail) {
        list->head = node;
        list->head->next = list->tail;
        list->tail->prev = list->head;
    }
    else {
        list->head->prev = node;
        list->head->prev->next = list->head;
        list->head = node;
    }
    return list->head;
}

```

```

MCNode* insert_midd(MCList* list, MCNode* node_1, MCNode* node_2, MCNode*
node_3) {
    if (!list || !node_1 || !node_2 || !node_3)
        return NULL;
    node_1->next = node_2;
    node_2->prev = node_1;
    node_2->next = node_3;
    node_3->prev = node_2;
    if (!list->head && !list->tail) {
        node_1->prev = NULL;
        list->head = node_1;
        list->tail = node_3;
        node_3->next = NULL;
        return node_1;
    }
    if (list->head == list->tail) {
        node_1->prev = NULL;
        list->head = node_1;
        list->tail->prev = node_3;
        node_3->next = list->tail;
    }
    else {
        int midd = nodes_count(list) >> 1;
        MCNode* iter = list->head;
        for (int i = 1; i < midd; ++i)
            iter = iter->next;
    }
}

```

```

        MCNode* iter_next = iter->next;
        node_1->prev = iter;
        iter->next = node_1;
        iter_next->prev = node_3;
        node_3->next = iter_next;
    }
    return node_1;
}

MCNode* insert_tail(MCList* list, MCNode* node) {
    if (!list || !node)
        return NULL;
    if (!list->head && !list->tail) {
        list->tail = list->head = node;
        list->tail->prev = list->head;
        list->head->next = list->tail;
        return list->tail;
    }
    if (list->head == list->tail) {
        list->tail = node;
        list->head->next = list->tail;
        list->tail->prev = list->head;
    }
    else {
        list->tail->next = node;
        list->tail->next->prev = list->tail;
        list->tail = node;
    }
    return list->tail;
}

MCNode* remove_head(MCList* list) {
    if (!list || !list->head)
        return NULL;
    if (list->head == list->tail) {
        freeMCNode(list->head);
        return list->head = list->tail = NULL;
    }
    list->head = list->head->next;

```

```

        freeMCNode(list->head->prev);
        list->head->prev = NULL;
        return list->head;
    }

MCNode* remove_tail(MCList* list) {
    if (!list || !list->tail)
        return NULL;
    if (list->head == list->tail) {
        freeMCNode(list->tail);
        return list->tail = list->head = NULL;
    }
    list->tail = list->tail->prev;
    freeMCNode(list->tail->next);
    list->tail->next = NULL;
    return list->tail;
}

int nodes_count(MCList* list) {
    if (!list)
        return 0;
    if (!list->head && !list->tail)
        return 0;
    if (list->head == list->tail)
        return 1;
    int count = 1;
    MCNode* iter;
    for (MCNode* iter = list->head; iter != list->tail; iter = iter->next)
        count++;
    return count;
}

void printMCList(MCList* list) {
    if (!list)
        return;
    for (MCNode* iter = list->head; iter != list->tail; iter = iter->next)
        printMCNode(iter);
    printMCNode(list->tail);
}

```

## Приложение Е.

Файл "main.c"

```
#include "MCNode.h"
#include "MCList.h"

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

int main() {
    MCList* playlist = createMCList();

    MCNode* song_1;
    MCNode* song_2;
    MCNode* song_3;

    char name[NAME_MAX_LENGTH];
    char author[AUTHOR_MAX_LENGTH];
    int year;

    char query;

    printf("\nInput query:\n1: Insertion song at begin of playlist\n2:
Insertion song at end of playlist\n3: Removing song from begin of playlist;\n4:
Removing song from end of playlist;\n5: Printing a playlist;\n6: Printing count
of songs in playlist;\n7: Theme of my course work;\nq: Quit;\n");
    do {
        printf("> ");
        scanf(" %c", &query);
        switch (query) {
            case '1':
                printf("<name> <author> <year>\n");
                scanf("%80s %80s %d", name, author, &year);
                insert_head(playlist, createMCNode(name, author, year));
                break;
            case '2':
                printf("<name> <author> <year>\n");
                scanf("%80s %80s %d", name, author, &year);
                insert_tail(playlist, createMCNode(name, author, year));
```

```

        break;
    case '3':
        remove_head(playlist);
        break;
    case '4':
        remove_tail(playlist);
        break;
    case '5':
        printMCList(playlist);
        break;
    case '6':
        printf("%d\n", nodes_count(playlist));
        break;
    case '7':
        printf("1. <name> <author> <year>\n");
        scanf("%s %s %d", name, author, &year);
        song_1 = createMCNode(name, author, year);
        printf("2. <name> <author> <year>\n");
        scanf("%s %s %d", name, author, &year);
        song_2 = createMCNode(name, author, year);
        printf("3. <name> <author> <year>\n");
        scanf("%s %s %d", name, author, &year);
        song_3 = createMCNode(name, author, year);
        insert_mid(playlist, song_1, song_2, song_3);
        break;
    case 'q':
        break;
    default:
        printf("Unknown query; Repeat input:\n");
    }
} while (query != 'q');

freeMCList(playlist);

return 0;
}

```

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

«Язык программирования С» Брайан Керниган, Деннис Ритчи.

«97 Things Every Programmer Should Know» Kevlin Henney.

«C elements of style» Steve Qualline.