**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. 6304 |  | Попов С.С. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2016

**ЗАДАНИЕ  
на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Попов С.С. | | |
| Группа 6304 | | |
| Тема работы: Линейные списки | | |
| Содержание пояснительной записки:   * Содержание * Введение * Теоретические сведения * Описание функций для работы со списком * Описание функций для разделения, копирования и вставки списка * Меню * Примеры работы программы * Заключение * Список использованных источников * Приложение А: Исходный код программы | | |
| Дата выдачи задания: 07.12.2016 | | |
| Дата сдачи реферата: \_\_\_\_\_\_\_\_\_\_ | | |
| Дата защиты реферата: \_\_\_\_\_\_\_\_\_ | | |
| Студент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Попов С.С. |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Берленко Т.А. |

**Аннотация**

В данной работе была разработана программа на языке С для работы со списком музыкальных композиций. Был разработан набор функций для работы со списком, который позволяет создавать, добавлять, удалять, считать и выводить на консоль элементы списка, а также разделять, копировать и вставлять этот список. Была описана структура элемента списка. Помимо этого, была проведена работа по оптимизации исходного кода программы для улучшения её быстродействия и оптимального использования ресурсов компьютера.

Содержание

[Введение 5](#_Toc470721660)

[1. Теоретические сведения 6](#_Toc470721661)

[2. Функции для работы со списком 6](#_Toc470721662)

[2.1 Создание элемента 6](#_Toc470721663)

[2.2 Создание двунаправленного списка музыкальных композиций 7](#_Toc470721664)

[2.3 Добавление нового элемента в конец списка 7](#_Toc470721665)

[2.4 Удаление элемента из списка 8](#_Toc470721666)

[2.5 Подсчёт количества элементов в списке 9](#_Toc470721667)

[2.6 Вывод имён элементов списка на консоль 9](#_Toc470721668)

[3. Функции для обмена местами первого и последнего элемента, функции для разделения, копирования и вставки списка 10](#_Toc470721669)

[3.1 Обмен местами первого и последнего элементов списка 10](#_Toc470721670)

[3.2 Разделение списка на две половины 11](#_Toc470721671)

[3.3 Копирование списка 12](#_Toc470721672)

[3.4 Объединение двух списков в один 12](#_Toc470721673)

[4. Меню 13](#_Toc470721674)

[5. Пример работы программы 13](#_Toc470721675)

[Заключение 15](#_Toc470721676)

[Список использованных источников 16](#_Toc470721677)

[Приложение А Исходный код программы 17](#_Toc470721678)

[Файлы заголовков 17](#_Toc470721679)

[main.h 17](#_Toc470721680)

[mc\_functionc.h 17](#_Toc470721681)

[Файлы .c 19](#_Toc470721682)

[main.c 19](#_Toc470721683)

[mc\_functionc.c 24](#_Toc470721684)

[Makefile 32](#_Toc470721685)

[Music.txt (файл, содержащий информацию для ввода для проверки работы программы) 33](#_Toc470721686)

# Введение

Необходимо создать двунаправленный список музыкальных композиций MusicalComposition и **api** (**a**pplication **p**rogrammin **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 для работы со списком

* Создание элемента типа MusicalComposition
* Создание двунаправленного списка связанных элементов типа MusicalComposition
* Добавление нового элемента в конец списка
* Удаление элемента из списка
* Подсчёт количества элементов в списке
* Вывод имён элементов списка на консоль
* Обмен местами первого и последнего элементов списка
* Разделение списка на две половины
* Копирование списка
* Объединение двух списков в один

# Теоретические сведения

Каждый узел двунаправленного (двусвязного) линейного списка (ДЛС) содержит два поля указателей - на следующий и на предыдущий узлы. Указатель на предыдущий узел корня списка содержит нулевое значение. Указатель на следующий узел последнего узла также содержит нулевое значение.

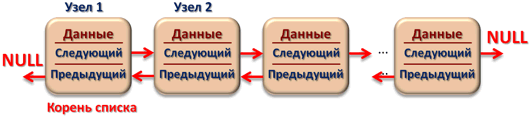
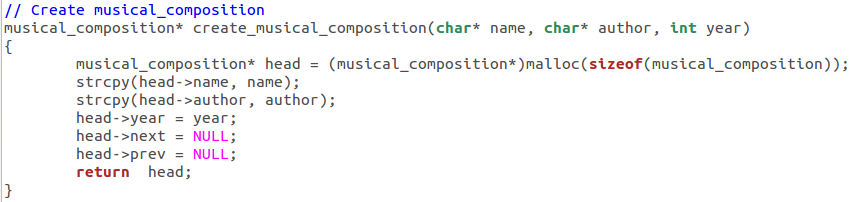


Рисунок 1.1 – Двунаправленный список [2]

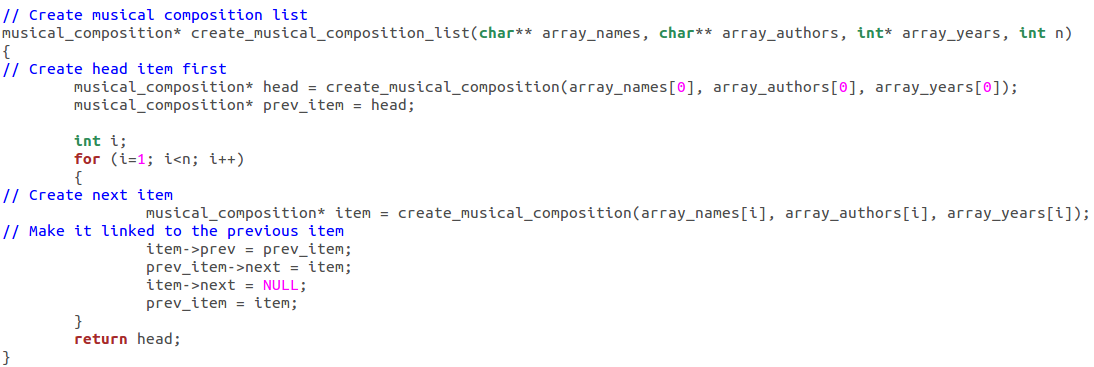
# Функции для работы со списком

## **Создание элемента**



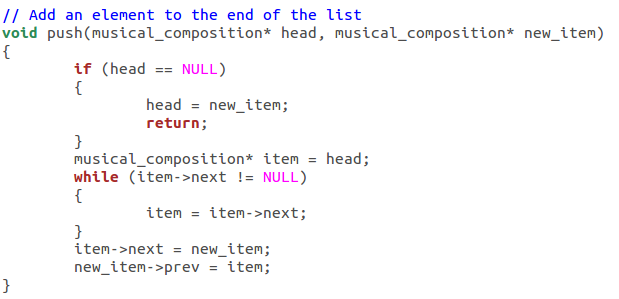
Функция принимает в качестве аргументов указатели на название композиции (char\* name) и её автора (char\* author), а также год создания (int year). Происходит выделение памяти для структуры типа MusicalComposition, и заполняются её переменные. А также добавляются указатели на предыдущий (head->prev = NULL) и следующий (head->next = NULL) элементы списка, которые будут заполнены при создании списка музыкальных композиций.

## Создание двунаправленного списка музыкальных композиций



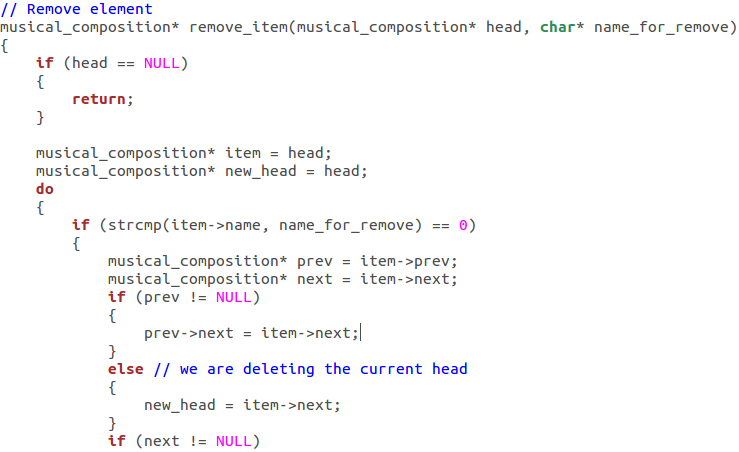
Функция принимает в качестве аргументов указатели на указатель на массив названий композиций (char\*\* array\_names) и их авторов (char\*\* array\_authors), указатель на массив лет создания (int\* array\_years), размер массива (int n). Сначала создаётся первый элемент списка (head) и происходит заполнение его элементов данными. Затем аналогичные операции выполняются для всех последующих n-1 элементов. А также заполняются указатели на предыдущий (head->prev) и следующий (head->next) элементы списка значениями адресов соответствующих элементов.

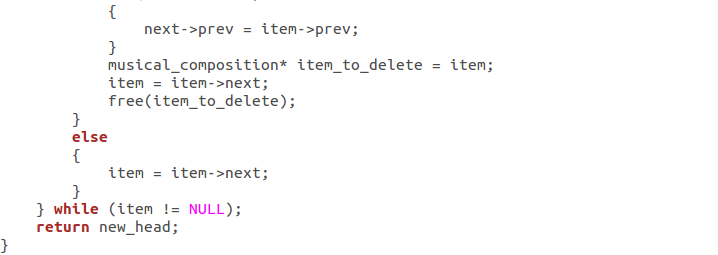
## Добавление нового элемента в конец списка



Функция принимает в качестве аргументов указатель на первый элемент списка (musical\_composition\* head) и указатель на элемент (musical\_composition\* new\_item), который необходимо добавить в конец списка. Если список в данный момент пуст, то новый элемент становится первым элементом списка. В противном случае происходит поиск конца списка и новый элемент добавляется после последнего.

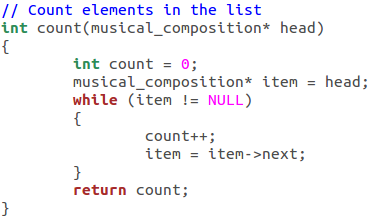
## Удаление элемента из списка





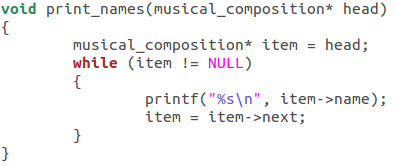
Функция принимает в качестве аргументов указатель на первый элемент списка (musical\_composition\* head) и указатель на элемент (musical\_composition\* name\_for\_remove), который необходимо удалить из списка. Функция проходит по списку и сравнивает название элементов с названием элемента для удаления. В функции предусмотрена защита от удаления, если список пуст, а также учтена возможность удаления первого элемента (в этом случае следующий элемент становится первым).

## Подсчёт количества элементов в списке



Функция принимает в качестве аргумента первый элемент списка (musical\_composition\* head). Она проходит по всему списку, считая элементы, пока следующий элемент не будет равен нулю (NULL).

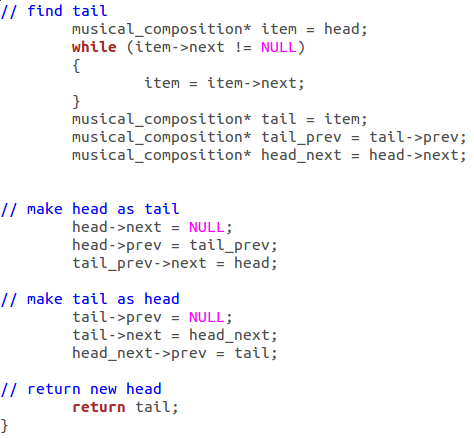
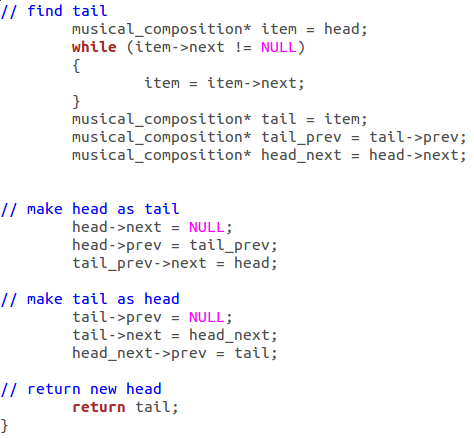
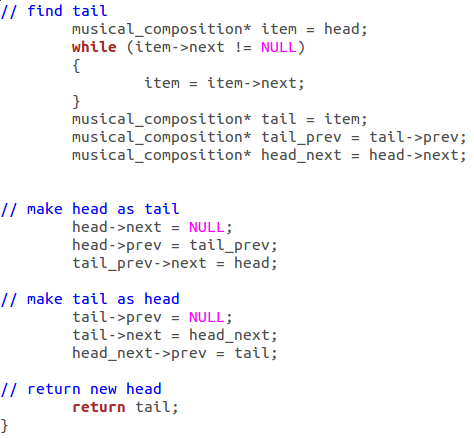
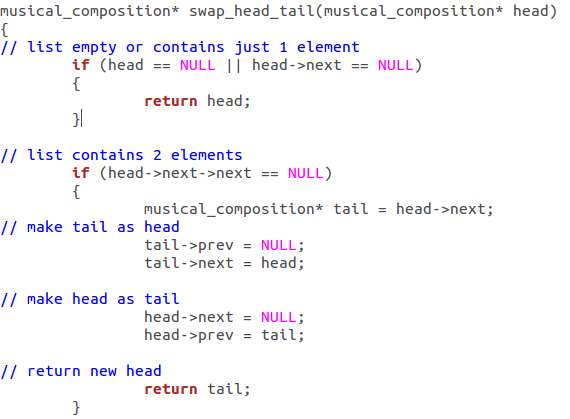
## Вывод имён элементов списка на консоль



Функция принимает в качестве аргумента первый элемент списка (musical\_composition\* head). Она проходит по всему списку, выводя имена композиций на консоль.

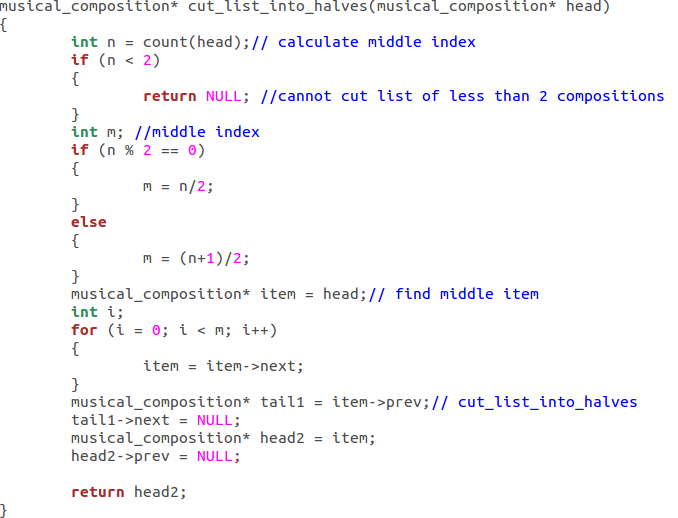
# Функции для обмена местами первого и последнего элемента, функции для разделения, копирования и вставки списка

## Обмен местами первого и последнего элементов списка



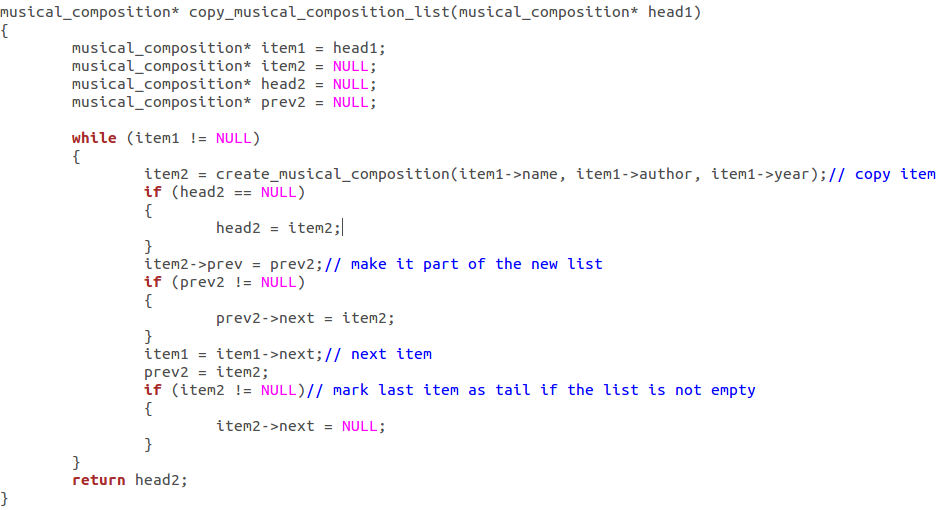
Функция принимает в качестве аргумента первый элемент списка (musical\_composition\* head). Происходит поиск второго и предпоследнего элемента, и с помощью дополнительных переменных и перестановки указателей первый элемент списка становится последним, а последний становится первым. Предусмотрена защита от ситуации, когда список пуст или состоит из одного элемента.

## Разделение списка на две половины



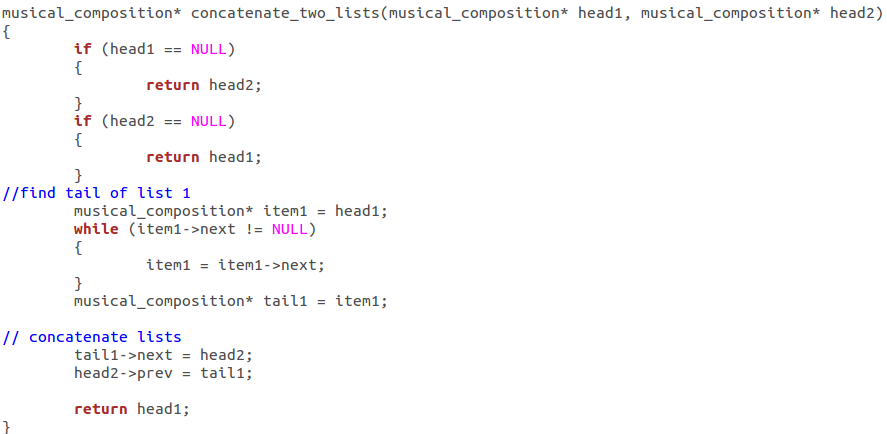
Функция принимает в качестве аргумента первый элемент списка (musical\_composition\* head). Происходит поиск центрального элемента с округлением индекса элемента в большую сторону, затем разделение списка на две части. Функция возвращает первый элемент второй части списка.

## Копирование списка



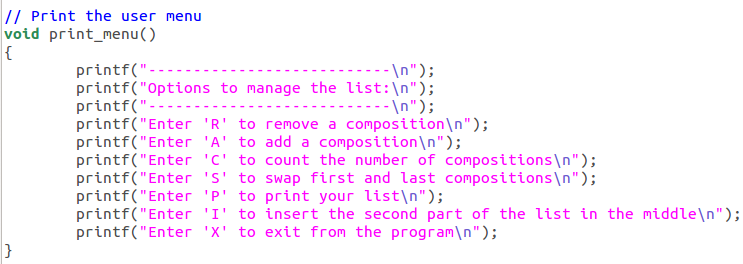
Функция принимает в качестве аргумента первый элемент списка (musical\_composition\* head). Происходит создание точной копии элементов и связывание их в новый список. Функция возвращает первый элемент копии списка.

## Объединение двух списков в один



Функция принимает в качестве аргуменов первые элементы списков, которые нужно объединить (musical\_composition\* head1), (musical\_composition\* head2). Путём расстановки указателей у граничных элементов списков мы получаем единый список.

# Меню



Функция выводит на консоль пользовательское меню.

# Пример работы программы

Для демонстрации работоспособности программы был создан файл Music.txt, который содержит в себе список музыкальных композиций.

*Таблица 5.1 – Ввод - Вывод*

|  |  |
| --- | --- |
| Ввод | Вывод |
| 5  N1 Born This Way  A1 Thousand Foot Krutch  2001  N2 Caravan  A2 Justin Hurwitz & Tim Simonec | =======================================  Welcome to Musical Composition List Manager!  =======================================  The list is:  N1 Born This Way  N2 Caravan  N2 Caravan  N3 Whiplash |

*Продолжение таблицы 5.1*

|  |  |
| --- | --- |
| 2002  N2 Caravan  A2.1 Justin Hurwitz & Tim Simonec  20021  N3 Whiplash  A3 Hank Levy  2003  N4 Breath  A4 Breaking Benjamin  2004 | N4 Breath  ---------------------------  Options to manage the list:  ---------------------------  Enter 'R' to remove a composition  Enter 'A' to add a composition  Enter 'C' to count the number of compositions  Enter 'S' to swap first and last compositions  Enter 'P' to print your list  Enter 'I' to insert the second part of the list in the middle  Enter 'X' to exit from the program |
| R | Enter the name of composition to remove |
| N2 Caravan | Composition removed |
| P | N1 Born This Way  N3 Whiplash  N4 Breath |
| S | Swap completed |
| P | N4 Breath  N3 Whiplash  N1 Born This Way |

# Заключение

В ходе работы над программой был изучен и закреплен на практике такой элемент программирования на языке С, как линейные двунаправленные списки.

Была реализована полноценная программа для работы со списком музыкальных композиций, которая включает в себя набор функций, позволяющих управлять списком.

Также был реализован пользовательский интерфейс через меню, который позволяет пользователю вызывать необходимые функции программы, не прерывая её работы.

Для удобства тестирования программы был разработан механизм считывания тестовых данных из внешнего файла.

# Список использованных источников

1. Язык С / Б.В. Керниган, Д.М. Ричи

2. Двусвязный линейный список [Электронный ресурс] Справочно-информационный материал, 2016.— Режим доступа:

[http://prog-cpp.ru/data-dls/](http://prog-cpp.ru/data-dls/%20) (дата обращения: 28.12.2016)

# **Приложение А Исходный код программы**

## Файлы заголовков

### main.h

// Define musical\_composition structure

typedef struct musical\_composition

{

char name[80];

char author[80];

int year;

struct musical\_composition\* next;

struct musical\_composition\* prev;

} musical\_composition;

// Print the user menu

void print\_menu();

### mc\_functionc.h

#include "main.h"

// Create musical composition

musical\_composition\* create\_musical\_composition(char\* name, char\* author, int year);

// Create list of musical compositions

musical\_composition\* create\_musical\_composition\_list(char\*\* array\_names, char\*\* array\_authors, int\* array\_years, int n);

// Copy list of musical compositions

musical\_composition\* copy\_musical\_composition\_list(musical\_composition\* head1);

// Add new musical composition to the end of the list

void push(musical\_composition\* head, musical\_composition\* new\_item);

// Remove musical composition by the name

musical\_composition\* remove\_item(musical\_composition\* head, char\* name\_for\_remove);

// Count musical compositions in the list

int count(musical\_composition\* head);

// Print names of musical compositions in the list

void print\_names(musical\_composition\* head);

// Swap tail and head in the list

musical\_composition\* swap\_head\_tail(musical\_composition\* head);

// Cut list into halves

musical\_composition\* cut\_list\_into\_halves(musical\_composition\* head);

// Concatenate two lists

musical\_composition\* concatenate\_two\_lists(musical\_composition\* head1, musical\_composition\* head2);

## Файлы .c

### main.c

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <stdbool.h>

#include "mc\_functions.h"

// Print the user menu

void print\_menu()

{

printf("---------------------------\n");

printf("Options to manage the list:\n");

printf("---------------------------\n");

printf("Enter 'R' to remove a composition\n");

printf("Enter 'A' to add a composition\n");

printf("Enter 'C' to count the number of compositions\n");

printf("Enter 'S' to swap first and last compositions\n");

printf("Enter 'P' to print your list\n");

printf("Enter 'I' to insert the second part of the list in the middle\n");

printf("Enter 'X' to exit from the program\n");

}

// Main function

int main ( int argc, char \*argv[] )

{

printf("============================================\n");

printf("Welcome to Musical Composition List Manager!\n");

printf("============================================\n");

if ( argc != 2 ) // argc should be 2 for correct execution

{

printf("Please use filename as the only 1 argument\n");

return 0;

}

// We assume argv[1] is a filename to open

FILE\* file = fopen( argv[1], "r" );

// fopen returns NULL pointer on failure

if ( file == NULL )

{

printf("Could not open file\n");

return 0;

}

int length;

fscanf(file, "%d\n", &length);

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

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

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

int i;

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

{

char name[80];

char author[80];

fgets(name, 80, file);

fgets(author, 80, file);

fscanf(file, "%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);

}

fclose(file);

musical\_composition\* head = create\_musical\_composition\_list(names, authors, years, length);

printf("The list is:\n");

print\_names(head);

print\_menu();

char c;

int ch;

while (1)

{

c = getchar();

// just remove \n characters left in input stream

while ( (ch = getchar()) != EOF && ch != '\n') { }

switch (c)

{

case 'r':

case 'R':

{

// read name to remove

printf("Enter the name of composition to remove\n");

char name\_for\_remove[80];

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

// replace trailing \n with 0;

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

// remove item

head = remove\_item(head, name\_for\_remove);

printf("Composition removed\n");

break;

}

case 'a':

case 'A':

{

// read name to push

printf("Enter the name of composition to add\n");

char name\_for\_push[80];

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

// replace trailing \n with 0;

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

// read author to push

printf("Enter the author of composition to add\n");

char author\_for\_push[80];

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

// replace trailing \n with 0;

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

// read year to push

printf("Enter the year of composition to add\n");

char year\_for\_push\_str[80];

fgets(year\_for\_push\_str, 80, stdin);

// replace trailing \n with 0;

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

// convert to int

int year\_for\_push = atoi(year\_for\_push\_str);

// create new composition and push it to the list

musical\_composition\* element\_for\_push = create\_musical\_composition(name\_for\_push, author\_for\_push, year\_for\_push);

push(head, element\_for\_push);

printf("Composition added\n");

break;

}

case 'c':

case 'C':

{

printf("Current list contains %d compositions\n", count(head));

break;

}

case 's':

case 'S':

{

head = swap\_head\_tail(head);

printf("Swap completed\n");

break;

}

case 'p':

case 'P':

{

print\_names(head);

break;

}

case 'i':

case 'I':

{

// cut list into halves

musical\_composition\* head2 = cut\_list\_into\_halves(head);

// copy the second half

musical\_composition\* head3 = copy\_musical\_composition\_list(head2);

// concatenate 3 lists together

head = concatenate\_two\_lists(head, head2);

head = concatenate\_two\_lists(head, head3);

printf("Insert completed\n");

break;

}

case 'x':

case 'X':

{

printf("=======\n");

printf("Bye-Bye\n");

printf("=======\n");

return 0;

}

case '\n':

break;

default:

{

printf("You've entered invalid character, try again\n");

print\_menu();

}

}

}

return 0;

}

### mc\_functionc.c

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include "mc\_functions.h"

// Create musical\_composition

musical\_composition\* create\_musical\_composition(char\* name, char\* author, int year)

{

musical\_composition\* head = (musical\_composition\*)malloc(sizeof(musical\_composition));

strcpy(head->name, name);

strcpy(head->author, author);

head->year = year;

head->next = NULL;

head->prev = NULL;

return head;

}

// Create musical composition list

musical\_composition\* create\_musical\_composition\_list(char\*\* array\_names, char\*\* array\_authors, int\* array\_years, int n)

{

// Create head item first

musical\_composition\* head = create\_musical\_composition(array\_names[0], array\_authors[0], array\_years[0]);

musical\_composition\* prev\_item = head;

int i;

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

{

// Create next item

musical\_composition\* item = create\_musical\_composition(array\_names[i], array\_authors[i], array\_years[i]);

// Make it linked to the previous item

item->prev = prev\_item;

prev\_item->next = item;

item->next = NULL;

prev\_item = item;

}

return head;

}

musical\_composition\* copy\_musical\_composition\_list(musical\_composition\* head1)

{

musical\_composition\* item1 = head1;

musical\_composition\* item2 = NULL;

musical\_composition\* head2 = NULL;

musical\_composition\* prev2 = NULL;

while (item1 != NULL)

{

item2 = create\_musical\_composition(item1->name, item1->author, item1->year);// copy item

if (head2 == NULL)

{

head2 = item2;

}

item2->prev = prev2;// make it part of the new list

if (prev2 != NULL)

{

prev2->next = item2;

}

item1 = item1->next;// next item

prev2 = item2;

if (item2 != NULL)// mark last item as tail if the list is not empty

{

item2->next = NULL;

}

}

return head2;

}

// Add an element to the end of the list

void push(musical\_composition\* head, musical\_composition\* new\_item)

{

if (head == NULL)

{

head = new\_item;

return;

}

musical\_composition\* item = head;

while (item->next != NULL)

{

item = item->next;

}

item->next = new\_item;

new\_item->prev = item;

}

// Remove element

musical\_composition\* remove\_item(musical\_composition\* head, char\* name\_for\_remove)

{

if (head == NULL)

{

return;

}

musical\_composition\* item = head;

musical\_composition\* new\_head = head;

do

{

if (strcmp(item->name, name\_for\_remove) == 0)

{

musical\_composition\* prev = item->prev;

musical\_composition\* next = item->next;

if (prev != NULL)

{

prev->next = item->next;

}

else // we are deleting the current head

{

new\_head = item->next;

}

if (next != NULL)

{

next->prev = item->prev;

}

musical\_composition\* item\_to\_delete = item;

item = item->next;

free(item\_to\_delete);

}

else

{

item = item->next;

}

} while (item != NULL);

return new\_head;

}

// Count elements in the list

int count(musical\_composition\* head)

{

int count = 0;

musical\_composition\* item = head;

while (item != NULL)

{

count++;

item = item->next;

}

return count;

}

void print\_names(musical\_composition\* head)

{

musical\_composition\* item = head;

while (item != NULL)

{

printf("%s\n", item->name);

item = item->next;

}

}

musical\_composition\* swap\_head\_tail(musical\_composition\* head)

{

// list empty or contains just 1 element

if (head == NULL || head->next == NULL)

{

return head;

}

// list contains 2 elements

if (head->next->next == NULL)

{

musical\_composition\* tail = head->next;

// make tail as head

tail->prev = NULL;

tail->next = head;

// make head as tail

head->next = NULL;

head->prev = tail;

// return new head

return tail;

}

// find tail

musical\_composition\* item = head;

while (item->next != NULL)

{

item = item->next;

}

musical\_composition\* tail = item;

musical\_composition\* tail\_prev = tail->prev;

musical\_composition\* head\_next = head->next;

// make head as tail

head->next = NULL;

head->prev = tail\_prev;

tail\_prev->next = head;

// make tail as head

tail->prev = NULL;

tail->next = head\_next;

head\_next->prev = tail;

// return new head

return tail;

}

musical\_composition\* cut\_list\_into\_halves(musical\_composition\* head)

{

int n = count(head);// calculate middle index

if (n < 2)

{

return NULL; //cannot cut list of less than 2 compositions

}

int m; //middle index

if (n % 2 == 0)

{

m = n/2;

}

else

{

m = (n+1)/2;

}

musical\_composition\* item = head;// find middle item

int i;

for (i = 0; i < m; i++)

{

item = item->next;

}

musical\_composition\* tail1 = item->prev;// cut\_list\_into\_halves

tail1->next = NULL;

musical\_composition\* head2 = item;

head2->prev = NULL;

return head2;

}

musical\_composition\* concatenate\_two\_lists(musical\_composition\* head1, musical\_composition\* head2)

{

if (head1 == NULL)

{

return head2;

}

if (head2 == NULL)

{

return head1;

}

//find tail of list 1

musical\_composition\* item1 = head1;

while (item1->next != NULL)

{

item1 = item1->next;

}

musical\_composition\* tail1 = item1;

// concatenate lists

tail1->next = head2;

head2->prev = tail1;

return head1;

}

## Makefile

zachet: mc\_functions.o main.o

gcc mc\_functions.o main.o -o kurswork.out

rm \*.o

mc\_functions: mc\_functions.h mc\_functions.c

gcc -c mc\_functions.c

main.o: main.c main.h

gcc -c main.c

## Music.txt (файл, содержащий информацию для ввода для проверки работы программы)

5

N1 Born This Way

A1 Thousand Foot Krutch

2001

N2 Caravan

A2 Justin Hurwitz & Tim Simonec

2002

N2 Caravan

A2.1 Justin Hurwitz & Tim Simonec

20021

N3 Whiplash

A3 Hank Levy

2003

N4 Breath

A4 Breaking Benjamin

2004