**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. 6304 |  | Ковынев М.В. |
| Преподаватель |  | Берленко Т.А, |

Санкт-Петербург

2016

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Ковынев М.В. | | |
| Группа 6304 | | |
| Тема работы: Линейные списки | | |
| Содержание пояснительной записки:   * Содержание * Введение * Описание функций * Работа с репозиторием * Примеры работы программы * Заключение * Список использованных источников * Приложение: Исходный код программы | | |
| Предполагаемый объем пояснительной записки:  Не менее 10 страниц. | | |
| Дата выдачи задания: 02.12.2016 | | |
| Дата сдачи реферата: 22.12.2016 | | |
| Дата защиты реферата: 22.12.2016 | | |
| Студент |  | Ковынев М.В. |
| Преподаватель |  | Берленко Т.А. |

# **Аннотация**

В данной работе был создан проект на языке программирования C, который позволяет работать с набором функций, отвечающих за список музыкальных композиций. Для функционирования списка были созданы и описаны необходимые функции, позволяющие добавлять, удалять, сортировать 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 быстродействия и оптимального использования памяти и ресурсов. Приведено полное описание исходного кода.

**Содержание**

[Аннотация 3](#_Toc470137644)

[Введение 5](#_Toc470137645)

[Цель работы 5](#_Toc470137646)

[Формулировка задачи 5](#_Toc470137647)

[Индивидуальное задание 6](#_Toc470137648)

[Решение задачи 6](#_Toc470137649)

[1. Создание структуры данных 6](#_Toc470137650)

[2. Функция создания нового экземпляра данного типа 6](#_Toc470137651)

[3. Функция создания списка 7](#_Toc470137652)

[4. Функция считывания значений списка 7](#_Toc470137653)

[5. Функция добавления элемента в конец списка 8](#_Toc470137654)

[6. Функция добавления нескольких элементов в конец списка 9](#_Toc470137655)

[7. Функция удаления элемента 9](#_Toc470137656)

[8. Функция подсчета элементов списка 11](#_Toc470137657)

[9. Функция вывода элементов списка 11](#_Toc470137658)

[10. Функция сортировки 11](#_Toc470137659)

[11. Функция удаления нечетных элементов списка 12](#_Toc470137660)

[12. Функция вывода ошибки при отсутствии списка 13](#_Toc470137661)

[13. Функция очистки экрана 13](#_Toc470137662)

[14. Функция задержки экрана 13](#_Toc470137663)

[15. Makefile 14](#_Toc470137664)

[Работа с репозиторием 14](#_Toc470137665)

[Заключение 15](#_Toc470137666)

[Список использованных источников 16](#_Toc470137667)

[Приложение 17](#_Toc470137668)

[Исходный код программы 17](#_Toc470137669)

# **Введение**

## Цель работы

Практика применения сложных типов (struct) в языке С. Использование их для реализации сложных структур данных. В частности, одно- и дву-направленных линейных списков. Создание API для работы с ними. Закрепление знаний об указателях (в том числе на сложные типы), динамической памяти, массивах, стандартном вводе-выводе и основных функций библиотек “stdio.h”, “stdlib.h”, “string.h”.

**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. Отсортировать список по году, по возрастанию (не вывести, а именно поменять местами, если придется, элементы списка).
2. Удалить все нечетные элементы списка.

**Решение задачи**

## Создание структуры данных

С помощью типа *struct* объявим новый сложный тип *struct MusicalComposition*, экземплярами которого и будем заполнять список.

/\* Структура "Музыкальная композиция \*/

struct MusicalComposition {

char name[80]; // Название композиции

char author[80]; // Автор композиции

int year; // Год написания композиции

struct MusicalComposition\* next; // Указатель на следующий элемент

struct MusicalComposition\* prev; // Указатель на предыдущий элемент

};

/\*Объявление типа данных MusicalComposition\*/

typedef struct MusicalComposition MusicalComposition;

## Функция создания нового экземпляра данного типа

Выделим память для нового элемента и проинициализируем поля переданными в функцию параметрами с помощью указателя на элемент.

/\* Функция создания и инициализации нового экземпляра структуры "MusicalComposition" \*/

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

{

/\* Выделим память для нового экземпляра структуры "MusicalComposition" \*/

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

/\* Проинициализируем поля параметрами, переданными в функцию \*/

strcpy(rr->name, name);

strcpy(rr->author, author);

rr->year = year;

rr->next = NULL;

rr->prev = NULL;

/\* Вернем указатель на новый экземпляр структуры "MusicalComposition" \*/

return rr;

}

## Функция создания списка

Создадим сначала один элемент и проинициализируем его поля, а затем с помощью еще двух указателей в цикле создадим остальные элементы и свяжем их и вернем указатель на первый элемент в списке.

/\* Функция создания списка \*/

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

{

/\* Указатель на первый элемент, который вернет функция \*/

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

/\* Указательи для движения по списку \*/

MusicalComposition \*prev = head;

MusicalComposition \*current;

int i = 1;

/\* Заполняем список данными, переданными массивами array\_names, array\_authors, array\_years \*/

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

{

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

/\* Двигаем указатели \*/

current->prev = prev;

prev->next = current;

prev = current;

}

/\* Возвращаем указатель на первый элемент списка - head \*/

return head;

}

## Функция считывания значений списка

Первоначально выполняем цикл пока пользователь не введет подходящее значение количества элементов - целое неотрицательное число (length). Получив значение переменной length, начинаем последовательно считывать названия композиций (name), авторов композиций(author) и года (&years[i]) и записываем их в массивы (names,authors,years) соответственно. Передаем функции createMusicalCompositionList() данные массивы для создания линейного списка.

/\* Функция считывания значений списка\*/

MusicalComposition\* add()

{

clear(); //Очищаем экран

int length = 0;

/\* Выполняем цикл пока не найдем подходящее значение количества элементов - целое неотрицательное число \*/

while (length <= 0)

{

/\* Считываем значение количества элементов \*/

printf("Enter the number of adding elements: ");

scanf("%d", &length);

/\* Если считанное значение не удовлетворяет условию, выводим сообщение об ошибке \*/

if (length <= 0)

printf("\nError - number of elements is a positive integer!\n\n");

}

/\* Выделяем память массивы \*/

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

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

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

int i = 0;

/\* Через цикл считываем элементы списка (имя, автор и год) и сохраняем в массивы \*/

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

{

printf("\nEnter element %d: - name, author, year (using spaces):\n", i + 1);

char name[80];

char author[80];

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

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);

}

return createMusicalCompositionList(names, authors, years, length);

}

## Функция добавления элемента в конец списка

Функция принимает в качестве аргумента указатель на новый элемент (head). Считаем название композиции, автора и год написания, которые мы хотим добавить в список, и заполним структуру (MusicalComposition \*element) полученными переменными. Циклом сместим указатель на последний элемент списка и добавим (MusicalComposition \*element) в список, используя указатели и связав его с последним элементом списка.

/\* Функция добавления элемента в конец списка\*/

void push(MusicalComposition\* head)

{

char name\_for\_push[80];

char author\_for\_push[80];

int year\_for\_push;

/\* Считываем имя, автора и год на добавление\*/

printf("Enter name, author and year (using spaces):\n");

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

/\* Выделяем память под новый элемент\*/

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

MusicalComposition \*current = head;

/\* Передвигаем указатель в конец списка \*/

while (current->next)

current = current->next;

/\* Добавляем элемент в конец списка и устанавливаем связь\*/

current->next = element;

element->prev = current;

element->next = NULL;

}

## Функция добавления нескольких элементов в конец списка

Функция pushmanyel(MusicalComposition\* head) добавления нескольких элементов в конец списка в качестве основы использует предыдущую функцию добавления. Отличие заключается в том, что в функции изначально считывает количество добавляемых элементов (scanf("%d", &N);) и использует цикл, в котором вызывает предыдущую функцию.

/\* Функция добавления нескольких элементов в конец списка \*/

void pushmanyel(MusicalComposition\* head)

{

int N = 0;

clear(); //Очищаем экран

/\* Считываем количество введенных элементов - N \*/

printf("Enter the number of adding elements: ");

scanf("%d", &N);

printf("\n");

int i = 0;

/\* Вызываем N-раз функцию для добавления элемента через цикл\*/

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

{

push(head);

printf("\n");

}

/\* Выводим сообщение о выполнении операции \*/

printf("Adding completed successfully!\n\n");

pause();

}

## Функция удаления элемента

Удаление происходит поиском совпадения между введенным значением имени и именами элементов списка. Если совпадение найдено, то выполняется следующее:

Для того чтобы выполнить удаление элемента, необходимо учитывать его положение в списке. Так как список двулинейный, то возможно 4 варианта.

* 1. Если элемент единственный (перед ним и после него отсутствуют другие элементы), тогда удаляем весь список и закрываем доступ ко все элементам меню (кроме выхода и создания списка) (\*CreatedList = 0;), так как при вызове функций с отсутствующим списком, ответ будет некорректным.
  2. Если элемент первый и после него идут другие элементы, тогда необходимо сместить первоначальное значение списка (head) на следующий элемент.
  3. Если элемент последний и перед ним идут элементы, тогда необходимо сместить указатели с последнего элемента на предпоследнее.
  4. Если элемент находится между другими элементами, тогда необходимо связать элемент, идущий до него с элементом, идущим после него.

/\* Функция удаления элемента \*/

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

{

/\* Двигаем указатель до конца списка \*/

while (head)

{

/\* Если находим в списке находим в списке удовлетворяющую нас позицию, то выполняем следующее:\*/

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

{

/\* 1-ый случай. Если элемент - едиснтвенный\*/

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

{

/\* Удаляем список\*/

free(head);

head = NULL;

/\* Выводим сообщение об удалении списка\*/

printf("\nYou removed the last element in the List\nTo continue use this programm create new List!\n");

\*CreatedList = 0; //Закрываем доступ к опрдеделенным пунктам меню

return;

}

/\*2-ой случай. Если элемент первый и после него идут другие элементы\*/

else if ((head->prev == NULL) && (head->next))

{

/\*Смещаем вперед указатель на следующий элемент\*/

\*head = \*head->next;

head->prev = NULL;

return;

}

/\* 3-ий случай. Если элемент последний и перед ним идут другие элементы \*/

else if ((head->next == NULL) && (head->prev))

{

/\*Смещаем назад указатель на предыдущий элемент\*/

head = head->prev;

head->next = NULL;

return;

}

/\*4-ый случай. Если до и после элемента есть другие элементы\*/

else if ((head->next) && (head->prev))

{

head->next->prev = head->prev;

head->prev->next = head->next;

return;

}

}

head = head->next;

}

}

## Функция подсчета элементов списка

Чтобы подсчитать количество элементов, необходимо использовать счетчик и смещать с помощью цикла указатель до последнего элемента.

/\* Функция подсчета элементов списка \*/

int count(MusicalComposition\*head)

{

MusicalComposition\* current = head;

int count = 0;

/\* Двигаем указатель пока существет список\*/

while (current)

{

count++; //Увеличиваем счетчик - количество элементов

current = current->next;

}

return count;

};

## Функция вывода элементов списка

Функция вывода элементов списка аналогична предыдущей функции, только вместо счетчика выполняется функция вывода названия композиции, автора и года выпуска.

/\* Функция вывода элементов списка\*/

void print\_names(MusicalComposition \*head)

{

MusicalComposition \*current = head;

int i = 0;

/\* Пока существует список выполняем цикл\*/

while (current)

{

/\* Выводим на экран имя, автора и год \*/

printf(" %d. %s %s %d\n", i++, current->name, current->author, current->year);

current = current->next; //Смещаем указатель вперед

}

}

## Функция сортировки

В основе функции сортировки лежит метод *«пузырька».* Между двумя элементами списка происходит обмен значениями, если они не стоят в нужном порядке. Затем происходит смещение на следующий элемент и происходит то же самое до тех пор, пока список не станет отсортированным.

void sort(MusicalComposition \*head)

{

clear(); //Очищаем экран

printf("Sort\n\n");

printf("The original List:\n");

print\_names(head); //Выводим исходный список

MusicalComposition \*curr\_i = head;

MusicalComposition \*curr\_j;

/\* Выполняем сортировку методом пузырька \*/

while (curr\_i->next)

{

curr\_j = head;

while (curr\_j->next)

{

if (curr\_j->year>curr\_j->next->year)

{

/\* Обмениваемся значениями между соседними элементами\*/

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

strcpy(temp->name, curr\_j->next->name);

strcpy(temp->author, curr\_j->next->author); temp->year = curr\_j->next->year;

strcpy(curr\_j->next->name, curr\_j->name);

strcpy(curr\_j->next->author, curr\_j->author);

curr\_j->next->year = curr\_j->year;

strcpy(curr\_j->name, temp->name);

strcpy(curr\_j->author, temp->author);

curr\_j->year = temp->year;

free(temp);

}

curr\_j = curr\_j->next;

}

curr\_i = curr\_i->next;

}

printf("\nUpdated List:\n");

print\_names(head); //Выводим измененный список

printf("\n");

pause();

}

## Функция удаления нечетных элементов списка

Если номер элемента в списке нечетен (нумерация начинается с нуля), тогда функции удаления передается имя элемента и происходит его удаление.

/\*Функция удаления нечетных элементов\*/

void removeoddel(MusicalComposition \*head)

{

clear();//Очищаем экран

printf("Remove all odd elements\n(numbering starts at zero)\n\n");

printf("The Original List:\n");

print\_names(head);//Выводим исходный список

int i = 0;

MusicalComposition \*current = head;

while (current)

{

if (!((++i) % 2)) //Если номер элемента нечетен, то вызываем функцию удаления

removeEl(head, current->name, 0);

current = current->next;

}

printf("\nUpdated List:\n");

print\_names(head); //Выводим исходный список

printf("\n");

pause();

}

## Функция вывода ошибки при отсутствии списка

Если пользователь не создал список, тогда выводится сообщение об ошибке и предлагает создать его.

/\* Функция вывода ошибки при отсутствии списка \*/

void CreatedList()

{

clear();

printf("You didn't create List!\nPlease, choose 1 to create List in Menu\n");

pause();

}

## Функция очистки экрана

Так как у Linux и Windows разные функции очистки экрана консоли system(“cls”) и system(“clear”), то необходимо учесть данную особенность в функции очистки экрана, которая будет выполнять это на разных платформах.

/\* Функция очистки экрана \*/

void clear()

{

/\* Очистка экрана для Windows и Linux \*/

system("@cls||clear");

}

## Функция задержки экрана

На платформе Linux не предусмотрена функция задержки экрана, поэтому воспользуемся функцией считывания произвольного символа – getchar()

/\* Функция задержки экрана \*/

void pause()

{

printf("Press any key to continue...\n");

getchar();

getchar();

}

## Makefile

Создадим поочерёдно объектные файлы функций и главной функции, после скомпилируем их вместе, также предусмотрена функция очистки после сборки проекта.

#Главная цель

Kursach: struct.h main.o function.o createlist.o

gcc main.o - o Kursach\_Main.out createlist.o function.o

rm \*.o

#Объектный файл для функции main

main.o : main.c struct.h

gcc - c main.c

#Объектный файл для функции createlist

createlist.o : createlist.c createlist.h struct.h function.h

gcc - c createlist.c

#Объектный файл для функции function

function.o : function.c function.h struct.h

gcc - c function.c

# **Работа с репозиторием**

1. Скопируем репозиторий группы 6304

*Git clone https://github.com/moevm/pr1-2016-6304.git*

1. От ветки master создаем локальную ветку с названием работы и именем:

*Git checkout –b “Kovynev\_kurswork”*

1. В папку с клонированным репозиторием скопируем файлы с работой с помощью команды *cp –r*
2. Индексируем добавленные файлы командой

*git add \*.c \*.h makefile*

1. Коммитим наше изменений, чтобы подготовить их к отправке на сервер

*Git commit –m “Done kurswork”*

1. Отправляем изменения в репозиторий командой

*Git push origin*

1. На сайте github создаем *pull request* в ветку *master.*

**примеры работы программы:**

Для демонстрации всех возможностей программы используется следующий формат ввода:

* 1. Выбор пункта меню
  2. Количество элементов списке
  3. Элементы списка
  4. Выбор в меню следующего действия
  5. Новый элемент списка
  6. Выбор в меню следующего действия
  7. Название композиции, которую нужно удалить

Для вывода:

1. Название, автор и год написания первого элемента списка
2. Количество элементов до добавления
3. Количество элементов после добавления
4. Список названий композиций после добавления
5. Количество элементов после удаления
6. Список элементов после удаления

|  |  |  |
| --- | --- | --- |
| № | ВВОд | вЫВОД |
| 1 | 1 // выбор в создание списка  6 // количество элементов списка  After Midnight // элементы для формирования списка  Blink-182  2011  Creep  Radiohead  1993  Master of Puppets  Metallica  1984  Numb  Linkin Park  2003  Hurt  Johny Cash  2003  Everlong  Foo Fighters  1995  4//выбор в меню добавления нового элемента  Snuff // элемент для добавления в список  Slipknot  2008  6//выбор в меню удаления элемента  Numb //имя элемента для удаления из списка | After Midnight Blink-182 2011//первый элемент списка  6 //количество элементов до добавления  7 //количество элементов после добавления  After Midnight //список названий композиций  Creep //после добавления элемента  Master of Puppets  Hurt  Everlong  Snuff  Numb  6 //количество элементов после удаления  After Midnight  Creep //список элементов после удаления  Master of Puppets  Hurt  Everlong  Snuff  3 // количество элементов после удаления нечетных элементов  // Список элементов после удаления  Creep // Нечетных элементов  Hurt  Snuff |
| 2 | 1 // выбор в создание списка  4 // количество элементов для формирования списка  Name1 // элементы для формирования списка  Lastname1  1  Name2  Lastname2  2  Name3  Lastname3  3  Name4  Lastname4  4  4//выбор в меню добавления нового элемента  Addname //элемент для добавления  Addlastname  5  6//выбор в меню удаления элемента  Addname //имя элемента для удаления | Name1 Lastname1 1 // первый элемент списка  4 // количество элементов до добавления  5 // количество элементов после добавления  Name1 // имена элементов списка после добавления  Name2  Name3  Name4  Addname  4 // количество элементов после удаления элемента  Name1 // имена элементов списка после удаления  Name2  Name3  Name4  2//количество элементов после удаления нечётных эл.  Name2 // имена элементов после удаления  Name4 |
| 3 | 1 // выбор в создание списка  3// количество элементов для формирования списка  Danil // элементы для формирования списка  Ivanov  1998  Nikita  Petrov  1998  Dmitriy  Ivanov  1999 | Danil Ilyashuk 1998 // первый элемент списка  3 // количество элементов до добавления  4 // количество элементов после добавления  Danil//имена элементов списка после добавления  Nikita  Dmitriy  3 // количество элементов после удаления элемента  Nikita // имена элементов списка после удаления  Dmitriy |

# **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.

# **Список использованных источников**

* 1. Язык программирования СИ / Керниган Б., Ритчи Д. СПб: Издательство «Невский Диалект», 2001. 352 с.
  2. UNIX. Программное окружение / Керниган Б., Пайк Р. СПб: Символ Плюс, 2003. 416с.

# **Приложение**

# **Исходный код программы**

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

struct MusicalComposition {

char name[80];

char author[80];

int year;

struct MusicalComposition\* next;

struct MusicalComposition\* prev;

};

typedef struct MusicalComposition MusicalComposition;

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

{

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

strcpy(rr->name, name);

strcpy(rr->author, author);

rr->year = year;

rr->next = NULL;

rr->prev = NULL;

return rr;

}

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]);

MusicalComposition \*prev = head;

MusicalComposition \*current;

int i = 1;

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

{

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

current->prev = prev;

prev->next = current;

prev = current;

}

return head;

}

MusicalComposition\* add()

{

clear();

int length = 0;

while (length <= 0)

{

printf("Enter the number of adding elements: ");

scanf("%d", &length);

if (length <= 0)

printf("\nError - number of elements is a positive integer!\n\n");

}

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

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

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

int i = 0;

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

{

printf("\nEnter element %d: - name, author, year (using spaces):\n", i + 1);

char name[80];

char author[80];

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

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);

}

return createMusicalCompositionList(names, authors, years, length);

}

void clear()

{

system("@cls||clear");

}

void pause()

{

printf("Press any key to continue...\n");

getchar();

getchar();

}

void print\_names(MusicalComposition \*head)

{

MusicalComposition \*current = head;

int i = 0;

while (current)

{

printf(" %d. %s %s %d\n", i++, current->name, current->author, current->year);

current = current->next;

}

}

void push(MusicalComposition\* head)

{

char name\_for\_push[80];

char author\_for\_push[80];

int year\_for\_push;

printf("Enter name, author and year (using spaces):\n");

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

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

MusicalComposition \*current = head;

while (current->next)

current = current->next;

current->next = element;

element->prev = current;

element->next = NULL;

}

void pushmanyel(MusicalComposition\* head)

{

int N = 0;

clear();

printf("Enter the number of adding elements: ");

scanf("%d", &N);

printf("\n");

int i = 0;

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

{

push(head);

printf("\n");

}

printf("Adding completed successfully!\n\n");

pause();

}

int count(MusicalComposition\*head)

{

MusicalComposition\* current = head;

int count = 0;

while (current)

{

count++;

current = current->next;

}

return count;

};

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

{

while (head)

{

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

{

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

{

free(head);

head = NULL;

printf("\nYou removed the last element in the List\nTo continue use this programm create new List!\n");

\*CreatedList = 0;

return;

}

else if ((head->prev == NULL) && (head->next))

{

\*head = \*head->next;

head->prev = NULL;

return;

}

else if ((head->next == NULL) && (head->prev))

{

head = head->prev;

head->next = NULL;

return;

}

else if ((head->next) && (head->prev))

{

head->next->prev = head->prev;

head->prev->next = head->next;

return;

}

}

head = head->next;

}

}

void sort(MusicalComposition \*head)

{

clear();

printf("Sort\n\n");

printf("The original List:\n");

print\_names(head);

MusicalComposition \*curr\_i = head;

MusicalComposition \*curr\_j;

while (curr\_i->next)

{

curr\_j = head;

while (curr\_j->next)

{

if (curr\_j->year>curr\_j->next->year)

{

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

strcpy(temp->name, curr\_j->next->name);

strcpy(temp->author, curr\_j->next->author);

temp->year = curr\_j->next->year;

strcpy(curr\_j->next->name, curr\_j->name);

strcpy(curr\_j->next->author, curr\_j->author);

curr\_j->next->year = curr\_j->year;

strcpy(curr\_j->name, temp->name);

strcpy(curr\_j->author, temp->author);

curr\_j->year = temp->year;

free(temp);

}

curr\_j = curr\_j->next;

}

curr\_i = curr\_i->next;

}

printf("\nUpdated List:\n");

print\_names(head);

printf("\n");

pause();

}

void removeoddel(MusicalComposition \*head)

{

clear();

printf("Remove all odd elements\n(numbering starts at zero)\n\n");

printf("The Original List:\n");

print\_names(head);

int i = 0;

MusicalComposition \*current = head;

while (current)

{

if (!((++i) % 2))

removeEl(head, current->name, 0);

current = current->next;

}

printf("\nUpdated List:\n");

print\_names(head);

printf("\n");

pause();

}

void CreatedList()

{

clear();

printf("You didn't create List!\nPlease, choose 1 to create List in Menu\n");

pause();

}

int main()

{

int decision = 0;

int createdList = 0;

MusicalComposition \*head = NULL;

while (decision != 9)

{

clear();

printf("\n ################################ Menu ###############################\n");

printf(" # #\n");

printf(" # #\n");

printf(" # 1. Create List #\n");

printf(" # 2. Print List #\n");

printf(" # 3. Sort List by year ascending #\n");

printf(" # 4. Add new element #\n");

printf(" # 5. Add several elements #\n");

printf(" # 6. Remove elements by search appropriate name #\n");

printf(" # 7. Remove all odd elements #\n");

printf(" # 8. Number of elements #\n");

printf(" # 9. Exit #\n");

printf(" # #\n");

printf(" # #\n");

printf(" #####################################################################\n\n\n");

printf(" Choose: ");

scanf("%d", &decision);

switch (decision)

{

case 1:

head = add();

createdList = 1;

break;

case 2:

if (createdList != 1)

CreatedList();

else

{

clear();

printf("List:\n");

print\_names(head);

printf("\n");

pause();

}

break;

case 3:

if (createdList != 1)

CreatedList();

else

sort(head);

break;

case 4:

if (createdList != 1)

CreatedList();

else

{

clear();

push(head);

printf("\nAdding elements completed successfully!\n\n");

pause();

}

break;

case 5:

if (createdList != 1)

CreatedList();

else

pushmanyel(head);

break;

case 6:

if (createdList != 1)

CreatedList();

else

{

clear();

char name\_for\_remove[80];

printf("Enter the name to be removed from the List\n");

scanf("%s", name\_for\_remove);

removeEl(head, name\_for\_remove, &createdList);

printf("\nRemovig completed successfully!\n\n");

pause();

}

break;

case 7:

if (createdList != 1)

CreatedList();

else

removeoddel(head);

break;

case 8:

if (createdList != 1)

CreatedList();

else

{

clear();

printf("Number of elements in List: %d\n\n", count(head));

pause();

}

break;

case 9: break;

default:

clear();

printf("There is no such option. Try again!\n\n");

pause();

}

}

clear();

return 0;

}