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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Программирование»

Тема: “Структуры данных, линейные списки”

Студент гр. 7381 _____ Габов Е.С

Преподаватель _____ Берленко Т.А

Санкт-Петербург

2017

Цель:
Научиться работать с двунаправленным списком. Закрепить имеющиеся знания по выделению и очистке динамической памяти. Написать функции для работы со списком: добавлять, удалять, создаваться и выводить элементы списка и их количество.

Задание:
Создать двунаправленный список музыкальных композиций MusicalComposition и **api** (***a****pplication* ***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

Основные теоретические положения:
Создана структура с именем MusicalComposition с членами типов char*,char*,int , и 2 члена с типом указателя на саму структуру, эти члены содержат адреса предыдущих и последующих элементов.

Далее создается ряд функция для работы с двухсвязным списком:

createMusicalComposition.
Создана функция createMusicalComposition. Эта функция создает элемент списка. На вход она получает 3 параметра с именем, автором и годом создания. В ней используются такие функции как malloc(динамическое выделение памяти) и strcpy(копирование символов).Функция возвращает указатель на первый элемент списка.

Push
Создана функция push. Эта функция добавляет элемент списка в конец списка. На вход получает указатель на первый элемент и элемент который нужно добавить в список. С помощью цикла while определяется местоположение последнего элемента. Функция ничего не возвращает.

createMusicalCompositionList
Создана функция **createMusicalCompositionList.** Эта функция создает целый список. На вход получает массив имен, авторов, лет издания и количество элементов которые будут в списке. Если количество элементов равняется нулю функция возвращает NULL. Иначе список заполняет при помощи цикла for. В нём вызываются описанные функции push и createMusicalComposition. Функция возвращает указатель на первый элемент списка.

removeEl
Создана функция removeEl. Эта функция удаляет элемент из списка, если значение члена name совпало с полученной функцией строкой. На вход поступает указатель на первый элемент и определенная строка символов. Функция ничего не возвращает.

Count
Создана функция count. Эта функция считает количество элементов в списке. На вход поступает указатель на первый элемент. В цикле while перебираются все элементы и счетчик с каждым шагом увеличивается на еденицу. Функция возвращает количество элементов.

Print_names
Создана функция print_names. Функция выводит на экран значение члена name всех элементов. На вход получает указатель на первый элемент списка. Функция ничего не возвращает.

Вывод
В данной лабораторной работе создан двунаправленный список с информацией о музыкальных композициях. В ходе лабораторной работе получены знания по нахождения утечек памяти и закреплены знания по её выделению. Освоены алгоритмы написания функций для работы с двусвязным списком.

Код программы:

Создание структуры:

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

createMusicalComposition:

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

	element_for_push->name = (char*)malloc(sizeof(char)*81);
	element_for_push->author = (char*)malloc(sizeof(char)*81);

	strcpy(element_for_push->name , name );
	strcpy(element_for_push->author,author);
	element_for_push->year = year;
	element_for_push->next = NULL;
	element_for_push->prev = NULL;

	return element_for_push;
}
```

Push:

```
void push(MusicalComposition* head, MusicalComposition* element)
{
	MusicalComposition* tmp = (MusicalComposition*)malloc(sizeof(MusicalComposition));
	if ( head->next == NULL )
	{
```

```
            element->next = NULL;
            element->prev = head;
            head->next = element;
            return;
        }

        tmp = head->next;

        while (tmp->next)
        {
            tmp = tmp->next;
        }

        element->next = NULL;
        element->prev = tmp;
        tmp->next = element;

}
```

createMusicalCompositionList:

```
MusicalComposition* createMusicalCompositionList(char** array_names, char**
array_authors, int* array_years,int length)
{
        if ( length == 0 )
           return NULL;

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

        head->name = (char*)malloc(sizeof(char)*81);
        head->author = (char*)malloc(sizeof(char)*81);

        strcpy(head->name , array_names[0]);
        strcpy(head->author , array_authors[0]);
        head->year = array_years[0];

        MusicalComposition* tmp=
(MusicalComposition*)malloc(sizeof(MusicalComposition));

        for ( int i = 1; i < length ; i++ )
        {
            tmp = createMusicalComposition( array_names[i] , array_authors[i] ,
array_years[i] );
            push(head , tmp);
```

```
	}

	return head;
}
removeEl:

void removeEl(MusicalComposition* head, char* name_for_remove)
{
	MusicalComposition *tmp ;
	tmp = head;
	while ( tmp )
	{
		if ( strcmp( tmp->name , name_for_remove ) == 0 )
		{
			tmp->next->prev = tmp->prev;
			tmp->prev->next = tmp->next;
			free(tmp);
			//return;
		}
		tmp = tmp->next;
	}

}

Count:

int count(MusicalComposition* head)
{
	int i=1;

	MusicalComposition* tmp =
(MusicalComposition*)malloc(sizeof(MusicalComposition));
	tmp = head->next;
	while (tmp)
	{
		tmp = tmp->next;
		i++;

	}
	return i;
}

print_names:

void print_names(MusicalComposition* head)
```

```
{MusicalComposition* tmp =
(MusicalComposition*)malloc(sizeof(MusicalComposition));
	tmp = head->next;

		printf( "%s\n" , head->name );
		printf( "%s\n" , tmp->name );
	while (tmp->next)
	{
		if ( tmp->next->year != -1 )
			printf("%s\n"  , tmp->next->name );
		tmp = tmp->next;
	}
}
```