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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Программирование»

Тема: Структуры данных, линейные списки

Студент гр. 7381 _____ Кревчик А.Б.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2017

Цель работы.

Ознакомиться с двусвязными списками в языке Си, научиться их создавать, добавлять и удалять элементы списка, получать нужную информацию об элементах списка.

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]**).

  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

В функции main написана некоторая последовательность вызова команд для проверки работы вашего списка.
Функцию main менять не нужно.

Основные теоретические положения.

Язык Си поддерживает **структуры данных**. Синтаксис объявления структуры:

```
struct <имя> {
   <тип1> <поле1>;
   <тип2> <поле2>;
   ...
   <типN> <полеN>;
};
```

Объявление экземпляра структуры данных:
struct <структура> <имя>;

Список - некоторый упорядоченный набор элементов любой природы.

Линейный однонаправленный (односвязный) список - список, каждый элемент которого хранит помимо значения указатель на следующий
элемент. В последнем элементе указатель на следующий элемент равен NULL (константа нулевого указателя).

Оператор **typedef.** Стандартный синтаксис использования:

typedef <type> <name>;
где

- type - любой тип
- name - новое имя типа (при этом можно использовать и старое имя).

Выводы.

В ходе данной работы были изучены структуры данных в языке Си, двунаправленные списки, основы работы с ними.

Исходный код проекта.

- *Файл Makefile:*

```
all: main.o
	gcc main.o
main.o: main.
	gcc -c main.c
```

- *Файл «main.c»:*

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


typedef struct MusicalComposition
 {
 char* name;
 char* author;
 int year;
 struct MusicalComposition* next;
 struct MusicalComposition* prev;
 }MusicalComposition;// Описание структуры MusicalComposition

// Создание структуры MusicalComposition

MusicalComposition* createMusicalComposition(char* name, char* author,int
year)
{    MusicalComposition* Musical_Composition;

Musical_Composition=(MusicalComposition*)malloc(sizeof(MusicalCompositio
n));
    Musical_Composition->name=name;
    Musical_Composition->author=author;
    Musical_Composition->year=year;
    Musical_Composition->prev=NULL;
    Musical_Composition->next=NULL;
 return Musical_Composition;
}

// Функции для работы со списком MusicalComposition

MusicalComposition* createMusicalCompositionList(char** array_names,
char** array_authors, int* array_years, int n)
{  int i=0;
```

```
    MusicalComposition* head, *tmp, *pred=NULL;

    for(i=0; i<n;i++)
        {tmp = createMusicalComposition ( array_names[i],
array_authors[i],array_years[i]);
         tmp->prev=pred;

if (i!=0)
        pred->next=tmp;


else

    head=tmp;

         pred=tmp;
}
    return head;
}

void push(MusicalComposition* head, MusicalComposition* element)
{if (head==NULL)
    *head=*element;
 else{
for(;head->next!=NULL;head=head->next);

        head->next=element;
        element->prev=head;
 }
}

void removeEl(MusicalComposition** head_p, char* name_for_remove)
{
    MusicalComposition* head = *head_p;

for (;head!=NULL;head=head->next)
    if (strcmp(head->name,name_for_remove)==0)
    if (head->next==NULL && head->prev==NULL)
    {*head_p=NULL;
    free(head);
    }
    else if(head->next!=NULL && head->prev==NULL)
    {head->next->prev=NULL;
     *head_p=head->next;
    free(head);
```

```
 }
   else if (head->next==NULL && head->prev!=NULL)
   {head->prev->next=NULL;
   free (head);
 }
   else
  {head->next->prev=head->prev;
      head->prev->next=head->next;
      free (head);
    }



}

int count(MusicalComposition* head)
{int k=0;
for (;head!=NULL;head=head->next)
   k++;
 return k;
}

void print_names(MusicalComposition* head)
{
for (;head!=NULL;head=head->next)
   puts(head->name);
}


int main(){
   int length;
   scanf("%d\n", &length);

   char** names = (char**)malloc(sizeof(char*)*length);
   char** authors = (char**)malloc(sizeof(char*)*length);
   int* years = (int*)malloc(sizeof(int)*length);

   for (int i=0;i<length;i++)
   {
      char name[80];
      char author[80];

      fgets(name, 80, stdin);
      fgets(author, 80, stdin);
      fscanf(stdin, "%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);

    }
    MusicalComposition* head = createMusicalCompositionList(names, authors,
years, length);
    char name_for_push[80];
    char author_for_push[80];
    int year_for_push;

    char name_for_remove[80];

    fgets(name_for_push, 80, stdin);
    fgets(author_for_push, 80, stdin);
    fscanf(stdin, "%d\n", &year_for_push);
    (*strstr(name_for_push,"\n"))=0;
    (*strstr(author_for_push,"\n"))=0;

    MusicalComposition* element_for_push =
createMusicalComposition(name_for_push, author_for_push, year_for_push);

    fgets(name_for_remove, 80, stdin);
    (*strstr(name_for_remove,"\n"))=0;

    printf("%s %s %d\n", head->name, head->author, head->year);
    int k = count(head);

    printf("%d\n", k);
    push(head, element_for_push);

    k = count(head);
    printf("%d\n", k);

    removeEl(&head, name_for_remove);
    print_names(head);

    k = count(head);
    printf("%d\n", k);

    for (int i=0;i<length;i++){
```

```
        free(names[i]);
        free(authors[i]);
    }
    free(names);
    free(authors);
    free(years);

    return 0;

}
```