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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Программирование»

Тема: «Линейные списки»

Студент гр. 7381 ____ Лауцюс М.

Преподаватель ____ Берленко Т.А.

Санкт-Петербург

2017

Цель работы:

Научится работать со списками и структурами на языке Си. Научиться создавать списки, добавлять и удалять элементы списка, получать информацию о количестве элементов в списке…

Задание:

Создайте 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.

Заголовочные файлы стандартной библиотеки языка Си, используемые в данной лабораторной работе:

1. stdio.h - содержит определения макросов, константы и объявления функций и типов, используемых для различных операций стандартного ввода и вывода.

2. string.h- заголовочный файл для работы с Си-строками.

- int strcmp(const char * string1, const char * string2):

Описание:

Эта функция сравнивает символы двух строк, string1 и string2. Начиная с первых символов функция strcmp сравнивает поочередно каждую пару символов, и продолжается это до тех пор, пока не будут найдены различные символы или не будет достигнут конец строки.

Параметры:

string1

Первая сравниваемая Си-строка.

string2

Вторая сравниваемая Си-строка..

Возвращаемое значение:

Функция возвращает несколько значений, которые указывают на отношение строк:

Нулевое значение говорит о том, что обе строки равны. Значение больше нуля указывает на то, что строка string1 больше строки string2, значение меньше нуля свидетельствует об обратном.

3. stdlib.h- содержит функции для преобразования чисел в текст, выделения памяти, генерации случайных чисел и др. функций-утилит.

4. stddef.h - заголовочный файл стандартной библиотеки языка программирования C, определяющий макросы NULL и offsetof, а также типы ptrdiff_t, wchar t и size_t.

Список — базовая динамическая структура данных в информатике, состоящая из узлов, каждый из которых содержит как собственно данные, так и одну или сколько-то ссылок на следующий и/или предыдущий узел списка. Принципиальным преимуществом перед массивом является структурная гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задаётся его внутренними

связями.

Линейный список — это структура данных, состоящая из элементов одного типа, связанных между собой посредством указателей.

Доступ к элементам указателя на структуру:

(*pointer).element = pointer->element

Вывод:

В ходе выполнения лабораторной работы были созданы функции для работы с двунаправленным списком. В ходе выполнения лабораторной работы были приобретены навыки создания списков, работы с ними и их модификации.

Исходный код проекта:

Файл Makefile:

```
all: main.c functions.h
    gcc main.c
```

Файл main.c:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stddef.h>
#include "functions.h"

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;

        }
```

Файл functions.h:

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

MusicalComposition* createMusicalComposition(char* name, char*
author, int year)
{
    MusicalComposition* music =
(MusicalComposition*)malloc(sizeof(MusicalComposition));
    music->name = name;
    music->author = author;
    music->year = year;
    music->next = NULL;
    music->previous = NULL;
    return music;
}

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* current;
    MusicalComposition* previous_el=head;
    int i;
    for(i = 1; i < n; i++)
    {
        current = createMusicalComposition(array_names[i],
array_authors[i], array_years[i]);
        current->previous = previous_el;
        previous_el->next = current;
        previous_el = current;
    }
    return head;
}

void push(MusicalComposition* head, MusicalComposition*
element)
{
    MusicalComposition* current = head;
    while(current->next)
        current=current->next;
    current->next=element;
    element->previous=current;
}

void removeEl(MusicalComposition* head, char* name_for_remove)
{
    MusicalComposition* current = head;
    //MusicalComposition* pointer;
    while(current->next)
    {
        if(strcmp(current->name, name_for_remove) == 0)
        {
            //pointer=current;
```

```
            if(current->next != NULL && current->previous !=
NULL)
            {
                current->previous->next = current->next;
                current->next->previous = current->previous;
            }
            else if(current->next == NULL)
            {
                current->previous->next = NULL;
            }
            else if(current->previous == NULL)
            {

                current->next->previous = NULL;
                head = current->next;
            }
            free(current);
        }
        current = current->next;
    }

}

int count(MusicalComposition* head)
{
    MusicalComposition* current = head;
    int n = 0;
    while (current)
    {
        n++;
        current = current->next;
    }
    return n;
}

void print_names(MusicalComposition* head)
```

```
{
    MusicalComposition* current = head;
    while (current)
    {
        printf("%s\n", current->name);
        current = current->next;
    }
}
```