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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Программирование»
Тема: Линейные списки

Студент гр. 1304 _____ Павлов Д. Р.

Преподаватель _____ Чайка К. В.

Санкт-Петербург
2022

Цель работы.

Целью данной лабораторной работы является изучение линейных списков и структур данных.

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.

Выполнение работы.

Инициализируется структура MusicalComposition в которой существует 5 полей — name(имя композиции), author(автор композиции), year(год создания композиции), next(*типа структуры) которая является узлом, указывающем на следующую структуру и, аналогично, prev. Далее пишется функция creatMusicalComposition кторая будет создавать музыкальных композицию. Сначала выделяется память под переменную tmp типо

MusicalComposition и в поля данной структуры копируются заданные параметры. Так же у tmp поля next & prev приравниваются NULL. Далее идет функция createmusicalcompositionlist, которая будет создавать список из музыкальных композиций. Сначала с помощью функции createMusicalComposition создается корневой список head. Далее к нему приравнивается tmp и инициализируется список current. Далее создается цикл до n-го элемента, в котором мы так же с помощью createmusicalcomposition создаем композицию и приравниваем ее к current. Далее у списка tmp поле next приравнивается current, а у поля prev у current приравнивается к tmp. После tmp приравнивается к current. Далее пишется функция push которая будет добавлять структуру в список. Для ее реализации пишется бесконечный цилк внутри которого мы пишем условия: если tmp→next == NULL, то tmp→next = структуре, а поле prev у данной структуры равно tmp, после чего идет break; иначе — tmp = tmp→next. Далее пишется функция которая будет удалять структуру из списка. Для ее реализации инициализируется структура tmp, которая приравнивается к корневой структуре в списке. Далее пишется условие пока имя структуры не равно имени для удаления: tmp = tmp→next. После чего tmp→prev→next = tmp→next. Далее пишется функция которая считает количество элементов в списке. Внутри нее инициализируется список tmp который приравнивается к нашему списку и далее мы заводим цикл пока следующий элемент списка не равен null. Внутри которого увеличивается счетчик и tmp перемешается по полю next. Возвращается счетчик. Далее пишется функция print_names которая выводит все имена композиций в списке. Внутри него заводится цикл пока поле next не равно null внутри которого мы перемещаемся по полю next и выводим имя композиции.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	7 Fields of Gold Sting 1993 In the Army Now Status Quo 1986 Mixed Emotions The Rolling Stones 1989 Billie Jean Michael Jackson 1983 Seek and Destroy Metallica 1982 Wicked Game Chris Isaak 1989 Points of Authority Linkin Park 2000 Sonne Rammstein 2001	Fields of Gold Sting 1993 7 8 Fields of Gold In the Army Now Mixed Emotions Billie Jean Seek and Destroy Wicked Game Sonne 7	True

	Points of Authority		
2.			
3.			

Выводы.

Была написана программа, которая создается двунаправленный список музыкальных композиций, и api, для работы с ним.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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



// Описание структуры MusicalComposition
typedef struct Node{
    char name[80];
    char author[80];
    int year;

    struct Node* next;
    struct Node* prev;

}MusicalComposition;


// Создание структуры MusicalComposition

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

    strcpy(tmp->name, name);
    strcpy(tmp->author, author);
    tmp->year = year;

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

    return tmp;

}
```

```
// Функции для работы со списком MusicalComposition

MusicalComposition*  createMusicalCompositionList(char**  array_names,
char** array_authors, int* array_years, int n){
        MusicalComposition* head = createMusicalComposition(*array_names,
*array_authors, *array_years);

    MusicalComposition* tmp = head;
    MusicalComposition* current;

    for (int i = 0; i<n; i++){
                current  =  createMusicalComposition(array_names[i],
array_authors[i], array_years[i]);

        tmp->next = current;
        current->prev = tmp;

        tmp = current;

    }

    return head;

}

void push(MusicalComposition* head, MusicalComposition* element) {
    MusicalComposition* tmp = head;
    while(1){
        if (tmp->next == NULL) {
            tmp->next = element;
            element->prev = tmp;
            break;
        }else {
            tmp = tmp->next;
        }
    }

}

void removeEl(MusicalComposition* head, char* name_for_remove){
```

```
    MusicalComposition* tmp;
    tmp = head;
    while(strcmp(tmp->name, name_for_remove) != 0){
        tmp = tmp->next;
    }
    tmp->prev->next = tmp->next;

}

int count(MusicalComposition* head){
   MusicalComposition* tmp = head;
   int i = 0;
   while (tmp->next != NULL){
       tmp = tmp->next;
       i++;
   }
   return i;
}

void print_names(MusicalComposition* head){
    MusicalComposition* tmp = head;

    while (tmp->next != NULL){
        printf("%s\n", tmp->next->name);
        tmp = tmp->next;
    }

}


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;
        //!!!
        //print_names(head);

        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;

}
```