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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Программирование»

Тема: Линейные списки

Студент гр. 0382 _____ Кондратов Ю.А.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Цель работы.

Обучение работе с линейными списками.

Задание.

Создайте двунаправленный список музыкальных композиций MusicalComposition и api (application programming interface - 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.

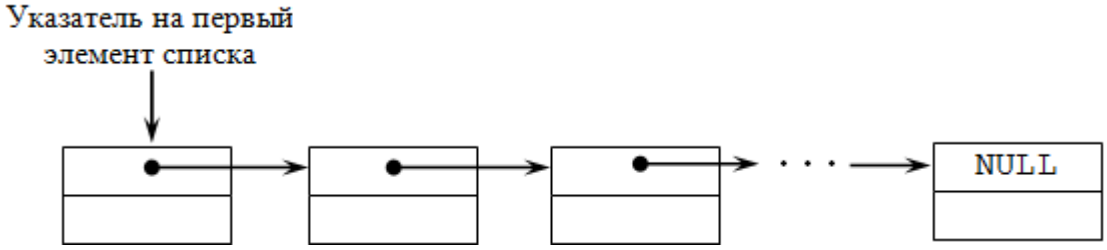
- void push(MusicalComposition* head, MusicalComposition* element); // добавляет element в конец списка musical_composition_list
- void removeEl (MusicalComposition* head, char* name_for_remove); // удаляет элемент element списка, у которого значение name равно значению name_for_remove
- int count(MusicalComposition* 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.

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



Выполнение работы.

Сначала было написано описание структуры *struct MusicalComposition* с элементами:

- *char* name* — хранит название композиции;
- *char* author* — хранит псевдоним автора композиции;
- *int year* — хранит год написания композиции;
- *struct MusicalComposition *next* — указатель на следующий элемент списка;
- *struct MusicalComposition *previous* — указатель на предыдущий элемент списка.

Далее была реализована функция для создания элемента списка *createMusicalComposition*. Функция принимает на вход название композиции, псевдоним автора и год написания. Сначала в функции динамически выделяется память для одного элемента структуры *MusicalComposition*, после чего значениями входных данный инициализируются соответствующие элементы структуры. Функции возвращает получившуюся структуру.

Для создания списка композиций реализована функция *createMusicalCompositionLis.* Функция принимает на вход массивы данных о композициях и количество композиций. В функции сначала создаётся первый элемент списка, после чего в цикле создаются все остальные элементы и присваиваются адреса. Функция возвращает адрес первого элемента («головы») списка.

Функция *push* реализует добавление элемента в конец списка. Для этого в цикле производится поиск последнего элемента (пока адрес следующего элемента не будет равняться NULL). Далее это адрес меняется на адрес элемента, который необходимо добавить, а адрес следующего за добавленным элемента делается равным NULL.

Функция *removeEl* реализуется удалении композиции из списка по названию. Сначала в цикле производится поиск элемента с заданным названием, после чего его указатель на следующий элемент присваивается полю *next* предыдущего элемента, а его указатель на предыдущий элемент присваивается полю *previous* следующего элемента. Далее производится очистка памяти и выход из функции.

Функции *count* и *print_names* работают аналогично. Они проходятся по всему списку, но первая считает количество элементов, а вторая выводит их на консоль.

Исходный код программы см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

Входные данные	Выходные данные	Комментарии
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 Points of Authority	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	Программа работает правильно

Выводы.

В ходе работы были изучены основные принципы работы с линейными списками и разработан API для работы со списком музыкальных композиций. API включает функции: создания элемента списка, создания списка, добавления элемента списка, удаления элемента из списка, подсчёта элементов в списке, вывода всех элементов списка.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ФАЙЛОВ ПРОЕКТА

1. Название файла: menu.c

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


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


// Создание структуры MusicalComposition

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

// Функции для работы со списком MusicalComposition

MusicalComposition*           createMusicalCompositionList(char**
array_names, char** array_authors, int* array_years, int n);

void push(MusicalComposition* head, MusicalComposition* element);

void removeEl(MusicalComposition* head, char* name_for_remove);

int count(MusicalComposition* head);

void print_names(MusicalComposition* head);


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;

    }

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

    MusicalComposition             *createMusicalCompositionList(char
**array_names, char **array_authors, int *array_years, int n) {
                                MusicalComposition      *first      =
createMusicalComposition(array_names[0],             array_authors[0],
array_years[0]);
        MusicalComposition *prev = first;
        prev->previous = NULL;
        for (int i = 1; i < n; i++) {
                                      MusicalComposition    *new    =
createMusicalComposition(array_names[i],             array_authors[i],
array_years[i]);
            prev->next = new;
            new->previous = prev;
            prev = new;
        }
        prev->next = NULL;
        return first;
    }

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

    void removeEl(MusicalComposition *head, char *name_for_remove) {
        while (head->next != NULL) {
            if (!strcmp(head->name, name_for_remove)){
                head->previous->next = head->next;
                head->next->previous = head->previous;
                MusicalComposition* tmp = head;
                head = head->previous;
                free(tmp);
            }
            head = head->next;
        }
    }
```

```
int count(MusicalComposition *head){
    int n = 1;
    while(head->next != NULL){
        n += 1;
        head = head->next;
    }
    return n;
}

void print_names(MusicalComposition *head){
    printf("%s\n", head->name);
    do{
        head = head->next;
        printf("%s\n", head->name);
    } while(head->next != NULL);
}
```