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: Структуры данных, линейные списки

Студент гр. 0382 Шангичев В. А.

Преподаватель Берленко Т. А.

Санкт-Петербург
2021

Цель работы.

Ознакомиться с механизмом работы линейных списков и реализовать функции для создания и управления линейным списком на языке Си.

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.

Заголовочный файл `<stdlib.h>` - одними из многих функций, реализованных в данном заголовочном файле являются функции сортировки и поиска элементов в массиве и функции для работы с памятью.

Заголовочный файл `<string.h>` - содержит функции для работы со

строками.

Синтаксическая конструкция `typedef` – позволяет задать имя существующему типу.

Функция `malloc` – используется для выделения памяти.

Функция `sizeof` – дает информацию о размере объекта в байтах.

Макрос `NULL` – используется в программе для обозначения нулевого указателя.

Функция `strcmp` – реализует сравнение двух строк.

Функция `free` – используется для освобождения блока памяти.

Выполнение работы.

В первых трех строках кода программы реализуется включение всех необходимых заголовочных файлов: `stdlib.h`, `stdio.h`, `string.h`. Затем объявляется структура `MusicalComposition`, название типа которой принимает

одноименное название с помощью синтаксической конструкции `typedef`. Поле `name` структуры отвечает за хранение названия композиции, в поле `author` сохраняется фамилия автора/название группы, поле `year` реализует хранение года выпуска музыкального произведения, поля `previous` и `next` содержат указатели на предыдущий и следующий элемент на в связном списке соответственно (если таких элементов нет, то поле содержит нулевой указатель).

После объявления структуры реализуется описание функции `createMusicalComposition`, с помощью которой будет создаваться новый объект структуры `MusicalComposition`. На вход функция принимает автора композиции, название и год ее создания. Функция возвращает указатель на новую композицию. В теле функции с помощью функции `malloc` выделяется память для хранения новой композиции. Полученный указатель сохраняется в переменной `mcomp`. Далее заполняются поля структуры в соответствии с переданными аргументами. Поля указателей на предыдущий и следующий элемент заполняются нулевыми указателями.

Следующая функция `push` используется для добавления нового элемента в конец списка. На вход подается указатель на первый элемент списка и указатель на элемент, который нужно вставить. Для идентификации текущего последнего элемента списка объявляется переменная `end`, начальное значение которой соответствует значению указателя на первый элемент списка. Далее реализуется цикл, где с помощью проверок значений поля `next` определяется последний элемент в списке. Полю `next` последнего элемента присваивается указатель на переданный в функцию элемент, а полю указателя на предыдущий элемент композиции, которую необходимо добавить, присваивается указатель на последний элемент.

Функция `createMusicalCompositionList` создает новый связный список по переданным массивам названий композиций, их авторов, годов создания и количеству элементов в новом списке. С помощью функции `createMusicalComposition` создается первая музыкальная композиция.

Указатель на нее сохраняется в переменной `list`. Далее реализуется цикл, в теле которого с помощью функций `createMusicalComposition` и `push` реализуется создание новой композиции и ее вставка в конец. Функция возвращает указатель на первый элемент списка.

Функция `removeEl` удаляет элемент из списка с соответствующим названием композиции. Элемент, который нужно удалить в последствии будет записан в переменную `needful_elem`. Далее реализуется цикл, продолжающийся до тех пор, пока строка, содержащая название композиции текущего элемента списка, не будет совпадать со строкой, переданной в функцию. Далее указателю на следующий элемент структуры, идущей до удаляемого элемента, присваивается указатель на следующий элемент удаляемого элемента (если удаляемый элемент не является первым). Аналогично происходит переопределение указателей на предыдущий элемент. После этого память, хранящая элемент, который надо удалить, освобождается.

Функция `count` возвращает количество элементов в связном списке, с помощью цикла, аналогичного используемым выше.

Функция `print_names` печатает названия всех композиций в списке.

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	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 корректно.

	1982 Wicked Game Chris Isaak 1989 Points of Authority Linkin Park 2000 Sonne Rammstein 2001 Points of Authority		

Выводы.

Была изучена механика работы связных списков. С помощью конструкций языка Си был реализован двунаправленный связный список. В ходе выполнения задания помимо приобретения новых навыков было закреплено умение обращения с динамической памятью.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: src/main.c

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


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

// Создание структуры MusicalComposition

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

// Функции для работы со списком MusicalComposition

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

MusicalComposition* createMusicalCompositionList(char** array_names, char**
array_authors, int* array_years, int n){
    int i;
    MusicalComposition* new_element;
    MusicalComposition* list = createMusicalComposition(array_names[0],
array_authors[0], array_years[0]);
    for (i = 1; i < n; i++){
        new_element = createMusicalComposition(array_names[i], array_authors[i],
array_years[i]);
        push(list, new_element);
    }
    return list;
}

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

```
        free(needful_elem);
    }

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


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

        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;

}
```