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. 3344 _____ Мурдасов М.К.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Изучение линейных списков в языке Си и их использование.

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

MusicalComposition. Были добавлены поля *prev* и *next* типа *MusicalComposition* для хранения указателей на предыдущий и следующий элемент списка соответственно.

createMusicalComposition. Функция создает внутри себя элемент структуры *MusicalComposition*, заполняет его переданными значениями, при этом поля *prev* и *next* равны *NULL*, и возвращает получившийся элемент.

CreateMusicalCompositionList. Функция создает первый элемент списка, заполняя его нулевыми элементами переданных массивов. Далее с помощью цикла функция определяет какой элемент является последним и приравнивает его указатель на следующий элемент списка указателю на новый элемент. Также в новый элемент списка вносится информация о предыдущем элементе. Это происходит столько раз, сколько, согласно входным данным, элементов в введенном списке. Функция возвращает получившийся список.

push. Для сохранения головного указателя при работе со списком функция использует его копию temp. Если головной указатель равен NULL, то новый элемент записывается в него. Если нет, то функция с помощью цикла ищет последний элемент списка и записывает новый после него, при этом записывая в новый элемент информацию о предыдущем.

removeEL. Для сохранения головного указателя при работе со списком функция использует его копию temp. Функция проверяет поле name каждого элемента списка на соответствие имени удаляемого элемента. При обнаружении такого элемента функция удаляет его из списка путем удаления информации о нем из соседних элементов. У предыдущего элемента изменяется информация о следующем и наоборот. Также если соседний элемент только один, то информация о текущем элементе удаляется только из него.

count. Функция перебирает все элементы списка, пока не встретит NULL, и считает каждый элемент.

print_names. Функция перебирает все элементы списка, пока не встретит NULL, и печатает их поля name.

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 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	Корректно
2.	3 Floods Pantera 1996 Points of Authority Linkin Park 2000 Seek and Destroy Metallica 1982 Angel of Death Slayer 1986	Floods Pantera 1996 3 4 Floods Seek and Destroy Angel of Death 3	Корректно

	Points of Authority		

Выводы

Было изучено строение линейных списков в языке Си. С использованием полученных знаний был написан музыкальный список и *api* к нему.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Murdasov_Mikhail_lb2.c

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

typedef struct MusicalComposition{
    char* name;
    char* autor;
    int year;
    struct MusicalComposition* next;
    struct MusicalComposition* prev;

} MusicalComposition;

MusicalComposition*   createMusicalComposition(char*   name,   char*
autor,int year){
                              MusicalComposition*       element       =
malloc(sizeof(MusicalComposition));
        element->name = name;
        element->autor = autor;
        element->year = year;
        element->prev = NULL;
        element->next = NULL;

        return element;
    }

    MusicalComposition*             createMusicalCompositionList(char**
array_names, char** array_authors, int* array_years, int n){
                      MusicalComposition*    MusicalCompositionList    =
createMusicalComposition(array_names[0],               array_authors[0],
array_years[0]);

        for(int i = 1; i<n; i++){
                                     MusicalComposition*    new_el    =
createMusicalComposition(array_names[i],array_authors[i],array_years[i]);
            MusicalComposition* temp = MusicalCompositionList;
            while(temp->next != NULL){
                temp = temp->next;
            }
            temp->next = new_el;
            new_el->prev = temp;
        }

        return MusicalCompositionList;
    }

    void push(MusicalComposition* head, MusicalComposition* element){
        MusicalComposition* temp = head;

        if(temp == NULL){
```

```
        head = element;
    }else{
        while(temp->next != NULL){
            temp = temp->next;
        }
        temp->next = element;
        element->prev = temp;
    }
}

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

    while(temp != NULL){
        if(strstr(temp->name, name_for_remove)){
            if(temp->prev != NULL && temp->next != NULL){
                temp->prev->next = temp->next;
                temp->next->prev = temp->prev;
            }
            if(temp->prev == NULL && temp->next != NULL){
                temp->next->prev = NULL;
            }
            if(temp->prev != NULL && temp->next == NULL){
                temp->prev->next = NULL;
            }
            break;
        }
        temp = temp->next;
    }
}

int count(MusicalComposition* head){
    MusicalComposition* temp = head;
    int count = 0;

    while(temp != NULL){
        count++;
        temp = temp->next;
    }

    return count;
}

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

    while(temp != NULL){
        printf("%s\n", temp->name);
        temp = temp->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 = malloc(sizeof(char)*80);
            char* author = malloc(sizeof(char)*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 = malloc(sizeof(char)*80);
        char* author_for_push = malloc(sizeof(char)*80);
        int year_for_push;

        char* name_for_remove = malloc(sizeof(char)*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->autor, 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;

}
```