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 Волохов М.

Преподаватель Глазунов С.А.

Санкт-Петербург

2024

Цель работы

Освоение создания и работы со списками на языке Си, посредством использования массивов и структур данных.

Задание.

Создайте двунаправленный список музыкальных композиций MusicalComposition и api (application programming interface - в данном случае набор функций) для работы со списком.

1. 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.

2. Функция для создания элемента списка (тип элемента MusicalComposition):

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

3. 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

Для работы с двусвязным списком структур MusicalComposition, написаны следующие функции:

- createMusicalComposition: Выделяет память и копирует переданные данные о композиции, создавая новую музыкальную композицию.
- createMusicalCompositionList: Создает элементы списка, добавляя их в конец, основываясь на переданных данных о композициях.
- push: Находит конец списка и добавляет новый элемент после него.
- removeEl: Проходит по списку, находит элемент с заданным названием для удаления, корректирует связи между соседними элементами и освобождает память, выделенную под этот элемент.
- count: Проходит по всем элементам списка и подсчитывает их количество.
- print_names: Проходит по списку и выводит названия всех композиций.

Программный код см. в приложении А

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№п/п	Входные данные	Выходные данные	Комментарий
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	-

Выводы

Было освоено создание и работа со списками на языке Си, посредством использования массивов и структур данных.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: 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 *next;
    struct MusicalComposition *prev;
} MusicalComposition;

// Создание структуры MusicalComposition
MusicalComposition* createMusicalComposition(char* name, char* author,
int year) {
    MusicalComposition* composition =
(MusicalComposition*)malloc(sizeof(MusicalComposition));
    composition->name = strdup(name);
    composition->author = strdup(author);
    composition->year = year;
    composition->next = NULL;
    composition->prev = NULL;
    return composition;
}

// Функции для работы со списком MusicalComposition

MusicalComposition* createMusicalCompositionList(char** array_names,
char** array_authors, int* array_years, int n) {
    MusicalComposition* head = NULL;
    MusicalComposition* tail = NULL;
    for (int i = 0; i < n; ++i) {
        MusicalComposition* composition =
createMusicalComposition(array_names[i], array_authors[i],
array_years[i]);
        if (head == NULL) {
            head = composition;
            tail = composition;
        } else {
            tail->next = composition;
            composition->prev = tail;
            tail = composition;
        }
    }
    return head;
}

void push(MusicalComposition* head, MusicalComposition* element) {
```

```
    MusicalComposition* current = head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = element;
    element->prev = current;
}

void removeEl(MusicalComposition* head, char* name_for_remove) {
    MusicalComposition* current = head;
    while (current != NULL) {
        if (strcmp(current->name, name_for_remove) == 0) {
            if (current->prev != NULL) {
                current->prev->next = current->next;
            }
            if (current->next != NULL) {
                current->next->prev = current->prev;
            }
            free(current->name);
            free(current->author);
            free(current);
            return;
        }
        current = current->next;
    }
}

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