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

Студентка гр. 3383 _____ Логинова А.Ю.

Преподаватель _____ Гаврилов А.В.

Санкт-Петербург

2024

Цель работы

Написать программу, реализовывающую двунаправленные линейные списки и функции, позволяющие с ними работать, такие как: добавление и удаление элементов, подсчет количества элементов в списке.

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

Двунаправленный линейный список был реализован с помощью структуры *musical_composition* с полями: *name, author, year, next, prev*. Первые три поля — информация о самом объекте структуры, последние два — указатели на следующий и предыдущий объект в списке.

Создание объектов списка реализовано в функции *createMusicalComposition(),* которая принимает на вход данные для полей *name, author, year*. По умолчанию указатели на предыдущий и следующий элемент равны *NULL*. Если выделить память для элемента не удалось, функция сообщает об ошибке. Далее поля инициализируются.

Функция добавления элемента *push()* добавляет элемент в конец, обновляя указатель на следующий элемент у предыдущего элемента списка. Если нет начала списка или элемента, функция сообщает об ошибке.

Функция создания двунаправленного списка реализована с помощью цикла *for,* который в себе вызывает функцию *push()* для добавления элемента.

Удаление элемента реализовано в четырех функциях: *find_el_by_name(), remove_first(), remove_last(), removeEl()*. Первая функция находит элемент, который необходимо удалить с помощью цикла *for* и функции сравнения строк *strcmp()*. Вторая функция заменяет все поля первого элемента на все поля второго, второй элемент удаляется с помощью функции *free*(). Третья функция изменяет указатель *next* предыдущего элемента на указатель *next* следующего элемента. Текущий элемент удаляется. *RemoveEl()* в себе вызывает две эти функции, проверяя, какой элемент нужно удалить. Если не первый и не последний, тогда происходит изменение указателей, где указатель предыдущего элемента next приравнивается к указателю следующего элемента next и таким же образом изменяется указатель prev.

Подсчитывание количества элементов в списке реализовано в функции *count()* с помощью цикла *for.*

Функция *print_names()* выводит значение поля *name* всех элементов списка. Была введена статическая переменная *need_lf* для того, чтобы перенос строки печатался во всех итерациях, кроме первой.

Результаты тестирования программы представлены в таблице 1.

Тестирование.

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 корректно.

Выводы

Была написана программа, позволяющая работать с двусвязными списками. Были изучены возможности структур и реализованы стандартные методы работы с контейнерами.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

// Описание структуры MusicalComposition

typedef struct musical_composition {
    char *name, *author;
    unsigned long year;
    struct musical_composition *next, *prev;
} 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);

void remove_first(MusicalComposition *current);

void remove_last(MusicalComposition *current);

MusicalComposition*  find_el_by_name(MusicalComposition  *head,  char
*name);

void print_error(char *func_name, char *msg);

MusicalComposition*   createMusicalComposition(char*   name,   char*
author, int year) {
                        MusicalComposition     *musicalComposition     =
malloc(sizeof(MusicalComposition));
        if(!musicalComposition) {
                       print_error("createMusicalComposition",   "memory
allocation");
            exit(errno);
        }
        musicalComposition->name = name;
```

```
        musicalComposition->author = author;
        musicalComposition->year = year;
        musicalComposition->next = NULL;
        musicalComposition->prev = NULL;
        return musicalComposition;
    }

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

    void push(MusicalComposition* head, MusicalComposition* element) {
        if (!head)
            return print_error("push", "head is invalid");
        if (!element)
            return print_error("push", "element for push is invalid");

        for (; head->next; head = head->next);
        head->next = element;
        element->prev = head;
    }

    void removeEl(MusicalComposition* head, char* name_for_remove) {
        if(!head)
            return print_error("removeEl", "head is invalid");
        if(!name_for_remove)
                return  print_error("removeEl",  "name  for  remove  is
invalid");

            MusicalComposition  *el_to_remove  =  find_el_by_name(head,
name_for_remove);

        if(!el_to_remove)
            return;
        if (!el_to_remove->prev)
            return remove_first(el_to_remove);
        if (!head->next)
            return remove_last(el_to_remove);

        el_to_remove->prev->next = el_to_remove->next;
        el_to_remove->next->prev = el_to_remove->prev;

        free(el_to_remove);
    }

    int count(MusicalComposition* head) {
        int count = 0;
        for(; head; head = head->next, count++);
        return count;
    }
```

```
void print_names(MusicalComposition*  head) {
    static int need_lf = 0;
    for(; head; head = head->next) {
        if (need_lf)
            printf("\n");
        else
            ++need_lf;
        printf("%s", head->name);
    }
}

void remove_first(MusicalComposition *current) {
    current->name = current->next->name;
    current->author = current->next->author;
    current->year = current->next->year;
    current->next = current->next->next;

    free(current->next->prev);
    current->next->prev = current;
}

void remove_last(MusicalComposition *current) {
    current->prev->next = current->next;
    free(current);
}

MusicalComposition*  find_el_by_name(MusicalComposition  *head,  char
*name) {
    for(; head && name; head = head->next)
        if(strcmp(head->name, name) == 0)
            return head;
    return NULL;
}

void print_error(char *func_name, char *msg) {
    if(func_name && msg)
        printf("[%s()] error: %s\n", func_name, msg);
}

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;

    }
```