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. 3341 _____ Кудин А.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

Цель работы

Для освоения методов работы с линейными структурами данных следует выполнить такие шаги:

1. Освоить программирование операций для списков на языке С.
2. Изучить структуру данных "список".
3. Применить полученные знания для разработки двунаправленного линейного списка.
4. Реализовать специфическую задачу, относящуюся к двусвязным спискам, согласно индивидуальному заданию.
5. Изучить и применить различные операции, применимые к спискам.

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

1. Создание двунаправленного списка музыкальных композиций MusicalComposition:
    - Структура Song определена верно, с полями для названия, автора, года, а также указателями на предыдущий и следующий элементы, что соответствует требованиям к двусвязному списку.
2. Создание элемента списка:
    - Функция createSong корректно выделяет память и инициализирует поля структуры Song. Таким образом, функция соответствует спецификации и позволяет создавать отдельные музыкальные композиции.
3. Функции для работы со списком:
    - Функция createSongList принимает массивы данных и собирает из них двусвязный список. Это соответствует указанию на создание списка композиций MusicalCompositionList.
    - Функция addSong добавляет новую композицию в конец списка, обновляя указатели next и prev соответствующих элементов.
    - Функция deleteSong осуществляет поиск по названию и удаление элемента из списка, также корректно обрабатывая связи с предыдущим и последующим элементами.
    - Функция countSongs возвращает количество элементов в списке, что соответствует требованиям задачи.
    - Функция printTitles выводит названия композиций, перебирая элементы списка.
4. Основная функция main:
    - В main выполняется чтение входных данных и создание списка, что соответствует ожидаемому использованию API списка.
    - Выполняется добавление нового элемента в список и удаление элемента по названию.
    - Подсчет и вывод количества элементов в списке также реализованы.

- В конце main происходит освобождение выделенной памяти, что является хорошей практикой управления ресурсами.

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	Результаты работы данного кода из задания
2.	7 Fields of Gold Sting 1993 In the Army Now Status Quo 1986 Mixed Emotions The Rolling Stones 1989 Billie Jean Michael Jackson	Fields of Gold Sting 1993 7 8 Fields of Gold Mixed Emotions Sonne 3	Удалено большее количество элементов списка

	1983 Seek and Destroy Metallica 1982 Wicked Game Chris Isaak 1989 Points of Authority Linkin Park 2000 Sonne Rammstein 2001 Points of Authority		

Выводы

В ходе данной работы было продемонстрировано умение создавать и манипулировать двусвязными линейными списками, что является фундаментальной навыком в изучении структур данных. Были разработаны и внедрены базовые операции для управления списками, включая добавление, удаление и подсчет элементов, а также изучены принципы динамического выделения памяти в языке программирования C. Полученные знания и навыки в управлении линейными структурами данных могут быть применены для реализации более сложных алгоритмов и систем, что подчеркивает значимость понимания и работы с линейными списками в компьютерных науках..

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

typedef struct Song {
    char* title;
    char* composer;
    int releaseYear;
    struct Song* next;
    struct Song* prev;
} Song;

Song* createSong(char* title, char* composer, int releaseYear) {
    Song* newSong = (Song*)malloc(sizeof(Song));
    newSong->title = (char*)calloc(strlen(title) + 1, sizeof(char));
    newSong->composer = (char*)calloc(strlen(composer) + 1, sizeof(char));
    strcpy(newSong->title, title);
    strcpy(newSong->composer, composer);
    newSong->releaseYear = releaseYear;
    newSong->next = NULL;
    newSong->prev = NULL;
    return newSong;
}

Song* createSongList(char** titles, char** composers, int* years, int count)
{
    Song* listHead = createSong(titles[0], composers[0], years[0]);
    Song* current = listHead;
    for (int i = 1; i < count; i++) {
        Song* nextSong = createSong(titles[i], composers[i], years[i]);
        nextSong->prev = current;
        current->next = nextSong;
        current = nextSong;
    }
    return listHead;
}

void addSong(Song* head, Song* newSong) {
    Song* end = head;
    while (end->next != NULL)
        end = end->next;
    end->next = newSong;
    newSong->prev = end;
}

void deleteSong(Song* head, char* titleToRemove) {
    Song* temp = head;
    while (temp != NULL && strcmp(temp->title, titleToRemove) != 0)
        temp = temp->next;
    if (temp != NULL && strcmp(temp->title, titleToRemove) == 0) {
        if (temp->prev != NULL)
```

```
            temp->prev->next = temp->next;
        if (temp->next != NULL)
            temp->next->prev = temp->prev;
        free(temp->title);
        free(temp->composer);
        free(temp);
    }
}

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

void printTitles(Song* head) {
    Song* current = head;
    while (current != NULL) {
        puts(current->title);
        current = current->next;
    }
}

int main() {
    int length;
    scanf("%d\n", &length);

    char** titles = (char**)malloc(sizeof(char*) * length);
    char** composers = (char**)malloc(sizeof(char*) * length);
    int* years = (int*)malloc(sizeof(int) * length);

    for (int i = 0; i < length; i++) {
        char title[80];
        char composer[80];

        fgets(title, 80, stdin);
        fgets(composer, 80, stdin);
        fscanf(stdin, "%d\n", &years[i]);

        title[strcspn(title, "\n")] = 0;
        composer[strcspn(composer, "\n")] = 0;

        titles[i] = (char*)malloc(sizeof(char) * (strlen(title) + 1));
        composers[i] = (char*)malloc(sizeof(char) * (strlen(composer) +
1));

        strcpy(titles[i], title);
        strcpy(composers[i], composer);
    }

    Song* head = createSongList(titles, composers, years, length);

    char titleForAdd[80];
    char composerForAdd[80];
```

```
    int yearForAdd;

    char titleForDelete[80];

    fgets(titleForAdd, 80, stdin);
    fgets(composerForAdd, 80, stdin);
    fscanf(stdin, "%d\n", &yearForAdd);
    titleForAdd[strcspn(titleForAdd, "\n")] = 0;
    composerForAdd[strcspn(composerForAdd, "\n")] = 0;

    Song* newSong = createSong(titleForAdd, composerForAdd, yearForAdd);

    fgets(titleForDelete, 80, stdin);
    titleForDelete[strcspn(titleForDelete, "\n")] = 0;

    printf("%s %s %d\n", head->title, head->composer, head->releaseYear);
    int k = countSongs(head);

    printf("%d\n", k);
    addSong(head, newSong);

    k = countSongs(head);
    printf("%d\n", k);

    deleteSong(head, titleForDelete);
    printTitles(head);

    k = countSongs(head);
    printf("%d\n", k);

    for (int i = 0; i < length; i++) {
        free(titles[i]);
        free(composers[i]);
    }
    free(titles);
    free(composers);
    free(years);

    return 0;
}
```