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. 3342 _____ Белаид Фарук

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Научиться реализовывать структурную и функциональную составляющую двунаправленных линейных списков в языке С. Изучить принцип создания подобных списков с помощью структур.

Задание

Создайте двунаправленный список музыкальных композиций MusicalComposition и api (application programming interface - 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]).
- 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.

Выполнение работы

1) Структура *MusicalComposition* содержит информацию о музыкальной композиции: название, автор и год выпуска. Она также содержит два указателя на другие элементы *prev* и *next*, которые используются для создания двусвязного списка.

2) Функция *createMusicalComposition* создает и возвращает новый элемент структуры *MusicalComposition*. В функции выделяется память для каждой строки *name* и *author*, и используется функция *strcpy* для копирования переданных строк в выделенную память.

3) Функция *createMusicalCompositionList* создает и возвращает двусвязный список *MusicalComposition*. Она принимает массивы строк *array_names* и *array_authors*, и массив целых чисел *array_years*, содержащие информацию о музыкальных композициях, а также количество элементов *n*. В функции создается новый элемент структуры *MusicalComposition* для каждой композиции, и каждый элемент связывается с предыдущим и следующим элементом, чтобы создать двусвязный список. Функция возвращает указатель на первый элемент списка.

4) Функция *push* добавляет новый элемент в конец двусвязного списка. Она принимает указатель на голову списка *head* и указатель на элемент, который нужно добавить *element*. Функция проходит по всем элементам списка до последнего элемента, и добавляет новый элемент после него.

5) Функция *removeEl* удаляет элемент из двусвязного списка. Она принимает указатель на голову списка *head* и строку *name_for_remove*, содержащую название композиции, которую нужно удалить. Функция находит элемент с соответствующим названием и удаляет его из списка. Если удаляемый элемент имеет предыдущий или следующий элемент, то эти элементы переустанавливают свои указатели, чтобы обойти удаленный элемент. Функция освобождает память, выделенную для удаляемого элемента.

6) Функция *count* считает количество элементов в двусвязном списке. Она принимает указатель на голову списка *head* и проходит по всем элементам списка, подсчитывая количество элементов.

7) Функция *print_names* выводит названия всех композиций в двусвязном списке. Она принимает указатель на голову списка *head* и проходит по всем элементам списка, выводя название каждой композиции.

В целом, данный код представляет собой реализацию двусвязного списка для хранения музыкальных композиций.

Разработанный программный код см. в приложении А.

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	Ответ верный.

Выводы

Разработанная программа написана на языке Си и предназначена для поиска заданного слова в строке. Сначала программа разбивает строку на слова, сортирует их в алфавитном порядке с помощью функции qsort, а затем использует функцию bsearch для поиска заданного слова в отсортированном массиве. Если слово найдено, программа выводит "exists", иначе - "doesn't exist". Эта программа демонстрирует применение стандартных функций языка Си для работы со строками и массивами, а также алгоритма бинарного поиска.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

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

MusicalComposition *createMusicalComposition(char *name, char *author,
int year) {
    MusicalComposition *composition = (MusicalComposition *)
malloc(sizeof(MusicalComposition));
    if (composition == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }
    composition->name = malloc(81 * sizeof(char));
    composition->author = malloc(81 * sizeof(char));
    if (composition->name == NULL || composition->author == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }
    composition->year = year;
    composition->prev = NULL;
    composition->next = NULL;

    strcpy(composition->author, author);
    strcpy(composition->name, name);
    return composition;
}

MusicalComposition *createMusicalCompositionList(char **array_names,
char **array_authors, int *array_years, int n) {
    MusicalComposition *head;
    MusicalComposition *last_elem;
    for (int i = 0; i < n; i++) {
        MusicalComposition *new_elem;
        if (i == 0) {
            new_elem = createMusicalComposition(array_names[i],
array_authors[i], array_years[i]);
            head = new_elem;
        } else {
            new_elem = createMusicalComposition(array_names[i],
array_authors[i], array_years[i]);
            last_elem->next = new_elem;
```

```
            new_elem->prev = last_elem;
        }
        last_elem = new_elem;
    }
    return head;
}

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

void removeEl(MusicalComposition *head, char *name_for_remove) {
    MusicalComposition *next_elem = head->next;
    while (next_elem != NULL) {
        if (strcmp(next_elem->name, name_for_remove) == 0) {
            if (next_elem->next != NULL) {
                next_elem->next->prev = next_elem->prev;
            }
            if (next_elem->prev != NULL) {
                next_elem->prev->next = next_elem->next;
            }
            free(next_elem->name); // Free memory for name
            free(next_elem->author); // Free memory for author
            free(next_elem); // Free memory for the element
            break;
        }
        next_elem = next_elem->next;
    }
}

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

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

    if (names == NULL || authors == NULL || years == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }

    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));

        if (names[i] == NULL || authors[i] == NULL) {
            fprintf(stderr, "Memory allocation failed\n");
            exit(EXIT_FAILURE);
        }

        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;
}
```