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. 3341 Мальцев К.Л.

Преподаватель Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Целью данной работы является освоение работы с линейными списками. Для достижения этой цели необходимо выполнить следующие задачи:

- изучить структуру «список»;

- ознакомиться с операциями, используемыми для работы со списками;

- изучить способы реализации этих операций на языке программирования C;

- разработать программу, которая будет реализовывать двусвязный линейный список и решать конкретную задачу в соответствии с заданием.

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]).

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.

Основные теоретические положения

Двусвязный список - это структура данных, состоящая из узлов, каждый из которых содержит данные и два указателя: один на предыдущий узел и один на следующий узел. Отличие двусвязного списка от односвязного заключается в том, что у двусвязного списка есть возможность двигаться в обоих направлениях: как вперед, так и назад по списку.

Основные теоретические положения о двусвязном списке:

1. У каждого узла есть указатели на предыдущий и следующий узел, что делает возможным эффективное добавление и удаление элементов в середине списка.

2. Двусвязный список обеспечивает быстрый доступ к любому элементу по индексу, за счет наличия указателя на ближайшие узлы.

3. Добавление и удаление элементов в начале и конце списка требует O(1) времени, что делает двусвязные списки эффективными структурами данных для определенных операций.

4. Реализация двусвязного списка требует дополнительного пространства памяти для хранения указателей на предыдущие узлы, что может увеличить использование памяти.

В целом, двусвязные списки широко используются в программировании, когда требуется эффективное добавление и удаление элементов в середине списка, а также когда необходимо быстро обращаться к элементам как вперед, так и назад.

Выполнение работы

1. Создание структуры данных MusicalComposition, которая содержит информацию о музыкальной композиции (название, автор, год выпуска) и указатели на следующий и предыдущий элементы в двусвязном списке.

2. Создание функции createMusicalComposition, которая выделяет память под новую музыкальную композицию, заполняет поля структуры данными и возвращает указатель на созданную композицию.

3. Создание функции createMusicalCompositionList, которая создает двусвязный список из массивов названий, авторов и годов выпуска композиций. Передается массивы и их размер n. Функция создает первый элемент списка, а затем добавляет последующие элементы в список, связывая их указателями prev и next.

4. Создание функции count, которая подсчитывает количество элементов в списке, начиная с головного узла.

5. Создание функции push, которая добавляет новый элемент в конец списка. Если список пустой, новый элемент становится головным. Иначе, новый элемент добавляется после последнего элемента, устанавливая связи с предыдущими элементами.

6. Создание функции removeEl, которая удаляет элемент из списка по заданному названию. Если удаляется головной элемент, то списка переустанавливается указатель на следующий элемент. В противном случае, устанавливаются правильные связи между соседними элементами для удаленного элемента.

7. Создание функции printNames, которая выводит на экран названия всех композиций из списка, начиная с головного элемента и до конца списка.

Таким образом, данный код реализует операции создания двусвязного списка музыкальных композиций, добавления новых элементов, удаления элементов по названию и вывода имен композиций на экран.

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.	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	Тест с e.moevm

2.	0 Sonne Rammstein 2001 Sonne	0 1 0	Проверка крайнего случая создания двусвязного списка на основе пустого массива, добавления в него 1 элемента и удаления из него этого же элемента (135 строка была закомментирована)

Выводы

В ходе данного исследования была поставлена цель освоения работы с линейными списками. Для достижения этой цели были выполнены следующие задачи:

1. Изучение структуры "список" как абстрактной структуры данных, позволяющей хранить и организовывать элементы в линейной последовательности.

2. Ознакомление с операциями, используемыми для работы со списками, такими как добавление элемента, удаление элемента, поиск элементов и т.д.

3. Изучение способов реализации этих операций на языке программирования C, включая работу с указателями и динамическим выделением памяти.

4. Разработка программы, которая реализует двусвязный линейный список и решает конкретную задачу в соответствии с индивидуальным заданием. Программа содержит функции для создания списка, добавления элементов, удаления элементов и вывода информации о списках.

Таким образом, выполнение поставленных задач позволило освоить работу с линейными списками и применить полученные знания при разработке программы на языке C.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

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

MusicalComposition*   createMusicalComposition(char*   name,   char*
author, int year) {
    MusicalComposition*            musicalComposition            =
(MusicalComposition*)malloc(sizeof(MusicalComposition));
    musicalComposition->name = name;
    musicalComposition->author = author;
    musicalComposition->year = year;
    musicalComposition->next = NULL;
    musicalComposition->prev = NULL;
    return musicalComposition;
}

MusicalComposition* createMusicalCompositionList(char** arrayNames,
char** arrayAuthors, int* arrayYears, int n) {
    if (n == 0) {
        return NULL;
    }
    MusicalComposition*                   head                    =
createMusicalComposition(arrayNames[0], arrayAuthors[0], arrayYears[0]);
    MusicalComposition* current = head;
    for (int i=1; i<n; i++) {
        current->next   =   createMusicalComposition(arrayNames[i],
arrayAuthors[i], arrayYears[i]);
        current->next->prev = current;
        current = current->next;
    }
    return head;
}

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

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

void removeEl(MusicalComposition** head, char* nameForRemove) {
    MusicalComposition* current = *head;
    if (strcmp((*head)->name, nameForRemove) == 0) {
        if ((*head)->next != NULL) {
            (*head)->next->prev = NULL;
        }
        (*head) = (*head)->next;
        return;
    }
    for (int i=0; i<count(*head); i++) {
        if (strcmp(current->next->name, nameForRemove) == 0) {
            if (current->next->next != NULL) {
                current->next->next->prev = current;
            }
            current->next = current->next->next;
            break;
        }
        current = current->next;
    }
}

void printNames(MusicalComposition* head) {
    MusicalComposition* current = head;
    for (int i=0; i<count(head); i++) {
        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 nameForPush[80];
        char authorForPush[80];
        int yearForPush;

        char nameForRemove[80];

        fgets(nameForPush, 80, stdin);
        fgets(authorForPush, 80, stdin);
        fscanf(stdin, "%d\n", &yearForPush);
        (*strstr(nameForPush,"\n"))=0;
        (*strstr(authorForPush,"\n"))=0;

        MusicalComposition*              elementForPush              =
createMusicalComposition(nameForPush, authorForPush, yearForPush);

        fgets(nameForRemove, 80, stdin);
        (*strstr(nameForRemove,"\n"))=0;

        printf("%s %s %d\n", head->name, head->author, head->year);
        int k = count(head);

        printf("%d\n", k);
        push(&head, elementForPush);

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

        removeEl(&head, nameForRemove);
        printNames(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;

    }
```