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 _____ Трофимов В.О.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Целью работы является освоение работы с линейными списками.

Для достижения поставленной цели требуется решить следующие задачи:

1. Ознакомиться со структурой данных «список».
2. Ознакомиться с операциями, используемыми для списков.
3. Изучить способы реализации этих операций на языке Си.
4. Написать программу, реализующую двусвязный линейный список и решающую задачу в соответствии с индивидуальным 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.

Функцию main менять не нужно.

Основные теоретические положения

Перед тем как задать определение линейного списка, введем пару терминов, которые будут использоваться далее:

• Узел – один элемент из списка, который связан с другими элементами;

• Голова списка (head) – первый элемент в списке;

• Хвост (tail) – все элементы списка, идущие после головы.

Линейный односвязный список – это структура данных, представляющая собой последовательность узлов, каждый из которых хранит какие-то полезные данные и указатель на следующий элемент. Важно, что в памяти элементы не находятся последовательно, в отличие от массивов.

Двусвязный список – это список с возможностью идти в обе стороны, в отличие от односвязного. Каждый элемент двусвязного списка имеет указатель на предыдущий элемент и на следующий.

Возможности двухсвязного списка:

1. Двусвязный список - это структура данных, состоящая из узлов, каждый из которых содержит два поля: данные и ссылки на предыдущий и следующий узлы.

2. Двусвязный список позволяет обращаться к любому элементу списка как в прямом, так и в обратном направлении, так как каждый узел имеет ссылку на предыдущий и следующий узел.

3. Вставка и удаление элементов в двусвязном списке происходит эффективно, так как не требуется переустановка ссылок на соседние узлы.

4. Операции доступа к элементам по индексу в двусвязном списке медленнее, чем в односвязном списке, так как для доступа к элементу необходимо пройти через все предшествующие или последующие элементы.

5. Двусвязный список часто используется в различных алгоритмах сортировки, как удобная структура данных для операций вставки и удаления элементов.

Выполнение работы

В программе объявлены следующие функции:

1) MusicalComposition* createMusicalComposition(char* name, char* author,int year);

2) MusicalComposition* createMusicalCompositionList(char** array_names, char** array_authors, int* array_years, int n)

3) void push(MusicalComposition* head, MusicalComposition* element);

4) void removeEl(MusicalComposition* head, char* name_for_remove);

5) int count(MusicalComposition* head);

6) void print_names(MusicalComposition* head);

Объявлена структура MusicalComposition, где определены поля: char* name, char* author, int year, а также указатели на следующий(next) и предыдущий(prev) элементы списка.

1. Функция createMusicalComposition создает новую композицию с заданными значениями name, author, year. Здесь выделяется память под новый элемент структуры, устанавливаются значения полей и возвращается указатель на созданный элемент.

2. Функция createMusicalCompositionList создает список композиций на основе переданных массивов char** array_names, char** array_authors, int* array_years и их размера int n. Она создает первый элемент списка head, а затем в цикле добавляет остальные элементы, связывая их между собой вперед и назад.

3. Функция push принимает указатели на два объекта типа MusicalComposition: head и element. Функция добавляет элемент element в конец списка, начинающегося с head. Сначала объявляется указатель current и инициализируется значением head. Затем выполняется проверка, если head равен NULL, то значит список пустой, и элемент element становится головой списка (head) и функция завершается. Если список не пустой, то происходит итерация по списку с помощью цикла while. Цикл завершается, когда текущий элемент current указывает на последний элемент списка (у которого next равен

NULL). После того как цикл завершается, текущий элемент current становится предпоследним элементом списка, его next указывает на новый элемент element, а prev нового элемента element указывает на текущий элемент current. Таким образом, элемент element добавляется в конец списка.

4. Функция removeEl удаляет элемент по заданному названию name_for_remove из списка. При этом происходит рассмотрение трех случаев: если удаляемый элемент является первым в списке, если он имеет предыдущий элемент и если он имеет следующий элемент.

5. Функция count считает количество элементов в списке, проходя его от начала до конца.

6. Функция print_names выводит названия композиций из списка, проходя по всем элементам.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
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
2.	1 Billie jean Michael Jackson 2001	Billie jean Michael Jackson 2001 1 2

	Feel Syncole 2020 Syncole	Billie jean Feel 2

Выводы

Цель работы была достигнута, освоена работа с линейными списками в языке С и изучена структура данных “список”, представляющая собой последовательность элементов данных, у которых каждый элемент связан с предыдущим и последующим элементом. Изучены операции, используемы при работе со списками, а именно добавление, удаление элемента из списка, поиск элемента в списке и прохождение по списку. Реализована программа, которая содержит двусвязный линейный список и решает задачу с использованием этой структуры данных в соответствии текстом задания.

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*    Composition    =    (MusicalComposition*)
malloc(sizeof(MusicalComposition));
    Composition->name = name;
    Composition->author = author;
    Composition->year = year;
    Composition->next = NULL;
    Composition->prev = NULL;
    return Composition;
}

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

void push(MusicalComposition** head, MusicalComposition* element){
    MusicalComposition* current = *head;
    if (*head == NULL){
        *head = element;
        return;
    }

    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){
                current->prev->next = current->next;
            }
            if (current->next){
                current->next->prev = current->prev;
            }
        }
        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;
    }
```