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

Студентка гр. 3341 _____ Шуменков А.П.

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:

- ознакомиться со структурой «список»;
- ознакомиться со списком операций используемых для списков;
- изучить способы реализации этих операций на языке С;
- 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.

Выполнение работы

Структура MusicalComposition, представляет музыкальную композицию с полями для имени, автора, года и ссылками на следующую и предыдущую композиции.

Функция createMusicalComposition принимает параметры name (имя композиции), author (автор композиции) и year (год создания композиции), затем выделяет динамическую память под новую структуру MusicalComposition, инициализирует поля структуры переданными значениями и возвращает указатель на новую структуру.

Функция createMusicalCompositionList принимает массивы array_names, array_authors и array_years, содержащие соответственно имена, авторов и года создания композиций, и их количество n. Вначале она создает первую композицию с помощью вызова функции createMusicalComposition и сохраняет указатель на нее в переменной head. Затем циклом создает и присоединяет к списку остальные композиции из массивов, устанавливая связи next и prev между композициями. В конце функция возвращает указатель на первую композицию (head). MusicalComposition* createMusicalComposition(char* name, char* author, int year) принимает указатель на название композиции, автора композиции и год создания, с помощью функции malloc выделяет память для одного элемента типа MusicalComposition, полям name, author, year данного элемента присваивает соответствующие значения, полям prev, next присваивает NULL, возвращает указатель на созданный элемент.

count(struct MusicalComposition* head):

- Функция принимает указатель на голову списка MusicalComposition и начинает считать количество элементов в списке.

- Инициализируется переменная counting равная 0.

- Устанавливается указатель curr_musical_composition на голову списка.

- Затем в цикле while проверяется, что curr_musical_composition не равен NULL.

- Внутри цикла указатель curr_musical_composition смещается на следующий элемент списка и увеличивается счетчик counting.

- По завершении цикла возвращается значение counting, которое является количеством элементов в списке.

push(MusicalComposition** head, MusicalComposition* element):

- Функция добавляет новый элемент element в конец списка, на который указывает head.

- Если указатель на голову head пустой, то новый элемент element становится головой списка.

- Если голова не пустая, то происходит проход до последнего элемента в списке.

- Затем указатель последнего элемента устанавливается на новый элемент element, а также устанавливается указатель prev этого нового элемента на предыдущий элемент списка.

removeEl(MusicalComposition* head, char* nameForRemove):

- Функция удаляет элемент из списка по имени nameForRemove.

- Если имя элемента в голове списка совпадает с nameForRemove, то голова списка смещается на следующий элемент.

- В противном случае происходит цикл while, который ищет элемент соответствующий имени nameForRemove.

- Найденый элемент удаляется из списка (устанавливается указатель на следующий элемент) и перестраивается связь между соседними элементами списка.

printNames(MusicalComposition* head):

- Функция выводит на экран имена элементов списка.

- Устанавливается временный указатель tmp на голову списка.

- Затем в цикле while выводится на экран имя элемента и tmp смещается на следующий элемент списка.

- Процесс повторяется пока tmp не станет равным NULL, что означает достижение конца списка.

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.	3 Fields of Gold Sting 1993 In the Army Now Status Quo 1986 Mixed Emotions The Rolling Stones 1989 Sonne Rammstein 2001 Mixed Emotions	Fields of Gold Sting 1993 3 4 Fields of Gold In the Army Now Sonne 3	*main* считывает в массив 3 названия композиций, авторов и лет; название, автора и год композиции, которую нужно будет добавить в список, затем название композиции, которую нужно удалить. Выводит название, автора и год первой композиции, количество элементов в списке (3), затем добавляет элемент и снова выводит количество элементов (4), удаляет элементы с заданным названием, после выводит названия композиций и количество элементов в списке.
2.	3 In the Army Now Sting 1993 Mixed Emotions	In the Army Now Sting 1993 3 4	В данном примере удаляется не одна, а 3 композиции с заданным именем.

	Status Quo 1986 Mixed Emotions The Rolling Stones 1989 Mixed Emotions Rammstein 2001 Mixed Emotions	In the Army Now 1	

Выводы

В ходе выполнения работы были изучены:

- основные принципы работы с линейными списками;
- структура списков и операции, применяемые к ним;
- способы реализации этих операций на языке С;
- написана программа, реализующаю двусвязный линейный список и решающаю задачу в соответствии с индивидуальным заданием.

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             *new_musical_composition            =
(MusicalComposition*)calloc(1, sizeof(MusicalComposition));
    new_musical_composition->name = name;
    new_musical_composition->author = author;
    new_musical_composition->year = year;
    new_musical_composition->next = NULL;
    new_musical_composition->prev = NULL;
    return new_musical_composition;
}


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

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

```
}

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){
    if(strcmp(head->name, nameForRemove) == 0){
        head = head->next;
        head->prev = NULL;
    } else {
    while(strcmp(head->next->name, nameForRemove) != 0){
        head = head->next;
    }
    head->next = head->next->next;
    if(head->next != NULL)
        head->next->prev = head;
    }
}

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

}
```