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.0382 _____ Диденко Д.В.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Цель работы.

Изучение принципов работы с линейными списками.

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.

Список - некоторый упорядоченный набор элементов любой природы.

Линейный однонаправленный (односвязный) список - список, каждый элемент которого хранит помимо значения указатель на следующий элемент(рис. 1). В последнем элементе указатель на следующий элемент равен NULL (константа нулевого указателя).

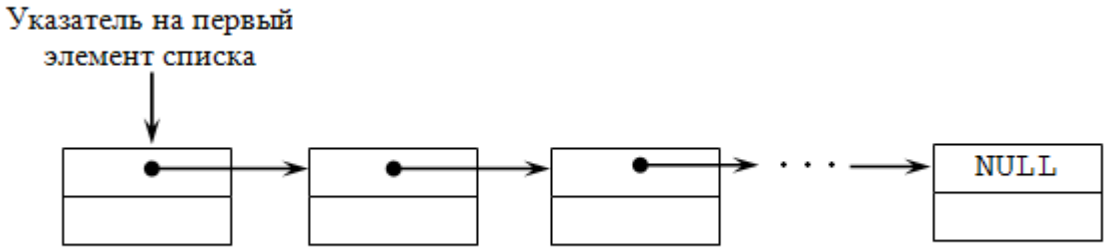

Рисунок 1 — Иллюстрация устройства списка

Каждый узел двунаправленного (двусвязного) линейного списка (ДЛС) содержит два поля указателей — на следующий и на предыдущий узлы. Указатель на предыдущий узел корня списка содержит нулевое значение.

Указатель на следующий узел последнего узла также содержит нулевое значение.

Основные действия, производимые над узлами ДЛС:

- Инициализация списка
- Добавление узла в список
- Удаление узла из списка
- Удаление корня списка
- Вывод элементов списка
- Вывод элементов списка в обратном порядке
- Взаимообмен двух узлов списка

Добавление узла в ДЛС включает в себя следующие этапы:

- создание узла добавляемого элемента и заполнение его поля данных;
- переустановка указателя «следующий» узла, предшествующего добавляемому, на добавляемый узел;
- переустановка указателя «предыдущий» узла, следующего за добавляемым, на добавляемый узел;
- установка указателя «следующий» добавляемого узла на следующий узел (тот, на который указывал предшествующий узел);
- установка указателя «предыдущий» добавляемого узла на узел, предшествующий добавляемому (узел, переданный в функцию).

Удаление узла ДЛС включает в себя следующие этапы:

- установка указателя «следующий» предыдущего узла на узел, следующий за удаляемым;
- установка указателя «предыдущий» следующего узла на узел, предшествующий удаляемому;
- освобождение памяти удаляемого узла.

Выполнение работы.

Исходный код решения задачи см.в приложении А.

Определяется структура MusicalComposition через typedef с полями name, author, year, максимум символов для name и author — 80. А также два дополнительных поля типа указателей на структуру MusicalComposition для реализации двусвязного списка — next и prev.

Функция createMusicalComposition создает структуру MusicalComposition. В теле выделяется память и определяются поля name, author, year.

Функция createMusicalCompositionList создает двусвязный список структур MusicalComposition с помощью поданных массивов names,authors и years и возвращает указатель на первый элемент списка — head. В поля next и и prev записываются соответствующие значения для каждого элемента списка.

Функция push добавляет структуру MusicalComposition в конец списка соответствующих структур. Принимает на вход указатель на первый элемент списка(head) и указатель на структуру, которую необходимо добавить(element). В цикле while находится последний элемент списка, в поле next которого записывает адрес нового элемента вместо NULL, в поле next нового элемента записывается NULL, а в поле prev — адрес предыдущего элемента(который был последним до добавления).

Функция removeEl удаляет элемент из списка по полю name. Принимает на вход указатель на head и строку, совпадающую с полем name элемента, который необходимо удалить. В цикле while функцией strcmp находится нужный элемент списка и присваивается его адрес в head. Далее обрабатываются три возможных сценария:

1. head первый.

2. head последний.

3. head между первым и последним.

Во всех случаях очищается память, выделенная под удаляемую структуру, и перезаписываются поля next и prev.

Функция count считает количество элементов списка. Принимает на вход указатель на head. Создается переменная-счетчик i, в цикле while перебираются элементы списка до последнего и на каждой итерации увеличивается на 1 значение i.

Функция print_names выводит на экран поля name элементов списка. Перебираются до последнего все элементы списка и поле name выводится на экран функцией printf.

*Во всех описанных функциях head не затирается, потому что фактически передается копия адреса.

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	Программа работает верно
2.	5 Fields of Gold	Fields of Gold Sting 1993	Программа работает верно

	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 Sonne Rammstein 2001 Billie Jean	5 6 Fields of Gold In the Army Now Mixed Emotions Seek and Destroy Sonne 5	

Выводы.

Были изучены принципы работы с линейными односвязными и двусвязными списками.

Разработана программа, которая использует двунаправленный список музыкальных композиций MusicalComposition и api (application programming interface - в данном случае набор функций) для работы со списком: функция для создания элемента списка; функция создания списка музыкальных композиций; функция добавления новой композиции в конец списка; функция удаления элемента списка по значению name; функция для вывода длины списка; функция вывода названий композиций.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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


// Описание структуры MusicalComposition
typedef struct MusicalComposition{
    char name[80];
    char author[80];
    int year;
    struct MusicalComposition* next;
    struct MusicalComposition* prev;
}MusicalComposition;

// Создание структуры MusicalComposition

MusicalComposition* createMusicalComposition(char* name, char*
autor,int year){
    MusicalComposition *New =
(MusicalComposition*)malloc(1*sizeof(struct MusicalComposition));
    strcpy(New->name,name);
    strcpy(New->author,autor);
    New->year = year;
    return New;
}

// Функции для работы со списком MusicalComposition

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 *p_next;
    head->prev = NULL;
    for (int i = 1; i < n-1;i++){
        p_next = createMusicalComposition(array_names[i],
array_authors[i],array_years[i]);
        p_next->prev = head;
        head->next = p_next;
        head = p_next;
    }
    p_next = createMusicalComposition(array_names[n-1],
array_authors[n-1],array_years[n-1]);
    p_next->prev = head;
    head->next = p_next;
    p_next->next = NULL;
    head = p_next;
    while (head->prev != NULL){
        head = head->prev;
    }
    return head;
}
```

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

void removeEl(MusicalComposition* head, char* name_for_remove){
    while (strcmp(head->name,name_for_remove) != 0){
        head = head->next;
    }
    if (head->prev == NULL){
       head->next->prev = NULL;
       free(head);
    }else if(head->next == NULL){
        head->prev->next = NULL;
        free(head);
    }else{
        head->prev->next = head->next;
        head->next->prev = head->prev;
        free(head);
    }
}


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


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