МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Математического обеспечения электронно-вычислительных машин

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Программирование»

ТЕМА: ЛИНЕЙНЫЕ СПИСКИ

Студентка гр. 0382 _____ Рубежова Н.А.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Цель работы.

Изучить линейные списки и их применение в языке Си, освоить их возможности, пользуясь ими в программном коде.

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.

Однонаправленный список.

Для удобства назовем первый элемент списка *head*:

Node head = (Node*)malloc(sizeof(Node));*

Второй элемент списка назовем *tmp*:

```
Node* tmp = (Node*)malloc(sizeof(Node));
```

Поле next первого элемента *head* должно ссылаться на второй элемент tmp:

head->next = tmp;

Создадим остальные элементы списка:

for(i = 1; i < 10; i++){ tmp->next = (Node)malloc(sizeof(Node)); // 1*

tmp->next->next = NULL; // 2

tmp = tmp->next; // 3

}

Выполнение работы.

1. Опишем структуру *MusicalComposition*. Чтобы каждый раз не писать *struct MusicalComposition* воспользуемся оператором *typedef* и переопределим тип *struct MusicalComposition* как *Elem.* У структуры *Elem* будет 5 полей: char *name[81], char author[81], int year, Elem* prev, elem* next* которые будут хранить строки –название композиции, автор и год выпуска, указатели на предыдущий и следующий элементы списка соответственно.

2. Определим функцию создания структуры *createMusicalComposition().*

Аргументами функции будут название композиции *char* name*, название автора *char* author,* год выпуска *int year*. Создаем указатель на структуру *Elem* my_elem,* который будет хранить адрес нашего элемента. И выделяем память под элемент размером *sizeof(Elem).* Далее, присваиваем соответствующие переданным данным значения полей нашему элементу-структуре, т.к. мы работаем с указателем, для этого воспользуемся оператором *'->'.* Поля с данными у нас заполнены, можно возвращать указатель на структуру: *return my_elem;*

3. Определим функцию создания списка *createMusicalCompositionList()*. Аргументы функции – массивы названий

композиций, авторов, годов выпуска, а также количество музыкальных композиций будущего списка. Для начала создадим указатель на структуру *Elem* head,* который будет указывать на первый элемент нашего списка. Вызовем функцию создания элемента из предыдущего пункта, которая вернет нам указатель на этот элемент и сохраним его в *head.* Data-данные для *head* у нас заполнены, осталось заполнить поля *prev* и *next.* Поле *head->prev=NULL,* так как это первый элемент нашего списка и предыдущего у него нет. Для того, чтобы заполнить поле *head->next,* создадим новый, следующий за ним элемент. Указатель на него вернет нам функция *createMusicalComposition(),* сохраним его в *Elem* cur,* который далее будет хранить адрес «текущего» элемента. Присвоим *head->next=cur, cur->prev=head.* Перейдем к созданию остальных элементов. Заходим в цикл *for*, чтобы сгенерировать оставшееся количество элементов списка. Воспользуемся вспомогательной переменной *tmp*, чтобы хранить указатель *cur* для дальнейшего присвоения и работы с полями *prev* и *cur.* В *tmp* сохраняем *cur.* Создаем новый элемент *cur.* И устанавливаем связь между *tmp(*прошлым элементом *cur)* и новым элементов *cur* с помощью блока присвоений: *tmp->next=cur; cur->prev=tmp;* После генерации всех элементов останется присвоить полю *next* последнего элемента *NULL*, что будет знаменовать конец списка, в случае его «прочтения» с первого по последний элемент. Возвращаем указатель на первый элемент списка *head.*

4. Определим функцию добавления элемента в конец списка *push().* Аргументы функции – указатель на первый элемент списка *Elem* head* и указатель на добавляемый элемент *Elem* element.* Сначала проходим по всему списку, чтобы дойти до последнего элемента с помощью цикла *while(head->next)head=head->next;* Далее «связываем» последний элемент нашего списка и добавляемый: *head->next=element; element->prev=head;* Так как теперь добавленный элемент – последний, значит, его поле *next* должно ссылаться на *NULL* – конец списка. Т.е. *element->next=NULL;* Все необходимые преобразования совершены.

5. Определим функцию удаления элемента с указанным названием композиции removeEl*().* Аргументы функции – указатель на первый элемент списка *Elem* head* и строка-название композиции *char* name_for_remove.* Перебираем элементы списка до совпадения с названием с помощью цикла *while* аналогично предыдущим пунктам. Если совпадающий элемент не последний, то полю *next* предыдущего элемента присваиваем указатель на следующий за найденным элементом и полю *prev* следующего за найденным элементом присвоим указатель на предыдущий найденному элементу: *head->prev->next=head->next; head->next->prev=head->prev.* Если же совпадающий элемент оказался последним, то чтобы убрать его из списка, достаточно полю *next* предыдущего элемента присвоить *NULL,* что будет ознаменовать конец списка.

6. Определим функцию *count(),* которая будет возвращать количество элементов списка. Аргумент функции – указатель на первый элемент списка *Elem* head.* Инициализируем переменную-счетчик *int c=1,* равна единице, так как перебирая элементы, мы уже находимся на первом элементе. Воспользуемся циклом *while(head->next),* на каждом проходе цикла увеличиваем счетчик и переходим к следующему элементу. Возвращаем полученное *c*.

7. Определим функцию вывода на экран названий композиций списка *print_names()*. Аргумент функции – указатель на первый элемент списка *Elem* head.* Функция ничего не возвращает, лишь осуществляет вывод на экран. Запускаем цикл *while(head)*, чтобы перебирать элементы от первого до последнего, пока не встретится *head->next=NULL*. Выводим на экран значение поля текущего элемента с помощью *printf("%s\n",head->name)* и присваиваем *head=head->next,* чтобы перейти к следующему элементу 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.	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	Вывод верный

Выводы.

Были изучены линейные списки и их применение в языке Си, а также освоены их возможности посредством использования их в программном коде.

Разработана программа, которая создает двунаправленный список музыкальных композиций, а затем реализует функции, работающие с этим списком: добавляет последний элемент к списку, удаляет элемент с заданным названием композиции, считает количество элементов в списке и выводит их на экран.

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[81];
    char author[81];
    int year;
    struct MusicalComposition* prev;
    struct MusicalComposition* next;
}Elem;

// Создание структуры MusicalComposition

Elem* createMusicalComposition(char* name, char* author,int year){
    Elem* my_elem=(Elem*)malloc(sizeof(Elem));
    strncpy(my_elem->name,name,81);
    strncpy(my_elem->author,author,81);
    my_elem->year=year;
    return my_elem;
}

// Функции для работы со списком MusicalComposition

Elem* createMusicalCompositionList(char** names, char** authors,
int* years, int n){
    int i=0;
    Elem*
head=createMusicalComposition(names[i],authors[i],years[i]);
    head->prev=NULL;
    Elem*
cur=createMusicalComposition(names[i+1],authors[i+1],years[i+1]);
    head->next=cur;
    cur->prev=head;
    Elem* tmp;
    for(i=2;i<n;i++){
        tmp=cur;

cur=createMusicalComposition(names[i],authors[i],years[i]);
        tmp->next=cur;
        cur->prev=tmp;
    }
    cur->next=NULL;
```

```
    return head;
}

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

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

int count(Elem* head){
    int c=1;
    while(head->next){
        head=head->next;
        c++;
    }
    return c;
}

void print_names(Elem* head){
    while(head){
        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);

        }
        Elem*  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;

        Elem*                    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;

}
```