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 Литягин С.М.

Преподаватель Берленко Т.А.

Санкт-Петербург
2021

Цель работы.

Изучение и использование линейных списков на языке Си.

Задание.

Создайте двунаправленный список музыкальных композиций MusicalComposition и **api (** ***a****pplication* ***p****rogramming* ***i****nterface - 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

Основные теоретические положения.

Функции библиотеки stdio.h:

- printf() – функция вывода на консоль;
- scanf() – функция ввода данных из консоли.

Циклы:

- for(){<переменная>; <условие>; <выражение_1>} – код в теле цикла будет выполняться до тех пор, пока объявленная в цикле переменная будет удовлетворять условию цикла, выражение_1 каким-либо способом меняет значение этой переменной.
- while(){} – каждая итерация проверяет, выполняется ли условие в круглых скобках, если оно верно, то выполняется код в фигурных скобках, а если неверно, то происходит выход из цикла;

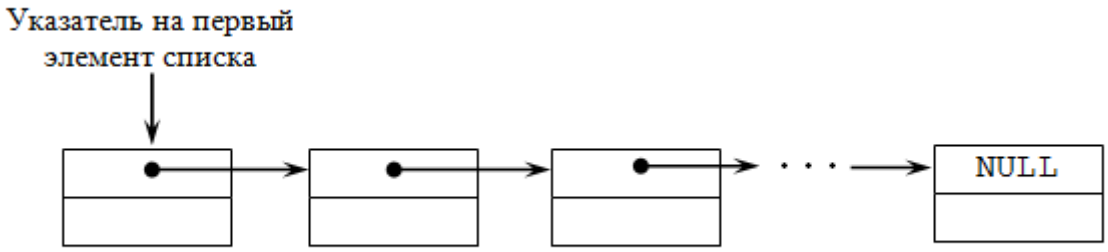
Операторы:

- *if(){} ... else{}* – если выполняется условия, указанное в круглых скобках, то выполняется код в фигурных скобках после *if*, иначе – в фигурных скобках после *else* (*else* не является обязательной частью конструкции)
- *typedef <type> <name>* - *type* - любой тип, *name* - новое имя типа (при этом можно использовать и старое имя)

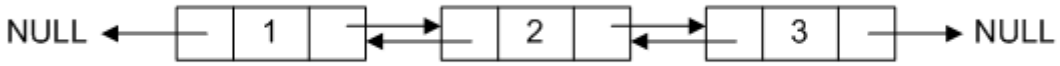
Линейные списки:

Список - некоторый упорядоченный набор элементов любой природы.

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



Двусвязный список - это структура данных, которая состоит из узлов, которые хранят полезные данные, указатели на предыдущий узел и следующий узел.



Выполнение работы.

Задаем структуру *MusicalComposition* (с помощью оператора *typedef* определяем одноименный тип данных) с 5 полями: *char* name* (наименование композиции), *char* author* (автор композиции), *int year* (год создания), *struct MusicalComposition next* (указатель на следующий элемент), *sruct MusicalComposition previous* (указатель на предыдущий элемент).

Функция создания элемента списка *MusicalComposition* createMusicalComposition(char* name, char* author,int year):*

Создается переменная *element* типа данных *MusicalComposition*, с помощью *malloc* динамически выделаем память для элементов заданного типа. Каждое поле структуры заполняет соответствующими значениями *name*, *author*, *year*, полученными в качестве аргументов функции. В поля *next*, *previous* записывается NULL. Возвращается *element*.

Функция для добавления элемента в конец списка *void push(MusicalComposition* head, MusicalComposition* element):*

Создается переменная *end* типа данных *MusicalComposition*, значение которой соответствует указателю на начало списка. С помощью цикла *while* находится последний элемент списка, путем проверки значения поля *next*. Полю *next* последнего элемента присваивается указатель на *element*, переданный в аргументах функции. А полю *previous* данного элемента, присваивается указатель на последний элемент.

Функция создания связанного списка музыкальных композиций *MusicalComposition* createMusicalCompositionList(char** array_names, char** array_authors, int* array_years, int n)*:

Создаются переменные *new_element* и *list* типа *MusicalComposition*. Переменной *list* присваивается указатель на первую музыкальную композицию, созданной с помощью функции *createMusicalComposition*.

Затем, с помощью цикла *for* переменной *new_element* присваивается указатель на n-ую композицию, созданной с помощью функции *createMusicalComposition*, а функция *push* добавляет его в конец списка *list*. Возвращается *list*.

Функция удаления элемента списка (указанного в аргументах функции) *void removeEl (MusicalComposition* head, char* name_for_remove):*

Создается переменная *element* типа *MusicalComposition*, которой присваивается указатель на первый элемент списка *head*. Функция *while* с помощью функции *strcmp()* находит элемент, который нужно удалить. Данный элемент удаляется из списка. Элементу списка, идущему до данного,

передается адрес элемента, следующего за данным. А элементу списка, следующему за ним, передается адрес элемента, идущему до данного.

Функция счета количества элементов *int count(MusicalComposition* head):*

Создается переменная *value* типа *int*, в которой будет храниться количество элеменнов списка. При помощи цикла *while* считается количество элементов. Возвращает *value*.

Функция, выводящая названия композиций, *void print_names(MusicalComposition* head):*

Создается переменная *element* типа *MusicalComposition*, которой присваивается указатель на первый элемент списка *head*. С помощью цикла *while* выводятся поля *name* элементов списка.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

*для удобства при тестировании бралось не 1000 чисел, а 10.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1	2 ss sor 12 tt tor 13 kk kor 14 tt	ss sor 12 2 3 ss kk 2	Программа работает правильно

2	3 ss sor 12 tt tor 13 ff for 14 kk kor 15 kk	ss sor 12 3 4 ss tt ff 3	Программа работает правильно

Выводы.

В ходе работы были изучены основные принципы работы с линейными списками, а также был создан двунаправленный список музыкальных композиций MusicalComposition и **api** (**a**pplication **p**rogramming **i**nterface - в данном случае набор функций) для работы со списком.

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;
    char* author;
    int year;
    struct MusicalComposition* next;
    struct MusicalComposition* previous;
}MusicalComposition;
// Создание структуры MusicalComposition

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

// Функции для работы со списком MusicalComposition
void push(MusicalComposition* head, MusicalComposition* element);

MusicalComposition* createMusicalCompositionList(char**
array_names, char** array_authors, int* array_years, int n){
    MusicalComposition* new_element;
    MusicalComposition* list =
createMusicalComposition(array_names[0], array_authors[0],
array_years[0]);
    for(int i = 1; i < n; i++){
        new_element = createMusicalComposition(array_names[i],
array_authors[i], array_years[i]);
        push(list, new_element);
    }
    return list;
}

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

```
void removeEl(MusicalComposition* head, char* name_for_remove){
    MusicalComposition* element = head;
    while(strcmp(element->name, name_for_remove)){
        if(element->next != NULL){
            element = element->next;
        }
        else{
            break;
        }
    }
    if(element -> previous == NULL){
        element -> next -> previous = NULL;
    }
    else {
        if(element -> next == NULL){
            element -> previous -> next = NULL;
        }
        else{
            element -> previous -> next = element -> next;
            element -> next -> previous = element -> previous;
        }
    }
}

int count(MusicalComposition* head){
    int value = 1;
    MusicalComposition* element = head;
    while(element->next!= NULL){
        value++;
        element = element -> next;
    }
    return value;
}

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