

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №4
по дисциплине «Программирование»
Тема: Структуры данных, линейные списки

Студент гр. 7381

Лукашев Р.С.

Преподаватель

Берленко Т.А.

Санкт-Петербург

2017

Цель работы.

Ознакомиться со структурами данных в языке Си, научиться создавать и управлять структурами.

Создать проект, реализующий структуру данных «линейный двусвязный список».

Основные теоретические положения.

Язык Си поддерживает структуры данных. Объявление структуры:

```
struct <имя структуры>{  
<тип1> поле1;  
<тип2> поле2;  
...  
}
```

Объявление экземпляра структуры данных:

```
struct <структура> <имя>;
```

Поля экземпляра структуры можно проинициализировать как при объявлении (аналогично массиву, по порядку), так и после (<имя экземпляра>.<поле> = <значение>; <имя экземпляра> → <поле> = <значение>;).

У структур не определена операция сравнения. Допустимыми операциями над структурами являются: копирование, присваивание, взятие адреса, доступ к элементам...

При использовании указателя на структуру, для доступа к полям структуры необходимо использовать следующие конструкции:

```
Структура → поле;  
*(структура).поле;
```

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). Для использования NULL необходимо подключить библиотеку `stddef.h`.

С помощью оператора `typedef` можно изменить имя типа. Синтаксис использования оператора:

```
typedef <type> <name>;
```

Для создания линейного списка необходимо создать структуру, содержащую хотя-бы один указатель (на следующий элемент списка) в качестве поля, а затем связать между собой различные экземпляры структуры, используя этот(и) указатель(и).

Ход работы.

Был создан 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_years` (`array_years[0]`). Аналогично для второго, третьего, ... $n-1$ -го элемента массива.

!длина массивов `array_names`, `array_authors`, `array_years` одинаковая и равна n .

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 списка.

Для проекта написан `Makefile`

Выводы.

Были изучены структуры данных в языке Си, основы работы с ними. Был реализован двунаправленный линейный список, и набор функций для работы с ним.

Исходный код проекта.

- Файл «Makefile»:

```
all: main.o
    gcc main.o
main.o: main.c
    gcc -c main.c
```

- Файл «main.c»:

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

```
// Описание структуры MusicalComposition
```

```
typedef struct MusicalComposition{
    char* name;
    char* author;
    int year;
    struct MusicalComposition* prev;
    struct MusicalComposition* next;
} MusicalComposition;
```

```
// Создание структуры MusicalComposition
```

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

```
// Функции для работы со списком MusicalComposition
```

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

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

```

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

int count(MusicalComposition* head);

void print_names(MusicalComposition* head);

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;

if(head) 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;

}

MusicalComposition* createMusicalCompositionList(char** array_names, char**
array_authors, int* array_years, int n){
    if(n==0) return NULL;
    MusicalComposition* List = createMusicalComposition(*array_names,
*array_authors, *array_years);
    MusicalComposition* Prev = List;
    for(int i = 1; i < n; i++){

```

```

        MusicalComposition* Next = createMusicalComposition(*(array_names+i),
*(array_authors+i), *(array_years+i));
        push(Prev, Next);
        Prev = Next;
    }
return List;
}

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

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

int count(MusicalComposition* head){
    int i = 0;

```



```

        while(head){
            i++;
            head = head->next;
        }
        return i;
    }

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

```