МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Программирование»

Тема: Линейные списки

Студент гр. 7381 Минуллин М. А.

Преподаватель Берленко Т. А.

Санкт-Петербург

2017

Цель работы.

Познакомиться со структурой данных: двусвязный список.

Изучить её реализацию, расположение в динамической памяти, применение.

Реализовать основные функции для работы со списком (конструктор, добавление/удаление элемента, и т. д.)

Задание.

Создать двунаправленный список музыкальных композиций `MusicalComposition` и `API` (`Application Programming Interface` – в данном случае набор функций) для работы со списком.

Структура элемента списка:

- `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 символом), автор композиции/музыкальная группа.
- `year` – целое число, год создания.

Функция создания элемента списка (тип элемента `MusicalComposition`):

```
MusicalComposition* createMusicalComposition(char* name, char* author, int year);
```

Функции для работы со списком:

Функция, создающая список музыкальных композиций `MusicalCompositionList` и возвращающая указатель на первый элемент списка.

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

- `n` – длина массивов `array_names`, `array_authors` и `array_years`.
- поле `name` `i`-ого элемента списка соответствует `i`-ому элементу `array_names` (`array_names[i]`)
- поле `author` `i`-ого элемента списка соответствует `i`-ому элементу `array_authors` (`array_authors[i]`)
- поле `year` `i`-ого элемента списка соответствует i-ому элементу `array_years` (`array_years[i]`)

Функция, добавляющая `element` в конец списка `MusicalCompositionList`:

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

Функция, удаляющая `element` списка, у которого значение поля `name` равно значению `name_for_remove`:

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

Функция, возвращающая количество элементов списка:

```
int count(MusicalComposition* head);
```

Функция, выводящая названия композиций:

```
void print_names(MusicalComposition* head
```

Основные теоретические положения.

Линейный однонаправленный список.

Список – некий упорядоченный набор элементов одной природы.

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

Чтобы использовать `NULL`, необходимо подключить заголовочный файл `<stddef.h>`.

Чтобы не писать каждый раз `struct MusicalComposition`, можно воспользоваться оператором `typedef`.

Стандартный синтаксис использования:

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

`type` – любой тип

`name` – новое имя типа (при этом можно использовать и старое имя).

Вывод.

В процессе выполнения лабораторной работы была изучена структура список (одно- и двусвязный). Закреплены знания по работе с указателями. Были реализованы основные функции для работы со списком: выделение памяти для элемента списка, удаление элемента по ключу, вставка элемента в конец списка, подсчёт количества элементов списка, перебор элементов списка. Были освоены константа нулевого указателя и оператор `typedef`.

Исходный код проекта:

Файл "main.c":

```
#include "MusicalComposition.h"

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

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;
}
```

Файл "MusicalComposition.h":

```
#ifndef   MUSICALCOMPOSITION_H___
#define   MUSICALCOMPOSITION_H___

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

MusicalComposition* createMusicalComposition(char*, char*, int);
MusicalComposition* createMusicalCompositionList(char**, char**, int*, int);
void push(MusicalComposition*, MusicalComposition*);
void removeEl(MusicalComposition*, char*);
int count(MusicalComposition*);
void print_names(MusicalComposition*);

#endif
```

Файл "MusicalComposition.c":

```
#include "MusicalComposition.h"

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

MusicalComposition* createMusicalComposition(char* name, char* author, int year) {
    MusicalComposition* mc = (MusicalComposition*)malloc(sizeof(MusicalComposition));
    mc->name = (char*)malloc(81 * sizeof(char));
    strcpy(mc->name, name);
    mc->author = (char*)malloc(81 * sizeof(char));
```

```
    strcpy(mc->author, author);
    mc->year = year;
    mc->next = NULL;
    mc->prev = NULL;
    return mc;
}

MusicalComposition* createMusicalCompositionList(char** array_names, char**
array_authors, int* array_years, int n) {
    MusicalComposition* mc = createMusicalComposition(array_names[0],
array_authors[0], array_years[0]);
    MusicalComposition* iter = mc;
    for (int i = 1; i < n; ++i) {
        iter->next = createMusicalComposition(array_names[i], array_authors[i],
array_years[i]);
        iter->next->prev = iter;
        iter = iter->next;
    }
    return mc;
}

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

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

```
            return;
        }
}

int count(MusicalComposition* head) {  int size
    = 0; MusicalComposition*  iter = head;
    for (iter = head; iter != NULL; iter = iter->next)
        ++size; return size;
}

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

Файл "Makefile":

```
objects = main.o MusicalComposition.o
executable = MusicalCompositions

all: $(objects)
    gcc -o $(executable) $(objects)

main.o: main.c MusicalComposition.h  gcc -c
    main.c

MusicalComposition.o: MusicalComposition.c MusicalComposition.h  gcc -c
    MusicalComposition.c

clean:
    rm $(objects) $(executable)
```