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. 9303 _____ Максимов Е.А.

Преподаватель _____ Чайка К.В.

Санкт-Петербург
2020

Цель работы.

Изучить понятие линейного списка и его применение.

Задание.

Создайте двунаправленный список музыкальных композиций и API (Application Programming Interface – набор функций) для работы со списком:

1) создание музыкальной композиции;

2) создание списка музыкальных композиций;

3) добавление музыкальной композиции в конец списка;

4) удаление музыкальной композиции по названию;

5) подсчёт количества композиций в списке.

Основные теоретические положения.

Список - некоторый упорядоченный набор элементов.

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).

Линейный двунаправленный (двусвязный) список – список, каждый элемент которого хранит помимо значения ещё два указателя: на следующий элемент и на предыдущий. В последнем элементе указатель на следующий элемент и в первом элементе указатель на предыдущий элемент равны *NULL*.

Для создания линейных списков в языке программирования С используют структуры. Пример структуры элемента односвязного списка:

```
struct Node{
      int x;
      int y;
      float r;
      struct Node* next;
};
```

В данном случае структура *Node* имеет 3 поля с данными (*x, y, r*) и поле, являющееся указателем на следующую структуру.

Пример инициализации элемента списка:

```
      struct Node* elem = malloc(sizeof(struct Node));
elem->x=1;
      elem->y=2;
      elem->r=2.5;
      elem->next=NULL;
```

Для создания линейного односвязного списка необходимо связать элементы этого списка с помощью указателей. Для этого, необходимо каждому элементу списка сопоставить указатель на следующий за ним элемент:

```
elem1->next = elem2;
```

У последнего элемента списка указатель на следующий равняется *NULL*.

Оператор *typedef* – оператор, с помощью которого можно определить новое имя уже существующему типу данных. Используется для облегчения создания программного кода. Для данной структуры будет выглядеть следующим образом:

```
typedef struct Node{
      int x;
      int y;
      float r;
      struct Node* next;
}Node;
```

Выполнение работы.

В программе использовались следующие переменные:

1. Структура *MusicalComposition*, состоящая из следующих полей:
    a. *char* name* – строка, содержащая название музыкальной композиции;
    b. *char* author* – строка, содержащая имя автора музыкальной композиции;
    c. *int year* – год написания музыкальной композиции;
    d. *struct MusicalComposition* p_prev* – указатель на предыдущий элемент двусвязного списка;
    e. *struct MusicalComposition* p_next* – указатель на следующий элемент двусвязного списка;

В программе реализованы следующие функции:

1. Функция *MusicalComposition* createMusicalComposition* принимает на вход 2 указателя на строки (название песни и имя автора) и год выпуска песни *char* name, char* author, int year* соответственно. Функция возвращает объект типа *MusicalComposition** – указатель на музыкальную композицию, элемент двусвязного списка.
2. Функция *MusicalComposition* createMusicalCompositionList* принимает на вход 3 указателя на массивы, содержащие названия песен, имена авторов, даты создания *char** array_names, char** array_authors, int* array_years* соответственно, и переменную *int n* – количество элементов в каждом массиве. Функция формирует двусвязный список из элементов данных массивов. Функция возвращает указатель на первый элемент двусвязного списка.
3. Функция *void push* принимает на вход первый элемент двусвязного списка *MusicalComposition* head* и новый элемент *MusicalComposition* element*, который добавляется в конец двусвязного списка. Функция не имеет возвращаемого значения.
4. Функция *void removeEl* принимает на вход первый элемент двусвязного списка *MusicalComposition* head* и имя элемента *char* name_for_remove*, который необходимо удалить из списка. Функция освобождает память, выделенную под объект, который она удаляет. Функция не имеет возвращаемого значения.
5. Функция *int count* принимает на вход первый элемент двусвязного списка *MusicalComposition* head*. Функция возвращает число – количество композиций в списке.

Разработанный программный код см. в приложении А.

Результаты тестирования.

№ п/п	Входные данные	Выходные данные	Комментарии
1.	7	Fields of Gold Sting	Тест пройден

	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	1993 7 8 Fields of Gold In the Army Now Mixed Emotions Billie Jean Seek and Destroy Wicked Game Sonne 7	

Выводы.

В ходе лабораторной работы было изучено понятия линейного списка и его реализация в языке программирования С.

Была разработана программа, которая считывает данные о музыкальных композициях с потока ввода и создаёт двусвязный список музыкальных композиций на основе структур языка программирования С.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

typedef struct MusicalComposition{
    char* name;
    char* author;
    int year;
    struct MusicalComposition* p_prev;
    struct MusicalComposition* p_next;
} MusicalComposition;

MusicalComposition* createMusicalComposition(char* name, char* autor,int
year);
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;

    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* createMusicalComposition(char* name, char* author,
int year){
    MusicalComposition* Composition = malloc(sizeof(MusicalComposition));
    Composition->name = name;
    Composition->author = author;
    Composition->year = year;
    Composition->p_prev = NULL;
    Composition->p_next = NULL;
    return Composition;
}

MusicalComposition* createMusicalCompositionList(char** array_names,
char** array_authors, int* array_years, int n){
    MusicalComposition* curr = NULL;
    MusicalComposition* prev = NULL;
    MusicalComposition* head = NULL;
```

```
    for(int i=0; i<n; i++){
        curr = createMusicalComposition(array_names[i], array_authors[i],
array_years[i]);
        if(i==0){
            head = curr;
            prev = curr;
            continue;
        }
        prev->p_next=curr;
        curr->p_prev=prev;
        prev=curr;
    }
    return head;
}

void push(MusicalComposition* head, MusicalComposition* element){
    MusicalComposition* curr = head;
    while(curr->p_next) curr = curr->p_next;
    curr->p_next=element;
    element->p_prev=curr;
    return;
}

void removeEl(MusicalComposition* head, char* name_for_remove){
    MusicalComposition* curr = head;
     while(strcmp(curr->name, name_for_remove)) curr = curr->p_next;
     (curr->p_next)->p_prev = curr->p_prev;
     (curr->p_prev)->p_next = curr->p_next;
     free(curr);
     return;
}

int count(MusicalComposition* head){
     int n=1;
     MusicalComposition* curr = head;
     while(curr->p_next){
          curr=curr->p_next;
          n++;
     }
     return n;
}

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