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 _____ Азаров М.С.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург
2021

Цель работы.

Изучить что такое линейные списки , и как они устроенны , а также освоить работу с ними.

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

В функции main написана некоторая последовательность вызова команд для проверки работы вашего списка.
Функцию main менять не нужно.

Основные теоретические положения.

Линейный однонаправленный список

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).

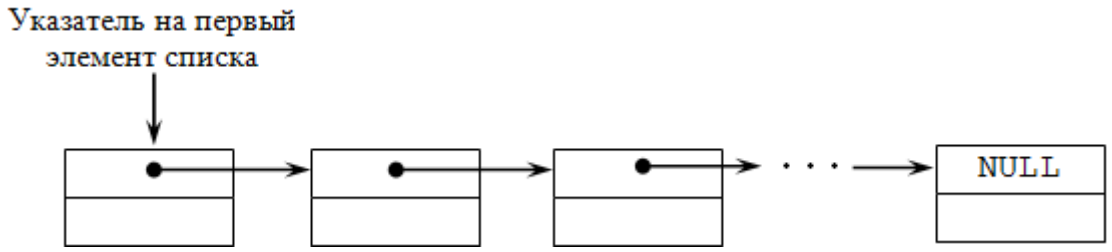

Рисунок 1 - Схематическая визуализация списка

! Чтобы использовать NULL, необходимо подключить #include <stddef.h>

Давайте сделаем из структуры Circle (урок 6.1 шаг 6) элемент списка Node:

```
struct Node{
      int x;
      int y;
      float r;
struct Node* next; // указатель на следующий элемент
};
```

И проинициализируем два элемента списка в функции main():

```
int main(){
   struct Node * p1 = (struct Node*)malloc(sizeof(struct Node));
   struct Node * p2 = (struct Node*)malloc(sizeof(struct Node));
   p1->x = 2; // используем -> поскольку p1 - указатель на
структуру Node
    p1->y = 2;
    p1->r = 2.5;
    p2->x = 5;
    p2->y = 5;
    p2->r = 5.5;
    p1→next = p2;
```

```
    p2->next = NULL;
    free(p1);
    free(p2);
    return 0;
}
```

У нас получился линейный список из двух элементов: p1 и p2.

Итак, мы описали структуру, которая позволяет создавать нам элементы списка.

Сейчас в нашем списке два элемента: p1 и p2. Но что будет, если нам потребуется 10 элементов?

Сейчас, используя код первого урока, мы вынуждены каждый раз создавать указатель на структуру и потом хранить этот указатель (например, аналогично 1му шагу, мы бы создали указатели p3, p4, ... p10).

Для удобства назовем первый элемент списка head:

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

Второй элемент:

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

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

```
head->next = tmp;
```

И создаем остальные элементы:

```
for(i = 1; i < 10; i++) {
      tmp->next = (Node*)malloc( sizeof(Node)); // 1
      tmp->next->next = NULL;   // 2
      tmp = tmp->next; // 3
}
```

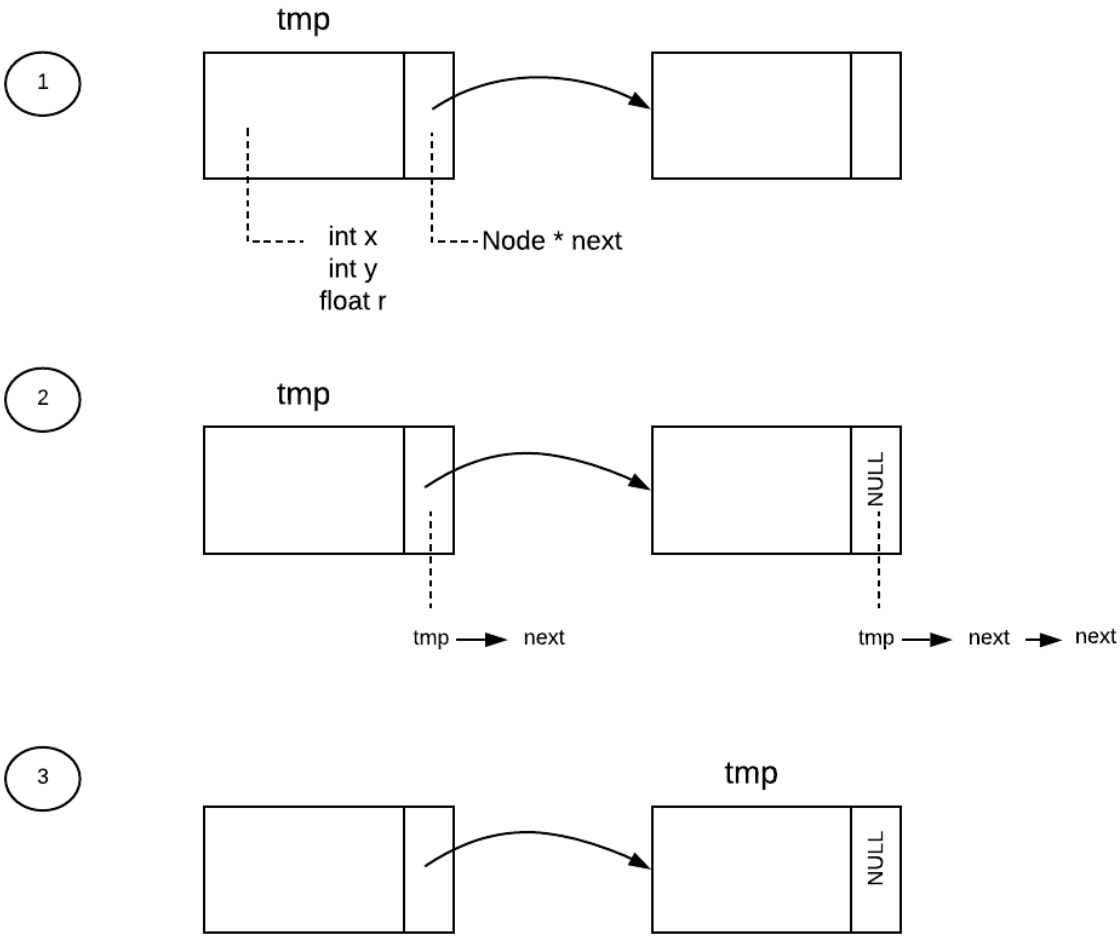

Рисунок 2 - Создание нового элемента списка

В конце у нас будет

- два указателя на элементы списка: head, который указывает на первый элемент и tmp, который указывает на последний
- список, который состоит из 10 элементов.

Выполнение работы.

Структуры:

Создана была *структура* ***MusicalComposition*** *,* которая содержит в себе *char** ***name*** - название композиции, *char** ***author*** - автор композиции/ музыкальная группа, *int* ***year*** - целое число, год создания.

Функция ***createMusicalComposition()*** **:**

Описание:

Функция для создания элемента списка (тип элемента MusicalComposition)

Переменные :

- Параметры функции:
  - *char** ***name -*** значение , которое нужно передать в поле ***name*** создаваемого элемента .
  - *char** ***autor -*** значение , которое нужно передать в поле ***autor*** создаваемого элемента .
  - *int* ***year -*** значение , которое нужно передать в поле ***year*** создаваемого элемента .
- Локальные переменные:
  - *MusicalComposition** ***tmp*** **—** указатель на созданный элемент .

Ход работы :

- Создаем указатель ***tmp*** и выделяем для него необходимое количество динамической памяти .
- Заполняем поля структуры ***tmp*** переданными в функцию значениями ***name , author , year.***
- Возвращаем указатель ***tmp***

Функция ***createMusicalCompositionList()****:*

Описание :

Создает список музыкальных композиций MusicalCompositionList.

Переменные:

- Параметры функции:
  - *int* ***n*** - длина массивов ***array_names, array_authors, array_years***
  - *char*** ***array_names*** **—** массив названий композицый
  - *char*** ***array_authors*** **—** массив имен авторов
  - *int** ***array_years*** **—** массив лет созданий каждой композиции
- Локальные переменные
  - *MusicalComposition** ***head*** **—** указатель на список (на его голову)
  - MusicalComposition* **tmp —** временный указатель

Ход работы :

- Проверка на передачу не пустых массивов .
- Создание первого элемента списка
- Создание остальных элементов с помощью цикла
- Возврат указателя на голову списка.

Функция ***push():***

Описание :

Добавляет ***element*** в конец списка ***musical_composition_list.***

Переменные:

- Параметры функции:
  - *MusicalComposition** ***head*** **—** указатель на список к которому добавляется элемент .

- *MusicalComposition** ***element*** — элемент который добавляется в список ***head.***

Ход работы :

- Проверка не пустой ли список
- Переход к последнему элементу списка
- Присоединение нового элемента в конец списка .

Функция ***removeEl():***

Описание :

Удаляет элемент ***element*** списка, у которого значение ***name*** равно значению ***name_for_remove***

Переменные:

- Параметры функции:
  - *MusicalComposition** ***head*** — Указатель на список
  - *char** ***name_for_remove*** — если значение ***name*** какого-то элемента списка равно этой переменной , то этот элемент нужно удалить из списка.

Ход работы :

- Проверка не пустой ли список .
- Переход к следующему элементу пока не найдется требуемый элемент или не закончится список .
- Выбрасывание искомого элемента из списка (если он нашелся)
- Очистка памяти выброшенного элемента .

Функция ***count():***

Описание :

Возвращает количество элементов списка

Переменные:

- Параметры функции:
  - *MusicalComposition** ***head*** **—** Указатель на список
- Локальные переменные
  - *int* ***count*** **—** счетчик элементов списка .

Ход работы :

- Переходим к следующему элементу и прибавляем +1 к ***count***
- Возвращаем значение ***count.***

Функция ***print_names():***

Описание :

Выводит названия композиций списка .

Переменные:

- Параметры функции:
  - *MusicalComposition** ***head*** **—** Указатель на список

Ход работы :

- Печатаем ***name*** текущего элемента списка и переходим к следующему .

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	2 Wicked Game Chris Isaak 1989 Points of Authority Linkin Park 2000 Sonne Rammstein 2001 Points of Authority	Wicked Game Chris Isaak 1989 2 3 Wicked Game Sonne 2	Программа работает правильно
2.	2 Wicked Game Chris Isaak 1989 Points of Authority Linkin Park 2000 Sonne Rammstein 2001 Hello world!	Wicked Game Chris Isaak 1989 2 3 Wicked Game Points of Authority Sonne 3	Программа работает правильно

Выводы.

Была изучена такая структура данных как список , и освоена работ с ним .

Разработана программа, выполняющая поставленную задачу, а именно хранение данных в списке и взаимодействие с ним через api . Для решения этой задачи были использованы полученные знания о том как устроен список и как с ним взаимодействовать .

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb_2.c

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

// Описание структуры MusicalComposition
struct MusicalComposition{
    char* name;
    char* author;
    int year;
    struct MusicalComposition* next;
    struct MusicalComposition* prev;
};

typedef struct MusicalComposition MusicalComposition;

// Создание структуры MusicalComposition

MusicalComposition* createMusicalComposition(char* name, char*
autor,int year){
    MusicalComposition* tmp =
(MusicalComposition*)malloc(sizeof(MusicalComposition));

    tmp->name = name;
    tmp->author = autor;
    tmp->year = year;

    return tmp;
}

// Функции для работы со списком MusicalComposition
```

```
MusicalComposition* createMusicalCompositionList(char** array_names,
char** array_authors, int* array_years, int n){
    if (n==0){return NULL;}

    MusicalComposition* head =
(MusicalComposition*)malloc(sizeof(MusicalComposition));
    MusicalComposition* tmp = head;

    head->name = array_names[0];
    head->author = array_authors[0];
    head->year = array_years[0];
    head->prev = NULL;

    for (int i = 1; i < n; i++){
        tmp->next =
(MusicalComposition*)malloc(sizeof(MusicalComposition));
        tmp->next->prev = tmp; //указ. на пред. идущий
        tmp = tmp->next;//переход
        tmp->name = array_names[i];
        tmp->author = array_authors[i];
        tmp->year = array_years[i];
    }
    tmp->next = NULL;
    return head;
}

void push(MusicalComposition* head, MusicalComposition* element){
    if (head == NULL){
        return;
    }

    while (head->next){
        head = head->next;
    }

    head->next = element;
```

```
    element->prev = head;
    element->next = NULL;
}

void removeEl(MusicalComposition* head, char* name_for_remove){
    if (head == NULL){
        return;
    }

    while (strcmp (head->name,name_for_remove)){
        head = head->next;
        if (head == NULL) {
            return;
        }
    }

    head->prev->next = head->next;
    head->next->prev = head->prev;

    free(head);
}

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

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

void print_names(MusicalComposition* 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);

    }
    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;

}
```