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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Программирование»

Тема: Структуры данных, линейные списки

Студентка гр. 7304 _____ Нгуен Т.Т. Зуен

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2017

Цель работы
Научиться работать со структурами данных, создавать линейные списки.

Основные теоретические положения.

Простейшие действия со структурой

Дана структура, описывающая некоторый прямоугольник со сторонами **a** и **b** следующего вида:

```
struct Rectangle{
    int a ;
    int b;
};
```

Указатель на структуру

Объявлен указатель на этой структуре

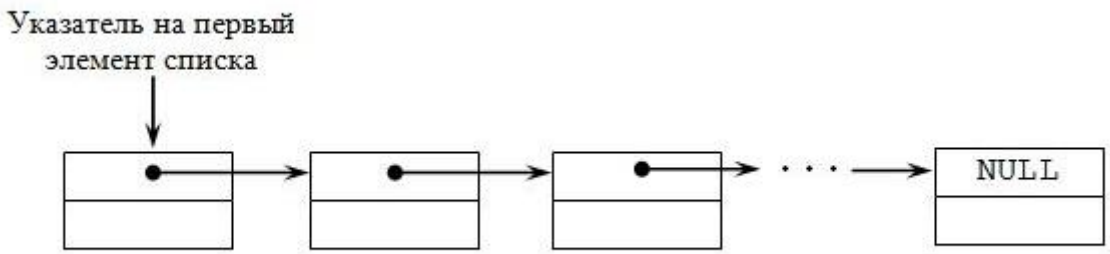
```
Struct Rectangle* p_retangle;
```

Для доступа к **a** нужно использовать: *p_rectangle->a* или *(*p_rectangle).a*

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).



Чтобы не писать каждый раз "struct Node", воспользуемся оператором typedef.

Стандартный синтаксис использования:

typedef <type> <name>;
где: type - любой тип
name - новое имя типа (при этом можно использовать и старое имя)

Итак, начальная запись вида

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

struct Node* p1 = (struct Node*)malloc(sizeof(struct Node));
```

с помощью оператора typedef приобретает следующий вид:

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

Node* p1 = (Node*)malloc(sizeof(Node));
```

Первый элемент списка head:

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

Второй элемент:

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

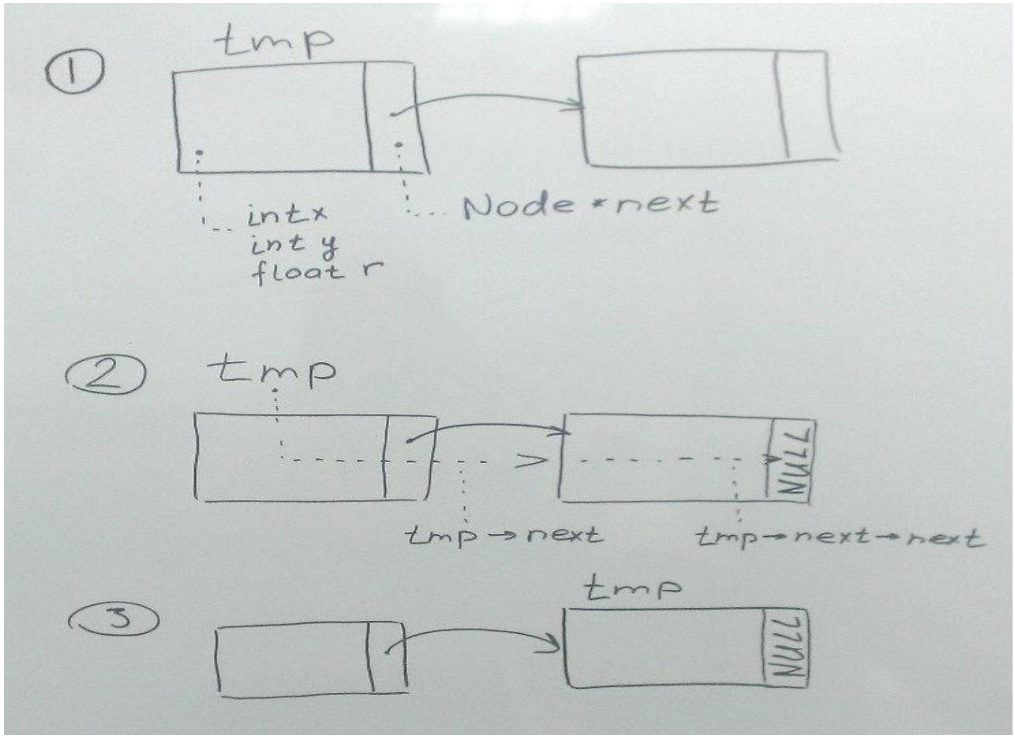
поле next первого элемента head должно ссылаться на второй элемент tmp:

```
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
}
```



Экспериментальные результаты.

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

Ход работы

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

// Описание структуры ====================================

struct MusicalCompisition
{
	char*name;
	char*author;
	int year;
	struct MusicalCompisition*next;//con tro lien ket giua cac node
	struct MusicalCompisition*last;
};
typedef struct MusicalCompisition MusicalCompisition;

// Создание структуры ====================================

MusicalCompisition* createMusicalComposition(char* name, char* author, int year)
{

MusicalCompisition*element=(MusicalCompisition*)malloc(sizeof(MusicalCompisition));
	(element->name) =
(MusicalCompisition*)malloc(strlen(name)*sizeof(MusicalCompisition));
	(element->author) =
(MusicalCompisition*)malloc(strlen(author)*sizeof(MusicalCompisition));

	strcpy(element->name,name);
	strcpy(element->author,author);
```

```
	element->year=year;
	element->next=NULL;
	element->last=NULL;
	return element;
}

void push(MusicalCompisition* head,  MusicalCompisition* element);

// Функции для работы со списком ==============================

MusicalCompisition* createMusicalCompositionList(char** array_names, char**
array_authors, int* array_years, int n)
{
	if(n==0)
		return NULL;
	int i=0;
	MusicalCompisition*head=createMusicalComposition(array_names[0],array_autho
rs[0],array_years[0]);
	for(i=1;i<n;i++)
	{
		MusicalCompisition*element =
createMusicalComposition(array_names[i],array_authors[i],array_years[i]);
		push(head,element);
	}
	return head;
}

// Добавляет element  в конец списка ==============================

void push(MusicalCompisition* head,  MusicalCompisition* element)
{
	MusicalCompisition*p_element=head; //p_element-указатель нужен добавлять
	{
		while(p_element->next!=NULL)
		{
			p_element=p_element->next;
		}
		p_element->next=element;
		element->last=p_element;
	}
}

// Удаляет элемент element списка, у которого значение name равно значению
name_for_remove==============================
void removeEl(MusicalCompisition* head, char* name_for_remove)
{
	MusicalCompisition*element=head;
	MusicalCompisition*remove; //указатель нужен удалить
```

```
	while(element!=NULL)
	{
		if((strcmp(element->name,name_for_remove))==0
		{
				remove=element;
				if(element->next!=NULL)
				{
					element->next->last=remove->last;
				}
				if(element->last!=NULL)
				{
					element->last->next=remove->next;
				}
				free(remove);
				return;
		}
		element=element->next;
	}
}

//Возвращает количество элементов списка==========================
int count(MusicalCompisition* head)
{
	int k=0;
	while(head!=NULL)
	{
		k+=1;
		head=head->next;
	}
	return k;
}

//Выводит названия композиций=====================================
void print_names(MusicalCompisition* head)
{
	MusicalCompisition*element=head;
	while(element!=NULL)
		{
			printf("%s\n",element->name);
			element=element->next;
		}
}


int main()
{
   int length;
   scanf("%d", &length);
```

```
    while(getchar()!='\n');
    char** names = (char**)malloc(sizeof(char*)*length);
    char** authors = (char**)malloc(sizeof(char*)*length);
    int* years = (int*)malloc(sizeof(int)*length);

   int i;
    for (i=0;i<length;i++)
    {
        char name[80];
        char author[80];

        fgets(name, 80, stdin);
        fgets(author, 80, stdin);
        fscanf(stdin, "%d", &years[i]);
                while(getchar()!='\n');

        (*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);

    }
    MusicalCompisition* 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;

    MusicalCompisition* 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 (i=0;i<length;i++){
        free(names[i]);
        free(authors[i]);
    }
    free(names);
    free(authors);
    free(years);

    return 0;
}
```