

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

Адамов Я.В.

Преподаватель

Берленко Т.А.

Санкт-Петербург

2017

Лабораторная работа №3

Использование указателей

Цель работы: познакомиться со структурой данных – двусвязными списками, и научиться с ними работать, а именно: создавать списки, добавлять и удалять элементы списка, получать информацию об элементах списка и их количество; узнать, как располагается список в динамической памяти.

Задание: Создать 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]).
- void push(MusicalComposition* head, MusicalComposition* element);

функция добавляет element в конец списка

musical_composition_list

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

функция удаляет элемент element списка, у которого значение name равно значению name_for_remove

- int count(MusicalComposition* head);

функция возвращает количество элементов списка

- void print_names(MusicalComposition* head);

функция выводит названия композиций

Основные теоретические положения:

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).

В линейном двунаправленном списке каждый элемент хранит также указатель на предыдущий элемент.

Заголовочные файлы стандартной библиотеки языка C, необходимые для выполнения данной лабораторной работы: `stdio.h`, `stdlib.h`, `string.h`, `stddef.h`.

- Библиотека `stdio.h` содержит прототипы функций стандартного ввода и вывода.
- Библиотека `stdlib.h` содержит прототипы функций для динамического выделения памяти:

1) **`void * malloc(size_t sizemem);`**

Описание: функция `malloc` выделяет блок памяти, размером `sizemem` байт, и возвращает указатель на начало блока. Содержание выделенного блока памяти не инициализируется, оно остается с неопределенными значениями.

Параметры:

- `sizemem` - размер выделяемого блока памяти в байтах.

Возвращаемое значение - указатель на выделенный блок памяти. Тип данных на который ссылается указатель всегда `void*`, поэтому это тип данных может быть приведен к желаемому типу данных. Если функции не удалось выделить требуемый блок памяти, возвращается нулевой указатель.

4) **`void free(void * ptrmem);`**

Описание: функция `free` освобождает место в памяти. Блок памяти, ранее выделенный с помощью вызова `malloc`, `calloc` или `realloc` освобождается. То есть освобожденная память может дальше использоваться программами.

Параметры:

- `ptrmem` -указатель на блок памяти, ранее выделенный функциями `malloc`, `calloc` или `realloc`, которую необходимо высвободить. Если в качестве аргумента передается нулевой указатель, никаких действий не происходит.

- Библиотека `string.h` содержит прототип следующие функции:

1) **`size_t strlen(const char * string);`**

Описание: длина Си-строки определяется по достижению нулевого символа — нуль терминатор. Функция `strlen` видит начало Си-строки и начинает сначала считать количество символов (байтов, отводимых под каждый символ), этот процесс выполняется до тех пор, пока не будет достигнут завершающий нулевой символ.

2) **`char * strcpy(char * destptr, const char * srcptr);`**

Описание: функция копирует Си-строку `srcptr`, включая завершающий нулевой символ в строку назначения, на которую ссылается указатель `destptr`.

Параметры:

- `destptr` - указатель на строку назначения, куда будет скопирована строка-источник.
- `srcptr` - указатель на копируемую строку.

Возвращаемое значение - указатель на строку назначения.

3) **`int strcmp(const char * string1, const char * string2);`**

Описание: функция сравнивает символы двух строк, `string1` и `string2`.

Начиная с первых символов функция `strcmp` сравнивает поочередно каждую пару символов, и продолжается это до тех пор, пока не будут найдены различные символы или не будет достигнут конец строки.

Параметры:

- `string1` - первая сравниваемая Си-строка.

- `string2` - вторая сравниваемая Си-строка.

Возвращаемое значение - функция возвращает несколько значений, которые указывают на отношение строк:

Нулевое значение говорит о том, что обе строки равны.

Значение больше нуля указывает на то, что строка `string1` больше строки `string2`, значение меньше нуля свидетельствует об обратном.

4) **`const char * strstr(const char * string1, const char * string2);`**

Описание: функция ищет первое вхождение подстроки `string2` в строке `string1`. Возвращает указатель на первое вхождение строки `string2` в строку `string1`, или пустой указатель, если строка `string2` не является частью строки `string1`. В данном поиске нуль-символ не учитывается.

Параметры:

- `string1` - строка, в которой выполняется поиск.
- `string2` - подстрока для поиска в строке `string1`.

Возвращаемое значение - указатель на первое вхождение в `string1` любой последовательности символов, указанных в `string2`. Нулевой указатель, если последовательность символов строки `string2` не входит в `string1`.

- Библиотека `ctype.h` содержит прототипы следующих функций:

1) **`int isdigit(int character);`**

Описание: функция `isdigit` проверяет аргумент, передаваемый через параметр `character`, является ли он десятичной цифрой.

Параметры:

- `character` - символ для проверки, передается в функцию как значение типа `int`, или EOF.

Возвращаемое значение - значение, отличное от нуля (т.е. истинно), если аргумент функции — это десятичная цифра . Ноль (т.е. ложь), в противном случае.

2) `int isspace(int character);`

Описание: функция `isspace` проверяет параметр `character`, является ли он символом пробела (пробела, табуляции, переноса строки).

Параметры:

- `character` - символ для проверки, передаётся в функцию как значение типа `int`, или `EOF`.

Возвращаемое значение - значение, отличное от нуля (т.е. истинно), если аргумент функции — это символ пробела. Ноль (т.е. ложь), в противном случае.

- Библиотеку `stdlib.h` необходимо подключить, чтобы использовать `NULL`.

Вывод: в ходе работы были изучены структуры данных: односвязные и двусвязные списки; а также освоена работа с ними: выделение динамической памяти под список, его создание и редактирование. Изучены новый оператор `typedef` и константа нулевого указателя. Также закреплены навыки работы с указателями, строками и динамической памятью.

Исходный код программы:

`menu.c:`

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

```
// Описание структуры MusicalComposition
```

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

```
// Создание структуры MusicalComposition
```

```
MusicalComposition* createMusicalComposition(char* name, char* author, int year){  
    MusicalComposition* new_composition = (MusicalComposition*)malloc(sizeof(MusicalComposition));  
    new_composition->name=(char*)malloc(81*sizeof(char));  
    new_composition->author=(char*)malloc(81*sizeof(char));  
    strcpy(new_composition->name, name);  
    strcpy(new_composition->author, author);  
    new_composition->year=year;  
    new_composition->next = NULL;  
    new_composition->prev = NULL;  
    return new_composition;  
}
```

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

```
//
```

```
// Создание списка музыкальных композиций
```

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

```
// Добавление новой композиции в конец списка
```

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

```
// Удаление элемента, у которого name равно name_for_remove
```

```
void removeEl(MusicalComposition* head, char* name_for_remove){  
    for (MusicalComposition* comp = head; comp!=NULL; comp=comp->next)
```



```

        if (strcmp(comp->name,name_for_remove)==0){
            comp->next->prev = comp->prev;
            comp->prev->next = comp->next;
            break;
        }
    }

// Возвращает количество композиций
int count(MusicalComposition* head){
    int count=0;
    for (MusicalComposition* comp = head; comp!=NULL; comp=comp->next)
        count++;
    return count;
}

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

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;

```

```

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);

char name_for_remove[80];
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;

}

```

Makefile

```

all: main.o
    gcc main.o
main.o: main.c
    gcc -c main.c
clean:
    rm main.o

```