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 _____ Машина Ю.Д.

Преподаватель _____ Берленко Т. А.

Санкт-Петербург

2017

Цель работы

Познакомиться со списками и структурами на языке СИ, научиться их создавать и производить операции над ними.

Задание

Создать 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

Заголовочные файлы, необходимые для создания проекта:

1.**<stdio.h>** - содержит прототип функции «int printf(const char* format [, argument]...);», которая используется для вывода в поток вывода.

Синтаксис: #include <
stdio.h >
int printf (const char *format, ...);

Аргументы:
format – указатель на строку с описанием формата.

Возвращаемое значение:
При успешном завершении вывода возвращается количество выведенных символов.
При ошибке возвращается отрицательное число.

2. **<string.h>** - содержит прототип функции «int strcmp(const char * string1, const char * string2);», которая сравнивает символы двух строк.

Синтаксис: #include <
string.h >
int strcmp(const char * string1, const char * string2);

Аргументы:
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.

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.

3. **<stdlib.h>** - содержит прототипы функций «void * malloc(size_t sizemem);» и «void free (void* ptr);», которые динамически выделяют память под массив данных и высвобождают динамически выделенную ранее память.

Синтаксис: #include <
stdlib.h >
void * malloc(size_t sizemem);

Аргументы:
sizemem - азмер 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 типом данных. Если функции не удалось выделить требуемый блок памяти, возвращается нулевой указатель.

Описание:
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.

Синтаксис: #include <
stdlib.h >
void free(void * ptrmem);

Аргументы:
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.

Возвращаемое значение:
Нет.

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 или ОС.

Вывод

В ходе работы были освоены структуры данных, линейные списки, двунаправленные списки, были освоены функции для работы с ними.

Исходный код проекта

- Файл main.c

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

typedef struct MusicalComposition
{
    char name[80];
    char author[80];
    int year;
    struct MusicalComposition* next;
    struct MusicalComposition* prev;
}MusicalComposition;
MusicalComposition* createMusicalComposition(char* name, char* author, int year)
{
   MusicalComposition* composition = (MusicalComposition*)
malloc(sizeof(MusicalComposition));
   strcpy(composition->name, name);
   strcpy(composition->author, author);
   composition->year = year;
   composition->next = NULL;
   composition->prev = NULL;
```

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

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

```
    for(;strcmp(head->name,name_for_remove);)
    {
        head = head->next;
    }
    head->prev->next = head->next;
    head->next->prev = head->prev;
}
int count(MusicalComposition* head)
{
    int amm = 0;
    for (MusicalComposition* list=head; list!=NULL; list=list ->next)
        amm++;
    return amm;
}
void print_names(MusicalComposition* head)
{
    for (MusicalComposition* list = head; list != NULL; list=list->next)
        printf("%s\n", list->name);
}
```

- Файл Makefile

```
all: main.o
  gcc main.o

main.o: main.c
  gcc -c main.c

clean:
  rm main.o
```