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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Структуры данных, линейные списки

Студент гр. 7381 _____ Павлов А.П.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург
2017

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Павлов Александр Павлович

Группа 7381

Тема работы: Структуры данных, линейные списки

Содержание пояснительной записки:

- Содержание
- Введение
- Описание структуры данных
- Описание функций
- Заключение
- Список использованных источников
- Приложение: Исходный код проекта

Предполагаемый объем пояснительной записки:

Не менее 11 страниц.

Дата выдачи задания: 23.11.2017

Дата сдачи реферата:

Дата защиты реферата:

Студент _____ Павлов А.П.

Преподаватель _____ Берленко Т. А.

АННОТАЦИЯ

В данной курсовой работе был создан проект на языке программирования C, который позволяет работать с набором функций, отвечающих за список музыкальных композиций. Для функционирования списка была описана структура элемента списка, созданы и описаны функции, позволяющие добавлять, удалять и выводить имена элементов списка и их количество на консоль. Кроме того была реализована функция, удаляющая элементы, длина поля которых меньше n. Был создан удобный интерфейс для работы со списком.

SUMMARY

In this course work was created a project in the programming language C, which allows you to work with a set of functions responsible for the list of musical compositions. To describe the list of described functions, functions and functions that allow you to add, delete and display list items and their number to the console. In addition, a function was implemented that removes elements whose field lengths are less than n. A convenient interface for working with the list was created.

Содержание

ВВЕДЕНИЕ

Цель работы

Практика применения сложных типов (struct) в языке C. Использование их для реализации сложных структур данных. В частности, двунаправленных линейных списков. Создание API для работы со списком.

Формулировка задачи

Создайте 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:

- 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

Индивидуальное задание.

Удалить все элементы списка, длина поля name которых меньше n(n подается с клавиатуры).

ОПИСАНИЕ РАБОТЫ

ОПИСАНИЕ СТРУКТУРЫ ДАННЫХ

С помощью типа struct объявляется новый сложный тип struct MusicalComposition, экземплярами которого будет заполняться список. Задан синоним MusicalComposition данному типу с помощью оператора typedef.

```
typedef struct
MusicalComposition{ char*
name; // Имя композиции char*
author; // Автор композиции int
year; // Год выпуска
композиции
struct MusicalComposition* prev; //Предыдущий
элемент struct MusicalComposition*
next;//Следующий элемент
} MusicalComposition;
```

ОПИСАНИЕ ФУНКЦИЙ

1. Функция createMusicalComposition

Назначение: Создание элемента списка**.**

Аргументы:
char* name — строка, имя музыкальной композиции.
char* author — строка, автор музыкальной
композиции. Int year — целое число, год
музыкальной композиции. Возвращаемое значение:
MusicalComposition* — указатель на экземпляр композиции.

1.2 Функция push
Назначение: вставка элемента в конец
списка. Аргументы:
MusicalComposition** list — указатель на указатель на первый элемент списка.
MusicalComposition* element — указатель на вставляемый

элемент. Возвращаемое значение:
void — функция ничего не возвращает.

1.3 Функция removeEl

Назначение: удаление элемента списка по заданному имени композиции. Аргументы:
MusicalComposition** list — указатель на указатель на первый элемент списка.
char* name_for_remove — строка, имя удаляемой композиции. Возвращаемое значение:
void — функция ничего не возвращает.

1.4 Функция count.

Назначение: подсчет количества элементов в списке. Аргументы:
MusicalComposition* list — указатель на первый элемент списка. Возвращаемое значение:
int — целое число, количество композиций в списке.

1.5 Функция print_names.

Назначение: выводит список имен музыкальных композиций в списке.
Аргументы:
MusicalComposition* list — указатель на первый элемент списка. Возвращаемое значение:
void — функция ничего не возвращает.

1.6 Функция freeList.

Назначение: очистка динамически выделенной памяти для списка, удаление списка.
Аргументы:
MusicalComposition** list — указатель на указатель на первый элемент удаляемого списка.
Возвращаемое значение:
void — функция ничего не возвращает.

1.7 Функция deleteEl

Индивидуальное задание
Аргументы:
MusicalComposition** list — указатель на указатель на первый элемент удаляемого списка.
Int n – длина поля name(n вводится с клавиатуры)
Возвращаемое значение:
void — функция ничего не возвращает.

ДЕМОНСТРАЦИЯ РАБОТЫ

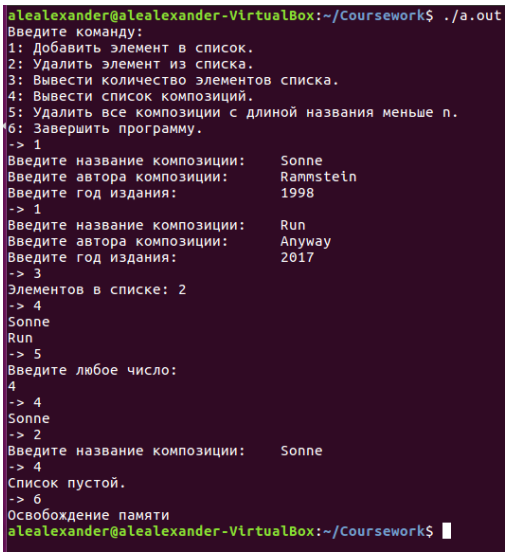

Рисунок 1 – выполнение всех функций

ЗАКЛЮЧЕНИЕ

Было освоено и закреплено на практике написание программ с применением структур в качестве элементов сложных типов представления данных, таких, как двунаправленные линейные списки, а также написание API для работы с ними, в частности, функции вставки, удаления, подсчета и вывода полей элементов списка. Усовершенствованы навыки работы с указателями, динамической памятью и функциями стандартных библиотек.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Язык программирования СИ / Керниган Б., Ритчи Д. СПб: Издательство

   «Невский Диалект», 2001. 352 л.

2. UNIX. Программное окружение / Керниган Б., Пайк Р. СПб: Символ

   Плюс, 2003. 416с.

Приложение А: Исходный код проекта

Файл main.c

```
#include "music.h"
#include "course.h"

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

int main() {
    char opt;

    MusicalComposition* list = NULL;

    char* name = (char*)malloc(81 * sizeof(char));
    char* author = (char*)malloc(81 * sizeof(char));
    int year;

    int n;

    printf("Введите команду:\n1: Добавить элемент в список.\n2: Удалить элемент из списка.\n3: Вывести количество элементов списка.\n4: Вывести список композиций.\n5: Удалить все композиции с длиной названия меньше n.\n6: Завершить программу.\n");
    do {
        printf("-> ");
        scanf(" %c", &opt);
        switch (opt) {
        case '1':
            printf("Введите название композиции:\t");
            scanf("%s", name);
            printf("Введите автора композиции:\t");
            scanf("%s", author);
            printf("Введите год издания:\t\t");
            scanf("%d", &year);
            push(&list, createMusicalComposition(name, author, year));
            break;
        case '2':
            printf("Введите название композиции:\t");
            scanf("%s", name);
            removeEl(&list, name);
            break;
        case '3':
            printf("Элементов в списке: %d\n", count(list));
```

```
            break;
        case '4':
            print_names(list);
            break;
        case '5':
            printf("Введите любое число:\n");
            scanf("%d", &n);
            deleteEl(&list, n);
            break;
        case '6':
            printf("Освобождение памяти\n");
            break;
        default:
            printf("Неизвестная команда\n");
            break;
        }
    } while (opt != '6');

    free(name);
    free(author);
    freeList(&list);

    return 0;
}
```

Файл course.c

```
#include "course.h"

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

void deleteEl(MusicalComposition** list, int n) {
    if (*list == NULL)
        return;
    MusicalComposition* iter = *list;
    while (iter) {
        if (strlen(iter->name) < n)
            removeEl(list, iter->name);
        iter = iter->next;
    }
}
```

```
void freeList(MusicalComposition** list){
	MusicalComposition* iter = *list;
	MusicalComposition* temp;
	if(!iter)
		return;
	while(iter){
		temp = iter;
		iter = iter->next;
		free(temp->name);
		free(temp->author);
		free(temp);
	}
}
```

Файл course.h

```
#pragma once

#include "music.h"

void deleteEl(MusicalComposition**, int);

void freeList(MusicalComposition**);
```

Файл music.c

```
#include "music.h"

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

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

```
    newmc->prev = NULL;
    return newmc;
}

void push(MusicalComposition** head, MusicalComposition *element)
{
    if (*head == NULL) {
        *head = element;
        return;
    }
    MusicalComposition* iter;
    for (iter = *head; iter->next; iter = iter->next);
    element->prev = iter;
    iter->next = element;
}

void removeEl(MusicalComposition** list, char* name_for_remove){
         if (*list == NULL)
        return;
     MusicalComposition* iter = *list;
     MusicalComposition* tmp;
     while (iter != NULL) {
        tmp = iter;
        iter = iter->next;
        if (strcmp(tmp->name, name_for_remove) == 0){
                                if (tmp->next == NULL && tmp->prev == NULL) {
                                free(tmp);
                                        *list = NULL;
                                }
                        else if (tmp->next == NULL) {
                        tmp = tmp->prev;
                        free(tmp->next);
                        tmp->next = NULL;
                        }
                        else if (tmp->prev == NULL){
                                *list = tmp->next;
                                tmp = tmp->next;
                        free(tmp->prev);
                                        tmp->prev = NULL;
                        }

                        else {
                                tmp->prev->next = tmp->next;
                                tmp->next->prev = tmp->prev;
                                free(tmp);
```

```
                        }
                        return;
                }
 }
 }
int count(MusicalComposition* list)
{
    MusicalComposition* iter = list;
    int n = 0;
    while (iter)
    {
       n++;
       iter = iter->next;
    }
    return n;
 }

void print_names(MusicalComposition* list) {
   MusicalComposition* iter = list;
        if (!iter)
                printf("Список пустой.\n");
   for(; iter!=NULL ; iter = iter->next)
        printf("%s\n", iter->name);
 }
```

Файл music.h

```
#pragma once

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

MusicalComposition* createMusicalComposition(char*, char*, int);

void push(MusicalComposition**, MusicalComposition*);
void removeEl(MusicalComposition**, char*);

int count(MusicalComposition*);
void print_names(MusicalComposition*);
```

Файл Makefile

```
all: main.o music.o course.o
	gcc main.o music.o course.o

main.o: main.c music.h course.h
	gcc -c main.c

music.o: music.c music.h
	gcc -c music.c

course.o: course.c course.h music.h
	gcc -c course.c

clean:
	rm main.o music.o course.o
```