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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Программирование»
Тема: Линейные списки

Студент гр. 3341 Гребенюк В.А.

Преподаватель Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Освоение работы со связанными списками на примере использующей их программы. Разработка функций для работы со связанными списками: создание элемента/списка, количество элементов, удаление элементов, добавление элемента в конец списка.

Задание

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:

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.

Выполнение работы

Создаётся 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.

next – указатель на следующий элемент (если есть)

Функции для работы со списком:

MusicalComposition createMusicalCompositionList(char** array_names, char** array_authors, int* array_years, int n);* – создает список музыкальных композиций

void push(MusicalComposition head, MusicalComposition* element);* – добавляет element в конец списка musical_composition_list

MusicalComposition 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.

Разработанный программный код см. в приложении А.

Выводы

Работа с линейными списками на основе использующей их программы освоена.

Линейные списки можно использовать как альтернативу если нет других структур с динамическим размером.

В линейном списке имеется преимущество в скорости удаления элементов, но и недостаток в виде невозможности быстро узнать позицию n-ого элемента путём операций с указателями.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

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

// Создание структуры MusicalComposition

MusicalComposition* createMusicalComposition(char* name, char* author,
int year) {
    MusicalComposition* _new = malloc(sizeof(MusicalComposition));
    _new->name = name;
    _new->author = author;
    _new->year = year;
    _new->next = NULL;
    return _new;
}

// Функции для работы со списком MusicalComposition

MusicalComposition*  createMusicalCompositionList(char**  array_names,
char** array_authors, int* array_years, int n) {
    // assuming n always >= 1
    MusicalComposition* head = createMusicalComposition(array_names[0],
array_authors[0], array_years[0]);
    MusicalComposition* cursor = head;

    for (size_t i = 1; i < n; i++) {
        cursor->next     =     createMusicalComposition(array_names[i],
array_authors[i], array_years[i]);
        cursor = cursor->next;
    }
    return head;
};

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

```
MusicalComposition*    removeEl(MusicalComposition*    node,    char*
name_for_remove) {
    if (node == NULL)
        return node;
    MusicalComposition*     next_valid     =     removeEl(node->next,
name_for_remove);

    if (strcmp(name_for_remove, node->name) == 0) {
        return next_valid;
    } else {
        node->next = next_valid;
        return node;
    }
};

int count(MusicalComposition* head) {
    MusicalComposition* cursor = head;
    int _count = 0;
    while (cursor != NULL) {
        _count++;
        cursor = cursor->next;
    }
    return _count;
};

void print_names(MusicalComposition* head) {
    MusicalComposition* cursor = head;
    while (cursor != NULL) {
        puts(cursor->name);
        cursor = cursor->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;
}
```