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

ОТЧЕТ

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

Студент гр. 3342 _____ Пушко К.Д.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Целью работы является освоение работы с линейными списками, а также использование их в языке программирования Си.

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

Структура MusicalComposition

Содержит поля year, name, author, prev и next. Поля prev и next являются указателями на предыдущий и следующие элементы списка соответственно. С использованием ключевого слова typedef создается новый псевдоним типа MusicalComposition.

Функция createMusicalComposition

Создает новую музыкальную композицию с заданным названием, автором и годом выпуска. Функция возвращает указатель на созданную композицию

Функция createMusicalCompositionList

Принимает массивы данных в качестве входных параметров, содержащих элементы, которые нужно разместить в соответствующих полях структуры. Функция createMusicalComposition создает первый элемент в списке – головной элемент, инициализируя его поля с помощью данных из входных массивов. Затем с помощью функции push последующие элементы связываются с предыдущими, образуя связанный список. В конце функция возвращает головной элемент, то есть начальный элемент списка.

Функция push

Принимает два указателя в качестве входных параметров: указатель на первый элемент в связанном списке и указатель на элемент, который требуется добавить в этот список. Затем мы проходим по всем элементам списка, начиная с первого элемента, у которых поле "next" не равно NULL, находим последний из таких элементов и связываем его поле с добавляемым элементом. Поле "prev" добавленного элемента указывает на головной элемент, а в поле "prev" последнего элемента списка записывается указатель на добавленный элемент. На выходе функция возвращает указатель на головной элемент связанного списка.

Функция removeEl

Принимает указатель на начальный элемент в связанном списке

и слово, по которому определяется, какой элемент в списке нужно удалить. С помощью цикла while Проходит по всем элементам списка, пока не встретится

элемент, поле "name" которого совпадает с переданным значением для удаления. После того как элемент найден, указатели переустанавливаются, таким образом, чтобы элемент, предшествующий удаляемому элементу, указывал на следующий элемент после удаляемого, а элемент, следующий за удаляемым элементом, указывал на предыдущий элемент перед удалаемым. Затем происходит освобождение выделенной памяти для удаляемого элемента.

Функция count

Принимает на вход указатель на начальный элемент списка. С помощью цикла while проходим по каждому элементу списка, каждый раз увеличивая счетчик на единицу. Функция возвращает количество элементов списка.

Функция print_names

Принимает на вход указатель на начальный элемент списка. Пробегаясь по всем элементам списка, выводим с новой строки значения поля name.

Разработанный программный код см. в приложении А.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	7 Fields of Gold Sting 1993 In the Army Now Status Quo 1986 Mixed Emotions The Rolling Stones 1989 Billie Jean Michael Jackson 1983 Seek and Destroy Metallica 1982 Wicked Game Chris Isaak 1989 Points of Authority Linkin Park 2000 Sonne	Fields of Gold Sting 1993 7 8 Fields of Gold In the Army Now Mixed Emotions Billie Jean Seek and Destroy Wicked Game Sonne 7

Выводы

Были изучены основы линейных списков и выявлена различия между списками и массивами. Также была рассмотрена реализация основных операций над линейными списками на языке программирования Си. Была написана программа, которая реализует двусвязный линейный список.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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


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


MusicalComposition*   createMusicalComposition(char*   name,   char*
author, int year)
{
    MusicalComposition*    composition    =    (MusicalComposition*)
malloc(sizeof(MusicalComposition));
    if (composition == NULL)
    {
        exit(1);
    }
    composition->name = name;
    composition->author = author;
    composition->year = year;
    composition->next = NULL;
    composition->previous = NULL;
    return composition;
}

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

    head->next=element;
    element->previous=head;
}

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]);
    for (int i = 1; i < n; ++i)
    {
```

```
            push(head,
createMusicalComposition(array_names[i],array_authors[i],array_years[i]));
        }
        return head;
    }

    void removeEl(MusicalComposition* head, char* name_for_remove)
    {
        MusicalComposition* originalHead = head;
        MusicalComposition * prev, * next;
        while (strcmp(head->name,name_for_remove))
        {
            head=head->next;
        }
        if (originalHead == head) {
            head -> next -> previous = NULL;
            originalHead = originalHead -> next;
        }
        else {
            prev = head -> previous;
            next = head -> next;
            prev -> next = next;
            next -> previous = prev;
        }
        free(head);
    }

    int count(MusicalComposition* head)
    {
        int counter = 0;
        while(head->next!=NULL)
        {
            counter++;
            head=head->next;
        }
        return counter+1;
    }

    void print_names(MusicalComposition* head)
    {
        while(head->next!=NULL)
        {
            printf("%s\n",head->name);
            head=head->next;
        }
        printf("%s\n",head->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;

        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;

}
```