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 _____ Лапшов К.Н.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

Цель работы

Цель данной задачи заключается в овладении навыками работы с линейными списками и их применением в языке программирования Си. В процессе выполнения работы предполагается освоение основных операций, таких как добавление, удаление и обработка элементов в связанных списках.

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

Структура данных содержит поля "year", "name", "author", "next" и "prev" где "next" является указателем на следующий элемент списка, а "prev" — указатель на предыдущий. Для удобства создается новый псевдоним типа "MusicalComposition" с использованием ключевого слова "typedef".

Функция createMusicalComposition

Создает и возвращает элемент списка, инициализирует его поля с помощью данных из входных параметров. "next" и "prev" инициализируются NULL указателями. В конце функция возвращает проинициализированный элемент списка.

Функция createMusicalCompositionList

Принимает массивы данных в качестве входных параметров, содержащих элементы, которые нужно разместить в соответствующих полях структуры, и с помощью createMusicalComposition создает список. Возвращает указатель на головной элемент списка.

Функция count

Эта функция принимает указатель на начало списка в качестве входных данных. Она использует цикл while для прохода по каждому элементу списка, увеличивая счетчик на единицу на каждой итерации. По завершении она возвращает количество элементов в списке.

Функция push

Эта функция принимает два указателя в качестве параметров: указатель на первый элемент в связанном списке и указатель на элемент, который предполагается добавить в этот список. Затем она проходит по всем элементам списка, начиная с первого элемента, у которых поле "next" не равно NULL. В процессе поиска находится последний из таких элементов, и его поле "next" связывается с добавляемым элементом. По завершении 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 и функции strcmp, проходит по всем элементам списка, пока не встретится элемент, поле "name" которого совпадает с переданным значением для удаления. После того как элемент найден, создается указатель на предыдущий элемент списка, и уже этому элементу в поле "next" передается указатель элемента, который идет после удаляемого. После этого, память удаляемого элемента очищается

Функция print_names

Принимает на вход указатель на начальный элемент списка. Итерируя все элемент 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 Rammstein 2001 Points of Authority	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 основные концепции линейных списков, а также выделены основные отличия между списками и массивами. Произведена реализация основных операций над линейными списками на языке программирования Си.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

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


void memoryError(){
    printf("Memory allocation error!");
    exit(0);
}

MusicalComposition*   createMusicalComposition(char*   name,   char*
autor,int year){
                                MusicalComposition*       newComp       =
malloc(sizeof(MusicalComposition));
    if(newComp == NULL){
        memoryError();
    }

    newComp->name = name;
    newComp->author = autor;
    newComp->year = year;
    newComp->next = NULL;
    newComp->prev = NULL;

    return newComp;
}


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* tmp = head;

    for (int i = 1; i < n; i++) {
                               MusicalComposition*   new_composition   =
createMusicalComposition(array_names[i],
array_authors[i],array_years[i]);

        tmp->next = (struct MusicalComposition* ) new_composition;
        new_composition->prev = (struct MusicalComposition* ) tmp;
```

```
            tmp = (MusicalComposition *) new_composition;
        }

        return head;
    }

    int count(MusicalComposition* head){
        MusicalComposition* tmp = head;
        int result = 0;

        while(tmp != NULL){
            tmp = (MusicalComposition *) tmp->next;
            result++;
        }

        return result;
    }

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

        tmp->next = (struct MusicalComposition *) element;
        element->prev = (struct MusicalComposition *) tmp;
    }

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


    void print_names(MusicalComposition* head){
        MusicalComposition* tmp = head;

        while (tmp != NULL){
            printf("%s\n", tmp->name);
            tmp = (MusicalComposition *) tmp->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);

    if(names == NULL || authors == NULL || years == NULL){
        memoryError();
    }

    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));
        if(names[i] == NULL || authors[i] == NULL){
            memoryError();
        }

        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;
}
```