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: «Структуры данных, линейные списки»

Студент гр. 3343 _____ Какира У.Н.

Преподаватель _____ Государкин Я. С.

Санкт-Петербург
2024

Цель работы.

Изучить понятие линейного списка в программировании, его строение и основные функции работы с ним, научиться реализовывать его в языке C при помощи структур.

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]).
- 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.

Выполнение работы.

1) Структура *MusicalComposition* содержит переменные name, author, year, которые хранят соответственно название композиции, автора и год создания. Она также содержит два указателя *prev* и *next* на другие элементы, являющиеся соответственно предыдущим и следующим в двунаправленном линейном списке.

2) Функция *createMusicalComposition* создает и возвращает новый элемент структуры *MusicalComposition*. В функции используется метод *strcpy* для копирования переданных строк в поля данной структуры. А так как мы создаем единственный элемент, то поля *next* и *prev* инициализируем как NULL. В конце возвращаем указатель на созданную композицию.

3) Функция *createMusicalCompositionList* создает и возвращает двусвязный список *MusicalComposition*. Она принимает массивы строк *array_names* и *array_authors*, и массив целых чисел *array_years*, содержащие информацию о всех музыкальных композициях, а также количество элементов *n*. В функции создается новый элемент структуры *MusicalComposition* для каждой композиции, и каждый элемент связывается с предыдущим и

следующим элементом, чтобы создать двусвязный список. Функция возвращает указатель на первый элемент списка.

4) Функция *push* добавляет новый элемент в конец двусвязного списка. Она принимает указатель на первый элемент списка *head* и указатель на элемент, который нужно добавить *element*. Функция проходит по всем элементам списка до последнего элемента, и добавляет новый элемент после него.

5) Функция *removeEl* удаляет элемент из двусвязного списка. Она принимает указатель на первый элемент списка *head* и строку *name_for_remove*, содержащую название композиции, которую нужно удалить. Функция находит элемент с соответствующим названием и удаляет его из списка. Если удаляемый элемент имеет предыдущий или следующий элемент, то эти элементы переустанавливают свои указатели, чтобы обойти удаленный элемент. Функция освобождает память, выделенную для удаляемого элемента.

6) Функция *count* подсчитывает количество элементов в двусвязном списке. Она принимает указатель на первый элемент списка *head* и проходит по всем элементам списка, увеличивая счетчик count. Функция возвращает количество элементов списка count.

7) Функция *print_names* выводит названия всех композиций в двусвязном списке. Она принимает указатель на первый элемент списка *head* и проходит по всем элементам списка, выводя название каждой композиции.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
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	Ответ верный.

Выводы.

Были изучены понятие линейного списка, его строение и функции работы с ним. Разработана программа, выполняющая считывание с клавиатуры количество элементов списка, их описание, а также добавляемый элемент и удаляемое имя. Для этого был реализован двунаправленный линейный список, в основе которого лежат структуры. Для управления списком были реализованы функции создания элемента списка, создания самого списка, добавления элемента в конец списка и удаления элемента по полю.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: *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* createComposition(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* createCompositionList(char** names, char**
authors, int* years, int n){
    MusicalComposition *head = createComposition(names[0], authors[0],
years[0]);
    if (n != 1){
        MusicalComposition *composition = createComposition(names[1],
authors[1], years[1]);
        head->next = composition;
        for (int i = 2; i < n; i++){
            composition->next = createComposition(names[i],
authors[i], years[i]);
            composition = composition->next;
        }
    }
    return head;
}

void pushComposition(MusicalComposition* head, MusicalComposition*
element){
    MusicalComposition *composition = head;
    while(composition->next){
        composition = composition->next;
    }
    composition->next = element;
}
```

```
void removeComposition(MusicalComposition* head, char*
name_for_remove){
    MusicalComposition *del_composition;
    if (strcmp(head->name, name_for_remove) == 0){
        del_composition = head;
        head = head -> next;
        free(del_composition);
    }else{
        while (head->next){
            if (strcmp(head->next->name, name_for_remove) == 0){
                del_composition = head->next;
                head->next = head->next->next;
                free(del_composition);
            }
            head = head->next;
        }
    }
}

int countCompositions(MusicalComposition* head){
    MusicalComposition *composition = head;
    int count = 0;
    while(composition){
        count++;
        composition = composition->next;
    }
    return count;
}

void printCompositionNames(MusicalComposition* head){
    MusicalComposition *composition = head;
    while (composition){
        puts(composition->name);
        composition = composition->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 = createCompositionList(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 =
createComposition(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 = countCompositions(head);

    printf("%d\n", k);
    pushComposition(head, element_for_push);

    k = countCompositions(head);
    printf("%d\n", k);

    removeComposition(head, name_for_remove);
    printCompositionNames(head);

    k = countCompositions(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;
}
```