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. 1304 _____ Крупин Н. С.

Преподаватель _____ Чайка К. В.

Санкт-Петербург

2022

Цель работы

Освоение работы с линейными списками.

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 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

Так как имена и авторы композиций хранятся в массивах, при создании элемента списка было принято решение обойтись указателями. Структура *MusicalComposition* состоит из полей *name*, *author* — указателей на строки и *year* — числа, а также указателей *prev* и *next* на предыдущий и последующий элементы списка соответственно.

Функция *createMusicalComposition()* динамически выделяет память под структуру, заполняет поля переданными значениями, не трогая указатели *prev* и *next*, и возвращает указатель на созданный элемент.

Функция *createMusicalCompositionList()* создаёт двунаправленный список элементов с помощью *createMusicalComposition()*, заполняя поля на основе переданных массивов, а указатели *prev* и *next* — в соответствии с их порядком в массивах, граничные указатели равны *NULL*. Функция возвращает указатель на первый элемент списка или *NULL*, если ожидается 0 элементов.

Функция *push()* в цикле находит последний элемент списка по переданному началу и связывает переданный элемент с концом списка, редактируя указатели *prev* и *next*. Было принято решение изменить сигнатуру функции с одобрения преподавателя, и передавать не указатель на первый элемент списка, а адрес указателя, чтобы можно было изменить указатель начала, если элемент добавляется к пустому списку.

Функция *removeEl()* по очереди сравнивает поле *name* каждого элемента списка со значением *name_for_remove*. Очищает память совпадающего элемента и связывает соседние элементы списка. Удаляет все элементы с искомым значением *name*. В случае удаления первого элемента возвращает новое значение указателя на начало списка (для этого сигнатура функции изменена с одобрения преподавателя, чтобы принимать не указатель, а его адрес из *main()*). В случае удаления всех элементов возвращает *NULL*.

Функция *count()* перебирает все элементы списка в цикле и возвращает их количество.

Функция *print_names()* печатает значения полей *name* (каждое на новой строке) по порядку, в котором элементы стоят в списке. Для пустого списка ничего не печатает.

В функции *main()* изменено:

- вызовы функций с исправленными сигнатурами (обговорено с преподавателем);
- опечатки в названиях типов при выделении памяти для строк внутри цикла (обговорено с преподавателем);
- печать полей первого элемента списка (теперь перед командой добавлена проверка на пустоту списка, чтобы программа не завершалась аварийно при обращении к полю несуществующего элемента).

Разработанный программный код см. в приложении А.

Тестирование

Результаты тестирования представлены в таблице 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	3 Люди Дайте танк (!) 2020 Люди Nautilus Pompilius 1989 Люди Мот 2020 Курьер Дайте танк (!) 2017 Люди	Люди Дайте танк (!) 2020 3 4 Курьер 1
2.	3 Люди Дайте танк (!) 2020 Люди Nautilus Pompilius 1989 Люди Мот 2020 Курьер Дайте танк (!) 2017 Курьер	Люди Дайте танк (!) 2020 3 4 Люди Люди Люди 3
3.	0 Люди Nautilus Pompilius 1989 Слова-паразиты	0 1 Люди 1
4.	0 Люди Nautilus Pompilius 1989 Люди	0 1 0

Выводы

Изучена структура данных — линейный список. Разработан API для работы с двунаправленным списком на языке Си.

Программой не предусмотрена очистка памяти, выделенной для списка.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

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

// Создание структуры MusicalComposition
MusicalComposition* createMusicalComposition(char* name, char* author,
                                                          int year){
    MusicalComposition* newMC = malloc(sizeof(MusicalComposition));
    newMC->name = name;
    newMC->author = author;
    newMC->year = year;

    return newMC;
}

// Функции для работы со списком MusicalComposition
MusicalComposition* createMusicalCompositionList(char** array_names,
                        char** array_authors, int* array_years, int n){
    if (n <= 0) return NULL;

    MusicalComposition* head = createMusicalComposition(array_names[0],
                                    array_authors[0], array_years[0]);
    MusicalComposition* tail = head;
    head->prev = NULL;
    for (int i = 1; i < n; i++){
        tail->next = createMusicalComposition(array_names[i],
                                    array_authors[i], array_years[i]);
        tail->next->prev = tail;
        tail = tail->next;
    }
    tail->next = NULL;

    return head;
}

void push(MusicalComposition** p_head, MusicalComposition* element){
    MusicalComposition* last = *p_head;
    if (last){
        while (last->next) last = last->next;
        last->next = element;
    } else *p_head = element;
    element->prev = last;
    element->next = NULL;
}
```

```
void removeEl(MusicalComposition** p_head, char* name_for_remove){
    MusicalComposition* current = *p_head;
    MusicalComposition* temp = NULL;
    while (current)
        if (!strcmp(current->name, name_for_remove)){
            if (current->next) current->next->prev = current->prev;
            if (current->prev) current->prev->next = current->next;
            else *p_head = current->next;
            temp = current;
            current = current->next;
            free(temp);
        } else current = current->next;
}

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

    return count;
}

void print_names(MusicalComposition* head){
    while (head){
        puts(head->name);
        head = head->next;
    }
}

// Основная программа
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;

    if (head) 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;

}
```