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.

Студентка гр. 1304 _____ Ярусова Т. В.

Преподаватель _____ Чайка К. В.

Санкт-Петербург
2022

Цель работы.

Изучить основные действия со структурами данных, научиться создавать и использовать линейные списки в программах на языке 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.

Описание структуры *MusicalComposition*:

Структура состоит из 5-ти полей: *char* name* – указатель на строку, в которой содержится название музыкальной композиции; *char* author* – указатель на строку, в которой содержится имя автора музыкальной композиции; *int year* – целое число, которое является годом создания композиции; *struct MusicalComposition* next* – указатель на структуру *MusicalComposition*, в котором содержится указатель на следующий элемент линейного списка; *struct MusicalComposition* previous* - указатель на структуру *MusicalComposition*, в котором содержится указатель на предыдущий элемент линейного списка .

Чтобы не писать каждый раз “*struct MusicalComposition*”, используется оператор *typedef*.

Функции:

MusicalComposition* createMusicalComposition(char* name, char* author,int year)

Функции на вход подаются указатели на строки *name*, *author* и целое число *year*. В функции создается указатель *mc* типа *MusicalComposition*, в котором будет храниться созданная структура. Функцией *malloc()* динамически выделяется память под элемент типа *MusicalComposition*. В соответствующие поля созданной структуры mc копируются данные, которые были получены на вход. В поле *next* и *previous* записывается нулевое значение *NULL*.

Из функции возвращается указатель на созданную структуру *mc*.

MusicalComposition* createMusicalCompositionList (char** array_names, char** array_authors, int* array_years, int n)

Функции на вход подаются массивы, содержащие названия музыкальных композиций, имена авторов данных композиций и года их создания, а также целое число *n* – число элементов в каждом из массивов. В

функции создаются указатели *head* и *next* типа *MusicalComposition*. В *head* будет храниться первый элемент списка. С помощью цикла for проходим по каждому элементу списка и связываем их между друг другом. Первый элемент *head* создается с помощью вышеописанной функции *createMusicalComposition()*, второй элемент *next* создается с помощью вышеописанной функции *createMusicalComposition()* и связываем его с предыдущем через оператор -> обращения к полю структуры, т.е. с *head*, все последующие элементы также создаются с помощью вышеописанной функции *createMusicalComposition()*, но при этом не объявляется новая переменная, а работа идет через обращение к полям структур.

Из функции возвращается указатель на начало списка *head*.

void push(MusicalComposition* head, MusicalComposition* element)

Функции на вход подаются указатель на первый элемент *head* списка и указатель на элемент *element*, который необходимо добавить в данный список. В функции объявляется указатель current типа *MusicalComposition*, которому присваивается указатель на начало списка *head*. С помощью цикла *while()* и оператора обращения к полю структуры *current-> next == NULL* происходит нахождение последнего элемента списка. В поле previous структуры по указателю *element* записывается указатель на последний элемент списка и в поле *next* последнего элемента списка записывается указатель на элемент *element*.

Из функции ничего не возвращается.

void removeEl(MusicalComposition* head, char* name_for_remove)

Функции на вход подается указатель на первый элемент *head* списка и указатель на строку, в которой хранится название композиции, элемент, в которой она хранится требуется удалить. В функции объявляется указатель *current* типа *MusicalComposition*, которому присваивается указатель на начало списка *head*. С помощью цикла *while()* происходит прохождение по каждому

элементу списка. С помощью условного оператора *if* и функции *strcmp* происходит сравнение строки, находящейся в текущем элементе в поле name и строки *name_for_remove*. Если строки равны, то проверяется какой текущей элемент списка.

Если элемент последний, то происходит обращение к полю предыдущего элемента, которое указывает на следующий элемент и присваивается *NULL* и с помощью функции *free()* очищается выделенная память под текущий элемент.

Если элемент находится в середине списка, то происходит обращение к полю, ссылающегося на предыдущий элемент, которое ссылается на следующий элемент и присваивается значение текущего элемента, ссылающегося на поле следующего элемента *(current->previous->next = current->next)* и также происходит обращение к полю, ссылающегося на следующий элемент, которое ссылается на предыдущий элемент и присваивается значение текущего элемента, ссылающегося на поле предыдущего элемента*(current->next->previous = current->previous)*. С помощью функции *free()* очищается выделенная память под текущий элемент.

В данной функции невозможно удалить первый элемент списка, потому что для этого из функции нужно вернуть указатель на новый первый элемент списка. А по условию задания функция *removeEl()* из функции ничего не возвращается и функцию *main()* не требуется менять.

int count(MusicalComposition* head)

Функции на вход подается указатель на первый элемент *head* списка. В функции объявляется указатель *current* типа *MusicalComposition*, которому присваивается указатель на начало списка *head* и целочисленная переменная *count*, которая будет отвечать за количество элементов в списке. С помощью цикла *while()* происходит прохождение по каждому элементу списка. В каждой итерации *count* увеличивается на единицу.

Из функции возвращается *count*.

void print_names(MusicalComposition* head)

Функции на вход подается указатель на первый элемент *head* списка. В функции объявляется указатель *current* типа *MusicalComposition*, которому присваивается указатель на начало списка *head*. С помощью цикла *while()* происходит прохождение по каждому элементу списка. В каждой итерации происходит печать строки, находящейся в поле *name*.

Из функции ничего не возвращается.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

В ходе выполнения лабораторной работы были изучены основные структуры данных языка программирования C.

Разработана программа, в которой был реализован двунаправленный список, а также функции для работы с ним.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Yarusova_Tatyana_lb2.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* mc =
(MusicalComposition*)malloc(sizeof(MusicalComposition));
    mc->name = name;
    mc->author = author;
    mc->year = year;
    mc->next = NULL;
    mc->previous = NULL;
    return mc;
}

MusicalComposition* createMusicalCompositionList(char** array_names,
char** array_authors, int* array_years, int n){
    MusicalComposition* head;
    MusicalComposition* next;
    for(int i = 0; i < n; i++){
        if(i == 0)
            head = createMusicalComposition(array_names[i],
array_authors[i], array_years[i]);
        else{
            if(i == 1){
                next = createMusicalComposition(array_names[i],
array_authors[i], array_years[i]);
                head->next = next;
                next->previous = head;
            }
            else{
                next->next = createMusicalComposition(array_names[i],
array_authors[i], array_years[i]);
                next->next->previous = next;
                next = next->next;
            }
        }
    }
    return head;

}

void push(MusicalComposition* head, MusicalComposition* element){
```

```
    MusicalComposition* current = head;
    while(current->next != NULL){
        current = current->next;
    }
    element->previous = current;
    current->next = element;
}

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

}

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

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

ПРИЛОЖЕНИЕ Б

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
2	7 Fields of Gold Sting 1993 In the Army Now Status Quo 1986 Mixed Emotions The Rolling Stones 1989	Fields of Gold Sting 1993 7 8 Fields of Gold In the Army Now Mixed Emotions Billie Jean Seek and Destroy Wicked Game Points of Authority

	Billie Jean Michael Jackson 1983 Seek and Destroy Metallica 1982 Wicked Game Chris Isaak 1989 Points of Authority Linkin Park 2000 Sonne Rammstein 2001 Sonne	7