**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 |  | Русанов А.И. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

## Цель работы

Целью данной работы является ознакомление со структурой данных «двунаправленный список» и реализация API для работы с ним на языке программирования С.

## 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

В программе определена структура MusicalComposition, содержащая такие поля, как char \* name (название композиции), char \* author (автор композиции), int year (год выхода композиции), struct MusicalComposition \* next (указатель на следующий элемент списка) и struct MusicalComposition \* prev (указатель на предыдущий элемент списка). Для удобства создается псевдоним "MusicalComposition" с использованием ключевого слова "typedef".

Функция createMusicalComposition принимает на вход название, автора и год выхода музыкальной композиции, выделяет память под элемент списка и инициализирует значения соответствующей структуры переданными в фукнцию аргументами. Указатели на следующий и предыдущий элементы инициализируются NULL. Функция возвращает указатель на созданную структуру.

Функция createMusicalCompositionList принимает на вход массив названий, авторов и годов создания музыкальных композиций и число элементов в этих массивах. Далее в цикле for с помощью функции createMusicalComposition инициализируются элементы двунаправленного списка. Функция возвращает указатель на первый элемент.

Функция push принимает на вход указатель на первый элемент списка и элемент типа MusicalComposition \*, который необходимо добавить в конец списка. Функция перебирает элементы списка до тех пор, пока не встретит последний элемент (указатель на следующий элемент которого равен NULL) и добавляет указатели next и prev соответстующих элементов нужным образом.

Функция removeEl удаляет элемент из двусвязного списка MusicalComposition. Она принимает указатель на «голову» списка (head) и указатель на строку с именем элемента, который нужно удалить (name\_for\_remove). Сначала функция проверяет, является ли головной элемент искомым элементом по имени. Если имя совпадает, то следующий элемент становится новой «головой» списка, а предыдущий указатель устанавливается в NULL и «голову» удаляют с помощью функции free(). Если имя головного элемента не совпадает, то функция переходит к следующему элементу списка до тех пор, пока не найдет элемент с нужным именем или не достигнет конца списка. Если элемент с нужным именем найден, то связи между предыдущим и следующим элементами обновляются, а сам элемент удаляется с помощью функции free().

Функция count получает на вход указатель на «голову» списка и перебирает все элементы списка, пока указатель на текущий элемент не станет равным NULL. Вместе с этим идет подсчет не равных NULL элементов списка.

Функция print\_names получает на вход первый элемент списка и выводит все его элементы списка, пока не встретит указатель на текущий элемент, который будет равен NULL.

Разработанный программный код см. в приложении А.

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |
| --- | --- | --- |
| № п/п | Входные данные | Выходные данные |
|  | 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

Была изучена структура данных «двунаправленный список» и реализована API для работы с ним на языке программирования С.

# 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 \*prev;

} MusicalComposition;

MusicalComposition \*createMusicalComposition(char \*name, char \*autor, int year)

{

MusicalComposition \*MusicComp = (MusicalComposition \*)malloc(sizeof(MusicalComposition));

if (MusicComp == NULL)

{

fprintf(stderr, "Memory allocation error!\n");

}

MusicComp->name = name;

MusicComp->author = autor;

MusicComp->year = year;

MusicComp->next = NULL;

MusicComp->prev = NULL;

return MusicComp;

}

void push(MusicalComposition \*head, MusicalComposition \*element)

{

MusicalComposition \*tmp = head;

while (tmp->next != NULL)

{

tmp = tmp->next;

}

tmp->next = element;

element->prev = tmp;

}

void removeEl(MusicalComposition \*head, char \*name\_for\_remove)

{

MusicalComposition \*tmp = head;

if (strcmp(tmp->name, name\_for\_remove) == 0)

{

head = tmp->next;

if (head != NULL)

{

head->prev = NULL;

}

free(tmp);

return;

}

while (strcmp(tmp->name, name\_for\_remove) != 0)

{

tmp = tmp->next;

}

tmp->prev->next = tmp->next;

tmp->next->prev = tmp->prev;

free(tmp);

}

int count(MusicalComposition \*head)

{

MusicalComposition \*tmp = head;

int count = 0;

while (tmp)

{

count++;

tmp = tmp->next;

}

return count;

}

void print\_names(MusicalComposition \*head)

{

MusicalComposition \*tmp = head;

while (tmp)

{

printf("%s\n", tmp->name);

tmp = tmp->next;

}

}

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 \*prev = head;

for (int i = 1; i < n; i++)

{

MusicalComposition \*current = createMusicalComposition(array\_names[i], array\_authors[i], array\_years[i]);

prev->next = current;

current->prev = prev;

prev = current;

}

return head;

}

int main()

{

int length;

scanf("%d\n", &length);

char \*\*names = (char \*\*)malloc(sizeof(char \*) \* length);

if (names == NULL)

{

fprintf(stderr, "Memory allocation error!\n");

}

char \*\*authors = (char \*\*)malloc(sizeof(char \*) \* length);

if (authors == NULL)

{

fprintf(stderr, "Memory allocation error!\n");

}

int \*years = (int \*)malloc(sizeof(int) \* length);

if (years == NULL)

{

fprintf(stderr, "Memory allocation error!\n");

}

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));

if (names[i] == NULL)

{

fprintf(stderr, "Memory allocation error!\n");

}

authors[i] = (char \*)malloc(sizeof(char \*) \* (strlen(author) + 1));

if (authors[i] == NULL)

{

fprintf(stderr, "Memory allocation error!\n");

}

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;

}