**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 |  | Белаид Фарук | |
| Преподаватель |  | | Глазунов С.А. |

Санкт-Петербург

2024

## Цель работы

Научиться реализовывать структурную и функциональную составляющую двунаправленных линейных списков в языке С. Изучить принцип создания подобных списков с помощью 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 информацию о музыкальной композиции: название, автор и год выпуска. Она также содержит два указателя на другие элементы *prev* и *next*, которые используются для создания двусвязного списка.

2) Функция *createMusicalComposition* создает и возвращает новый элемент структуры *MusicalComposition*. В функции выделяется память для каждой строки *name* и *author*, и используется функция *strcpy* для копирования переданных строк в выделенную память.

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 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 голову 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 голову 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* считает количество элементов в двусвязном списке. Она принимает указатель на голову списка *head* и проходит по всем элементам списка, подсчитывая количество элементов.

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

В целом, данный код представляет собой реализацию двусвязного списка для хранения музыкальных композиций.

Разработанный программный код см. в приложении A.

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

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

Разработанная программа написана на языке Си и предназначена для поиска заданного слова в строке. Сначала программа разбивает строку на слова, сортирует их в алфавитном порядке с помощью функции qsort, а затем использует функцию bsearch для поиска заданного слова в отсортированном массиве. Если слово найдено, программа выводит "exists", иначе - "doesn't exist". Эта программа демонстрирует применение стандартных функций языка Си для работы со строками и массивами, а также алгоритма бинарного поиска.

# Приложение А Исходный код программы

Название файла: main.c

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

struct MusicalComposition {

char \*name;

char \*author;

int year;

struct MusicalComposition \*prev;

struct MusicalComposition \*next;

};

typedef struct MusicalComposition MusicalComposition;

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

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

if (composition == NULL) {

fprintf(stderr, "Memory allocation failed\n");

exit(EXIT\_FAILURE);

}

composition->name = malloc(81 \* sizeof(char));

composition->author = malloc(81 \* sizeof(char));

if (composition->name == NULL || composition->author == NULL) {

fprintf(stderr, "Memory allocation failed\n");

exit(EXIT\_FAILURE);

}

composition->year = year;

composition->prev = NULL;

composition->next = NULL;

strcpy(composition->author, author);

strcpy(composition->name, name);

return composition;

}

MusicalComposition \*createMusicalCompositionList(char \*\*array\_names, char \*\*array\_authors, int \*array\_years, int n) {

MusicalComposition \*head;

MusicalComposition \*last\_elem;

for (int i = 0; i < n; i++) {

MusicalComposition \*new\_elem;

if (i == 0) {

new\_elem = createMusicalComposition(array\_names[i], array\_authors[i], array\_years[i]);

head = new\_elem;

} else {

new\_elem = createMusicalComposition(array\_names[i], array\_authors[i], array\_years[i]);

last\_elem->next = new\_elem;

new\_elem->prev = last\_elem;

}

last\_elem = new\_elem;

}

return head;

}

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

MusicalComposition \*next\_elem = head->next;

while (next\_elem->next != NULL) {

next\_elem = next\_elem->next;

}

next\_elem->next = element;

element->prev = next\_elem;

}

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

MusicalComposition \*next\_elem = head->next;

while (next\_elem != NULL) {

if (strcmp(next\_elem->name, name\_for\_remove) == 0) {

if (next\_elem->next != NULL) {

next\_elem->next->prev = next\_elem->prev;

}

if (next\_elem->prev != NULL) {

next\_elem->prev->next = next\_elem->next;

}

free(next\_elem->name); // Free memory for name

free(next\_elem->author); // Free memory for author

free(next\_elem); // Free memory for the element

break;

}

next\_elem = next\_elem->next;

}

}

int count(MusicalComposition \*head) {

int cnt = 0;

MusicalComposition \*next\_elem = head;

while (next\_elem != NULL) {

cnt++;

next\_elem = next\_elem->next;

}

return cnt;

}

void print\_names(MusicalComposition \*head) {

MusicalComposition \*next\_elem = head;

while (next\_elem != NULL) {

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

next\_elem = next\_elem->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) {

fprintf(stderr, "Memory allocation failed\n");

exit(EXIT\_FAILURE);

}

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) {

fprintf(stderr, "Memory allocation failed\n");

exit(EXIT\_FAILURE);

}

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;

}