**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

Курсовая РАБОТА

**по дисциплине «Программирование»**

Тема: Линейные списки

|  |  |  |
| --- | --- | --- |
| Студент гр. 6303 |  | Ваганов Н.А. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2016

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Ваганов Н.А. | | |
| Группа 6303 | | |
| Тема работы: Линейные списки | | |
|  | | |
| Содержание пояснительной записки:   * Содержание * Введение * Описание функций, необходимых для работы с двунаправленным линейным списком * Описание функций создание «среза» * Примеры работы программы * Заключение * Список использованных источников * Приложение А. Исходный код программы | | |
| Предполагаемый объем пояснительной записки:  Не менее 10 страниц. | | |
| Дата выдачи задания: 02.12.2016 | | |
| Дата сдачи реферата: 21.12.2016 | | |
| Дата защиты реферата: 21.12.2016 | | |
| Студент |  | Ваганов Н.А. |
| Преподаватель |  | Берленко Т.А. |

**Аннотация**

В данной работе была разработана программа на языке программирования C, позволяющая работать с набором функций, отвечающих засписок музыкальных композиций. Для функционирования списка были созданы и описаны необходимые функции, позволяющие добавлять, удалять, сортировать и выводить элементы списка и их количество в консоль, а также описана структура элемента списка. Помимо этого, была проведена работа над оптимизацией исходного кода программы для ускорения ее быстродействия и оптимального использования памяти и ресурсов клиента. Приведено полное описание исходного кода.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 4 |
| 1. | Описание функций, необходимых для работы с двунаправленным линейным списком | 6 |
| 1.1. | Созданиеэлемента | 6 |
| 1.2. | Создание двунаправленного списка связанных элементов | 6 |
| 1.3. | Добавление элемента в конец списка | 6 |
| 1.4. | Удаление элементов из списка | 7 |
| 1.5. | Подсчет количества элементов в списке | 7 |
| 1.6. | Вывод элементов списка в консоль | 7 |
| 2. | Описание функции создания «среза» | 8 |
| 2.1. | Создание «среза» | 8 |
|  | Примеры работы программы | 9 |
|  | Заключение | 11 |
|  | Список использованных источников | 12 |
|  | Приложение А. Исходный код программы | 13 |

**введение**

Необходимо создать двунаправленный список музыкальных композиций MusicalComposition и **api** (***a****pplication****p****rogramming****i****nterface - 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.

Функции, необходимые для работы программы

* Создание элемента типа MusicalComposition
* Создание двунаправленного списка связанных элементов типа MusicalComposition
* Добавление элемента в конец списка
* Удаление определенного элемента (или элементов) из списка
* Подсчет количества элементов в списке
* Вывод элементов списка в консоль
* Сортировка элементов в списке
* Смена местами двух соседних элементов списка

**1. функции для работы с двунаправленным линейным списком**

* 1. **Создание элемента**

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

{

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

next->name = name;

next->author = author;

next->year = year;

next->next=NULL;

next->prev=NULL;

return next;

}

Функция принимает в качестве аргументов указатели на название композиции (**char**\* name) и ее автора (**char**\* author), а также год написания (**int** year). Происходит выделение памяти для структура типа MusicalComposition и заполняются ее переменные.

* 1. **Создание двунаправленного списка связанных элементов**

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

{

MusicalComposition\* head;

int i=1;

head=createMusicalComposition(array\_names[0], array\_authors[0], array\_years[0]);

MusicalComposition\* ptr=head;

for (i;i<n;i++)

{

ptr->next=createMusicalComposition(array\_names[i], array\_authors[i], array\_years[i]);

ptr->next->prev = ptr;

ptr->next->next=NULL;

ptr=ptr->next;

}

return head;

}

Функция принимает в качестве аргументов указатели на указатель на массив названий композиций (**char**\*\* array\_names) и их авторов (**char**\*\* array\_authors), указатель на массив лет написания (**int\*** array\_years), размер массива (**int** n). Сначала создается “голова” списка и происходит заполнение ее переменных данными, затем аналогичные операции происходят для всех последующих n-1 элементов списка

* 1. **Добавление элемента в конец списка**

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

{

while (head->next != NULL)

{

head=head->next;

}

head->next=element;

element->next=NULL;

}

Функция принимает в качестве аргументов указатель на “голову” списка (MusicalComposition\* head) и элемент (MusicalComposition\* element), который нужно добавить в конец списка. В первую очередь происходит поиск последнего элемента списка, далее новый элемент вставляется после текущего.

* 1. **Удаление элементов из списка**

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

{

MusicalComposition\* buf=head;

while (head != NULL)

{

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

{

if (head->prev == NULL)

{

\*head = \*head->next;

head->prev=NULL;

buf=head;

}

else if (head->next == NULL)

{

head->prev->next = NULL;

buf=head;

}

else

{

head->prev->next = head->next;

head->next->prev = head->prev;

buf=head;

}

free(buf);

}

head=head->next;

}

}

Функция принимает в качестве аргументов указатель на “голову” списка (MusicalComposition\* head) и название композиций (**char**\* name\_for\_remove), которые нужно удалить из списка. Для каждого его элемента происходит сравнение названия композиции с именем произведений, которые нужно удалить.

* 1. **Подсчет количества элементов в списке**

int count(MusicalComposition\* head)

{

int counter = 0;

while(head)

{

counter++;

head=head->next;

}

return counter;

}

Функция принимает в качестве аргументов указатель на “голову” списка (MusicalComposition\* head). Далее происходит подсчет элементов в списке, пока следующий элемент существует

* 1. **Вывод элементов списка в консоль**

void print\_names(MusicalComposition\* head)

{

while(head)

{

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

head=head->next;

}

}

Функция принимает в качестве аргументов указатель на “голову” списка (MusicalComposition\* head). Далее происходит вывод названия и даты написания композиции в списке, пока следующий элемент существует.

1. **функции СОздания «среза»**
2. **Создание «среза»**

int cut(MusicalComposition\* head, int i, int j)

{

int count = 1;

if (i < j && i != 0 && j != 0 && head != NULL)

{

for (count;count <= j;count++)

{

if (count != i && count < i)

{

\*head=\*head->next;

head->prev=NULL;

}

if (count > i && count < j)

{

head=head->next;

}

else if (count == j) head->next=NULL;

}

return 0;

}

else return 1;

}

Функция принимает в качестве аргумента указатель на “голову” списка (MusicalComposition\* head), а также целые числа i и j, где i – первый элемент «среза» и j – второй. Последовательно переставляя «голову» списка, доходим до i-того элемента, таким образом делая элемент новой «головой». Далее идем до j-того элемента, чей указатель на следующий элемент делаем равным NULL. В результате получаем новый список, начинающийся с i-того элемента и заканчивающийся j-тым.

**примеры работы программы**

Для демонстрации всех возможностей программы используется следующий формат ввода:

* Количество элементов в списке
* Элементы списка
* Элемент для добавления в список
* Название композиции, которую нужно удалить

И вывода:

* Название, автор и год написания первого элемента списка
* Количество элементов до добавления нового
* Количество элементов после добавления
* Названия всех композиций и год написания после удаления
* Количество элементов после удаления
* Названия всех композиций, оставшихся в результате создания «среза»

|  |  |  |
| --- | --- | --- |
| № | ВВОд | вЫВОД |
| 1 | 6  My Propeller  Arctic Monkeys  2005  Radioactive  Imagine Dragons  2011  Sweater Weather  The Neighbourhood  2012  The Sharpest lives  My Chemical Romance  2013  Feeling Good  Muse  2006  Counting Stars  One Republic  2014  Kids With Guns  Gorillaz  2007  Sweater Weather  3 5 | My Propeller Arctic Monkeys 2005  //первый элемент списка  6 //количество элементов  7 //кол-во после добавления  My Propeller  Radioactive  The Sharpest lives  Feeling Good  Counting Stars  Kids With Guns  6 //кол-во после удаления элемента  The Sharpest lives //сформированный «срез» списка  Feeling Good //с 3 по 5 элементы включительно  Counting Stars |
| 2 | 4  Feeling Good  Muse  2006  Counting Stars  One Republic  2014  Kids with Guns  Gorillaz  2007  Sweater Weather  The Neighbourhood  2012  My Propeller  Arctic Monkeys  2005  Kids with Guns  1 3 | Feeling Good Muse 2006  4 //количество элементов  5 //кол-во после удаления  Feeling Good  Counting Stars  Sweater Weather  My Propeller  4  Feeling Good //сформированный срез списка  Counting Stars //с 1 по 3 элементы включительно  Sweater Weather |
| 3 | 5  Feeling Good  Muse  2006  Counting Stars  One Republic  2014  Kids With Guns  Gorillaz  2007  Old Yellow Bricks  Arctic Monkeys  2011  Arabella  Arctic Monkeys  2012  Red Flag  Billy Talent  2015  Arabella  3 4 | Feeling Good Muse 2006  5 //количество элементов  6 //кол-во после удаления  Feeling Good  Counting Stars  Kids With Guns  Old Yellow Bricks  Red Flag  5  Kids With Guns //сформированный срез списка  Old Yellow Bricks //с 3 по 4 элементы включительно |

**заключение**

Был создан и описан двунаправленный список музыкальных композиций MusicalComposition и api для работы со списком. Также были созданы и описаны все необходимые функции и структуры для работы программы.

**список использованных источников**

1. Язык программирования СИ / Керниган Б., Ритчи Д. СПб.: Издательство "Невский Диалект", 2001. 352 с.
2. UNIX.  Программное окружение / Керниган Б., Пайк Р. СПб.: Символ Плюс, 2003. 416 с.

**приложение А**

**Исходный код программы**

#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\* author, int year){

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

next->name = name;

next->author = author;

next->year = year;

next->next=NULL;

next->prev=NULL;

return next;

}

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

MusicalComposition\* head;

int i=1;

head=createMusicalComposition(array\_names[0], array\_authors[0], array\_years[0]);

MusicalComposition\* ptr=head;

for (i;i<n;i++){

ptr->next=createMusicalComposition(array\_names[i], array\_authors[i], array\_years[i]);

ptr->next->prev = ptr;

ptr->next->next=NULL;

ptr=ptr->next;

}

return head;

}

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

while (head->next != NULL){

head=head->next;

}

head->next=element;

element->next=NULL;

}

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

MusicalComposition\* buf=head;

while (head != NULL){

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

if (head->prev == NULL){

\*head = \*head->next;

head->prev=NULL;

buf=head;

}

else if (head->next == NULL){

head->prev->next = NULL;

buf=head;

}

else {

head->prev->next = head->next;

head->next->prev = head->prev;

buf=head;

}

free(buf);

}

head=head->next;

}

}

int count(MusicalComposition\* head){

int counter = 0;

while(head){

counter++;

head=head->next;

}

return counter;

}

void print\_names(MusicalComposition\* head){

while(head){

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

head=head->next;

}

}

int cut(MusicalComposition\* head, int i, int j) {

int count = 1;

if (i < j && i != 0 && j != 0 && head != NULL){

for (count;count <= j;count++) {

if (count < i) {

\*head=\*head->next;

head->prev=NULL;

}

if (count > i && count < j) {

head=head->next;

}

else if (count == j){

head=head->next;

head->next=NULL;

}

}

return 0;

}

else return 1;

}

int main(){

int length,c,d;

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);

printf("Введите начало и конец среза (числа через пробел)\n");

scanf("%d %d",&c,&d);

cut(head, c, d);

print\_names(head);

while (head->next){

MusicalComposition\* buf=head;

head=head->next;

free(buf);

}

return 0;

}