**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 |  | Филиппенко Д.Р. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2016

**ЗАДАНИЕ**

**для курсовой работы**

|  |  |  |
| --- | --- | --- |
| Студент: Филиппенко Д.Р. | | |
| Группа: 6303 | | |
| Тема работы: Линейные списки | | |
|  | | |
| Содержание пояснительной записки:   * Содержание * Введение * Описание функций, необходимых для работы с двунаправленным линейным списком * Описание функции добавления элементов после четного элемента списка * 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 курсовой работы: | | |
| Дата защиты курсовой работы: | | |
| Студент |  | Филиппенко Д.Р. |
| Преподаватель |  | Берленко Т.А. |

**Аннотация**

В данной работе была разработана программа на языке программирования C, позволяющая работать с набором функций, отвечающих за созданиесписка музыкальных композиций. Для работы со списком были созданы и описаны необходимые функции, позволяющие добавлять, удалять, выводить элементы списка и их количество на консоль, а также описана структура элемента списка. Помимо этого, была проведена работа над оптимизацией исходного кода. Приведено полное описание исходного кода.

Содержание

[Введение 5](#_Toc470196914)

[1. Функции для работы с двунаправленным линейным списком. 6](#_Toc470196915)

[1.1. Создание элемента 6](#_Toc470196916)

[1.2. Создание двунаправленного списка 6](#_Toc470196917)

[1.3. Добавление элемента в конец списка 7](#_Toc470196918)

[1.4. Удаление элементов списка 7](#_Toc470196919)

[1.5. Подсчет количества элементов в списке 8](#_Toc470196920)

[1.6. Вывод элементов на консоль 8](#_Toc470196921)

[2. Функция добавления двух элементов после каждого четного элемента списка 9](#_Toc470196922)

[2.1. Создание функции 9](#_Toc470196923)

[Примеры работы программы 11](#_Toc470196924)

[Заключение 13](#_Toc470196925)

[Список использованных источников 14](#_Toc470196926)

[Приложение А 15](#_Toc470196927)

[Исходный код программы 15](#_Toc470196928)

### 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

## Функции для работы с двунаправленным линейным списком.

## Создание элемента

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

{

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

MusComp->name=name;

MusComp->author=author;

MusComp->status = 1;

MusComp->year=year;

MusComp->prev=NULL;

MusComp->next=NULL;

return MusComp;

}

В качестве аргументов функция принимает два указателя: на название композиции (char\* name) и на ее автора (char\* author), и год создания (int year). Необходимо выделить память для структуры типа MusicalComposition, а также присвоить значения ее переменным. Переменная status будет необходима при удалении элементов из списка.

## Создание двунаправленного списка

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;

MusicalComposition \*actual;

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

{

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

actual->prev = prev;

prev->next = actual;

prev = actual;

}

return head;

}

Функция принимает в качестве аргументов указатель на указатель массива названий композиций (char\*\* array\_names), указатель на указатель массива композиторов песен (char\*\* array\_authors) и указатель на массив, содержащий даты создания композиций (int\* array\_years), размер массива (int n). Создаем «голову» списка. Заполняем ее данными. Далее те же действия происходят со всеми

(n-1) элементами массива.

## Добавление элемента в конец списка

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

{

MusicalComposition\* actual = head;

MusicalComposition\* new = createMusicalComposition(element->name,

element->author, element->year);

if (head-> status == 0)

\*head = \*new;

else

{

while(actual->next != NULL)

actual = actual->next;

actual->next = new;

new->prev = actual;

}

}

Аргументами функции являются указатель на «голову» списка (MusicalComposition\* head) и указатель на элемент (MusicalComposition\* element), добавляемый в конец списка. В теле функции происходит поиск последнего элемента, затем в конец списка добавляется новый элемент.

## Удаление элементов списка

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

{

MusicalComposition\* actual = head;

while(actual != NULL )

if (strcmp(actual->name, name\_for\_remove) == 0)//strcmp сравнивает name и name\_for\_remove

{

if(actual->next != NULL && actual->prev != NULL)

{

actual->prev->next = actual->next;

actual->next->prev = actual->prev;

actual = actual->next;

}

else if(actual->prev == NULL && actual->next != NULL)

{

\*actual = \*(actual->next);

actual->prev = NULL;

}

else if(actual->next == NULL)

{

if (actual->prev == NULL)

{

head->status = 0;

actual=NULL;

}

else

{

actual->prev->next = NULL;

actual=NULL;

}

}

}

else

actual = actual->next;

}

На вход функции подаются указатель на «голову» списка (MusicalComposition\* head) и указатель на названия композиций, которые нужно удалить (char\* name\_for\_remove). С помощью strcmp выполняется сравнивание имени элемента списка с именем элемента, который необходимо удалить (name\_for\_remove). (head->status = 0) осуществляется, если в списке удалены все элементы, кроме последнего. Т.к. по умолчанию status = 1, то при удалении всех элементов списка переменная не будет меняться.

## Подсчет количества элементов в списке

int count(MusicalComposition\* head)

{

MusicalComposition\* actual = head;

int count = 0;

while(actual)

if (actual->status == 0)

return 0;

else

{

count++;

actual = actual->next;

}

return count;

}

Функция принимает на вход указатель на «голову» списка (MusicalComposition\* head). Пока существует следующий элемент, будет происходить подсчет элементов.

## Вывод элементов на консоль

void print\_names(MusicalComposition\* head)

{

MusicalComposition\* actual = head;

if (actual->status == 1)

while(actual)

{

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

actual = actual->next;

}

}

В качестве аргумента функция принимает указатель на «голову» списка (MusicalComposition\* head).

Пока существует следующий элемент списка, на консоль будет выводиться название композиции.

## Функция добавления двух элементов после каждого четного элемента списка

## Создание функции

void push\_double(MusicalComposition\* head, char \*name, char \*author, int year)

{

MusicalComposition\* actual = head;

MusicalComposition\* head\_temp = head;

MusicalComposition \*next;

int i = 1;

while(actual->next)

{

if((i%2) == 0)

{

MusicalComposition\* element1 = createMusicalComposition(name, author, year);

MusicalComposition\* element2 = createMusicalComposition(name, author, year);

next = actual->next;

actual->next = element1;

element1->prev = actual;

element1->next = element2;

element2->prev = element1;

element2->next = next;

next->prev = element2;

actual = next;

}

else

actual = actual->next;

i++;

}

if ((i%2) == 0)

{

MusicalComposition\* element1 = createMusicalComposition(name, author, year);

MusicalComposition\* element2 = createMusicalComposition(name, author, year);

actual->next = element1;

element1->prev = actual;

element1->next = element2;

element2->prev = element1;

}

}

На вход функции подаются указатели на структуры (MusicalComposition\* head, char \*name, char \*author, int year). Текущему элементу присваивается «голова» списка (MusicalComposition\* actual = head). Также создадим временный указатель, которому тоже присвоим значение «головы» списка и указатель на следующий элемент (MusicalComposition \*next). Пока существует следующий элемент списка, проверяется условие: если элемент четный, то после него добавить 2 произвольных элемента. Если условие не выполняется для элемента (он нечетный), то переходим к следующему, увеличивая счетчик на единицу. Когда встречается последний элемент списка, он также проверяется условием, и если он четный, то после него добавляются 2 элемента.

### Примеры работы программы

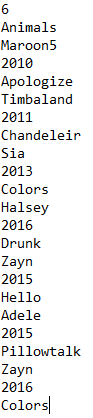
Для демонстрации всех возможностей программы используется следующий формат ввода:

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

И вывода:

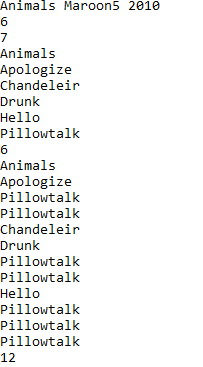
* Название, автор и год написания первого элемента списка
* Количество элементов до добавления нового
* Количество элементов после добавления
* Названия всех композиций после удаления
* Количество элементов после удаления
* Названия всех композиций после выполнения функции добавления элементов.
* Количество элементов после добавления.

ВВОД

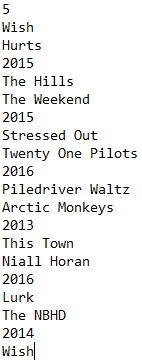


На вход подается 6 значений, которые добавляются в список, затем подается значение, которое добавляется 7-ым элементом, и после него вводится название композиции, которую необходимо удалить.

ВЫВОД



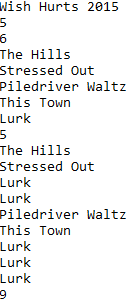
После выполнения программы выводится следующее: первый элемент списка, число композиций, число композиций после добавления нового элемента, список элементов после добавления, число элементов после удаления, список и число композиций, после выполнения функции push\_double.



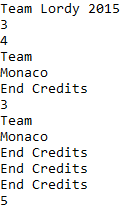
На вход подается 5 значений, которые добавляются в список, затем подается значение, которое добавляется 6-ым элементом, и после него вводится название композиции, которую необходимо удалить.



На вход подается 3 значения, которые добавляются в список, затем подается значение, которое добавляется 4-ым элементом, и после него вводится название композиции, которую необходимо удалить.



После выполнения программы выводится следующее: первый элемент списка, число композиций, число композиций после добавления нового элемента, список элементов после добавления, число элементов после удаления, список и число композиций, после выполнения функции push\_double.



После выполнения программы выводится следующее: первый элемент списка, число композиций, число композиций после добавления нового элемента, список элементов после добавления, число элементов после удаления, список и число композиций, после выполнения функции push\_double.

### 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.
3. Программирование на языке Си / Подбельский В.В., Фомин С.С. / М.: Финансы и статистика, 2004. 600 с.

## Приложение А

## Исходный код программы

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

struct MusicalComposition

{

char\* name;

char\* author;

int status;

int year;

struct MusicalComposition \*prev, \*next;

};

typedef struct MusicalComposition MusicalComposition;

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

{

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

MusComp->name=name;

MusComp->author=author;

MusComp->status = 1;

MusComp->year=year;

MusComp->prev=NULL;

MusComp->next=NULL;

return MusComp;

}

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;

MusicalComposition \*actual;

int i;

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

{

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

actual->prev = prev;

prev->next = actual;

prev = actual;

}

return head;

}

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

{

MusicalComposition\* actual = head;

MusicalComposition\* new = createMusicalComposition(element->name, element-> author, element->year);

if (head-> status == 0)

\*head = \*new;

else

{

while(actual->next != NULL)

actual = actual->next;

actual->next = new;

new->prev = actual;

}

}

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

{

MusicalComposition\* actual = head;

while(actual != NULL )

if (strcmp(actual->name, name\_for\_remove) == 0)//strcmp сравнивает name и name\_for\_remove

{

if(actual->next != NULL && actual->prev != NULL)

{

actual->prev->next = actual->next;

actual->next->prev = actual->prev;

actual = actual->next;

}

else if(actual->prev == NULL && actual->next != NULL)

{

\*actual = \*(actual->next);

actual->prev = NULL;

}

else if(actual->next == NULL)

{

if (actual->prev == NULL)

{

head->status = 0;

actual=NULL;

}

else

{

actual->prev->next = NULL;

actual=NULL;

}

}

}

else

actual = actual->next;

}

int count(MusicalComposition\* head)

{

MusicalComposition\* actual = head;

int count = 0;

while(actual)

if (actual->status == 0)

return 0;

else

{

count++;

actual = actual->next;

}

return count;

}

void print\_names(MusicalComposition\* head)

{

MusicalComposition\* actual = head;

if (actual->status == 1)

while(actual)

{

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

actual = actual->next;

}

}

void push\_double(MusicalComposition\* head, char \*name, char \*author, int year)

{

MusicalComposition\* actual = head;

MusicalComposition\* head\_temp = head;

MusicalComposition \*next;

int i = 1;

while(actual->next)

{

if((i%2) == 0)

{

MusicalComposition\* element1 = createMusicalComposition(name, author, year);

MusicalComposition\* element2 = createMusicalComposition(name, author, year);

next = actual->next;

actual->next = element1;

element1->prev = actual;

element1->next = element2;

element2->prev = element1;

element2->next = next;

next->prev = element2;

actual = next;

}

else

actual = actual->next;

i++;

}

if ((i%2) == 0)

{

MusicalComposition\* element1 = createMusicalComposition(name, author, year);

MusicalComposition\* element2 = createMusicalComposition(name, author, year);

actual->next = element1;

element1->prev = actual;

element1->next = element2;

element2->prev = element1;

}

}

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);

int i;

for(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);

push\_double(head, name\_for\_push, author\_for\_push, year\_for\_push);

print\_names(head);

k = count(head);

printf("%d\n",k);

return 0;

}