МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

Курсовая РАБОТА

по дисциплине «Программирование»

Тема: Структуры данных. Линейные списки

Студент гр. 7381 _____ Адамов Я.В.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2017

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Адамов Я.В.

Группа 7381

Тема работы: Структуры данных. Линейные списки

Исходные данные:

- Создать 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
- void sort(MusicalComposition* head); // Сортирует список по году (по убыванию)
- Создать удобный интерфейс для работы.

Предполагаемый объем пояснительной записки:
Не менее 18 страниц.

Дата выдачи задания: 23.11.2017

Дата сдачи реферата:

Дата защиты реферата:

Студент _____ Адамов Я.В.

Преподаватель _____ Берленко Т.А.

АННОТАЦИЯ

В результате работы выполнения задания была создана программа для создания и работы с двунаправленным списком музыкальных композиций, а именно: добавление и удаление элементов в список, вывод количества и названий композиций, их сортировка. Также был создан удобный интерфейс для работы.

SUMMARY

As a result of the job, a program was created to create and work with a bidirectional list of musical compositions, namely: adding and removing elements to the list, outputting the number and names of tracks, and sorting them. Also, a convenient interface for work was created.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является закрепление знаний по работе с функциями, указателями и динамической памятью. Научиться работать с двунаправленными списками и создать функции для работы с ними: добавлять и удалять элементы списка, получать информацию о их количестве и названиях, сортировать.

1. ОПИСАНИЕ ТЕЛА ФУНКЦИИ MAIN

1.1. С помощью функции scanf считывается первоначальное количество композиций в списке.

1.2. Динамически выделяется память для массивов строк названий композиций и их авторов, а также массива лет создания этих композиций.

1.3. Вводятся эти данные, которые дальше передаются в функцию createMusicalCompositionList, которая возращает указатель на первый элемент двусвязного списка.

1.4. Пользователю предоставляется выбор дальнейших действий со списком копозиций:

- 1 - вывод количесива композиций.
- 2 - вывод названий композиций.
- 3 - добавление элемента в список.
- 4 - удаление элемента из списка.
- 5 - сортировка списка по году.
- 6 - завершение работы.

1.5. Пользователь может выполнить любое количество задач, пока не завершит работу.

1.6. С помощью функции free() очищается вся выделенная память.

2. ОПИСАНИЕ ФУНКЦИЙ ДЛЯ РАБОТЫ СО СПИСКОМ

2.1. Функция createMusicalCompositionList принимает массивы указателей на строки, содержащие имена и авторов композиций, а также массив int, содержащий года создания этих композиций. Функция вызывает функцию createMusicalCompositon столько раз, сколько нужно создать композиций. Функция создаёт список композиций и возвращает указатель на первый элемент.

2.2. Функция createMusicalComposition принимает два указателя char* на строки, содержащие нвазание и автора композиции, и один int, в котором содержится год создания композиции. Динамически выделяется память для нового элемента списка музыкальных композиций. Функция возвращает указатель на первый элемент (в данном случае он единственный).

2.3. Функция count, которая принимает указатель на первый элемент списка. С помощью цикла while перебираются все элементы до последнего, после чего функция возращает их количество.

2.4. Функция print_names, которая принимает указатель на первый элемент списка, после чего с помощью цикла while перебираются все элементы списка и выводятся их названия.

2.5. Функция push добавляет элемент в конец списка. Функция принимает указатель на новый элемент списка, после чего добавляет его в конец списка. Функция ничего не возвращает.

2.6. Функция removeEl, которая удаляет элемент. Функция принемает строку и указатель на указатель головы списка. С помощью цикла for функция проходит по всем элементам, пока не найдётся композиции, имя которой совпадает с принятой строкой. Элемент удаляется из списка, также освобождается память с помощью функции free(). Если совпадений нет, то функция ничего не делает.

2.7. Функция sort которая принимает указатель на первый элемент списка, после чего с помощью цикла for проходит по всем элементам списка. Год каждого

элемента сравнивается с годом всех впереди стоящих, если он меньше, что значения двух композиций меняются местами.

3. ЗАКЛЮЧЕНИЕ

В ходе работы была создана программа для работы с двунаправленным списком. Она может создавать и редактировать списки, а также выводить информацию о них. Были получены знания по работе со структурой данных линейными списками, закреплены знания по работе с динамической памятью и указателями.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Демидович Е. «Основы алгоритмитмизации и программирования. Язык СИ»

2. Подбельский В.С., Фомин С.С. «Курс программирования на СИ»

3. Керниган Б., Риччи Д. «Язык программирования СИ»

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

1) MAIN.C

```
#include
<stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stddef.h>
#include "api.h"

int main(){

// создание списка

int length;
printf("Создание списка композиций.\nВведите количество композиций: ");
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);

MusicalComposition* head = NULL;
if (length==0){
printf("Вы создали пустой список.\n");
} else{
for (int i=0;i<length;i++){
char name[80];
char author[80];

printf("Введите название %d-й композиции (до 80 символов): ", i+1);
fgets(name, 80, stdin);
printf("Введите автора этой композиции (до 80 символов): ");
fgets(author, 80, stdin);
printf("Введите год создания этой композиции: ");
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);
}
head = createMusicalCompositionList(names, authors, years, length);
```

```
}


// работа со списком
int rabota = 1;
int fun = 0;
char name_for_push[80];
char author_for_push[80];
int year_for_push;
char name_for_remove[80];

printf("Выберите действие, которое хотите совершить:\n");
printf("1 - вывести на экран количество композиций.\n");
printf("2 - вывести на экран названия композиций.\n");
printf("3 - добавить композицию.\n");
printf("4 - удалить композицию.\n");
printf("5 - отсортировать список по году (по убыванию).\n");
printf("6 - завершить работу программы.\n");
while (rabota){
scanf("%d\n", &fun);
switch(fun){

case 1:
if (head == NULL)
printf("Список пуст.\n");
else
printf("Количество композиций: %d\n", count(head));
break;

case 2:
if (head == NULL)
printf("Список пуст.\n");
else{
printf("Названия композиций:\n");
print_names(head);
}
break;

case 3:
printf("Чтобы добавить композицию, введите её название (до 80 символов): ");
fgets(name_for_push, 80, stdin);
(*strstr(name_for_push,"\n"))=0;

printf("Введите имя автора композиции (до 80 символов): ");
fgets(author_for_push, 80, stdin);
(*strstr(author_for_push,"\n"))=0;

printf("Введите год создания композиции: ");
```

```
fscanf(stdin, "%d\n", &year_for_push);
MusicalComposition* element_for_push =
createMusicalComposition(name_for_push, author_for_push, year_for_push);
if (head == NULL)
head = element_for_push;
else
push(head, element_for_push);
break;

case 4:
if (head == NULL)
printf("Список пуст, нечего удалять.\n");
else{
printf("Чтобы удалить композицию, введите её название: ");
fgets(name_for_remove, 80, stdin);
(*strstr(name_for_remove,"\n"))=0;
removeEl(&head, name_for_remove);
}
break;

case 5:
if (head == NULL)
printf("Список пуст, нечего сортировать.\n");
else{
sort(head);
printf("Список отсортирован.\n");
}
break;

case 6:
rabota = 0;
printf("Работа завершена.");
break;

default: printf("Для данной клавиши не придусмотренно никаких действий.\n");
}
if (rabota)
printf("Выберите ещё одно действие или завершите работу программы.\n");
}


// освобождение памяти

for (int i=0;i<length;i++){
free(names[i]);
free(authors[i]);
}
free(names);
```

```
free(authors);
free(years);

if (head!=NULL){ // если в списке нет элементов
if (head->next!=NULL){ // если в списке больше 1 элемента
head = head->next;
while(head->next!=NULL){
free (head->prev->name);
free (head->prev->author);
free (head->prev);
head = head->next;
}
}
free (head->name);
free (head->author);
free (head);
}

return 0;
}
```

2) API.H

```
#pragm
a once

typedef struct MusicalComposition{
char *name;
char *author;
int year;
struct MusicalComposition* next;
struct MusicalComposition* prev;
} MusicalComposition;

MusicalComposition* createMusicalComposition(char*, char*, int);
MusicalComposition* createMusicalCompositionList(char**, char**, int*, int);
void push(MusicalComposition*, MusicalComposition*);
void removeEl(MusicalComposition**, char*);
int count(MusicalComposition*);
void print_names(MusicalComposition*);
void sort(MusicalComposition*);
```

3) API.C

```
#include
<stdlib.h>
#include <stdio.h>
#include <string.h>
```

```
#include "api.h"

// Создание структуры MusicalComposition
MusicalComposition* createMusicalComposition(char* name, char* author, int year)
{
MusicalComposition* new_composition =
(MusicalComposition*)malloc(sizeof(MusicalComposition));
new_composition->name=(char*)malloc(81*sizeof(char));
new_composition->author=(char*)malloc(81*sizeof(char));
strncpy(new_composition->name, name, 80);
strncpy(new_composition->author, author, 80);
new_composition->year=year;
new_composition->next = NULL;
new_composition->prev = NULL;
return new_composition;
}


// Функции для работы со списком MusicalComposition
//
// Создание списка музыкальных композиций
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 *comp = head;
int i=0;
for (i=1;i<n;i++){
comp->next = createMusicalComposition(array_names[i], array_authors[i],
array_years[i]);
comp->next->prev=comp;
comp=comp->next;
}
return head;
}

// Добавление новой композиции в конец списка
void push(MusicalComposition* head, MusicalComposition* element){
MusicalComposition* comp = head;
while (comp->next!=NULL)
comp=comp->next;
comp->next=element;
element->prev=comp;
}


// Удаление элемента, у которого name равно name_for_remove
void removeEl(MusicalComposition **head, char* name_for_remove){
```

```
MusicalComposition* comp;
for (comp = *head; comp!=NULL; comp=comp->next)
if (strcmp(comp->name,name_for_remove)==0){
if (comp->prev == NULL){ // удаляется head
if (comp->next == NULL){ // в списке всего 1 элемент
*head = NULL;
} else{ // удаляется head, в списке больше 1 элемента
comp->next->prev=NULL;
*head = comp->next;
}
} else{
if (comp->next == NULL){ // удаляется последний элемент
comp->prev->next = NULL;
} else{ // остальные случаи
comp->next->prev = comp->prev;
comp->prev->next = comp->next;
}
}
free (comp->name);
free (comp->author);
free (comp);
break;
}
}

// Возвращает количество композиций
int count(MusicalComposition* head){
int count=0;
for (MusicalComposition* comp = head; comp!=NULL; comp=comp->next)
count++;
return count;
}

// Выводит названия композиций
void print_names(MusicalComposition* head){
for (MusicalComposition* comp=head; comp!=NULL; comp=comp->next)
printf("%s\n",comp->name);
}

// сортировка списка по убыванию года
void sort(MusicalComposition* head){
char *name_for_swap=(char*)malloc(81*sizeof(char));
char *author_for_swap=(char*)malloc(81*sizeof(char));
int year_for_swap;
for (MusicalComposition* comp_1 = head; comp_1!=NULL; comp_1=comp_1-
>next){
for (MusicalComposition* comp_2 = head; comp_2!=NULL; comp_2=comp_2-
>next){
```

```
if (comp_1->year > comp_2->year){
strncpy (name_for_swap, comp_1->name, 80);
strncpy (comp_1->name, comp_2->name, 80);
strncpy (comp_2->name, name_for_swap, 80);
strncpy (author_for_swap, comp_1->author, 80);
strncpy (comp_1->author, comp_2->author, 80);
strncpy (comp_2->author, author_for_swap, 80);
year_for_swap = comp_1->year;
comp_1->year = comp_2->year;
comp_2->year = year_for_swap;
}
}
}
free (name_for_swap);
free (author_for_swap);
}
```