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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Линейные списки

Студент гр. 7381 _____ Лукашев Р.С.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2017

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Лукашев Роман Сергеевич

Группа 7381

Тема работы: Линейные списки

Содержание пояснительной записки:

- Содержание
- Введение
- Описание структуры данных
- Описание функций
- Заключение
- Список использованных источников
- Приложение: Исходный код проекта

Предполагаемый объем пояснительной записки:

Не менее 11 страниц.

Дата выдачи задания: 23.11.2017

Дата сдачи реферата: 23.12.2017

Дата защиты реферата: 23.12.2017

Студент _____ Лукашев Р.С.

Преподаватель _____ Берленко Т. А.

АННОТАЦИЯ

В данной работе был создан проект на языке программирования C, который позволяет работать с набором функций, отвечающих за список музыкальных композиций. Для функционирования списка была описана структура элемента списка, созданы и описаны функции, позволяющие добавлять, удалять, сортировать и выводить имена элементов списка и их количество на консоль. Кроме того была реализована функция вставки двух элементов после каждого четного элемента списка. Была проведена работа над оптимизацией исходного кода программы для ускорения ее быстродействия и оптимального использования памяти и ресурсов. Приведено полное описание исходного кода. Приведены примеры работы программы.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы

Практика применения сложных типов (struct) в языке С. Использование их для реализации сложных структур данных. В частности, двунаправленных линейных списков. Создание API для работы со списком. Закрепление знаний об указателях (в том числе на сложные типы), динамической памяти, массивах, стандартном вводе-выводе и основных функций библиотек “stdio.h”, “stdlib.h”, “string.h”.

Формулировка задачи

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]).

  Аналогично для второго, третьего, ... n-1-го элемента массива.

! длина массивов array_names, array_authors, array_years одинаковая и равна n, это проверять не требуется.

Функция возвращает указатель на первый элемент списка.

- 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

Индивидуальное задание.

Добавить после каждого четного элемента списка еще 2 элемента (имя, автор, год — произвольные), на вход функции подаются 3 указателя на структуру.

1 ОПИСАНИЕ РАБОТЫ

1.1 ОПИСАНИЕ СТРУКТУРЫ ДАННЫХ

С помощью типа struct объявляется новый сложный тип struct MusicalComposition, экземплярами которого будет заполняться список. Задан синоним MusicalComposition данному типу с помощью оператора typedef.

```
typedef struct MusicalComposition{
char* name; // Имя композиции
char* author; // Автор композиции
int year; // Год выпуска композиции
struct MusicalComposition* prev; //Предыдущий элемент
struct MusicalComposition* next;//Следующий элемент
} MusicalComposition;
```

1.2 ОПИСАНИЕ ФУНКЦИЙ

1.2.1 Функция createMusicalComposition

Назначение:

Создание элемента списка.

Аргументы:

char* name — строка, имя музыкальной композиции.

char* author — строка, автор музыкальной композиции.

Int year — целое число, год музыкальной композиции.

Возвращаемое значение:

MusicalComposition* — указатель на экземпляр композиции.

1.2.2 Функция createMusicalCompositionList

Назначение:

Создание списка.

Аргументы:

char** array_names — массив строк с именами музыкальных композиций.

char** array_authors — массив строк с авторами композиций.

int* array_years — массив целых чисел, года выпуска каждой из композиций.

int n — количество композиций.

Возвращаемое значение:

MusicalComposition* — указатель на первый элемент списка.

1.2.3 Функция push

Назначение: вставка элемента в конец списка.

Аргументы:

MusicalComposition** head — указатель на указатель на первый элемент списка.

MusicalComposition* element — указатель на вставляемый элемент.

Возвращаемое значение:

void — функция ничего не возвращает.

1.2.4 Функция removeEl

Назначение: удаление элемента списка по заданному имени композиции.

Аргументы:

MusicalComposition** head — указатель на указатель на первый элемент списка.

char* name_for_remove — строка, имя удаляемой композиции.

Возвращаемое значение:

void — функция ничего не возвращает.

1.2.5 Функция count.

Назначение: подсчет количества элементов в списке.

Аргументы:

MusicalComposition* head — указатель на первый элемент списка.

Возвращаемое значение:

int — целое число, количество композиций в списке.

1.2.6 Функция print_names.

Назначение: выводит список имен музыкальных композиций в списке.

Аргументы:

MusicalComposition* head — указатель на первый элемент списка.

Возвращаемое значение:

void — функция ничего не возвращает.

1.2.7 Функция free_list.

Назначение: очистка динамически выделенной памяти для списка, удаление списка.

Аргументы:

MusicalComposition** list — указатель на указатель на первый элемент удаляемого списка.

Возвращаемое значение:

void — функция ничего не возвращает.

1.2.8 Функция insert_task.

Индивидуальное задание

Аргументы:

MusicalComposition* head — указатель на первый элемент списка

MusicalComposition* item1 — указатель на первый элемент для вставки

MusicalComposition* item2 — указатель на второй элемент для вставки

Возвращаемое значение:

void — функция ничего не возвращает.

2. ДЕМОНСТРАЦИЯ РАБОТЫ

2.1. Тест №1

```
Length of Initial List:
> 0
Type in command:
0: exit
1: push
2: print names
3: remove element by name
4: insert task
5: count number of musical compositions in list
> 1
Name:
> Sober
Author:
> James Abrams
Year:
> 2004
Type in command:
0: exit
1: push
2: print names
3: remove element by name
4: insert task
5: count number of musical compositions in list
> 2
Names of Musical Compositions currently in list:
Sober

Type in command:
0: exit
1: push
2: print names
3: remove element by name
4: insert task
5: count number of musical compositions in list
> 0
```

Рис. 1. Демонстрация работы первого теста

Вставка элемента в пустой Список.

1.2. Тест №2

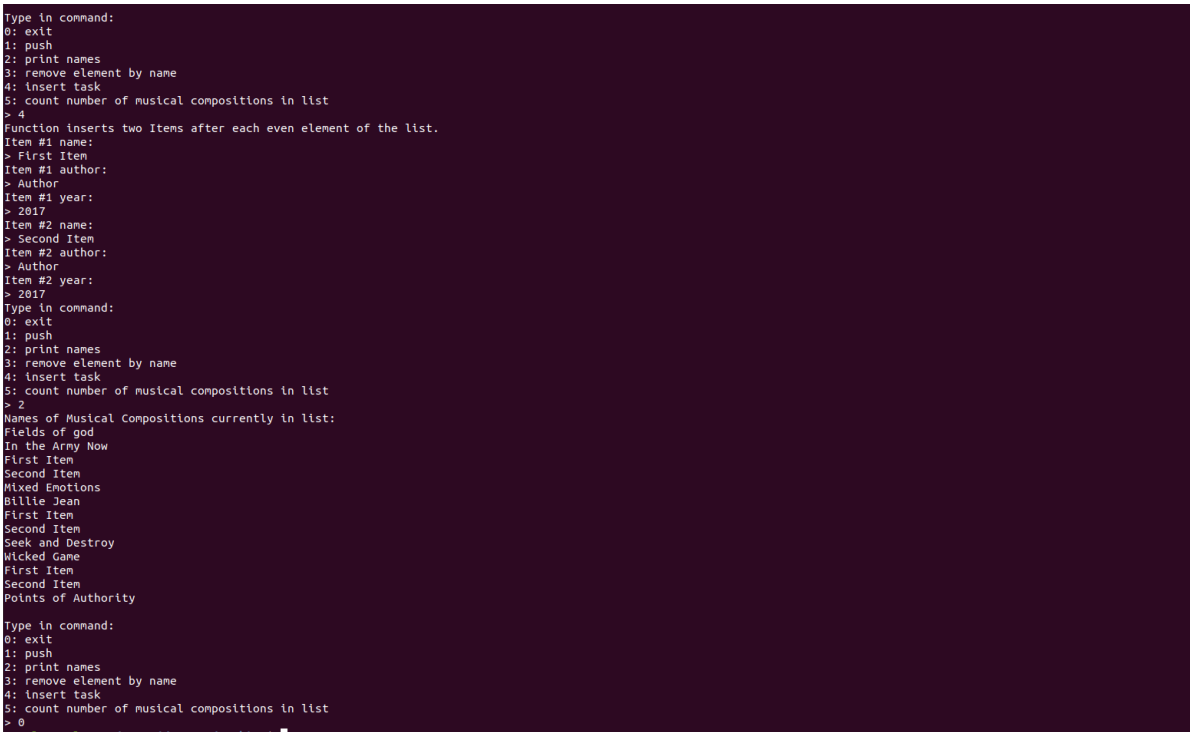

Рис. 2. Демонстрация работы второго теста

Демострация работы функции insert_task индивидуального задания.

ЗАКЛЮЧЕНИЕ

Было освоено и закреплено на практике написание программ с применением структур в качестве элементов сложных типов представления данных, таких, как двунаправленные линейные списки, а также написание API для работы с ними, в частности, функции вставки, удаления, подсчета и вывода полей элементов списка. Повторена работа с проектом в целом: разбиение его на заголовки (header files) и файлы кода (source files), написание makefile для сборки программы. Усовершенствованы навыки работы с указателями, динамической памятью, массивами и функциями стандартных библиотек.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

ПРИЛОЖЕНИЕ

ИСХОДНЫЙ КОД ПРОЕКТА

1. main.c

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

int main(){
   MusicalComposition* List = NULL;
   int input;
   int length;
   char c;

   printf("Length of Initial List:\n> ");
   scanf("%d", &length);
   c = getchar();
   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 name0[80];
      char author0[80];

        printf("Name:\n> ");
      fgets(name0, 80, stdin);

        printf("Author:\n> ");
      fgets(author0, 80, stdin);

        printf("Year:\n> ");
      fscanf(stdin, "%d", &years[i]);

        c = getchar();

      (*strstr(name0,"\n"))=0;
      (*strstr(author0,"\n"))=0;

      names[i] = (char*)malloc(sizeof(char*) * (strlen(name0)+1));
```

```
        authors[i] = (char*)malloc(sizeof(char*) * (strlen(author0)+1));

        strcpy(names[i], name0);
        strcpy(authors[i], author0);

    }
    List = createMusicalCompositionList(names, authors, years, length);

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

    free(names);
    free(authors);
    free(years);

    while(1){
        input = -1;
        printf("Type in command:\n0: exit\n1: push\n2: print names\n3: remove
element by name\n4: insert task\n5: count number of musical compositions in list\n>
");

        scanf("%d", &input);
        while(c = getchar() != '\n') {}

        switch (input){
                case 0:{
                free_list(&List);
                return 0;
                }

                case 1:{

                char* name = (char*)malloc(80*sizeof(char));
                char* author = (char*)malloc(80*sizeof(char));
                int year;

                printf("Name:\n> ");
                fgets(name, 80, stdin);

                printf("Author:\n> ");
               fgets(author, 80, stdin);
```

```
                printf("Year:\n> ");
              fscanf(stdin, "%d", &year);

              c = getchar();

                (*strstr(name,"\n"))=0;
              (*strstr(author,"\n"))=0;

                MusicalComposition* insert =
createMusicalComposition(name,author,year);

                push(&List, insert);

                free(name);
                free(author);
                break;
                }

                case 2:{
                printf("Names of Musical Compositions currently in list:\n");
                print_names(List);
                printf("\n");
                break;
                }

                case 3:{
                printf("Name of composition to delete:\n> ");

                char* name = (char*)malloc(80*sizeof(char));

                fgets(name, 80, stdin);
            (*strstr(name,"\n"))=0;

                removeEl(&List, name);

                free(name);
                break;
                }

                case 4:{
                printf("Function inserts two Items after each even element of the
list.\n");

                char* name = (char*)malloc(80*sizeof(char));
```

```
            char* author = (char*)malloc(80*sizeof(char));
            int year;

               char* name1 = (char*)malloc(80*sizeof(char));
            char* author1 = (char*)malloc(80*sizeof(char));
            int year1;

               printf("Item #1 name:\n> ");
               fgets(name, 80, stdin);

               printf("Item #1 author:\n> ");
            fgets(author, 80, stdin);

               printf("Item #1 year:\n> ");
            fscanf(stdin, "%d", &year);

            c = getchar();

               (*strstr(name,"\n"))=0;
            (*strstr(author,"\n"))=0;

               printf("Item #2 name:\n> ");
               fgets(name1, 80, stdin);

               printf("Item #2 author:\n> ");
            fgets(author1, 80, stdin);

               printf("Item #2 year:\n> ");
            fscanf(stdin, "%d", &year1);

               c = getchar();

               (*strstr(name1,"\n"))=0;
            (*strstr(author1,"\n"))=0;

               MusicalComposition* item1 = createMusicalComposition(name, author,
year);
               MusicalComposition* item2 = createMusicalComposition(name1,
author1, year1);

               insert_task(List, item1, item2);

               free_list(&item1);
               free_list(&item2);
```

```
                free(name);
                free(name1);
                free(author);
                free(author1);

                break;
                }

                case 5:{
                printf("Number of items in list: %d\n", count(List));
                break;
                }
        }
   }
return 0;
}
```

2. MusicalComposition.c

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

// Создание структуры MusicalComposition

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


MusicalComposition* createMusicalCompositionList(char** array_names, char**
array_authors, int* array_years, int n){
```

```
        if(!n) return NULL;
      MusicalComposition* List = createMusicalComposition(*array_names,
*array_authors, *array_years);
        for(int i = 1; i < n; i++){
      MusicalComposition* Next = createMusicalComposition(*(array_names+i),
*(array_authors+i), *(array_years+i));
        push(&List, Next);
        }
return List;
}

void push(MusicalComposition** head, MusicalComposition* element){
        if(!(*head)){
         *head = element;
         return;
        }
        MusicalComposition* temp = *head;
        while(temp->next){
         temp = temp->next;
        }
        element->prev = temp;
        element->next = NULL;
        temp->next = element;
}

void removeEl(MusicalComposition** head, char* name_for_remove){
  MusicalComposition* temp = *head;
  while((*head)){
        if(!strcmp((*head)->name, name_for_remove)){
         if(!((*head)->prev) && !((*head)->next)){ free_list(head); return; }
         if(!((*head)->prev)){
          *head = (*head)->next; ((*head)->prev)->next = NULL; free_list(&((*head)-
>prev)); (*head)->prev = NULL; return;
         }
         if(!((*head)->next)){
          (*head)->prev->next = NULL; free_list(head); *head = temp; return;
         }
         (*head)->prev->next = (*head)->next;
         (*head)->next->prev = (*head)->prev;
         (*head)->next = NULL;
         free_list(head);
         *head = temp;
         return;
        }
```

```
  (*head) = (*head)->next;
  }
}

int count(MusicalComposition* head){
        int i = 0;
        while(head){
         i++;
         head = head->next;
        }
        return i;
}

void print_names(MusicalComposition* head){
while(head){
  printf("%s\n", head->name);
  head = head->next;
 }
}

void free_list(MusicalComposition** list){
        if(!(*list)) return;
        while(((*list)->next)){
        *list = (*list)->next;
        free(((*list)->prev)->name);
       free(((*list)->prev)->author);
        free((*list)->prev);
        (*list)->prev = NULL;
        }
       free((*list)->name);
       free((*list)->author);
       free((*list));
        (*list) = NULL;
}

void insert_task(MusicalComposition* head, MusicalComposition* item1,
MusicalComposition* item2){
        int i = 0;
        while(head){
         if(i == 1){
          i = -1;
          MusicalComposition* insert1 = createMusicalComposition(item1->name,
item1->author, item1->year);
          insert1->prev = head;
```

```
            MusicalComposition* insert2 = createMusicalComposition(item2->name,
item2->author, item2->year);
            insert1->next = insert2;
            insert2->prev = insert1;
            insert2->next = head->next;
            if(head->next) head->next->prev = insert2;
            head->next = insert1;
            head = head->next->next;
           }
          i++;
          head = head->next;
         }
}
```

3. MusicalComposition.h

```
// Описание структуры MusicalComposition
typedef struct MusicalComposition{
 char* name;
 char* author;
 int year;
 struct MusicalComposition* prev;
 struct MusicalComposition* next;
} MusicalComposition;

// Создание структуры MusicalComposition

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

// Функции для работы со списком MusicalComposition

MusicalComposition* createMusicalCompositionList(char** array_names, char**
array_authors, int* array_years, int n);

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

void removeEl(MusicalComposition** head, char* name_for_remove);

int count(MusicalComposition* head);

void print_names(MusicalComposition* head);

void free_list(MusicalComposition** list);
```

```
void insert_task(MusicalComposition* head, MusicalComposition* item1,
MusicalComposition* item2);
```

4. Makefile

```
all: main.o MusicalComposition.o
	gcc MusicalComposition.o main.o
main.o: main.c
	gcc -c main.c
MusicalComposition.o: MusicalComposition.h MusicalComposition.c
	gcc -c MusicalComposition.c
clean:
	rm main.o MusicalComposition.o
```