МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МОЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Структура данных и линейные списки

Студентка гр. 7381 _____ Кушкоева А.О.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2017

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студентка Кушкоева А.О.

Группа 7381

Тема работы: Структура данных и линейные списки

Исходные данные:

Двунаправленный список музыкальных композиций MusicalComposition и api (application programming interface - в данном случае набор функций) для работы со списком.

MusicalComposition* swap(MusicalComposition* head); //меняет местами соседние четные и нечетные элементы местами(1-ый и 2-ой, 3-ий и 4-ый, …);

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 23.11.2017

Дата сдачи реферата:

Дата защиты реферата:

Студентка _____ Кушкоева А.О.

Преподаватель _____ Берленко Т.А.

АННОТАЦИЯ

В ходе выполнения лабораторной работы была создана программа, которая обрабатывает список музыкальных композиций. Для этого были написаны такие функции как: удаление элемента, добавление элемента в конец списка, перестановка соседних четных и нечетных элементов, подсчет количества элементов, вывод всех элементов списка.

SUMMARY

During the execution of the laboratory work was created by a program that processes a list of songs. This was written such functions as: delete item, add item to the end of the list, swapping adjacent even and odd elements, counting the number of items, withdrawal of all elements of the list.

СОДЕРЖАНИЕ

Оглавление

ВВЕДЕНИЕ

В данной работе необходимо освоить структуры данных, а также работу двунаправленных списков и функции для работы с ними, закрепляя полученные знания созданием программы на языке Си.

1. ФУНКЦИИ ДЛЯ РАБОТЫ СО СПИСКОМ

1.1. Функция создания списка музыкальных композиций MusicalCompositionList

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

Функция возвращает указатель на первый элемент списка.

1.2. Функция добавления элемента element в конец списка

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

В функцию подается указатель на голову списка и элемент, который требуется добавить в список. Указатель перемещается по списку, пока не доходит до последнего элемента(не NULL), который связывается с помощью указателей с добавляемым элементом.

1.3. Функция удаления элемента element списка, у которого значение name равно значению name_for_remove

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

В функцию подается указатель на голову списка и указатель на название композиции, которую необходимо удалить. Указатель перемещается по списку, сверяя название каждой композиции с переданной строкой, пока не находит совпадения. В этом случае с помощью указателей связываются следующая и предыдущая композиции. А после происходит удаление элемента.

1.4. Функция перестановки соседних четных и нечетных элементов списка

MusicalComposition* swap(MusicalComposition* head);

В функцию подается указатель на голову списка. Функция проверяет список на наличие хотя бы больше двух элементов, если условие неверно, то возвращается указатель на голову списка. Если же условие верно, то с помощью указателей на следующий и предыдущий элемент меняются местами соседние четные и нечетные элементы. После перестановки всех элементов возвращается указатель на голову списка.

1.5. Функция, которая возвращает количество элементов списка

int count(MusicalComposition* head);

В функцию подается указатель на голову списка. Пока указатель на следующий элемент не NULL, происходит счет всех элементов с помощью указателя, который перемещается по списку. Функция возвращает число, которое равно количеству элементам списка.

1.6. Функция, которая выводит названия композиций

void print_names(MusicalComposition* head);

В функцию подается указатель на голову списка. Перебирая все элементы, функция печатает название каждой композиции.

2. ФУНКЦИЯ MAIN

- Вводится количество композиций.
- Выделяется динамическая память.
- Считываются название, автор, год композиций.
- Создается двунаправленный список по полученным данным.
- Обработка списка:
    1. Добавление элемента;
    2. Вывод количества элементов;
    3. Удаление элемента;
    4. Перестановка соседних четных и нечетных элементов;
    5. Вывод количества элементов;
    6. Вывод названий композиций;
- Очищение выделенной динамической памяти.

3. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

3.1. Поступающие данные

```
nastya@nastya-VirtualBox:~/kr$ make
gcc -c main.c
gcc main.o
nastya@nastya-VirtualBox:~/kr$ ./a.out
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
```

Рис.1

3.2. Результат без функции swap

```
--------------------------
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
```

Рис.2

3.3. Результат с функцией swap

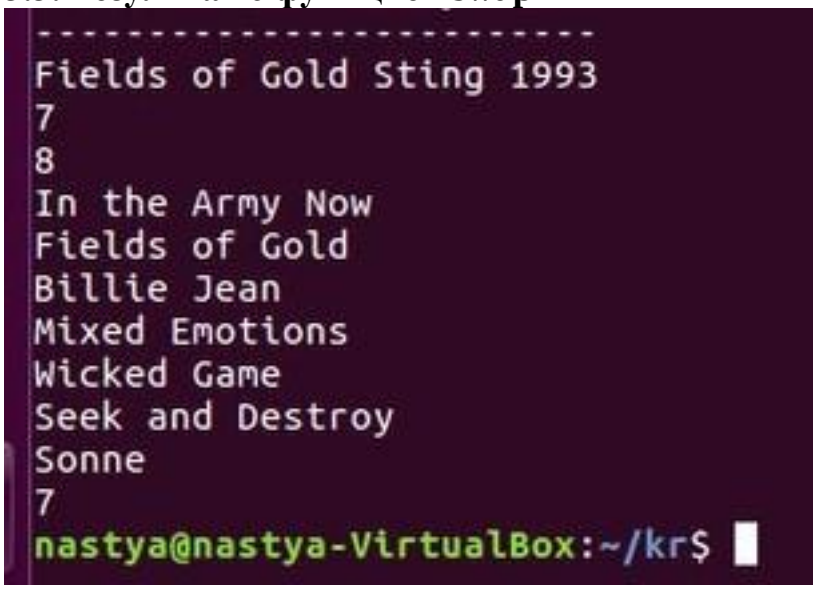

Рис.3

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были освоены структуры данных, двунаправленные линейные списки и работа с ними. Полученные знания были закреплены на практике.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Б.Керниган, Д.Риччи «Язык программирования Си»

К.Паппас, У.Мэррей «Руководство программиста»

ПРИЛОЖЕНИЕ

Makefile

```
all: main.o
	gcc main.o
main.o: main.c functions.h
	gcc -c main.c
clean:
	rm -rf *.o
```

functions.c

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

MusicalComposition* createMusicalComposition(char* name, char* author,int year)
	{
	MusicalComposition* song=(MusicalComposition*)malloc(sizeof(MusicalComposition));
	song->name=name;
	song->author=author;
	song->year=year;
	song->next=NULL;
	song->previous=NULL;
	return song;
	}
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* list=head;
        int i;
        for(i=1; i<n; i++)
                {
                list-
>next=createMusicalComposition(array_names[i],array_authors[i],array_years[i]);
                list->next->previous=list;
                list=list->next;
                }
        return head;
        }

void push(MusicalComposition* head, MusicalComposition* element)
        {
        MusicalComposition* list=head;
        while (list->next!=NULL)
    list=list->next;
        list->next=element;
        element->previous=list;
        }

void removeEl(MusicalComposition* head, char* name_for_remove)
        {
        MusicalComposition* list=head;
        while(list->next!=NULL)
        {
                if(strcmp(name_for_remove, list->name)==0)
                {
                list->next->previous=list->previous;
                list->previous->next=list->next;
                }
        list=list->next;
        }
        }

MusicalComposition* swap(MusicalComposition* head)
        {
        MusicalComposition* list=head;
        if(head!=NULL)
        {
        if(list->next==NULL && list->previous==NULL)
        return head;
         list->next->previous=list->previous;
           list->previous=list->next;
```

```
            list->next=list->next->next;
            list->previous->next=list;
            if(list->next!=NULL)
                  {
                  list->next->previous=list;
            list=list->next;
                  }
          head=head->previous;
          }
          while(list->next!=NULL && list!=NULL)
                   {
                   list->previous->next=list->next;
                   list->next->previous=list->previous;
                   list->previous=list->next;
                   list->next=list->next->next;
                   list->previous->next=list;
                   if(list->next!=NULL)
                            {
                            list->next->previous=list;
                   list=list->next;
                            }
                   else
                   break;
                   }

          return head;
          }

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


void print_names(MusicalComposition* head)
          {
          MusicalComposition* list=head;
```

```
        while(list!=NULL)
        {
        printf("%s\n", list->name);
        list=list->next;
        }
        }
```

main.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stddef.h>
#include "functions.h"
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);

   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("------------------------\n");
    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);
    head=swap(head);
    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;
}
```