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

Студентка гр.7381 _____ Процветкина А. В.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2017

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Процветкина А. В.

Группа 7381

Тема работы: Структуры данных, линейные списки

Исходные данные:

Создать двунаправленный список музыкальных композиций MusicalComposition и **api** (***a****pplication* ***p****rogramming* ***i****nterface - в данном случае набор функций*) для работы со списком.

Создать функцию для удаления всех элементов списка, год которых нацело делится на 4.

Создать удобный интерфейс для работы программы.

Предполагаемый объем пояснительной
записки: Не менее 10 страниц.

Дата выдачи задания: 23.11.2017

Дата сдачи реферата:

Дата защиты реферата:

Студент _____ Процветкина А. В.

Преподаватель _____ Берленко Т. А.

АННОТАЦИЯ

В результате выполнения курсовой работы была создана программа на языке СИ. Она позволяет работать со списком и с функциями для него, такими как: удаление элемента, добавление элемента в конец списка, подсчет количества элементов, удаление всех элементов, год которых нацело делится на 4, а также вывод самого списка. Создан приятный для работы интерфейс программы.

SUMMARY

As a result of this course work, a C program was created. It allows one to work with the list and functions for it such as: removing an element, adding an element to the end of the list, counting the number of elements, deleting all the elements in which year is divisible by 4, and also list output. A user-friendly interface for working with the program was created.

СОДЕРЖАНИЕ

Оглавление

Введение

В данной работе необходимо научиться работать с двунаправленными списками, уметь создавать функции для работы с ними. Для этого была создана программа на языке СИ, наглядно показывающая данные умения.

Функции программы

1. ФУНКЦИЯ MAIN.C

1.1. Вводится количество элементов списка.

1.2. Динамически выделяется память под массивы для хранения названий композиций, авторов и дат создания.

1.3. Заполняются данные массива

1.4. Создается двунаправленный список по данным из массива.

1.5. Пользователю предлагают многократный выбор дальнейших действий

- 1.5.1. №1 – Вывести количество элементов списка.
- 1.5.2. №2 – Вывести названия композиций.
- 1.5.3. №3 – Добавить элемент в конец списка.
- 1.5.4. №4 – Удалить элемент с данным названием.
- 1.5.5. №5 – Удалить все элементы, поле год которых нацело делится на 4.
- 1.5.6. №9 – Закончить работу программы.

1.6. Освобождается вся выделенная динамически память.

2. ФУНКЦИИ ДЛЯ РАБОТЫ СОСПИСКОМ

2.1. MusicalComposition* createMusicalComposition(char* name,char* author,int year)

Данной функции передают три параметра: название (char* name), имя автора (char* author) и год создания композиции (int year); которые записываются в элемент после выделения памяти для структуры MusicalComposition. Функция возвращает указатель на созданный элемент списка.

2.2. void push(MusicalComposition** head_p, MusicalComposition*element)

Данной функции передается указатель на голову списка (указатель на указатель на первый элемент списка) и элемент, который необходимо добавить (MusicalComposition* element). Функция проходит по списку, пока не дойдет до последнего элемента, а затем соединяет последний и новый элементы списка посредством указателей, если же список был пуст, то указатель на голову теперь указывает на данный элемент. Функция ничего не возвращает.

2.3. MusicalComposition* createMusicalCompositionList(char** array_names, char** array_authors, int* array_years, intn)

Функция получает количество существующих композиций и указатели на три массива: массив названий (char** array_names), массив авторов (char** array_authors) и массив годов созданий (int* array_years); и формирует линейный двунаправленный список. Функция возвращает указатель на первый элемент списка.

2.4. void removeEl(MusicalComposition** head_p, char* name_for_remove)

Функция получает указатель на голову списка (указатель на указатель на первый элемент списка) и название композиции (char* name_for_remove), которую нужно удалить. Проходя по списку, она сравнивает название композиции с

переданной строкой. При совпадении значений функция удаляет элемент определенным способом в зависимости от положения элемента. Первый случай, если элемент единственный в списке. Значение указателя на голову становится равным NULL, память под элемент списка очищается. Второй случай, если элемент последний в списке. Указатель на следующий элемент списка предыдущего элемента становится равным NULL. Память под элемент списка очищается. Третий случай, если элемент головной. Указатель на голову станет ссылаться на следующий элемент, а память под первый элемент высвободится. И последний случай для любых других ситуаций: когда существует и предыдущий и последующий элементы списка. Предыдущий элемент теперь ссылается на следующий (после удаляемого элемента), а следующий на предыдущий. Память высвобождается. Функция ничего не возвращает.

2.5. int count(MusicalComposition*head)

Функция получает указатель на первый элемент списка (MusicalComposition* head), и, пробегая весь список, подсчитывает количество элементов в нем. Функция возвращает количество элементов списка.

2.6. void print_names(MusicalComposition*head)

Данной функции передается указатель на первый элемент списка (MusicalComposition* head). Переходя от одного элемента списка к другому, функция печатает названия каждой композиции. Функция ничего не возвращает.

2.7. void remove_div(MusicalComposition ** head_p)

Функция получает указатель на голову списка (указатель на указатель на первый элемент списка).

Пробегая весь список, функция проверяет, удовлетворяет ли год композиции условию удаления, и в противном случае удаляет этот элемент (аналогично обычному удалению, описанному в п.2.4). Функция ничего не возвращает.

Заключение

В ходе выполнения данной курсовой работы был создан двунаправленный линейный список, функции для работы с ним и удобный интерфейс для пользования. Были получены необходимые навыки для работы со списками и функциями для них. Также были закреплены знания по выделению и очищению динамической памяти.

Список использованных источников

- Б. Керниган, Д. Ритчи «Язык программирования Си»
- Подбельский В. В., Фомин С. С. «Курс программирования на Си: учебник»

Приложение А
Исходный код программы

- Файл “main.c”

```
#include <stdlib.h>
#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* createMusicalCompositionList(char** array_names, char**
array_authors, int* array_years, int n);

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

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

int count(MusicalComposition* head);

void print_names(MusicalComposition* head);

void remove_div(MusicalComposition** head_p);

int main(){
  int length;

   char name_for_push[80];
   char author_for_push[80];
   int year_for_push;
   char name_for_remove[80];

    char name[80];
    char author[80];
    MusicalComposition* head = NULL;
        MusicalComposition** main_head = NULL;
    printf("Input the number of elements the List will consist of\n");
    scanf("%d", &length);

    char** names = (char**)malloc(sizeof(char*)*(length));
```

```
    char** authors = (char**)malloc(sizeof(char*)*(length));
    int* years = (int*)malloc(sizeof(int)*length);

    if (length == 0)
      printf("Well done! An empty list!\n");
    else {
    for (int i=0; i < length; i++){
        getchar();
        printf("Input the title of the composition\n");
        fgets(name, 80, stdin);

        printf("Input its author\n");
        fgets(author, 80, stdin);

        printf("Input the year of its release\n");
        scanf("%d", &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);
}
         main_head = &head;
    int choice, exit_flag = 0;

    do {
    printf("Choose what you're willing to do, Sir:\n");
    printf("1 to print the number of the compositions\n");
    printf("2 to print all the compositions\n");
    printf("3 to add a composition\n");
    printf("4 to remove a composition by its title \n");
    printf("5 to remove all the compositions which year of release is divisible by
4\n");
    printf("9 to quit\n");

    scanf("%d", &choice);

    switch(choice){

      case 1:

          printf("%d\n", count(head));
          break;
```

```
    case 2:

        if (length == 0)
          printf("Nothing to print! Your List is empty :c \n");
        else
          print_names(head);
        break;

    case 3:
        getchar();
        printf("Input the title of the composition (up to 80 characters)\n");
        fgets(name_for_push, 80, stdin);

        printf("Input its author\n");
        fgets(author_for_push, 80, stdin);

        printf("Input the year of its release\n");
        scanf("%d", &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);
        push(head, element_for_push);
        break;

    case 4:
        getchar();
        printf("Input the title of the composition to remove it:\n");
        fgets(name_for_remove, 80, stdin);
        (*strstr(name_for_remove, "\n")) = 0;
        removeEl(main_head, name_for_remove);
        break;

    case 5:

        remove_div(main_head);
        break;

    case 9:

        printf("Session finished successfully!\n");
        exit_flag = 1;
        break;

    default:
        printf("Wrong input! Have another try!\n");
    }
  }
```

```
    while(!exit_flag);

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

    for (; head != NULL; head = head->next){
         free(head->prev);
         if (head->next == NULL)
          free(head);
      }
    return 0;
}

MusicalComposition* createMusicalComposition(char* name, char* author,int year)
{
    MusicalComposition* Musical_Composition = (MusicalComposition*)
malloc(sizeof(MusicalComposition));
    Musical_Composition->name = name;
    Musical_Composition->author = author;
    Musical_Composition->year = year;
    Musical_Composition->next = NULL;
    Musical_Composition->prev = NULL;
    return Musical_Composition;
}

MusicalComposition* createMusicalCompositionList(char** array_names, char**
array_authors, int* array_years, int n){
    int i;
    MusicalComposition* current, *prev = NULL;
    MusicalComposition* head = NULL;

  for(i = 0; i < n; i++){
    MusicalComposition* current = (MusicalComposition*)
malloc(sizeof(MusicalComposition));
    if (head == NULL)
        head = current;
    else
        prev->next = current;

    current->next = NULL;
    current->name = array_names[i];
    current->author = array_authors[i];
    current->year = array_years[i];
    current->prev = prev;
    prev = current;
  }
```

```
  return head;
}

void push(MusicalComposition* head, MusicalComposition* element){
  if (head == NULL)
    head = element;
  else {
    for(;head->next != NULL; head = head->next);
    head->next = element;
    }
}

void removeEl(MusicalComposition** head_p, char* name_for_remove){
  MusicalComposition *head = * head_p;
  for(;head != NULL; head = head->next)
    if (strcmp(head->name,name_for_remove) == 0){
      if (head->next == NULL && head->prev == NULL) {// case of only 1 element
existing
        free(head);
                *head_p = NULL;
        }
      else if (head->next == NULL) {// case of removing the last element
        head = head->prev;
        free(head->next);
        head->next = NULL;
      }
      else if (head->prev == NULL){ // case of removing the head
        *head_p = head->next;
        head = head->next;
        free(head->prev);
                head->prev = NULL;
        }
      else {                    // anything else
      head->prev->next = head->next;
      head->next->prev = head->prev;
      free(head);
      }
      return;
    }
}

int count(MusicalComposition* head){
    int i;

     if(head == NULL)
      return 0;

    for(i = 1; head->next != NULL; head = head->next)
        i++;
    return i;
}
```

```
void print_names(MusicalComposition* head){
        if (head == NULL)
                printf("Nothing to print! Your list is empty.\n");
    for(; head != NULL; head = head->next)
        puts(head->name);
}

void remove_div(MusicalComposition** head_p){

   MusicalComposition *head = * head_p;
  for(;head != NULL; head = head->next)
    if (head->year % 4 == 0){
      if (head->next == NULL && head->prev == NULL) {// case of only 1 element
existing
        free(head);
                *head_p = NULL;
        }
      else if (head->next == NULL) {// case of removing the last element
       head = head->prev;
       free(head->next);
       head->next = NULL;
      }
      else if (head->prev == NULL){ // case of removing the head
       *head_p = head->next;
       head = head->next;
       free(head->prev);
                head->prev = NULL;
       }
      else {                    // anything else
      head->prev->next = head->next;
      head->next->prev = head->prev;
      free(head);
      }
    }
}
```