Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

КУРСОВАЯ РАБОТА

по курсу
"Информатика"
"Архитектура ЭВМ, системное программное обеспечение"
II семестр
Задание 8 "Линейные списки"

Студент: Пашкевич А. Р.
Группа: 08-107, № по списку 10

Руководитель: Ридли М. К.
Ридли А. Н.

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2017

АННОТАЦИЯ

В данном документе описывается программа, написанная в соответствии с постановкой задачи на курсовое проектирование по теме "Линейные списки" по дисциплине "Архитектура ЭВМ, системное программное обеспечение". Данная программа предназначена для обработки линейного списка заданной организации. Результаты тестирования доказывают, что программа правильно выполняет все операции по обработке входных данных и формирования выходных данных.

1. ПОСТАНОВКА ЗАДАЧИ

Составить и отладить программу на языке Си для обработки линейного списка заданной организации с отображением на динамические структуры. Навигацию по списку следует реализовать с применением итераторов. Тип элементов списка — целый. Предусмотреть выполнение одного нестандартного и четырех стандартных действий:

1. Печать списка;
2. Вставка нового элемента в список;
3. Удаление элемента списка;
4. Подсчет длины списка;
5. **Удалить каждый k-ый элемент списка**

2. ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

Данная программа осуществляет обработку однонаправленного линейного списка с применением итераторов.

3. ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММЕ

Необходимое программное и аппаратное обеспечение: компилятор gcc

Операционная система: GNU/Linux, UNIX, MS Windows

Язык: Си

Система программирования: Си

Способ вызова и загрузки: bash

4. ОРГАНИЗАЦИЯ ВХОДНЫХ И ВЫХОДНЫХ ДАННЫХ

Входные данные:

На стандартный ввод программе подаются команды пяти типов:

1. — печать списка;

2. — вставка перед указанным элементом элемента с введенным значением

3. — удаление указанного элемента из списка

4. — вывод длины списка

5. — выполнение заданного вариантом действия — удаление каждого k-того элемента списка. Вводится значение k.

Выходные данные:

После чтения и выполнения каждой команды программа выводит результат операции.

Вывод всех элементов списка в формате 25 -> 5 -> NULL.

Выводит длину списка.

При удалении каждого k-того элемента выводит значения удаленных элементов и печатает элементы списка

5. ВНУТРЕННЕЕ ПРЕДСТАВЛЕНИЕ ДАННЫХ

```
typedef struct Elem elem_t;
struct Elem {
	int data;
	elem_t *next;
};

typedef struct {
	elem_t *head;
	int size;
} List;

typedef struct {
	elem_t *node;
} Iterator;
```

3. ОРГАНИЗАЦИЯ ИСХОДНОГО КОДА

Программа поделена на 3 части: заголовочный файл с описанием интерфейса списка, файл с реализаций списка и основной файл, в котором производится обработка команд пользователя и вызов соответствующих функций.

Таблица 1: Структура файлов проекта:

/include	
list.h	Заголовочный файл с описанием структуры элементов списка, итераторов и линейного списка
/src	
main.c	Инициация структуры **list** Вывод меню. Обработка команд пользователя, вызов соответствующих функций, функция удаления k-того элемента
list.c	Файл с кодом для работы со структурой elem, list, iterator.
makefile	Используется для управления сборкой проекта. Служит для сборки проекта и очистки каталогов проекта от образованных в процессе сборки файлов.

4. КАК ЭТО РАБОТАЕТ

Программа создает пустой список, после чего выводится меню действий состоящее из следующих пунктов:

Please, enter your choice:
1 --- to print list.
2 --- to insert into the list an element before some element (index)
3 --- to delete an element from the list (index)
4 --- length of list
5 --- remove each k-th element, write some like 2
6 --- exit.

void **function**(List *, int)
удаляет каждый k-тый элемент из списка

int **menu**()
возвращает какой элемент меню выбран

List* **CreateList**()
Инициализация списка

Iterator* **first**(List *)
Создание итератора из ссылочной компоненты первого элемента (head) списка

Iterator* **last**(List *)
создает итератор из последнего элемента списка

bool **is_empty**(const List *)
проверяет не пустой ли список

Iterator* **find_item_by_index**(List *, const int)
ищет элемент списка по заданному индексу и возвращает итератор на него

void **print_list**(List *)
печать элементов списка

void **destroy_list**(List *)
уничтожает список, перебирая его элементы и удаляя их

void **insert**(List *, const int, int)
вставляет элемент в список перед заданным

void **delete**(List *, const int)
удаляет элемент на который указывает индекс

```
Please, enter your choice:
1 --- to print list.
2 --- to insert into the list an element before some element (index)
3 --- to delete an element from the list (index)
4 --- length of list
5 --- remove each k-th element, write some like 2
6 --- exit.

Select action: 2
add: 1
25

The list is: (1)
25 -> NULL

...

The list is: (12)
25 -> 3 -> 12 -> 63 -> 8 -> 36 -> 10 -> 45 -> 6 -> 1 -> 77 -> 25 -> NULL

Please, enter your choice:
1 --- to print list.
2 --- to insert into the list an element before some element (index)
3 --- to delete an element from the list (index)
4 --- length of list
5 --- remove each k-th element, write some like 2
6 --- exit.

Select action: 5
```

```
function
Enter number of k-th element: 3
deleted = 12
deleted = 36
deleted = 6
deleted = 25

The list is: (8)
25 -> 3 -> 63 -> 8 -> 10 -> 45 -> 1 -> 77 -> NULL
```

ЗАКЛЮЧЕНИЕ

Данная программа, составлена в соответствии с постановкой задачи на курсовое проектирование по теме "Линейные списки" по дисциплине "Архитектура ЭВМ, системное программное обеспечение". При написании программы использованы методические указания по курсовому проектированию по дисциплине. Тестирование программы подтвердило, что программа правильно выполняет обработку данных и выдаёт верные результаты. Всё это свидетельствует о работоспособности программы и позволяет сделать вывод о пригодности программы к решению практических задач по обработке линейных списков.

List.h

```
#ifndef __LIST_H__
#define __LIST_H__

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

typedef struct Elem elem_t;
struct Elem {
        int data;
    elem_t *next;
};

typedef struct {
    elem_t *head;
    int size;
} List;

typedef struct {
    elem_t *node;
} Iterator;

List* CreateList();
Iterator* first(List *);
Iterator* last(List *);
bool is_empty(const List *);
Iterator* find_item_by_index(List *, const int);
void print_list(List *);
void destroy_list(List *);
void insert(List *, const int, int);
void delete(List *, const int);

#endif
```

List.c

```
#include "list.h"

// выделяет память под дескриптор списка, создат пустой список
List* CreateList() {
    List *ls = (List*)malloc(sizeof(List));
    ls->head = (elem_t*)malloc(sizeof(elem_t));
    ls->head->next = NULL;
    ls->head->data = 0;
    ls->size = 0;
    return ls;
}

// Создание итератора из ссылочной компоненты list->head списка
Iterator *first(List *ls) {
        Iterator *it = (Iterator*)malloc(sizeof(Iterator));
    it->node = ls->head;
    return it;
}

// Функция Last создает итератор из последнего элемента списка
Iterator* last(List *ls){
        elem_t *p = ls->head;
        while (p->next) {
                p = p->next;
        }
        Iterator *it = (Iterator*)malloc(sizeof(Iterator));
    it->node = p;
    return it;
}

// Список пуст???
bool is_empty(const List * ls) {
    return ls->size == 0;
}

// Ищем в списке нужный индекс
Iterator* find_item_by_index(List *ls, const int index) {
    int counter = 0;
    Iterator *it = (Iterator*)malloc(sizeof(Iterator));
    if (index == 0) {
        it = first(ls);
```

```
        return it;
    }
    if (is_empty(ls)) return NULL;

    it = first(ls);
    while (it->node->next) {
        if (counter == index) return it;
        it->node = it->node->next;
        counter++;
    }
    return it;
}

// Выводим список на экран
void print_list(List *ls) {
    if (is_empty(ls)) {
        printf("\nError: List is empty!\n");
        return;
    }
    printf("\nThe list is: (%d)\n",ls->size);
    Iterator *it = (Iterator*)malloc(sizeof(Iterator));
    it = first(ls);
    while (it->node != NULL) {
        printf("%d -> ", it->node->data);
        it->node = it->node->next;
    }
    printf("NULL\n\n");
}

// Функция Destroy уничтожает список, перебирая его элементы и удаляя их
void destroy_list(List * ls) {
    elem_t *p = ls->head->next;
    while (p != ls->head) {
        elem_t *tmp = p;
        p = p->next;
        free(tmp);
    }
    free(ls->head);
    ls->head = NULL;
    ls->size = 0;
}

// Функция Insert вставляет элемент в список перед заданным.
void insert(List * ls, const int i, int t) {
       if (ls->size == 0) {
              ls->head->data = t;
              ls->head->next = NULL;
       } else {
              elem_t *tmp = (elem_t*)malloc(sizeof(elem_t));
              tmp->data = t;
              if (i == 0) {
                     tmp->next = ls->head;
                     ls->head = tmp;
              } else {
                     Iterator *it = find_item_by_index(ls, i-1);
                     tmp->next = it->node->next;
                     it->node->next = tmp;
              }
    }
    ls->size++;
    print_list(ls);
}

// функция Delete удаляет элемент на который указывает индекс
void delete(List *ls, const int i) {
    Iterator *tmp = last(ls);
       Iterator *it = find_item_by_index(ls, i);
    if (!ls->size) return;
    tmp->node = it->node->next;
    if (i>0) {
              Iterator *prev = find_item_by_index(ls, i-1);
              prev->node->next = tmp->node;
              free(prev);
       } else ls->head = tmp->node;

    printf("deleted = %d\n",it->node->data);
    free(it->node);
    it->node = tmp->node;
    ls->size--;
}
```

```
#include "list.h"

void function(List *ls, int k) {
        Iterator *it = first(ls);
        int count = 0;
        while (it->node->next) {
                int i = 0;
                while (i < k-1) ++i;
                count +=i;
                if (count == ls->size) break;
                it = find_item_by_index(ls, count+1);
                delete(ls, count);
        }
        print_list(ls);
}

int menu(){
    int action;
    printf("\nPlease, enter your choice:\n"
    "1 --- to print list.\n"
    "2 --- to insert into the list an element before some element (index)\n"
    "3 --- to delete an element from the list (index)\n"
    "4 --- length of list\n"
    "5 --- remove each k-th element, write some like 2\n"
    "6 --- exit.\n");
    printf("\nSelect action: ");
    scanf("%d",&action);
    return action;
}

int main(void){
    List* ls = CreateList();
    int index;
    int value;
    int action;
    do {
        action = menu();
        switch (action) {
            case 1:
                print_list(ls);
                break;
            case 2:
                printf("add: ");
                if (scanf("%d %d",&index, &value)==2){
                    insert(ls,index,value);
                }else {
                    printf("Error!\n");
                }
                break;
            case 3:
                printf("Delete: ");
                if (scanf("%d",&index)==1) {
                    delete(ls,index);
                } else {
                    printf("Error!\n");
                }
                print_list(ls);
                break;
            case 4:
                                printf("length of list = %d\n\n", ls->size);
                break;
            case 5:
                printf("function\n");
                printf("Enter number of k-th element: ");
                scanf("%d",&index);
                function(ls,index);
                break;
            case 6: break;
            default:
                printf("Command is missing\n");
                printf("Enter the correct code: ");
                break;
        }
    } while (action != 6);
    return 0;
}
```