Московский авиационный институт

(Национальный исследовательский университет)

Факультет прикладной математики и физики

Курсовой проект

по курсу

«Методы и языки программирования»

II семестр

Задание 8

Выполнила: Довженко А.

Группа: М8О-107Б-16, №5

Проверили: Ридли А.Н.

Ридли М.К.

Оценка:

Дата:

Москва

2017г.

Задача:

Составить и отладить программу на языке Си для обработки линейного списка заданной организации с отображением списка на динамические структуры. Навигацию по списку следует реализовать с применением итераторов. Предусмотреть выполнение одного нестандартного действия и четырех стандартных действий: печать списка, вставка нового элемента в список, удаление элемента из списка, подсчет длины списка.

Тип элемента списка: комплексный.
Вид списка: кольцевой двунаправленный.
Нестандартное действие: исключить из списка последние k элементов. Если в списке менее k элементов, то не менять его.

Введение:

Список довольно распространенная структура данных. Каждый элемент списка имеет указатель на следующий элемент (в случае двусвязного списка – еще и на предыдущий). Программисту доступен указатель на начало списка. Зная его, мы можем получить доступ до любого элемента, пройдя последовательно предыдущие элементы. Удобно реализовать итератор для навигации по списку, экономя время прохода по нему. Таким образом, списки удобно применять там, где достаточно только последовательного доступа к данным. Списки можно организовать на массивах, хотя удобнее использовать динамическую память.

Решение:

Структура списка представляет собой стандартную СД список с наложенными условиями. Так, двунаправленность реализована как существование двух указателей из каждой записи, первый – на следующую запись, второй – на предыдущую. Закольцованность списка означает, что указатель на следующую запись из последней записи указывает на первую запись, а указатель на предыдущую запись из первой записи указывает на последнюю запись. Ввод списка осуществляется со стандартного потока ввода. Действия, доступные пользователю: печать списка, удаление записи из списка, вставка новой записи в список, подсчет длины списка, удаление последних k записей из списка.

Организация программы:

main.c:

Программа управляется через этот файл. Команды insert after_node new_node и ins after_node new_node добавляют запись в список после указанной, если добавлемая запись первая, то она добавляется с начало списка. Команды print и p печатают список. Команды run k и r k удаляют последние k записей, если записей меньше k, то список не изменяется. Команды lenght и len выводят длину списка. Команды delete и del удаляют запись из списка. Команды destroy и des удаляют список. Команды quit и q организуют выход из программы.

list.h, list.c:

```
typedef complex double elem_type;
typedef struct _node Node;
typedef struct _list *List;
struct _node {
   elem_type elem;
   struct _node *next;
   struct _node *prev;
};

struct _list {
   Node *head;
   Node *iter;
   uint32_t elements;
};
```

Тип значений записей в списке – комплексный. Запись хранит значение и указатели на предыдущий и следующий элементы, т.к. список двунапрвленный. В структуре _list хранятся указатель на условно первую запись списка, итератор (запись, в которой мы находимся в данный момент), и количество элементов списка для удобства.

```
void iterator_next(List list);
void iterator_prev(List list);
void iterator_first(List list);
void iterator_last(List list);
elem_type iterator_get(List list);
```

void iterator_set(List list, elem_type value);
void iterator_delete(List list);

Реализация навигации по списку с помощью итератора.

List list_create();
Создание списка.

bool list_is_empty(List list);
Проверка, пуст ли список.

void list_insert(List list, Node *after_node, elem_type elem);
Вставка с список новой записи после заданной.

uint32_t list_length(List list);
Длина списка.

void list_print(List list);
Печать списка.

void list_node_delete(List list, Node *del_node);
Удаление заданной записи из списка.

void list_destroy(List *list);
Удаление списка.

Node *node_search(List list, elem_type value);
Поиск записи в списке.

void delete_last_k_elements(List list, uint32_t k);
Удаление последних k записей в списке.

Вывод:

Связный список используется практически везде — от ОС до игр, и второе — на его основе строится множество других структур данных. Структура списка вариативна, ее выбор зависит от нужд программиста. Список может быть однонаправленным или двунаправленным, кольцевым или линейным, с барьерными элементами или без них. Такая возможность выбора позволяет нам управлять сложностью своих программ, делая их доступнее для понимания, а также создавать высокопроизводительные программы, эффективно работающие с память. Правильно выбранная реализация может кардинально упростить код, устраняя запутанную логику. Когда учитываются производительность или память программы, правильный выбор структуры значительно сказывается на работе.

Код программы:

main.c

```
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <inttypes.h>
#include "list.h"
uint32_t main(void)
{
    double x_val = 0.0, y_val = 0.0;
    double x_node = 0.0, y_node = 0.0;
    //elem_type val = 0;
    //elem_type node = 0;
    uint32_t k = 0;
    List list = list_create();
    char s[8];
    printf("\nДля получения помощи в использовании программы напишите 'help' или 'h'.\n\n");
    while (1) {
        scanf("%7s", s);
        if (!strcmp(s, "insert") || !strcmp(s, "ins")) {
            if(list == NULL) {
                list = list_create();
            }
            scanf("%lf %lf %lf %lf", &x_node, &y_node, &x_val, &y_val);
            elem_type node = x_node + y_node * I;
            elem_type val = x_val + y_val * I;
            Node *after_node = NULL;
            after_node = node_search(list, node);
            //if(after_node != NULL) printf("lol\n");
            list_insert(list, after_node, val);

        } else if (!strcmp(s, "delete") || !strcmp(s, "del")) {
            if(list == NULL) printf("Списка не существует, воспользуйтесь командами 'help' или 'h'.\n");
            else if (list_length(list) == 0) {
                printf("Список пуст.\n");
            } else {
                scanf("%lf %lf", &x_node, &y_node);
                elem_type node = x_node + y_node * I;
                Node *after_node = NULL;
                after_node = node_search(list, node);
                list_node_delete(list, after_node);

            }
        } else if (!strcmp(s, "quit") || !strcmp(s, "exit") || !strcmp(s, "q")) {
            if (list != NULL) list_destroy(&list);
            break;
        } else if (!strcmp(s, "run") || !strcmp(s, "r")) {
            if(list == NULL) printf("Списка не существует, воспользуйтесь командами 'help' или 'h'.\n");
            else {
                scanf("%d", &k);
                delete_last_k_elements(list, k);
            }
        } else if (!strcmp(s, "print") || !strcmp(s, "p")) {
            if (list == NULL) printf("Списка не существует, воспользуйтесь командами 'help' или 'h'.\n");
```

```
            else {
                printf("\n");
                list_print(list);
                printf("\n");
            }
        } else if (!strcmp(s, "length") || !strcmp(s, "len")) {
            if (list == NULL) printf("Списка не существует, воспользуйтесь командами 'help' или 'h'.\n");
            else {
                printf("%d\n", list_length(list));
            }
        } else if (!strcmp(s, "destroy") || !strcmp(s, "des")) {
            if (list == NULL) printf("Списка не существует, воспользуйтесь командами 'help' или 'h'.\n");
            else {
                list_destroy(&list);
            }
        } else if (!strcmp(s, "help") || !strcmp(s, "h")) {
            printf("\n\nКоманды 'insert after_node new_node' и 'ins after_node new_node' добавляют значения в список, где after_node -- значение, после которого
вставляем, new_node -- вставляемое значение.\n\n");
            printf("\n\nКоманды 'print' и 'p' печатают список.\n\n");
            printf("\n\nКоманды 'run k' и 'r k' удаляют последние k записей, если k больше длины списка, то список остается без изменений.\n\n");
            printf("\n\nКоманды 'length' и 'len' возвращают длину списка.\n\n");
            printf("\n\nКоманды 'delete' и 'del' удаляют вершину из списка.\n\n");
            printf("\n\nКоманды 'destroy' и 'des' удаляют список.\n\n");
            printf("\n\nКоманды 'quit' и 'q' выходят из программы.\n\n");
        }
    }
    return 0;
}
```

list.h

```
#ifndef _LIST_H_
#define _LIST_H_
#include <stdbool.h>
#include <inttypes.h>
#include <complex.h>
typedef complex double elem_type;
//typedef int elem_type;
typedef struct _node Node;
typedef struct _list *List;
List list_create();
bool list_is_empty(List list);
void list_insert(List list, Node *after_node, elem_type elem);
uint32_t list_length(List list);
void list_print(List list);
void list_node_delete(List list, Node *del_node);
void list_destroy(List *list);
Node *node_search(List list, elem_type value);
void delete_last_k_elements(List list, uint32_t k);
#endif
```

list.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <complex.h>
#include "list.h"
struct _node {
    elem_type elem;
    struct _node *next;
    struct _node *prev;
};
struct _list {
    Node *head;
    Node *iter;
    uint32_t elements;
};
void iterator_next(List list);
void iterator_prev(List list);
void iterator_first(List list);
void iterator_last(List list);
elem_type iterator_get(List list);
void iterator_set(List list, elem_type value);
```

```
void iterator_delete(List list);
List list_create()
{
    List list = (List) malloc(sizeof(*list));
    list->head = NULL;
    list->elements = 0;
    iterator_first(list);
    return list;
}
void iterator_next(List list)
{
    list->iter = list->iter->next;
}
void iterator_prev(List list)
{
    list->iter = list->iter->prev;
}
void iterator_first(List list)
{
    list->iter = list->head;
}
void iterator_last(List list)
{
    iterator_first(list);
    iterator_prev(list);
}
void iterator_set(List list, elem_type value)
{
    list->iter->elem = value;
}
elem_type iterator_get(List list)
{
    return list->iter->elem;
}
void iterator_delete(List list)
{
    if(list->iter == NULL && list_length(list) > 0) {
        printf("Указанной вершины нет в списке\n");
    } else {
        if(list->iter->next == list->iter) {
            free(list->iter);
            list->iter = NULL;
            list->head = NULL;
        } else {
            Node *tmp = list->iter->next;
            if(list->iter == list->head) {
                list->head = tmp;
            }
            list->iter->prev->next = tmp;
            tmp->prev = list->iter->prev;
            free(list->iter);
            list->iter = tmp;
        }
        list->elements--;
    }
}
bool list_is_empty(List list)
{
    return (list->elements > 0 ? true : false);
}
void list_insert(List list, Node *after_node, elem_type value)
{
    Node *new_node = (Node *) malloc(sizeof(Node));
    if(!new_node) {
        fprintf(stderr, "Error: no memory\n");
        exit(1);
    }
    list->iter = after_node;
    new_node->elem = value;
    if(list->iter == NULL && list_length(list) == 0) {
        new_node->next = new_node;
        new_node->prev = new_node;
        list->iter = new_node;
        list->head = new_node;
        list->elements++;
    } else if(list->iter == NULL && list_length(list) > 0) {
        free(new_node);
        new_node = NULL;
        printf("Указанной вершины нет в списке\n");
    } else {
        new_node->next = list->iter->next;
        new_node->prev = list->iter;
        list->iter->next->prev = new_node;
        list->iter->next = new_node;
        list->elements++;
    }
}
```

```
Node *node_search(List list, elem_type value)
{
    iterator_first(list);
    //printf("check2\n");
    for (uint32_t i = 0; i < list_length(list); ++i) {
        //printf("check\n");
        if(value == iterator_get(list)) {
            //printf("kek\n");
            return list->iter;
        }
        iterator_next(list);
    }
    return NULL;
}
uint32_t list_length(List list)
{
    return list->elements;
}
void list_print(List list)
{
    iterator_first(list);
    for (uint32_t i = 0; i < list_length(list); ++i) {
        printf("(%.2lf + %.2lfi) \n", creal(iterator_get(list)), cimag(iterator_get(list)));
        iterator_next(list);
    }
}
void list_node_delete(List list, Node *del_node)
{
    list->iter = del_node;
    iterator_delete(list);
}
void list_destroy(List *list)
{
    if((*list)->iter != NULL) {
        iterator_first(*list);
        while((*list)->head->next != (*list)->head) {
            iterator_delete(*list);
        }
        iterator_delete(*list);
    }
    free(*list);
    *list = NULL;
}
void delete_last_k_elements(List list, uint32_t k)
{
    if (list_length(list) < k) {
        printf("Длина списка меньше, чем k.\n");
    } else {
        for (uint32_t i = 0; i < k; i++) {
            iterator_last(list);
            iterator_delete(list);
        }
    }
}
```