МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Факультет № 8

«Информационные технологии и прикладная математика»

Кафедра 806

**КУРСОВОЙ ПРОЕКТ**

**по курсу «Практикум на ЭВМ»**

на тему

“Обработка последовательной файловой структуры на языке Си”

Выполнил:

Студент 1 курса, гр. М8О-106Б-19

Кондратьев Егор Алексеевич

Москва

2020 г.

# Содержание

[Содержание 2](#_Toc43929280)

[Задание 3](#_Toc43929281)

[Общий метод решения 3](#_Toc43929282)

[Общие сведения о программе 3](#_Toc43929283)

[Функциональное назначение 3](#_Toc43929284)

[Текст программы 4](#_Toc43929285)

[Распечатка протокола 13](#_Toc43929286)

[Заключение 14](#_Toc43929287)

[Список используемых источников 14](#_Toc43929288)

# Задание

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.

*Вариант*

13

*Тип элемента списка*:

Комплексный

*Вид списка:*

Односвязный динамический

*Нестандартное действие:*

Очистить список, если в нем есть заданный элемент

# Общий метод решения

Односвязный список – это структура данных, представляющая собой последовательность элементов, последний элемент которой не содержит указатель на первый элемент списка.

Циклический однонаправленный список похож на линейный однонаправленный список, но его последний элемент содержит указатель, связывающий его с первым элементом.

Для полного обхода такого списка достаточно иметь указатель на первый элемент.

# **Общие сведения о программе**

**Язык и система программирования:** GNU C

# Функциональное назначение

Программа представляет собой программную реализацию такой последовательной структуры данных как односвязный список. Кроме того, показаны основные функции для полноценной работы со списками:

* Генерация списка
* Вставка элемента
* Удаление элемента
* Удаление списка
* Печать списка
* ….

Также, реализовано нестандартное действие

# Текст программы

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

#include <ctype.h>

#include <stdlib.h>

typedef struct list list;

typedef struct list\_el list\_el;

typedef struct

{

    int val;

    int val2;

    int s1;

    int s2;

} list\_com;

struct list\_el

{

    list\_com val;

    list\_el \*next;

};

struct list

{

    list\_el \*first;

    list\_el \*last;

    size\_t size;

};

typedef struct

{

    list \*l;

    list\_el \*\*pos;

} iter;

void l\_init(list \*l)

{

    l->first = NULL;

    l->last = NULL;

    l->size = 0;

}

void l\_destroy(list \*l)

{

    list\_el \*el = l->first;

    list\_el \*next\_el;

    while (el != NULL)

    {

        next\_el = el->next;

        free(el);

        el = next\_el;

    }

    l->size = 0;

    l->first = NULL;

    l->last = NULL;

}

list\_com l\_remove(list \*l, list\_el \*\*pos)

{

    list\_el \*tofree = \*pos;

    list\_com value = tofree->val;

    if (l->first == l->last)

        l->last = NULL;

    else if (l->last == \*pos)

    {

        list\_el \*\*it = &(l->first);

        while ((\*it)->next->next != NULL)

            it = &((\*it)->next);

        l->last = \*it;

    }

    \*pos = (\*pos)->next;

    free(tofree);

    l->size--;

    return value;

}

bool l\_insert\_before(list \*l, list\_el \*\*pos, list\_com value)

{

    list\_el \*new = (list\_el \*)malloc(sizeof(list\_el));

    if (new == NULL)

        return false;

    new->val = value;

    new->next = \*pos;

    if (l->last == NULL)

        l->last = new;

    else if (\*pos == NULL)

        l->last = new;

    \*pos = new;

    l->size++;

    return true;

}

size\_t l\_get\_len(list \*l)

{

    return l->size;

}

iter iter\_begin(list \*l)

{

    iter it;

    it.pos = &(l->first);

    it.l = l;

    return it;

}

iter iter\_end(list \*l)

{

    iter it;

    if (l->last == NULL)

        it.pos = &(l->first);

    else

        it.pos = &(l->last->next);

    it.l = l;

    return it;

}

bool iter\_equals(iter a, iter b)

{

    return (a.pos == b.pos && a.l == b.l);

}

void iter\_move\_next(iter \*it)

{

    it->pos = &((\*it->pos)->next);

}

list\_com iter\_get\_value(iter it)

{

    list\_el \*\*pos = it.pos;

    return (\*pos)->val;

}

void iter\_set\_value(iter it, list\_com val)

{

    (\*it.pos)->val = val;

}

void iter\_insert\_before(iter \*it, list\_com val)

{

    l\_insert\_before(it->l, it->pos, val);

}

void iter\_remove(iter \*it)

{

    l\_remove(it->l, it->pos);

}

bool iter\_move\_to(int i, iter \*it)

{

    list \*l = it->l;

    if (i < 0 || i > l->size)

    {

        return false;

    }

    it->pos = &l->first;

    while (i > 0)

    {

        iter\_move\_next(it);

        i--;

    }

    return true;

}

list l;

void help()

{

    printf("-------------------------------------------\n");

    printf("|  Commands   |   Description             |\n");

    printf("-------------------------------------------\n");

    printf("|  help       |   print menu              |\n");

    printf("-------------------------------------------\n");

    printf("|  add i val  |   add element in i pos    |\n");

    printf("-------------------------------------------\n");

    printf("|  del i      |   delete element in i pos |\n");

    printf("-------------------------------------------\n");

    printf("|  print      |   print list              |\n");

    printf("-------------------------------------------\n");

    printf("|  func val   |   delete list if find val |\n");

    printf("-------------------------------------------\n");

    printf("|  quit       |   quit                    |\n");

    printf("-------------------------------------------\n\n");

}

void del(int i)

{

    iter it = iter\_begin(&l);

    if (!iter\_move\_to(i, &it))

    {

        printf("bad index\n\n");

        return;

    }

    if (i == l\_get\_len(&l))

    {

        printf("bad index\n\n");

        return;

    }

    iter\_remove(&it);

    printf("delete\n\n");

}

void add(int i, list\_com val)

{

    iter it = iter\_begin(&l);

    if (!iter\_move\_to(i, &it))

    {

        printf("bad index\n\n");

        return;

    }

    iter\_insert\_before(&it, val);

    printf("\n");

}

void func(list \*l, list\_com x)

{

    bool tmp = false;

    iter end = iter\_end(l);

    for (iter it = iter\_begin(l); !iter\_equals(it, end); iter\_move\_next(&it))

    {

        if ((\*it.pos)->val.val == x.val &&

            (\*it.pos)->val.val2 == x.val2 &&

            (\*it.pos)->val.s1 == x.s1 &&

            (\*it.pos)->val.s2 == x.s2)

        {

            tmp = true;

        }

    }

    if (tmp)

    {

        printf("clear all list\n\n");

        l\_destroy(l);

        l\_init(l);

    }

    else

        printf("not found\n\n");

}

void print(list \*l)

{

    if (l->first == NULL)

    {

        printf("List empty\n\n");

        return;

    }

    iter end = iter\_end(l);

    for (iter begin = iter\_begin(l); !iter\_equals(begin, end); iter\_move\_next(&begin))

    {

        list\_com tmp = iter\_get\_value(begin);

        tmp.val = tmp.s1 \* tmp.val;

        tmp.val2 = tmp.s2 \* tmp.val2;

        if (tmp.s2 == 0)

            printf("[ %d ]\n", tmp.val);

        if (tmp.s2 == -1)

            printf("[ %d%di ]\n", tmp.val, tmp.val2);

        if (tmp.s2 == 1)

            printf("[ %d+%di ]\n", tmp.val, tmp.val2);

    }

    printf("[ NULL ]\n\n");

}

list\_com tocom(char \*a)

{

    list\_com x;

    x.val = 0;

    x.val2 = 0;

    x.s1 = 1;

    x.s2 = 0;

    int i = 0;

    if (a[0] == '-')

    {

        x.s1 = -1;

        i++;

    }

    if (a[0] == '+')

    {

        x.s1 = 1;

        i++;

    }

    for (i; a[i] != '\0'; i++)

    {

        if (a[i] == '+')

            x.s2 = 1;

        else if (a[i] == '-')

            x.s2 = -1;

        if (a[i] >= '0' && a[i] <= '9')

        {

            if (x.s2 == 0)

                x.val = x.val \* 10 + a[i] - '0';

            else

                x.val2 = x.val2 \* 10 + a[i] - '0';

        }

    }

    return x;

}

enum commands

{

    ADD,

    DELETE,

    PRINT,

    FUNC,

    FAIL,

    HELP,

    QUIT

};

enum commands parse(char \*command)

{

    if (command == NULL)

        return FAIL;

    switch (command[0])

    {

    case 'a':

        if (command[1] != '\0')

        {

            char tmp[] = "add";

            if (strcmp(command, tmp) == 0)

                return ADD;

            return FAIL;

        }

        return ADD;

    case 'd':

        if (command[1] != '\0')

        {

            char tmp[] = "del";

            if (strcmp(command, tmp) != 0)

                return FAIL;

        }

        return DELETE;

    case 'p':

        if (command[1] != '\0')

        {

            char tmp[] = "print";

            if (strcmp(command, tmp) != 0)

                return FAIL;

        }

        return PRINT;

    case 'f':

        if (command[1] != '\0')

        {

            char tmp[] = "func";

            if (strcmp(command, tmp) != 0)

                return FAIL;

        }

        return FUNC;

    case 'h':

        if (command[1] != '\0')

        {

            char tmp[] = "help";

            if (strcmp(command, tmp) != 0)

                return FAIL;

        }

        return HELP;

    case 'q':

        if (command[1] != '\0')

        {

            char tmp[] = "quit";

            if (strcmp(command, tmp) != 0)

                return FAIL;

        }

        return QUIT;

        break;

    }

    return FAIL;

}

bool command(enum commands com, char \*arg1, char \*arg2)

{

    switch (com)

    {

    case PRINT:

        print(&l);

        return true;

    case DELETE:

        if (arg1 == NULL)

            break;

        del(atoi(arg1));

        return true;

    case ADD:

        if (arg2 == NULL)

            break;

        add(atoi(arg1), tocom(arg2));

        return true;

    case FUNC:

        if (arg1 == NULL)

            break;

        func(&l, tocom(arg1));

        return true;

    case HELP:

        if (arg1 == NULL)

            break;

        help();

        return true;

    case QUIT:

        l\_destroy(&l);

        \_Exit(0);

    case FAIL:

        printf("Invalid command\nTry command help\n");

        return false;

    }

    printf("Invalid command or arguments \n\tTry command help\n");

    return false;

}

int main()

{

    help();

    l\_init(&l);

    char string[50] = "";

    char \*commands = "";

    char \*arg1 = NULL;

    char \*arg2 = NULL;

    while (true)

    {

        fgets(string, 50, stdin);

        char end[] = " \t\n";

        commands = strtok(string, end);

        arg1 = strtok(NULL, end);

        arg2 = strtok(NULL, end);

        enum commands com = parse(commands);

        command(com, arg1, arg2);

        arg1 = NULL;

        arg2 = NULL;

    }

    return 0;

}

# Распечатка протокола

werwerwerwerw@Du:/mnt/c/Users/egork/Desktop/labs/kp8$ gcc kp8.c    
werwerwerwerw@Du:/mnt/c/Users/egork/Desktop/labs/kp8$ ./a.out      
--------------------------------------------------------  
|  Commands   |   Description                    |  
--------------------------------------------------------

|  help              |   print menu                      |  
--------------------------------------------------------  
|  add i val       |   add val in i pos               |  
--------------------------------------------------------  
|  del i              |   delete element in i pos   |  
--------------------------------------------------------  
|  print              |   print list                          |  
--------------------------------------------------------  
|  func val        |   delete list if find val        |  
--------------------------------------------------------  
|  quit               |   quit                                |  
--------------------------------------------------------

add 0 12-2i

add 1 23-2i

print  
[ 12-2i ]  
[ 23-2i ]  
[ NULL ]

add 2 -23

print  
[ 12-2i ]  
[ 23-2i ]  
[ -23 ]  
[ NULL ]

func 23  
not found

func -23

clear all list

print  
List empty

quit  
werwerwerwerw@Du:/mnt/c/Users/egork/Desktop/labs/kp8$

# Заключение

Списки бывают линейные и кольцевые, с барьерным элементом и без, однонаправленные и двунаправленные.

*Достоинства списков:*

* эффективное (за константное время) добавление и удаление элементов
* размер ограничен только объёмом памяти компьютера и разрядностью указателей.
* динамическое добавление и удаление элементов.

*Недостатки:*

Недостатки связных списков вытекают из их главного свойства — последовательного доступа к данным:

* сложность прямого доступа к элементу, а именно определения физического адреса по его индексу (порядковому номеру) в списке
* на поля-указатели (указатели на следующий и предыдущий элемент) расходуется дополнительная память (в массивах, например, указатели не нужны)
* некоторые операции со списками медленнее, чем с массивами, так как к произвольному элементу списка можно обратиться, только пройдя все предшествующие ему элементы

# Список используемых источников

* Поляков К.​“Динамические структуры данных”.
* <https://www.cyberforum.ru/>
* https://prog-cpp.ru/data-dcs/
* https://ru.wikipedia.org/wiki/Связанный\_список
* Односвязный список на C++
* <http://itnotesblog.ru/note.php?id=178>