Московский авиационный институт (национальный исследовательский университет)

Институт № 8 «Информационные технологии и прикладная математика» Кафедра 806 «Вычислительная математика и программирование»

ПРАКТИКУМ ПРОГРАММИРОВАНИЯ

по курсу   
«Алгоритмы и структуры данных»   
2 семестр

на тему   
«Линейные списки»

|  |  |
| --- | --- |
| Студент: | Абдисаламов Э. |
| Группа: | М8О – 106Б – 22 |
| Преподаватель: | Дубинин А.В. |
| Подпись: |  |
| Оценка: |  |

Москва, 2023

[Введение 3](#_Toc134627382)

[Теоретическая часть 3](#_Toc134627383)

[Линейный список 3](#_Toc134627384)

[Итераторы 5](#_Toc134627385)

[Практическая часть 7](#_Toc134627386)

[Задание: 7](#_Toc134627387)

[Метод реализации 7](#_Toc134627388)

[Код программы: 8](#_Toc134627389)

[iterator.h: 8](#_Toc134627390)

[iterator.c: 8](#_Toc134627391)

[mylist.h: 8](#_Toc134627392)

[mylist.c: 8](#_Toc134627393)

[main.c: 8](#_Toc134627394)

[Оценка сложности 8](#_Toc134627395)

[Список источников: 9](#_Toc134627396)

# Введение

По мере изучения новых структур данных мы знакомимся с различными способами оперирования базовыми типами данных выбранного языка программирования. Так как для нас основной язык — Си, большую часть структур данных реализовывать будем сами. В данном задании практикума рассмотрим связные списки, а также присущую им теорию.

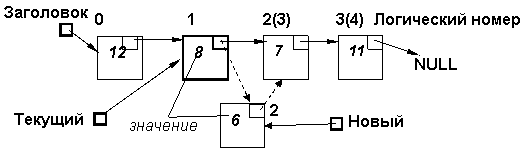
Разберемся, зачем нужны связные списки, как их можно использовать, какие для них существуют конфигурации, также что можно с помощью них хранить.

# Теоретическая часть

## Линейный список

Реальное многообразие структур данных базируется всего на двух основных способах получения адреса хранимого элемента: вычисление (массив) и хранение (указатель). До сих пор основной компонентой структуры данных являлся массив (обычный массив, массив указателей). Если же попытаться построить структуру данных, исходя только из указателей, то получается цепочка (последовательность) элементов, содержащих указатели друг на друга. В простейшем случае она может быть линейной (список), в более сложных случаях – ветвящейся (деревья, графы).

Итак, список – линейная последовательность элементов, каждый из которых содержит указатели (ссылается) на своих соседей.



Сразу же отметим основную особенность: физическое размещение в памяти элементов списка не имеет никакого значения, все определяется наличием ссылок на него в других элементах и извне. У массива всегда есть «начало». У списка по определению отсутствует фиксированная привязка к памяти.

Перечислим основные свойства списка:

1) Элемент списка доступен в программе через указатель. «Смысл» этого указателя отражает функциональное назначение элемента списка в программе: первый, последний, текущий, предыдущий, новый и т.п.. Между указателем и элементом списка имеется такая же взаимосвязь, как между индексом в массиве и элементом массива.

2) в программе список задается посредством заголовка **–** указателя на первый элемент списка;

3) порядок следования элементов определяется последовательностью связей между элементами. Изменение порядка следования элементов (вставка, удаление) осуществляются изменением переустановкой указателей на соседние элементы.

4) логический (порядковый) номерэлемента списка также задается его естественной нумерацией в цепочке элементов;

5) список является структурой данных с последовательным доступом.Для получения n-го по счету элемента необходимо последовательно пройти по цепочке от элемента, на который имеется указатель (например, от заголовка);

6) список удобен для использования именно как *динамическая структура данных*: элементы списка обычно создаются как динамические переменные, а связи между ними устанавливаются программно (динамически);

7) список обладает свойством локальности изменений:при вставке/удалении элемента изменения касаются только текущего и его соседей. Вспомним массив: при вставке/удалении его элементов происходит физическое перемещение (сдвиг) всех элементов от текущего до конца.

Следовательно, преимущества списков проявляются в таких структурах данных, где изменения порядка превалируют над операциями доступа и поиска

К основным операциям над линейными списками относятся:

· Создание списка;

· Подсчет узлов (элементов) списка;

· Доступ к узлу;

· Конкатенация (соединение) списков;

· Поиск минимального (максимального) элемента, суммы, произведения;

· Изменение чисел в списке;

· Исключение (включение) узла;

· Разбиение списка;

· Упорядочение узлов и др.

К основным способам реализации операций над линейными списками относятся:

· Итерационный алгоритм;

· Рекурсивный алгоритм.

К специальным видам линейных списков относятся *стеки* и *очереди*. Это динамические структуры данных, которые часто используют в программировании для определения порядка выполнения подзадач. *Стек* – это линейный список, в котором все включения и исключения производятся на одном конце списка, называемом вершиной стека («последним пришел – первым ушел»). *Очередь* – это линейный список, в котором все включения производятся на одном конце списка (в конце очереди), а все исключения – на другом его конце (в начале очереди) («первым пришел – первым ушел»).

## Итераторы

Итератор — это объект, который позволяет перемещаться (итерироваться) по элементам некоторой последовательности.

Использование таких объектов классифицируется как паттерн программирования.

Последовательность может быть как готовым набором объектов в памяти, так и состоять объектов которые создаются "на лету" при перемещении итератора (например читаются из файла).

В отличие от разнообразных последовательностей элементов (массивы, списки, файлы), итераторы имеют одинаковый интерфейс: получение текущего элемента, перемещение к следующему. Это позволяет писать более общие алгоритмы, которые работают с любыми итераторами, поддерживающими этот минимальный набор функций.

Итераторы стандартной библиотеки C++ повторяют интерфейс указателя. Это позволяет использовать указатели на элементы массива в качестве итераторов. При этом большинство алгоритмов используют только некоторый минимальный набор операций, например только перемещение вперед.

Использование итераторов позволяет абстрагироваться от внутреннего представления структуры данных и использовать ее более удобным способом.

# Практическая часть

## Задание:

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

1. Печать списка.

2. Вставка нового элемента в список.

3. Удаление элемента из списка.

4. Подсчет длины списка.

Тип элемента списка: unsigned int

Вид списка: кольцевой двунаправленный

Нестандартное действие: удалить элементы списка со значениями, находящимися в заданном диапазоне;

## Метод реализации

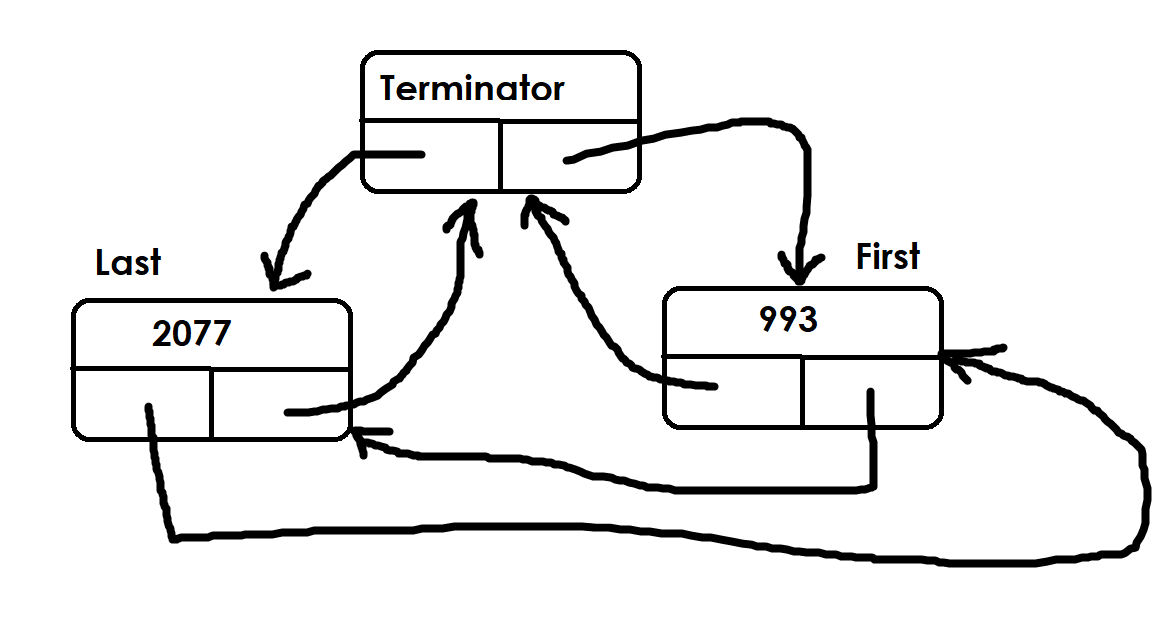
Так как список должен быть двунаправленный и закольцованный, воспользуемся терминатором (элементом без значения, о котором программист знать не должен), который позволит нам реализовать кольцо, а также будем хранить в каждом узле ссылку на предыдущий и на следующий узел.

Барьерный элемент позволит нам определять конец списка и начало.

Ниже на схеме представлен пример списка, который будет впоследствии реализован. В каждом элементе помимо ссылок на соседей хранится само значение.

Пользователь будет вводить в консоль одну из 6 команд:

* add
* rm
* len
* rm range
* print
* help



Код программы:

iterator.h:

#pragma once

#include "mylist.h"

#include <stdbool.h>

typedef struct {

    node\* cur;

    list\* list;

} iterator;

iterator iter\_begin(list \*list);

bool iter\_not\_end(iterator \*iter);

void iter\_next(iterator \*iter);

void iter\_prev(iterator \*iter);

unsigned int iter\_val(iterator \*iter);

void iter\_delete(iterator \*iter);

iterator.c:

#include "mylist.h"

#include "iterator.h"

#include "stdlib.h"

#include "stdbool.h"

iterator iter\_begin(list \*list) {

    iterator iter;

    iter.cur = list->terminator->next;

    iter.list = list;

    return iter;

}

unsigned int iter\_val(iterator \*iter){

    return iter->cur->val;

}

bool iter\_not\_end(iterator \*iter) {

    if (iter->cur == iter->list->terminator) return false;

    return true;

}

void iter\_next(iterator \*iter) {

    iter->cur = iter->cur->next;

}

void iter\_prev(iterator \*iter) {

    iter->cur = iter->cur->prev;

}

void iter\_delete(iterator \*iter){

    node\* node = iter->cur;

    iter\_prev(iter);

    delete\_node(node);

    iter->list->len--;

}

mylist.h:

#pragma once

typedef struct node node;

struct node {

    unsigned int val;

    node \*next;

    node \*prev;

};

typedef struct {

    node \*terminator;

    node \*last;

    int len;

} list;

void init(list \*list);

void print\_list(list \*list);

void insert(list \*list, unsigned int val);

void delete\_by\_val(list \*list, unsigned int val);

void delete\_node(node\* node);

int length(list \*list);

void delete\_range(list \*list, unsigned int min, unsigned int max);

void destroy(list \*list);

mylist.c:

#include <stdio.h>

#include <stdlib.h>

#include "mylist.h"

#include "iterator.h"

void init(list \*list) {

    list->terminator = malloc(sizeof(node));

    list->terminator->next = list->terminator->prev = list->terminator;

    list->last = list->terminator;

    list->len = 0;

}

void insert(list \*list, unsigned int val) {

    node \*new\_node = malloc(sizeof(node));

    new\_node->val = val;

    new\_node->prev = list->last;

    new\_node->next = list->terminator;

    list->terminator->prev = new\_node;

    list->last->next = new\_node;

    list->last = new\_node;

    list->len++;

}

void delete\_by\_val(list \*list, unsigned int val) {

    if (list->len == 0) return; // если список пустой

    node\* elem = NULL;

    // поиск нужного элемента

    for (iterator iter = iter\_begin(list); iter\_not\_end(&iter); iter\_next(&iter)){

        if (iter\_val(&iter) == val){

            elem = iter.cur;

            break;

        }

    };

    if (elem == NULL) return; // если элемент не найден

    elem->prev->next = elem->next;

    elem->next->prev = elem->prev;

    free(elem);

    list->len--;

}

void delete\_node(node\* node) {

    node->prev->next = node->next;

    node->next->prev = node->prev;

    free(node);

}

void print\_list(list \*list) {

    for (iterator iter = iter\_begin(list); iter\_not\_end(&iter); iter\_next(&iter)) {

        printf("%u ", iter\_val(&iter));

    }

    printf("\n");

}

int length(list \*list) {

    return list->len;

}

void delete\_range(list \*list, unsigned int min, unsigned int max) {

    for (iterator iter = iter\_begin(list); iter\_not\_end(&iter); iter\_next(&iter)) {

        unsigned int elem\_val = iter\_val(&iter);

        if (( elem\_val >= min) && (elem\_val <= max)) {

            iter\_delete(&iter);

        }

    }

}

void destroy(list \*list){

    for (iterator iter = iter\_begin(list); iter\_not\_end(&iter); iter\_next(&iter)) {

        iter\_delete(&iter);

    }

    free(list->terminator);

}

main.c:

#include <stdio.h>

#include <stdbool.h>

#include "struct/mylist.h"

#include "struct/iterator.h"

#include <string.h>

#include <stdlib.h>

void clean\_input(){

    char s;

    while (s != '\n') s = getchar();

    fflush(stdin);

}

bool check\_extra\_input(){

    int ch;

    while ((ch = getchar()) != '\n' && ch != EOF) {

        if (ch != ' ' && ch != '\t') { // игнорируем пробелы и табуляцию

            return false;

        }

    }

    return true;

}

bool input\_errors(){

    if (!check\_extra\_input()) {

        clean\_input();

        printf("error: extra input after command\n");

        return true;

    }

    return false;

}

bool correct\_input\_uint(unsigned int \*val) {

    int stat = scanf("%u", val);

    if (stat != 1) {

        clean\_input();

        printf("error: invalid input format\n");

        return false;

    }

    return true;

}

void process\_unknown\_command(){

    clean\_input();

    printf("there is no such command\n");

}

void print\_manual(){

    printf("This program provides the ability to use the list structure with values unsigned int. \nThe following commands are defined: \n1. add \"val\"\t\t - Add an element to the list \n2. rm \"val\"\t\t - Remove an element from the list \n3. len\t\t\t - Length of the list \n4. rm range \"min\" \"max\"\t - Remove list elements with values within a specified range \n5. print \t\t - Print list \n6. help\t\t\t - Print manual\n7. end\t\t\t - Finish the program. \n");

}

int main() {

    setbuf(stdout, NULL);

    print\_manual();

    char command[10];

    list list;

    init(&list);

    while (1){

        printf(">>> ");

        scanf("%s", command);

        if ((strcmp(command, "end") == 0) || feof(stdin)) break;

        else if (strcmp(command, "add") == 0) {

            unsigned int val;

            if (correct\_input\_uint(&val)){

                if (!input\_errors()){

                    insert(&list, val);

                }

            }

        } else if (strcmp(command, "rm") == 0) {

            char subcommand[10];

            scanf("%s", subcommand);

            if (strcmp(subcommand, "range") == 0) {

                unsigned int min, max;

                int stat = scanf("%u %u", &min, &max);

                if (stat != 2) {

                    clean\_input();

                    printf("error: invalid input format\n");

                } else {

                    if (!input\_errors()){

                        delete\_range(&list, min, max);

                    }

                }

            } else {

                unsigned int val = atoi(subcommand);

                if (!input\_errors()){

                    delete\_by\_val(&list, val);

                }

            }

        } else if (strcmp(command, "len") == 0) {

            if (!input\_errors()){

                printf("len: %d\n", length(&list));

            }

        } else if (strcmp(command, "print") == 0) {

            if (!input\_errors()){

                printf("list: "); print\_list(&list);

            }

        } else if (strcmp(command, "help") == 0) {

            print\_manual();

        } else {

            process\_unknown\_command();

        }

    }

    destroy(&list);

    return 0;

}

# Оценка сложности

insert – O(1)

delete\_by\_val – O(n), где n – размер списка.

print\_list – O(n), n – размер списка.

delete\_range – O(n), n – размер списка.

destroy – O(n), n – размер списка.

# Заключение

# Список источников:

Понятие итератора:  
<https://ru.stackoverflow.com/questions/270697/Что-такое-итераторы-и-зачем-они-нужны>

Списки, их основные виды и способы реализации:  
<https://megalektsii.ru/s19674t1.html>

Линейные списки:  
<http://ermak.cs.nstu.ru/cprog/html/063.htm>

Связный список:  
<https://ru.wikipedia.org/wiki/Связный_список>