Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**КУРСОВАЯ РАБОТА**

**по курсу "Вычислительные системы"**

**II семестр**

**Задание 8**

**«Линейные списки»**

Студент: Почечура А.А.

Группа: М8О-106Б-20

Оценка:

Дата:

Подпись:

Оглавление

[1.Общая информация 3](#_Toc73233397)

[2.Постановка задачи 3](#_Toc73233398)

[3.Общий метод решения 4](#_Toc73233399)

[4.Общие сведения о программе 4](#_Toc73233400)

[5.Таблица переменных и констант 4](#_Toc73233401)

[6.Функциональное назначение 5](#_Toc73233402)

[7.Сложностные оценки 6](#_Toc73233403)

[8.Программа на Си 6](#_Toc73233404)

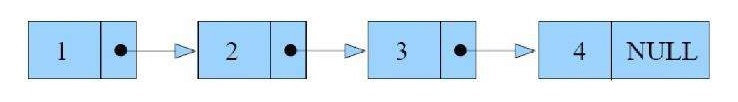
[9.Заключение 19](#_Toc73233405)

[10.Список используемых источников 19](#_Toc73233406)

# 1.Общая информация

Связный список — базовая динамическая структура данных в информатике, состоящая из узлов, каждый из которых содержит как собственно данные, так и одну или две ссылки («связки») на следующий и/или предыдущий узел списка. Принципиальным преимуществом перед массивом является структурная гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задаётся его внутренними связями.

В программе использован линейный однонаправленный список. Это значит, что ссылки в каждом узле указывают на последующий узел в списке.



Список будет отображён на массиве, в котором все пустые элементы также связаны. При добавлении элемента в список, он будет ставиться на место первого свободного, и теперь первым свободным станет следующий элемент этой связи. При удалении элемента будет наоборот происходить “подвязывание” элемента в начало связи пустых элементов.

# 2.Постановка задачи

Составить и отладить программу на языке Си для обработки линейного списка заданной организации с отображением списка на массив (только с индексным доступом, без применения ссылок и указателей).

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.

**Вид списка:** линейный однонаправленный.

**Тип элемента списка:** беззнаковое целое (unsigned long int)

**Нестандартное действие** – удалить из списка все элементы, предшествующие и последующие заданному значению (вариант 4).

# 3.Общий метод решения

Работа идёт с тремя программами языка Си – основной и двумя вспомогательными для работы с самим линейным списком и итераторами для него, одним заголовочным файлом (содержит описание структур, необходимых для работы соответственных программ и список используемых функций).

Основная программа считывает команду, которую необходимо исполнить и нужные для неё операнды, после чего исполняет её, взаимодействуя со списком.

# 4.Общие сведения о программе

Программное и аппаратное обеспечения для запуска данной программы на ПК не ограничено в выборе. Операционная система семейства Linux - Ubuntu. Язык программирования Си.

Имена файлов:

1. kursach\_8.c;
2. list.h;
3. list\_fucntions.c;
4. iter\_functions.c;

# 5.Таблица переменных и констант

Описание переменных и структур в заголовочном файле “ list.h”:

Структура list\_el:

|  |  |
| --- | --- |
| unsigned int val | Значение элемента списка |
| int next | Следующий элемент в списке |

Структура list:

|  |  |
| --- | --- |
| list\_el \*buf | Указатель на буфер массива |
| int cap | Вместительность массива |
| int top\_empty | Индекс первого свободного элемента в массиве |
| int size | Размер списка |
| int first | Индекс первого элемента в списке |
| int last | Индекс последнего элемента в списке |

Структура iter:

|  |  |
| --- | --- |
| list \*list; | Указатель на список |
| int prev; | Индекс предыдущего элемента в списке |

# 6.Функциональное назначение

Программа представляет собой программную реализацию таких структур данных, как односвязные списки и итераторы для работы с ними. Кроме того, показаны основные функции для полноценной работы со списками:

* Генерация списка;
* Вставка элемента по индексу;
* Удаление элемента по индексу;
* Получение значения элемента;
* Удаление списка;
* Печать списка.

Для навигации по списку реализован итератор со следующими функциями:

* Получение итератора на первый/последний элемент;
* Перемещение итератора на следующий элемент;
* Проверка итераторов на равенство;
* Получение значения элемента;
* Вставка/удаление элемента.

Также, реализовано нестандартное действие – переставление двух элементов списка местами.

# 7.Сложностные оценки

Добавление элемента в список –

Удаление элемента из списка –

Печать списка -

Задание -

# 8.Программа на Си

kursach\_8.c:

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include "list.h"

void task(list \*l, unsigned int val){

iter i=begin\_iter(l);

bool h=false;

iter i1=end\_iter(l);

while(!iter\_equal(i,end\_iter(l))){

if(iter\_val(i)==val){

if(!iter\_equal(iter\_next(i),end\_iter(l)) && iter\_val(iter\_next(i))!=val){

iter\_remove(&i);

if(h==true){

i=iter\_next(i);

iter\_remove(&i1);

}

if(h!=true && !iter\_equal(i,end\_iter(l))){

i=iter\_next(i);

i1=iter\_next(i1);

}

h=false;

} else if(!iter\_equal(iter\_next(i),end\_iter(l))){

i=iter\_next(i);

iter\_remove(&i1);

h=true;

} else {

if(h==true){

i=iter\_next(i);

iter\_remove(&i1);

} else {

i=iter\_next(i);

}

}

} else if(!iter\_equal(iter\_next(i),end\_iter(l)) && iter\_val(iter\_next(i))==val){

i=iter\_next(i);

iter\_remove(&i1);

} else {

i=iter\_next(i);

i1=iter\_next(i1);

}

}

}

void usage(){

printf("\nUSAGE:\n\na <EL> <POS> - add element on specified position.\nd <POS> - delete element for specified position.\n");

printf("t <EL> - task: удалить из списка все элементы, предшествующие и последующие заданному значению.\n\n");

printf("Possible values for <EL>: unsigned int\n");

printf("Possible values for <POS>: int,'b' - back of list, 'f' - front of list\n\n");

printf("p - print list.\n\nh - help.\n\n");

}

void reading(char t){

while(t!='\n'){

t=getchar();

}

}

int main(){

usage();

list l;

l\_create(&l);

char a;

unsigned int val;

int pos;

char test;

while(scanf("%c",&a)!=-1){

while(a==' ' || a=='\n' || a=='\t'){

scanf("%c",&a);

}

test=getchar();

if(a=='h' && (test==' ' || test=='\n' || test=='\t')){

if(test!='\n'){

scanf("%c",&test);

while(test==' ' || test=='\t'){

scanf("%c",&test);

}

}

if(test!='\n'){

printf("Error: use 'h' without any arguments.\n");

reading(test);

} else {

usage();

}

} else if(a=='a' && (test==' ' || test=='\n' || test=='\t')){

if(test!=' ' && test!='\t'){

printf("Error: wrong usage of command 'a'. Use 'h' for help.\n");

continue;

}

while(test==' ' || test=='\t'){

test=getchar();

}

if(test=='\n'){

printf("Error: lack of arguments for command 'a'. Use 'h' for help.\n");

continue;

} else if(test>='1' && test<='9'){

ungetc(test,stdin);

scanf("%u",&val);

test=getchar();

} else {

printf("Error: wrong value for 'a'. Use 'h' for help.\n");

reading(test);

continue;

}

if(test!=' ' && test!='\n' && test!='\t'){

printf("Error: wrong value for 'a'. Use 'h' for help.\n");

reading(test);

continue;

}

while(test==' ' || test=='\t'){

test=getchar();

}

if(test=='\n'){

printf("Error: lack of arguments for command 'a'. Use 'h' for help.\n");

continue;

}

if(test>='1' && test<='9'){

ungetc(test,stdin);

scanf("%d",&pos);

test=getchar();

} else if(test=='b'){

pos=l.size+1;

test=getchar();

} else if(test=='f'){

pos=1;

test=getchar();

} else {

printf("Error: wrong position for 'a'. Use 'h' for help.\n");

reading(test);

continue;

}

if(test!=' ' && test!='\n' && test!='\t'){

printf("Error: wrong position for 'a'. Use 'h' for help.\n");

reading(test);

continue;

}

while(test==' ' || test=='\t'){

test=getchar();

}

if(test!='\n'){

printf("Error: wrong number of arguments for 'a'. Use 'h' for help.\n");

reading(test);

} else {

if(pos>l.size+1){

printf("Error: incorrect position for this list.\n");

} else if(l\_insert(&l,val,pos)){

printf("The addition was successful\n");

} else {

printf("Error\n");

}

}

} else if(a=='d' && (test==' ' || test=='\n' || test=='\t')){

if(test!=' ' && test!='\t'){

printf("Error: wrong usage of command 'd'. Use 'h' for help.\n");

continue;

}

while(test==' ' || test=='\t'){

scanf("%c",&test);

}

if(test=='\n'){

printf("Error: lack of arguments for command 'd'. Use 'h' for help.\n");

continue;

}

if(test>='1' && test<='9'){

ungetc(test,stdin);

scanf("%d",&pos);

test=getchar();

} else if(test=='b'){

pos=l.size;

test=getchar();

} else if(test=='f'){

pos=1;

test=getchar();

} else {

printf("Error: wrong position for 'd'. Use 'h' for help.\n");

reading(test);

continue;

}

if(test!=' ' && test!='\n' && test!='\t'){

printf("Error: wrong position for 'd'. Use 'h' for help.\n");

reading(test);

continue;

}

while(test==' ' || test=='\t'){

scanf("%c",&test);

}

if(test!='\n'){

printf("Error: wrong number of arguments for 'd'. Use 'h' for help.\n");

reading(test);

} else {

if(pos>l.size){

printf("Error: incorrect position for this list.\n");

} else {

l\_remove(&l,pos);

printf("The deletion was successful\n");

}

}

} else if(a=='p' && (test==' ' || test=='\n' || test=='\t')){

while(test==' ' || test=='\t'){

scanf("%c",&test);

}

if(test!='\n'){

printf("Error: use 'p' without arguments.\n");

reading(test);

} else {

l\_print(&l);

}

} else if(a=='t' && (test==' ' || test=='\n' || test=='\t')){

if(test!=' ' && test!='\t'){

printf("Error: wrong usage of command 't'. Use 'h' for help.\n");

continue;

}

while(test==' ' || test=='\t'){

scanf("%c",&test);

}

if(test=='\n'){

printf("Error: lack of arguments for command 't'. Use 'h' for help.\n");

continue;

} else if(test>='1' && test<='9'){

ungetc(test,stdin);

scanf("%u",&val);

test=getchar();

} else {

printf("Error: wrong value for 't'. Use 'h' for help.\n");

reading(test);

continue;

}

if(test!=' ' && test!='\n' && test!='\t'){

printf("Error: wrong value for 't'. Use 'h' for help.\n");

reading(test);

continue;

}

while(test==' ' || test=='\t'){

scanf("%c",&test);

}

if(test!='\n'){

printf("Error: wrong number of arguments for command 't'. Use 'h' for help.\n");

reading(test);

continue;

} else {

task(&l,val);

printf("Task done\n");

}

} else {

printf("Wrong key. Use 'h' for help\n");

reading(test);

}

}

l\_destroy(&l);

}

list.h:

#ifndef \_LIST\_H

#define \_LIST\_H

#include <stdbool.h>

#include <stdlib.h>

typedef struct{

unsigned int val;

int next;

} list\_el;

typedef struct{

int first;

int last;

int size;

int cap;

list\_el\* buf;

int top\_empty;

} list;

typedef struct{

int prev;

list \*l;

} iter;

bool l\_create(list \*l);

bool l\_is\_empty(list \*l);

void iter\_swap(iter \*it1, iter \*it2);

iter find\_iter(list \*l, int i);

bool l\_insert(list \*l, unsigned int val, int i);

void l\_remove(list \*l, int i);

void l\_print(list \*l);

void l\_destroy(list \*l);

unsigned int iter\_val(iter it);

iter iter\_next(iter it);

bool iter\_equal(iter it1, iter it2);

iter begin\_iter(list \*l);

iter end\_iter(list \*l);

bool iter\_insert(iter it, unsigned int val);

void iter\_remove(iter \*it);

#endif

list\_fucntions.c:

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include "list.h"

bool l\_create(list \*l){

l->first=-1;

l->last=-1;

l->size=0;

l->cap=10;

l->top\_empty=0;

list\_el \*tmp=malloc(sizeof(list\_el)\*l->cap);

if(tmp==NULL){

return false;

}

l->buf=tmp;

for(int i=0;i<l->cap-1;i++){

l->buf[i].next=i+1;

}

l->buf[l->cap-1].next=-1;

return true;

}

bool l\_is\_empty(list \*l){

return l->size==0;

}

iter find\_iter(list \*l, int i){

if(i==1){

return end\_iter(l);

}

iter tmp=begin\_iter(l);

for(int j=1;j<i-1;j++){

tmp=iter\_next(tmp);

}

return tmp;

}

bool l\_insert(list \*l, unsigned int val, int i){

iter tmp=find\_iter(l,i);

if(!iter\_insert(tmp,val)){

return false;

}

return true;

}

void l\_remove(list \*l, int i){

iter tmp=find\_iter(l,i);

iter\_remove(&tmp);

}

void l\_print(list \*l){

for(iter i=begin\_iter(l);!iter\_equal(i,end\_iter(l));i=iter\_next(i)){

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

}

printf("\n");

}

void l\_destroy(list \*l){

l->first=-1;

l->last=-1;

l->size=0;

l->cap=0;

l->top\_empty=0;

free(l->buf);

}

iter\_functions.c:

#include <stdbool.h>

#include <stdlib.h>

#include "list.h"

unsigned int iter\_val(iter it){

return it.l->buf[it.prev].val;

}

iter iter\_next(iter it){

if(it.prev==-1){

it.prev=it.l->first;

} else{

it.prev=it.l->buf[it.prev].next;

}

return it;

}

void iter\_swap(iter \*it1, iter \*it2){

unsigned int c=iter\_val(\*it1);

it1->l->buf[it1->prev].val=iter\_val(\*it2);

it2->l->buf[it2->prev].val=c;

return;

}

bool iter\_equal(iter it1, iter it2){

return it1.prev==it2.prev;

}

iter begin\_iter(list \*l){

iter res;

res.prev=l->first;

res.l=l;

return res;

}

iter end\_iter(list \*l){

iter res;

res.prev=-1;

res.l=l;

return res;

}

bool grow\_buffer(list \*l){

list\_el \*tmp=realloc(l->buf,sizeof(list\_el)\*l->cap\*3/2);

if(tmp==NULL){

return false;

}

l->buf=tmp;

for(int i=l->cap;i<l->cap\*3/2-1;i++){

l->buf[i].next=i+1;

}

l->top\_empty=l->cap;

l->cap=l->cap\*3/2;

l->buf[l->cap-1].next=-1;

return true;

}

bool iter\_insert(iter it, unsigned int val){

if(it.l->size+1==it.l->cap){

if(!grow\_buffer(it.l)){

return false;

}

}

int tmp=it.l->top\_empty;

it.l->top\_empty=it.l->buf[it.l->top\_empty].next;

it.l->buf[tmp].val=val;

if(iter\_equal(it,end\_iter(it.l))){

it.l->buf[tmp].next=it.l->first;

it.l->first=tmp;

if(l\_is\_empty(it.l)){

it.l->last=tmp;

}

it.l->size++;

return true;

}

if(it.prev==it.l->last){

it.l->last=tmp;

}

it.l->buf[tmp].next=it.l->buf[it.prev].next;

it.l->buf[it.prev].next=tmp;

it.l->size++;

return true;

}

void iter\_remove(iter \*it){

if(iter\_equal(\*it,end\_iter(it->l))){

int tmp=it->l->first;

it->l->first=it->l->buf[tmp].next;

it->l->buf[tmp].next=it->l->top\_empty;

it->l->top\_empty=tmp;

it->l->size--;

if(l\_is\_empty(it->l)){

it->l->last=it->l->first;

}

return;

}

if(it->l->buf[it->prev].next==it->l->last){

it->l->last=it->prev;

}

int tmp=it->l->buf[it->prev].next;

it->l->buf[it->prev].next=it->l->buf[tmp].next;

it->l->buf[tmp].next=it->l->top\_empty;

it->l->top\_empty=tmp;

it->l->size--;

}

# 9.Заключение

Реализация линейного списка на массиве – очень интересный процесс. Добавление и удаление элементов в нём реализуется не так просто, как в обычном массиве, поэтому написание данных алгоритмов требует смекалки. Итераторы оказались очень полезны при написании процедур удаления и вставки элементов, так как через них можно взаимодействовать с самим списком, в отличие от указателей.

# 10.Список используемых источников

1. Методические указания к выполнению курсовых работ. Зайцев В. Е.
2. Что такое двусвязный список? <https://www.youtube.com/watch?v=128BR4KvVtk>