**C3 SDA**

Functiile care implementeaza **stiva si coada** (statice)

- creare , adaugare, stergere

**Adaugare in stiva:**

a - elementul de adaugat

daca stiva nu e plina atunci

incrementeaza top

STACK[top] <- a

[]

altfel Eroare

**Extragere (stergere) in stiva**:

a - elementul de adaugat

daca stiva nu e vida atunci

a= STACK[top]

decrementeaza top

intoarce a

[]

altfel Eroare

Cozi (implementare statica)

coada vida - > front = 0, rear = 0

coada plina -> rear = N-1

Simplificare:

introducerea unui contor de elemente; cnt

coada vida - > cnt=0

coada plina -> cnt= N-1

**inserarea in coada:**

daca coada nu e plina atunci

incrementeza rear

QUEQUE[rear]=element nou

incrementeaza contor

[]

altfel Eroare

**extragerea din coada:**

daca coada nu e vida atunci

incrementeza front

element extras = QUEQUE[rear]

decrementeaza contor

intoarce element extras

[]

altfel Eroare

**Exemplu de implementare stiva si coada (static)**

============================================================================

#include <stdio.h>

#define N 4

// Stiva - implementare statica

**// prototipuri functii stiva**

int stack\_empty(int t);

int stack\_full(int t, int n);

int add\_stack(int s[], int t, int a, int n);

int del\_stack(int s[], int t, int \*a);

**// coada - implementare statica**

int queue\_empty(int c);

int queue\_full(int c);

int add\_queue(int q[], int \*r, int a, int c);

int del\_queue(int q[], int \*f, int \*a,int c);

**// prototipuri functii stiva**

int main() {

int x;

**// definire stiva cu N-1 elemente (static)**

int top=0;

int STACK[N];

**// definire coada cu N-1 elemente (static)**

int front = 0;

int rear = 0;

int cnt = 0;

int QUEUE[N];

**// adaugare element stiva**

top = add\_stack(STACK, top, 1,N);

top = add\_stack(STACK, top, 2,N);

**// extragere element stiva**

top = del\_stack(STACK,top, &x);

**// adaugare element stiva**

top = add\_stack(STACK, top, 3,N);

top = add\_stack(STACK, top, 4,N);

**// adaugare element in coada**

cnt = add\_queue(QUEUE,&rear, 10, cnt );

cnt = add\_queue(QUEUE,&rear, 20, cnt );

**// extragere element coada**

cnt=del\_queue(QUEUE, &front, &x, cnt);

**// adaugare element in coada**

cnt = add\_queue(QUEUE,&rear, 30, cnt );

cnt = add\_queue(QUEUE,&rear, 40, cnt );

return 0;

}

**// intrare virful stivei**

**// intoarce 1 - stiva vida, 0 - stiva nevida**

int stack\_empty(int t)

{

if (t!=0) return 0;

else return 1;

}

**// intrare virful stivei, dimensiunea stivei + 1**

**// intoarce 1 - stiva plina, 0 - stiva nu e plina**

int stack\_full(int t, int n)

{

if (t==n-1) return 1;

else return 0;

}

**// intrare: tabloul care defineste stiva, virful stivei, valoare de adaugat, dimensiunea stivei +1**

**// intoarce nou virf al stivei**

int add\_stack(int s[], int t, int a, int n)

{

int res;

res = stack\_full(t,n);

if (res==0)

{

t++;

s[t]=a;

return t;

}

else return t;

}

**// intrare: tabloul care defineste stiva, virful stivei, pointer la valoare de sters, dimensiunea stivei +1**

**// intoarce nou virf al stivei**

int del\_stack(int s[], int t, int \*a)

{

int res;

res = stack\_empty(t);

if (res==0)

{

\*a= s[t];

t--;

return t;

}

else return t;

}

**// intrare nr de elemente din coada**

**// intoarce 1 daca coada e vida, 0 in rest**

int queue\_empty(int c)

{

if (c==0) return 1;

else return 0;

}

**// intrare nr de elemente din coada**

**// intoarce 1 daca coada e plina, 0 in rest**

int queue\_full(int c)

{

if (c==N-1) return 1;

else return 0;

}

**// intrare tabloul de elemente, indexul rear, elementul de introdus, nr de elemente din coada**

**// intoarce noul numar de elemente din coada**

int add\_queue(int q[], int \*r, int a, int c)

{

int res;

res = queue\_full(c);

if (res==0)

{

\*r = (\*r+1)%N;

q[\*r]=a;

c=(c+1)%N;

return c;

}

else return c;

}

**// intrare tabloul de elemente, indexul front, pointer la elementul extras, nr de elemente din coada**

**// intoarce noul numar de elemente din coada**

int del\_queue(int q[], int \*f, int \*a,int c)

{

int res;

res = queue\_empty(c);

if (res==0)

{

\*f= (\*f+1)%N;

\*a = q[\*f];

c=(c-1)%N;

return c;

}

else return c;

}

============================================================================

**Liste dinamice**

**Definitia unui nod lsi**

struct node {

DATA data;

struct node \* next;

};

struct node\* NOD;

creare nod: alocare dinamica a memoriei si initializarea cimpurilor

struct node\* NOD;

alocare memorie cu malloc

NOD = (struct node\*)malloc(sizeof(struct node));

NOD->data = valoare ...

NOD->next=NULL

**Parcurgere lista (prelucrare)**

atit timp cit nodul curent este diferit de nul

prelucreaza noul curent

avanseaza la nodul urmator

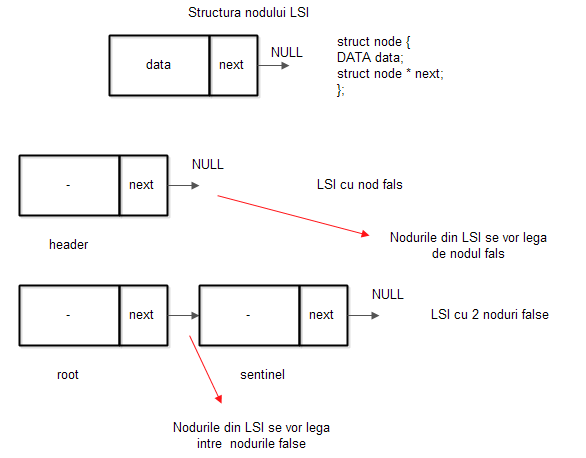
[]

*accesul la elementele lsi (noduri)*

p - pointer la nod

p->data : informatia din nodul p

p->next : urmatorul nod



**C4 SDA**

Ex: lista ordonata, lista circulara, Liste dublu inlantuite, Liste circulare, Exemple de manevrare a listelor

1. **Inserare unui nod dat intr-o lista simplu inlantuita**

- elemente generale:

- se creaza un nod nou (alocare dinamica si actualizare cimpuri)

- se "leaga" nodul nou in lista (se actualizeaza legaturile, in functied e pozitia dorita)

- inserare dupa un nod dat

- **crearea nod nou - malloc**

struct node\* new\_node(int a)

{

struct node\* p;

p=(struct node\*)malloc(sizeof(struct node));

p->data=a;

p->next=NULL;

return p;

}

**// se insereaza un nod nou cu info a dupa un nod dat p**

void insert\_after(struct node\* p, int a)

{

struct node\* q;

if (p!=NULL) {

q=new\_node(a);

q->next=p->next;

p->next=q;

}

}

**- inserare inainte de un nod dat**

**// se insereaza un nod nou cu info a inainte de un nod dat p**

void insert\_before(struct node\* p, int a)

{

struct node\* q;

int tmp;

if (p!=NULL) {

q=new\_node(a);

tmp=p->data;

p->data=q->data;

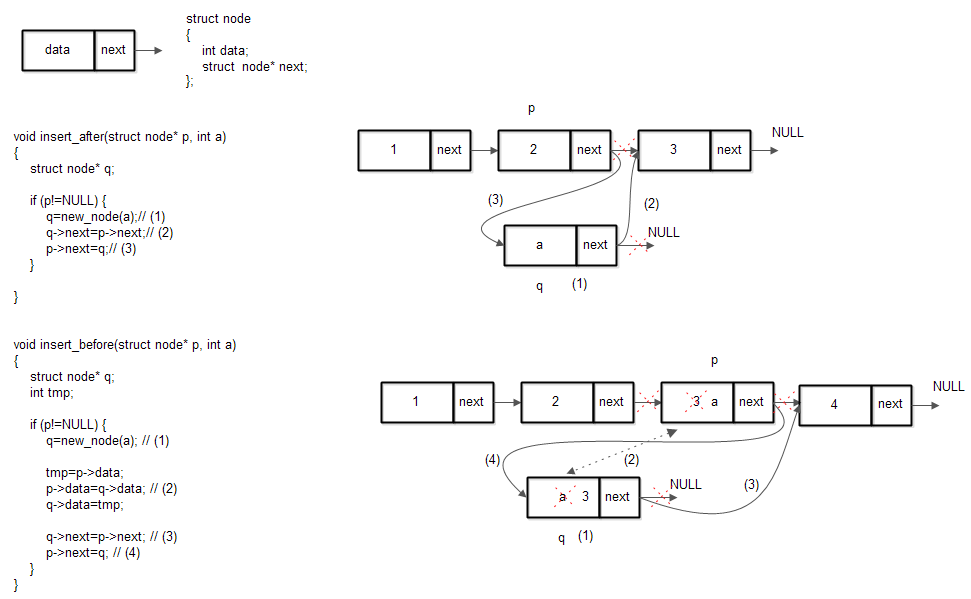
q->data=tmp;

q->next=p->next;

p->next=q;

}

}



***2. Stergerea unui nod dat intr-o lista simplu inlantuita***

- elemente generale:

- se refac legaturile intre noduri, conform pozitiei nodului de sters

- se elibereaza memoria alocata nodului de ster (free)

stergerea unui nod dupa un nod dat

**// se sterge nodul de dupa nodul p**

**// se intoarce info din nodul sters (nu e obligatoriu)**

int del\_after(struct node\* p)

{

struct node\* q;

int r = -1;

if (p != NULL && p->next != NULL)

{

q=p->next;

p->next=q->next;

r=q->data;

free(q);

}

return r;

}

**stergerea unui nod dat**

// se sterge nodul p

// intoarce valoarea cimpului de informatie din nodul sters

int del(struct node\* p)

{

struct node\* q;

int r = -1;

if (p != NULL && p->next != NULL)

{

q=p->next;

r=p->data;

p->data=q->data;

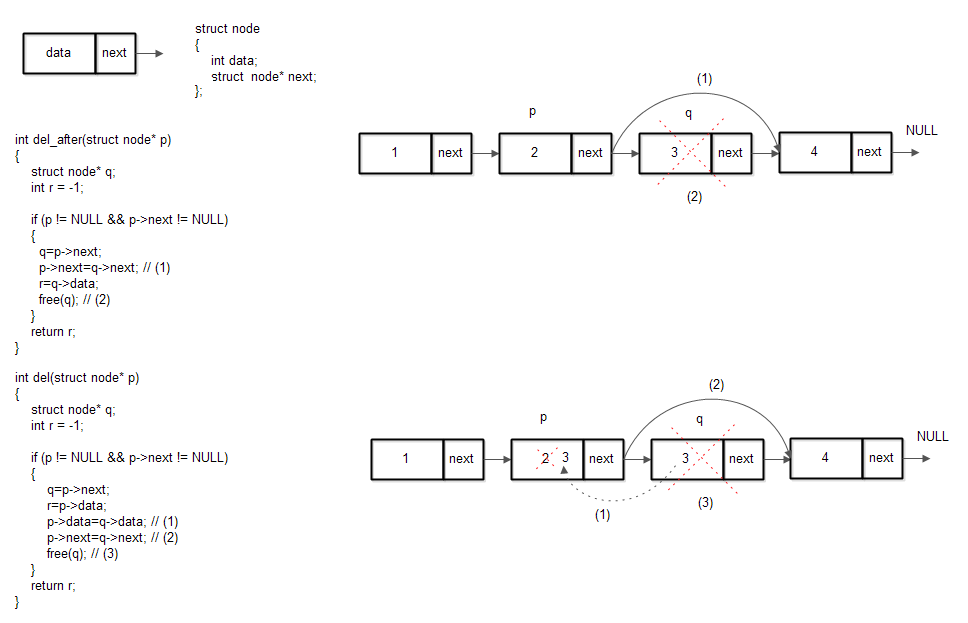
p->next=q->next;

free(q);

}

return r;

}



***3. Stive si cozi - implementate ca liste simplu inlantuite***

stiva vida - top=NULL

**inserare in stiva**

// la intrare - top stiva si val nodul de inserat

// intoarce noul top

struct node\* add\_stack(struct node\* t, int a)

{

struct node\* p;

if(t==NULL)

{

t=new\_node(a);

t->next=NULL;

}

else

{

p=new\_node(a);

p->next=t;

t=p;

}

return t;

}

**extragere din stiva**

// intrare: virful stivei, pointer la un element unde se va scrie valoarea extrasa

// intoarce noul virf al stivei

struct node\* del\_stack(struct node\* t, int \*a)

{

struct node\* p;

if (t!=NULL)

{

\*a=t->data;

p=t;

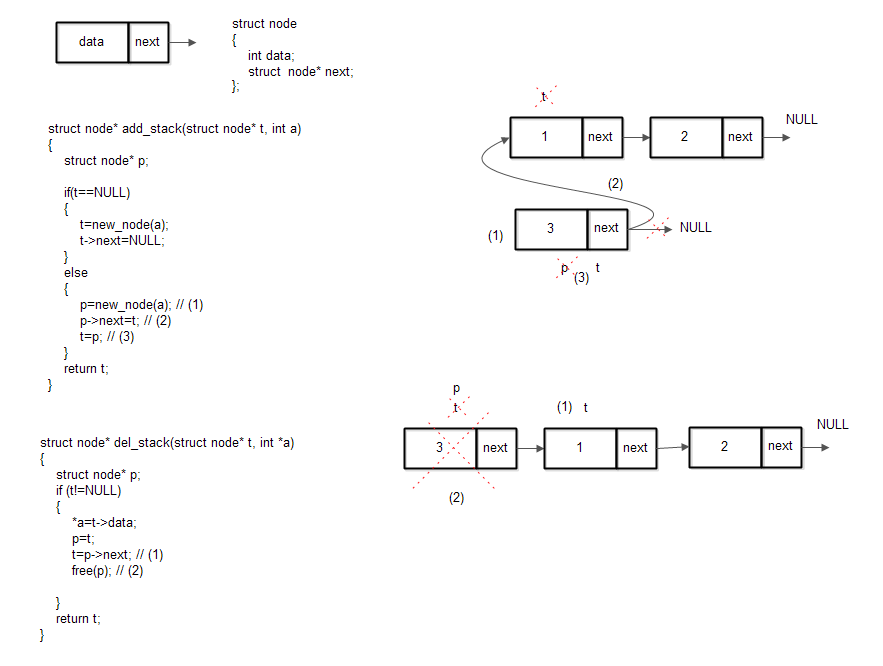
t=p->next;

free(p);

}

return t;

}



**Crearea cozii vide (cu 2 noduri false)**

Coada vida -> front->next = rear

**Inserare in coada**

// intrare -> rear si info din nodul nou

// nu intoarce nimic

void add\_queue(struct node\* r, int a)

{

struct node\* p;

r->data = a;

p=new\_node(-1);

r->next=p;

r=p;

}

**Extragere din coada**

// intrari front si rear

// iesire -> valoarea cimpului de info a nodului extras

int del\_queue(struct node\* f, struct node\* r)

{

struct node\* p;

int x;

if (f->next != r )

{

p=f->next;

x=p->data;

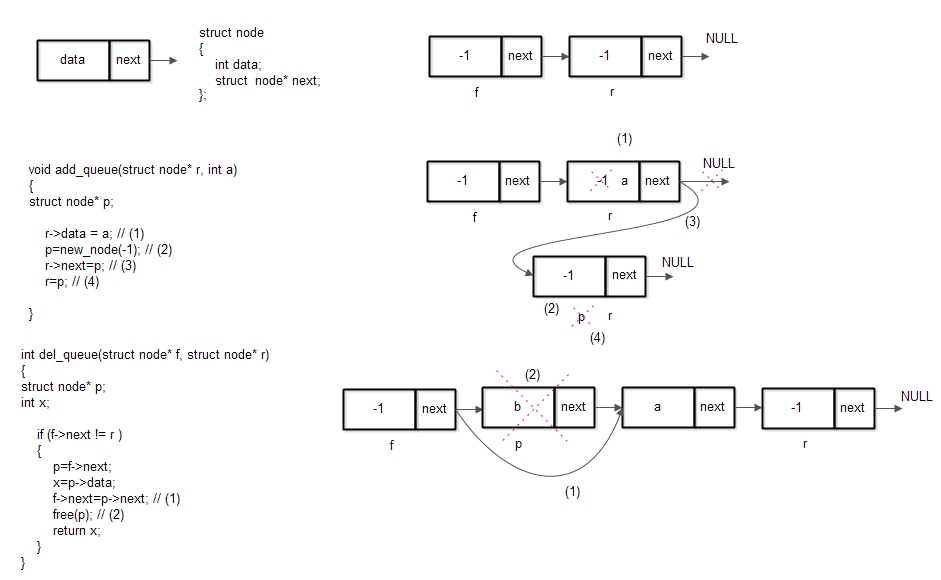
f->next=p->next;

free(p);

return x;

}

}



4. Alte exemple de manevrare cu lsi

- liste ordonate (inainte de inserare se cauta pozitia nodului conform unei relatii de ordine date - uzual ordine crescatoare

**Exemplu: implementare cu doua noduri false (root si sentinel)**

void ins\_ord(struct node\* r, struct node\* s, int a)

{

struct node\* x;

struct node\* y;

struct node\* z;

x=r;

y=x->next;

s->data=a;

while (y->data < a)

{

x=y;

y=x->next;

}

if (y->data = a && y != s)

{

// nod identic

}

else

{

z = new\_node(a);

z->next = y;

x->next = z;

}

}

*Liste circulare -> ultimul nod e legat cu primul nod*

**Liste dublu inlantuite (ldi)**

- extindere a conceptului de lista dinamica (lsi) -> introducerea unui nou cimp de legatura

next la lsi -> o singura directie

left si right -> 2 directii

**nod nou**

// intare - valoare cimp info nod nou

// intoarce - un pointer la nodul nou creat

struct dnode\* new\_dnode(int a)

{

struct dnode\* p;

p=(struct dnode\*)malloc(sizeof(struct dnode));

p->info = a;

p->left = NULL;

p->right = NULL;

return p;

}

**aduugare la dreapta unui nod**

// intrari - pointer la nodul la dreapta caruia inseram, val info nod nou

void add\_r\_dlist( struct dnode\* p, int a)

{

struct dnode\* h;

if (p!=NULL)

{ h=new\_dnode(a);

h->right=p->right;

h->left=p;

p->right->left=h;

p->right=h;

}

}

**adaugare la stinga -> similar (right <-> left)**

**stergere unui nod**

// intrare pointer la nodul de sters

// intoarce - info din nodul sters (nu e obligatoriu)

int del\_dlist( struct dnode\* p)

{

int x=-1;

if (p!=NULL)

{

p->left->right=p->right;

p->right->left=p->left;

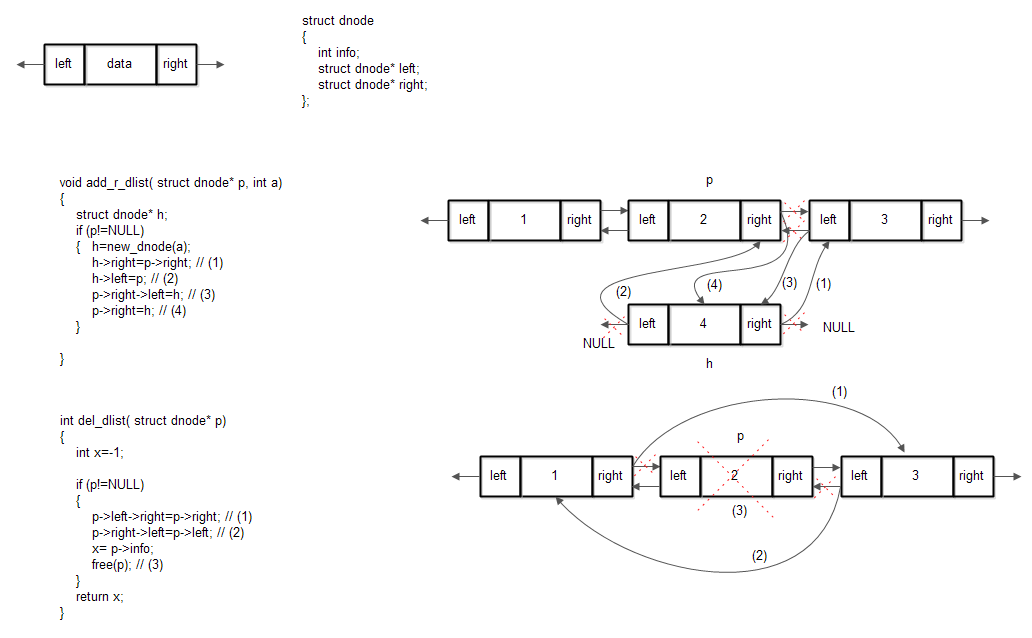
x= p->info;

free(p);

}

return x;

}



**stergeri la dreapta sau la stinga**

- se va folosi functia de stergere a nod dat p (apelata cu p->right sau p->left)

**Obs** la ldi- parcurgerile se pot face la stinga sau la dreapta

**Parcurgere lista ldi (prelucrare)**

atit timp cit nodul curent este diferit de nul

prelucreaza noul curent

avanseaza la nodul urmator (catre una din directiile posibile)

[]