

2

STRUCTURA DE LISTĂ ÎN LIMBAJUL PASCAL

Unul din principalele atribute ale datelor este structura sau modul de organizare care le caracterizează. Structurile rezultă prin gruparea într-un anumit mod a colecțiilor de date primitive. Exemple de structuri de date sunt: tablourile, mulțimile, înregistrările. În continuare vor fi prezentate noi tipuri de organizări, și anume: listele simplu și dublu înălțuite și cazurile particulare cunoscute sub numele de stive și cozi.

2.1 Noțiuni introductive

Un tip de structură de date poate fi gândit logic, făcând abstracție de modul particular în care datele sunt reprezentate în memoria calculatorului. El se reprezintă particularizat prin diverse implementări din limbajele de programare.

Specificarea unui anume tip de organizare pentru datele procesate de un program presupune definirea mulțimii de valori permise pe care o variabilă sau parametru aparținând acestui tip de date le poate avea, precum și a mulțimii de operații care pot fi efectuate cu astfel de date. Eficiența unei scheme de prelucrare a datelor este determinată în mare măsură de tipurile particulare de date utilizate.

Limbajul de programare Pascal oferă suficientă flexibilitate pentru reprezentarea unor tipuri complexe de date, dar controlul corectitudinii modului în care sunt utilizate revine programatorului. Structurile de date Pascal pentru reprezentarea colecțiilor de date pot fi clasificate în structuri statice și dinamice. Structurile statice cuprind: *array*, *set* și *file* pentru reprezentarea datelor de același tip (omogene) și *record*, pentru reprezentarea datelor eterogene. Structurile dinamice se construiesc pe baza tipurilor de date Pascal *referință legată* și *referință liberă* (desemnată prin tipul predefinit *pointer*).

Definirea unei date reprezentată printr-o structură statică presupune alocarea unui spațiu de memorie de dimensiune invariabilă pe durata evoluției programului, în timp ce dimensiunea unei date reprezentată prin intermediul unei structuri dinamice poate fi modificată pe durata execuției.

Pentru anumite clase de probleme, structurile statice de date nu numai că nu sunt suficiente, dar se dovedesc chiar imposibil de folosit, datorită limitărilor la care sunt supuse. În primul rând, spațiul de memorie aferent unor astfel de date se definește și se rezervă în momentul compilării programului (rezervare statică), la o dimensiune maximă (cuprinzătoare). Acest spațiu nu poate fi disponibilizat și nici împărțit cu alte date, chiar dacă nu este în întregime utilizat în anumite momente ale execuției programului. În al doilea rând, componentele structurilor statice ocupă locuri prestabilite în spațiul rezervat, determinate de relația de ordonare specifică fiecărei structuri. În al treilea rând, limbajul definește operațiile admise cu valorile componentelor, potrivit tipului de bază al structurii, astfel încât numărul maxim și ordinea componentelor structurii nu pot fi modificate.

În aceste condiții, structurile statice sunt dificil de utilizat în rezolvarea problemelor care prelucrează mulțimi de date pentru care numărul și ordinea componentelor se modifică frecvent în timpul execuției programului. Pentru astfel de situații, există posibilitatea utilizării datelor de tip dinamic, cărora li se pot aloca și elibera zone de memorie pe parcursul execuției programului.

2.2 Structura de listă

Organizarea de tip listă corespunde unei structurări lineare a datelor, la nivelul fiecărei componente dispunându-se de informație suficientă pentru identificarea următoarei componente a colecției. Datele dintr-o colecție astfel structurată sunt referite de obicei prin termenii de noduri, celule, componente etc.

Reprezentarea unei liste în limbajul Pascal poate fi realizată prin intermediul structurii de date *array*, ordinea componentelor fiind dată de ordinea pe domeniul valorilor corespunzătoare indexării și, în consecință, următoarea componentă este implicit specificată. Memorarea unei mulțimi de date $\{d_1, d_2, \dots, d_n\}$ prin intermediul unei structuri statice poate fi realizată în limbajul Pascal utilizând un masiv unidimensional.

Definirea tipului de date Pascal pentru memorarea ca listă a datelor $\{d_1, d_2, \dots, d_n\}$ este:

```
const max=500;
```

```
type lst=array[1..max] of tip_informatie;
```

unde $n \leq \text{max}$ și *tip_informatie* este numele tipului de date Pascal utilizat pentru memorarea fiecărei date din mulțimea $\{d_1, d_2, \dots, d_n\}$. Dacă *lista* este variabila de tip *lst* utilizată pentru memorarea colecției $\{d_1, d_2, \dots, d_n\}$, atunci data d_i este memorată în componenta *lista[i]*, $1 \leq i \leq n$.

Dezavantajele majore în utilizarea reprezentării statice rezidă în volumul de calcule necesare efectuării operațiilor de inserție/eliminare de noduri și în necesitatea păstrării unei zone de memorie alocată, indiferent de lungimea efectivă a listei. Aceste dezavantaje sunt eliminate prin opțiunea de utilizare a structurilor dinamice. Componentele unei liste dinamice sunt eterogene, fiecare nod conținând o parte de informație și câmpuri de legătură care permit identificarea celulelor vecine. Câmpurile de legătură sunt reprezentate de date de tip referință (adresă). În cazul listelor cu un singur câmp de legătură (simplu înlanțuite), valoarea câmpului indică adresa nodului următor, în timp ce, în cazul listelor cu dublă legătură (dublu înlanțuite), valorile memorate în câmpurile de legătură sunt adresele componentelor care preced și, respectiv, urmează celulei. În ambele situații, câmpul de legătură pentru indicarea celulei următoare corespunzător ultimei componente a listei are valoarea *nil* în cazul listelor “deschise” (lineare) și respectiv indică adresa primei componente din listă în cazul listelor “închise” (circular).

Se presupune că mulțimea de date $\{d_1, d_2, \dots, d_n\}$ este memorată ca listă lineară. Convențional, structura rezultată poate fi reprezentată grafic fie prin a) (listă simplu înlanțuită), fie prin b) (listă dublu înlanțuită), unde simbolul \equiv desemnează valoarea *nil* a câmpului de legătură (figura 2.1). În cazul în care pentru memorarea datelor $\{d_1, d_2, \dots, d_n\}$ se dorește utilizarea unei structuri de listă circulară, reprezentarea grafică este dată în figura 2.2.

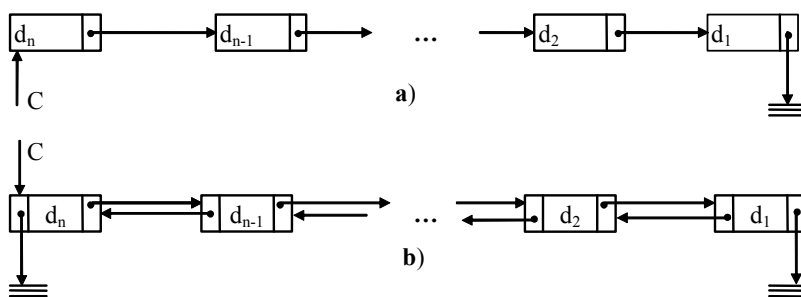


Fig. 2.1 Liste lineare

Declarările tipurilor de date Pascal pentru definirea structurilor de liste dinamice simplu și, respectiv, dublu înlanțuite sunt:

a) Listă simplu înlanțuită

b) Listă dublu înlanțuită

```
type lista=^nod;
nod=record
    inf:tip_informatie;
    leg: lista;
end;
```

```
type lista=^nod;
nod=record
    inf:tip_informatie;
    st,dr: lista;
end;
```

unde *tip_informatie* este numele tipului de date Pascal utilizat pentru memorarea fiecărei date din mulțimea $\{d_1, d_2, \dots, d_n\}$.

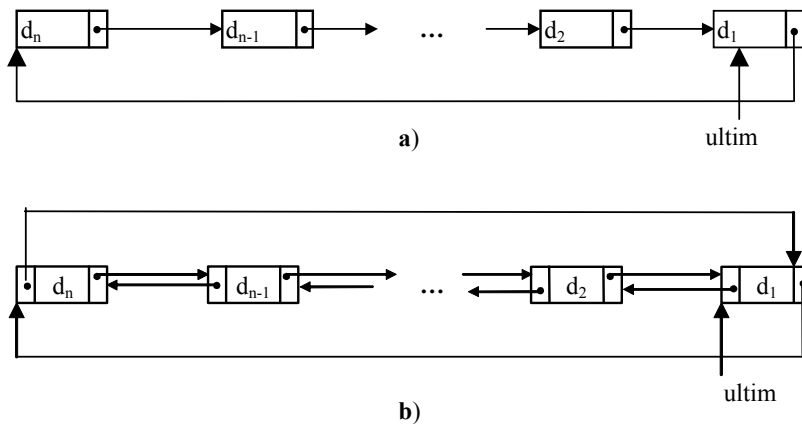


Fig. 2.2 Liste circulare

2.3 Operații primitive asupra listelor

Accesul la informația stocată într-o variabilă de tip listă revine efectuând una sau mai multe dintre operațiile primitive: regăsirea nodului (dacă există) care corespunde unei chei date (condiție impusă asupra valorii câmpului de informație), inserarea unei noi componente în listă, eliminarea componentei (componentelor) cu proprietatea că valorile câmpurilor de informație satisfac o anumită cerință și înlocuirea câmpului de informație corespunzător unei componente printr-o informație dată.

Accesarea componentelor unei liste reprezentată printr-o structură statică poate fi realizată atât secvențial, cât și direct, utilizând valorile indicelui considerat pentru indexare, în timp ce accesarea componentelor unei liste dinamice se realizează, de regulă, numai secvențial, începând cu prima componentă și continuând cu următoarele, pe baza valorilor câmpurilor de legătură. Convențional, numim *cap* al listei dinamice pointerul a cărui valoare este adresa primei componente a listei.

1. *Parcurerea integrală a datelor memorate într-o listă.* Se presupune că declarațiile de tip pentru definirea structurilor de liste menționate anterior sunt globale, relativ la procedurile descrise în continuare.

a) Lista reprezentată prin structură statică (masiv unidimensional)

```
procedure parcurgere1(var l:lst; n:word);
var i:word;
begin
    for i:=1 to n do
        prelucrare(l[i]);
    end;
```

b) Lista reprezentată prin structură dinamică simplu înlănțuită

```
procedure parcurgere2( var cap:lista);
var p:lista;
begin
    p:=cap;
    while(p<>nil) do
        begin
            prelucrare(p^.inf);
            p:=p^.leg;
        end;
    end;
```

c) Lista reprezentată prin structură dinamică dublu înlănțuită

```
procedure parcurgere3( var cap:lista);
var p:lista;
begin
    p:=cap;
    while(p<>nil) do
        begin
            prelucrare(p^.inf);
            p:=p^.dr;
        end;
    end;
```

2. Regăsirea unei date *d* într-o colecție memorată într-o listă

În continuare se presupune că funcția *egal(a,b:tip_informatie):boolean* returnează valoarea *true* dacă *a* și *b* coincid, altfel returnează *false*. Funcțiile Pascal *exista1*, *exista2* și *exista3* calculează valoarea *true*, dacă data *d* se află în colecția memorată în lista dată ca argument, altfel calculează valoarea *false*.

a) Lista reprezentată prin structură statică (masiv unidimensional)

```
function exista1(l:lst; n:word; d:tip_informatie):boolean;
var i:word;
    c:boolean;
begin
    c:=false;
    i:=1;
    while(i<=n)and(not c) do
        if egal(d,l[i]) then c:=true
        else i:=i+1;
    end;
    exista1:=c;
```

b) Lista reprezentată prin structură dinamică simplu înlănțuită

```
function exista2(cap:lista;d:tip_informatie):boolean;
var p:lista;
```

```

        c:boolean;
begin
    p:=cap;
    c:=false;
    while(p<>nil)and(not c) do
        if egal(d,p^.inf) then c:=true
        else p:=p^.leg;
    exista2:=c;
end;

```

c) Lista reprezentată prin structură dinamică dublu înălănțuită

```

function exista3(cap:lista;d:tip_informatie):boolean;
var p:lista;
    c:boolean;
begin
    p:=cap;
    c:=false;
    while(p<>nil)and(not c) do
        if egal(d,p^.inf) then c:=true
        else p:=p^.dr;
    exista3:=c;
end;

```

3. Inserarea unei date, *d*, într-o listă

Includerea unei noi componente într-o listă poate fi realizată, în funcție de cerințele problemei particulare, la începutul listei, după ultima componentă din listă, înaintea/după o componentă cu proprietatea că valoarea câmpului de informație îndeplinește o anumită condiție. Presupunem că funcția *condiție* (*a,b:tip_informatie*):*boolean* calculează valoarea *true* dacă *a* și *b* verifică condiția formulată pentru inserare, altfel calculează valoarea *false*. De asemenea, procedura *atribuie*(*var destinatie:tip_informatie; sursa:tip_informatie*) realizează copierea datei *sursa* în *destinatie*.

Deoarece prin inserarea unei componente se poate ajunge la depășirea spațiului disponibil de memorie, este necesară verificarea prealabilă a posibilității realizării operației (dacă se poate alocă spațiu de memorie pentru componenta de inserat). În cazul listelor reprezentate prin structuri statice, această verificare va fi realizată prin compararea lungimii efective *n* a listei cu dimensiunea declarată *max*. În cazul listelor dinamice, este utilizată funcția Pascal predefinită *MaxAvail*, care calculează lungimea maximă a zonelor nealocate contigue din memoria Heap. Valoarea returnată pentru parametrul *test* este *true*, dacă inserarea a fost posibilă, altfel este *false*.

De exemplu, inserarea la începutul listei decurge astfel:

a) Lista reprezentată prin structură statică (masiv unidimensional)

```

procedure inserare_la_inceput1(var l:lst; var n:word;
d:tip_informatie;var test:boolean);

```

```
var i:word;
begin
test:=true;
if n=max then test:=false
else
    begin
        for i:=n downto 1 do atribuire(l[i+1],l[i]);
            atribuire(l[1],d);
            n:=n+1;
        end;
    end;
end;
```

b) Lista reprezentată prin structură dinamică simplu înlănțuită

```
procedure inserare_la_inceput2(var
cap:lista;d:tip_informatie;var test:boolean);
var p:lista;
begin
test:=true;
if MaxAvail<sizeof(nod) then test:=false
else
    begin
        new(p);
        atribuire(p^.inf,d);
        p^.leg:=cap;
        cap:=p;
    end;
end;
```

c) Lista reprezentată prin structură dinamică dublu înlănțuită

```
procedure inserare_la_inceput3(var
cap:lista;d:tip_informatie;var test:boolean);
var p:lista;
begin
test:=true;
if MaxAvail<sizeof(nod) then test:=false
else
    begin
        new(p);
        atribuire(p^.inf,d);
        p^.dr:=cap;p^.st:=nil;
        cap:=p;
    end;
end;
```

Procedurile de inserare la începutul listei pot fi folosite și pentru crearea unei liste în care să fie memorată o colecție de date $\{d_1, d_2, \dots, d_n\}$. Procedura *copiază*(var *d:tip_informatie*; var *i:word*) realizează copierea în *d* a datei *d_i* din colecția considerată. Utilizând procedurile de inserare deja descrise, operația de creare a unei liste poate fi realizată astfel:

a) Lista reprezentată prin structură statică (masiv unidimensional)

```
procedure creeazal(var l:lst; n:word; var test:boolean);
var i,j:word;
```

```
      d:tip_informatie;
begin
  test:=true;
  if n>max then test:=false
  else
    begin
      i:=1;
      while(i<=n) do
        begin
          copiaza(d,i);
          inserare_la_inceput1(l,i,d,test)
        end;
      end;
    end;
end;
```

b) Lista reprezentată prin structură dinamică simplu înlănțuită

```
procedure creeaza2(var cap:lista;n:word;var test:boolean);
var i:word;
begin
  i:=0; cap:=nil;
  test:=true;
  while(i<n)and(test) do
    begin
      i:=i+1; copiaza(d,i);
      inserare_la_inceput2(cap,d,test);
    end;
end;
```

4. Eliminarea (ștergerea) unei date, *d*, dintr-o listă

Modificarea conținutului unei liste prin eliminarea uneia sau mai multor componente poate fi descrisă secvențial, astfel încât este suficient să dispunem de o procedură care realizează eliminarea unei singure componente.

Criteriile de eliminare pot fi formulate diferit, cele mai uzuale fiind: prima componentă, ultima componentă, prima componentă care îndeplinește o anumită condiție, respectiv componenta ce precede/urmează prima componentă care îndeplinește o condiție dată.

În aceste cazuri este necesară verificarea existenței în lista considerată a componentei ce trebuie eliminată. Verificarea asigură și testarea faptului că lista prelucrată este vidă sau nu. Analog operațiilor de inserare, în cadrul procedurilor următoare, parametrul *test* returnează valoarea *true*, dacă eliminarea este efectivă, altfel returnează *false*.

De exemplu, eliminarea primei componente dintr-o listă are loc astfel:

a) Lista reprezentată prin structură statică (masiv unidimensional)

```
procedure elimina_prima(var l:lst; var n:word; var
d:tip_informatie;var test:boolean);
var i:word;
begin
  test:=true;
  if n=0 then test:=false
  else
    begin
      atribuire(d,l[1]);
```



```
        for i:=1 to n-1 do atribuire(l[i],l[i+1]);
        n:=n-1;
    end;
end;
```

b) Lista reprezentată prin structură dinamică simplu înlănțuită

```
procedure elimina_prima2(var cap:lista;var d:tip_informatie;var
test:boolean);
var p:lista;
begin
    test:=true;
    if cap=nil then test:=false
    else
        begin
            p:=cap;
            cap:=cap^.leg;
            atribuire(d,p^.inf);
            dispose(p);
        end;
    end;
end;
```

c) Lista reprezentată prin structură dinamică dublu înlănțuită

```
procedure elimina_prima3(var cap:lista;var d:tip_informatie;var
test:boolean);
var p:lista;
begin
    test:=true;
    if cap=nil then test:=false
    else
        begin
            p:=cap;
            cap:=cap^.dr;
            cap^.st:=nil;
            atribuire(d,p^.inf);
            dispose(p);
        end;
    end;
end;
```

2.4 Liste circulare

În anumite cazuri, este preferabilă renunțarea la structura de tip linear a listelor și utilizarea unei legături de la ultima componentă către capul listei, rezultând ceea ce se numește *listă circulară* (figura 2.2).

Avantajul utilizării acestui tip de structură este posibilitatea de accesare dintr-un element al listei a oricărui alt element. În continuare sunt prezentate module pentru realizarea unor operații de bază în lucrul cu liste circulare.

```
type
    tip_informatie=string;
    clista=^nod;
    nod=record
        inf:tip_informatie;
        leg:clista;
    end;
```

```
procedure atribuire(var d:tip_informatie;s:tip_informatie);
begin
d:=s;
end;
```

```
procedure inserare_la_inceput(var
ultim:clista;d:tip_informatie;var test:boolean);
var
    p:clista;
    cap:clista;
begin
    test:=true;
    if MaxAvail<sizeof(nod) then test:=false
    else
        begin
            new(p);
            atribuire(p^.inf,d);
            if ultim=nil then
                begin
                    ultim:=p;
                    ultim^.leg:=ultim;
                end
            else
                begin
                    cap:=ultim^.leg;
                    p^.leg:=cap;
                    ultim^.leg:=p;
                end;
            end;
        end;
end;
```

```
procedure inserare_la_sfarsit(var
ultim:clista;d:tip_informatie;var test:boolean);
var
    p:clista;
    cap:clista;
begin
    test:=true;
    if MaxAvail<sizeof(nod) then test:=false
    else
        begin
            new(p);
            atribuire(p^.inf,d);
            if ultim=nil then
                begin
                    ultim:=p;
                    ultim^.leg:=ultim;
                end
            else
                begin
                    cap:=ultim^.leg;
                    p^.leg:=cap;
                    ultim^.leg:=p;
                    ultim:=p;
                end;
            end;
        end;
end;
```

```
procedure elimina_la_inceput(var ultim:clista;var
d:tip_informatie;var test:boolean);
var
    cap:clista;
```

```
begin
    test:=true;
    if ultim=nil then test:=false
    else
        begin
            cap:=ultim^.leg;
            atribuire(d, cap^.inf);
            if cap=ultim then
                begin
                    dispose(ultim);
                    ultim:=nil;
                end
            else
                begin
                    ultim^.leg:=cap^.leg;
                    dispose(cap);
                end;
            end;
        end;
end;

procedure elimina_la_sfarsit(var ultim:clista;var
d:tip_informatie;var test:boolean);
var
    p:clista;
    cap:clista;
begin
    test:=true;
    if ultim=nil then test:=false
    else
        begin
            cap:=ultim^.leg;
            atribuire(d, ultim^.inf);
            if cap=ultim then
                begin
                    dispose(ultim);
                    ultim:=nil;
                end
            else
                begin
                    p:=cap;
                    while(p^.leg<>ultim) do p:=p^.leg;
                    p^.leg:=cap;
                    dispose(ultim);
                    ultim:=p;
                end;
            end;
        end;
end;

procedure afisare(ultim:clista);
var p, cap:clista;
begin
    if ultim=nil then writeln('Lista vida')
    else
        begin
            cap:=ultim^.leg;
            p:=cap;
            repeat
                write(p^.inf, ' ');
                p:=p^.leg;
            until p=cap;
            writeln;
        end;
    end;
end;
```

2.5 Stive și cozi

Accesul la informația memorată într-o listă lineară pentru efectuarea operațiilor de inserare și eliminare este permis la oricare dintre componentele colecției. Așa după cum a rezultat în §2.3, identificarea poziției în care trebuie efectuată inserarea/eliminarea presupune inițierea unei secvențe de operații de căutare, ceea ce determină creșterea timpului de lucru. Pe de altă parte, o mulțime de aplicații pot fi modelate utilizând liste lineare în care introducerea și, respectiv, eliminarea informațiilor sunt permise numai la capete. Astfel au fost definite tipurile de listă *stivă* și *coadă*, care impun un tip de organizare a aplicării operațiilor de inserare și eliminare.

Stiva

Se numește *stivă* o listă organizată astfel încât operațiile de inserare și eliminare sunt permise numai la prima componentă. Acest mod de organizare corespunde unei gestiuni LIFO (Last In First Out) a informației stocate.

Modelul corespunde unei stive de cărți. Adăugarea unei noi cărți în stivă se face deasupra primei cărți, iar extragerea este posibilă numai pentru prima carte.

Operațiile de inserare și eliminare într-o stivă pot fi descrise prin intermediul procedurilor *inserare_la_inceput1*, *inserare_la_inceput2*, *elimina_prima1*, *elimina_prima2*. Operațiile elementare pentru gestiunea informației memorate într-o stivă sunt:

- *push(S,d,test)* – inserarea informației *d* în stiva *S*;
- *pop(d,S,test)* – preluarea cu eliminare a informației memorate în prima celulă a stivei *S*;
- *top(d,S,test)* – preluarea fără eliminare a informației deținute de prima componentă a stivei *S*.

Parametrul *test* returnează *true* dacă operația este posibilă, altfel *false*.

Coadă

Se numește *coadă* o listă organizată astfel încât operația de inserare este permisă la ultima componentă, iar operația de eliminare este permisă numai la prima componentă. Acest mod de organizare corespunde unei gestiuni FIFO (First In First Out) a informației stocate.

Modelul corespunde unei cozi de așteptare la un magazin. O nouă persoană se așază la coadă după ultimul cumpărător, iar persoana care își achită nota de plată (primul cumpărător) părăsește coada.

Implementarea unei liste coadă poate fi efectuată atât printr-o structură statică (masiv unidimensional), cât și printr-o structură dinamică de tip listă. În scopul eficientizării operațiilor de inserare/extragere, în cazul implementării cozilor prin structuri dinamice lineare, este necesară utilizarea a două informații: adresa

primei componente și adresa ultimei componente. Aceste informații pot fi menținute explicit prin utilizarea a doi pointeri sau prin utilizarea unui pointer și a unei structuri de listă circulară.

1. *Operațiile de inserare și eliminare pentru listă coadă reprezentată static* sunt descrise de procedurile *inserare_la_sfarsit1* și *eliminar_la_inceput* prezentate în cadrul secțiunii §2.3.

2. *Reprezentarea unei liste coadă printr-o structură dinamică circulară* utilizează o singură variabilă adresă, *ultim*, pentru referirea ultimei componente a listei. Operațiile de inserare și eliminare sunt descrise de procedurile *inserare_coadă* și *eliminar_coadă*.

```
procedure inserare_coadă(var ultim:lista; d:tip_informatie; var
test:boolean);
var p:lista;
begin
    test:=true;
    if MaxAvail<sizeof(nod) then test:=false
    else
        begin
            new(p);
            atribuire(p^.inf,d);
            if ultim=nil then
                begin
                    ultim:=p;
                    ultim^.leg:=ultim;
                end
            else
                begin
                    p^.leg:=ultim^.leg;
                    ultim^.leg:=p;
                    ultim:=p;
                end;
            end;
        end;
    end;
procedure elimina_coadă(var ultim:lista; var d:tip_informatie;
var test:boolean);
var prim:lista;
begin
    test:=true;
    if ultim=nil then test:=false
    else
        begin
            prim:=ultim^.leg;
            atribuire(d,prim^.inf);
            if ultim^.leg=ultim then
                begin
                    dispose(ultim);
                    ultim:=nil;
                end
            else
                begin
                    ultim^.leg:=prim^.leg;
                    dispose(prim);
                end;
            end;
        end;
    end;
end;
```