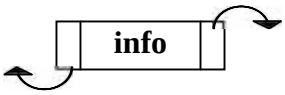
LABORATOR 4

LISTA DUBLU ÎNLĂNŢUITĂ

Lista dublu înlănţuită este formată din noduri care conţin:

- informaţie
- adresa următorului nod
- adresa precedentului nod



Avantajul utilizării listelor dublu înlănţuite rezultă din posibilitatea parcurgerii (traversării) listei în ambele sensuri: de la primul la ultimul, respectiv, de la ultimul la primul nod. Acest lucru permite o manipulare mai flexibilă a nodurilor listei

Structura unui nod al listei dublu înlănţuite este următoarea:

```
struct nod
        {
                //declaraţii de date – câmpuri ale informaţiei
                struct nod *urm; //adresa următorului nod
                struct nod *prec; //adresa nodului precedent
        }
```

În exemplele următoare vom utiliza un tip de date utilizator prin care specificăm tipul nodurilor listei:

```
typedef struct nod
    {int cheie;
    …..//alte câmpuri
    struct nod *pre;
    struct nod* urm;
    }tnod;
```

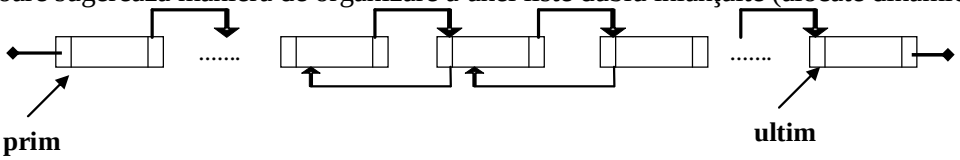
Ca şi la lista simplu înlănţuită, principalele operaţii sunt:

- crearea;
- accesul la un nod; parcurgerea listei
- adăugarea unui nod;
- ştergerea unui nod,
- ştergerea listei.

Gestionarea unei liste dublu înlănţuite se face în maniera similară listelor simplu înlănţuite prin adresele nodurilor *prim* şi *ultim*. În plus, nodul prim se marchează prin stabilirea adresei precedentului său la Null: **prim->prec=0**.

```
tnod *prim,*ultim;
```

Figura următoare sugerează maniera de organizare a unei liste dublu înlănţuite (alocate dinamic):



Crearea listei dublu înlănţuită

Crearea listei vide presupune iniţializarea celor doi pointer de control prim şi ultim cu valoarea 0 (Null): **prim=ultim=0**. Crearea unei liste cu mai mult de un nod se rezumă la apelul repetat al subrutinelor de adăugare a unui nou nod (înaintea primului sau după ultimul nod).

Funcţia următoare este un exemplu prin care se poate crea o listă dublu înlănţuită prin adăugarea noilor noduri după ultimul nod. Un caz particular al procedurii de adăugare a noului nod este tratat distinct: în situaţia în care lista este vidă, după adăugarea unui nod trebuie marcate nodurile *prim* şi *ultim*. Funcţia incarca_nod este cea definită în capitolul dedicat listelor simplu înlănţuite.

```
void creare()
     {
     tnod *p;
```

```
int rasp;
//creare lista vida
prim=ultim=0;
printf("\nIntroduceti? (1/0)");scanf("%d",&rasp);
while (rasp==1)
  {
 p=incarca_nod(); //alocare memorie şi încărcare nod
 if (prim==0)
       {//creare primul nod
       prim=p;ultim=p;
       prim->pre=0; ultim->urm=0; //marcare capete listă
       }
 else
       {
       ultim->urm=p;
       p->pre=ultim;
       ultim=p;
       ultim->urm=0;
       }
 printf("\nIntroduceti? (1/0)");scanf("%d",&rasp);
  }}
```

Parcurgerea listei dublu înlănţuită

Spre deosebire de listele simplu înlănţuite, listele dublu înlănţuite pot fi parcurse în ambele sensuri. Prezentăm în continuare funcţii de afişare a informaţiei nodurilor parcurse în două variante: prin folosirea legăturii următor (*urm*), respectiv, succesor (*prec*).

```
void tiparireDirecta()
{
tnod *p;
if (prim==0)
        {printf("\nLista e vida!");
        return;
        }
p=prim; //iniţializare adresă nod curent
while(p!=0)
  {
   printf("\n %d",p->cheie);
   p=p->urm; //trece la următorul nod
  }
}
```

```
void tiparireInversa()
{
tnod *p;
if (prim==0)
        {printf("\nLista e vida!");
        return;
        }
p=ultim; //iniţializare adresă nod curent
while(p!=0)
  {
   printf("\n %d",p->cheie);
   p=p->prec; //trece la precedentul nod
  }
}
```

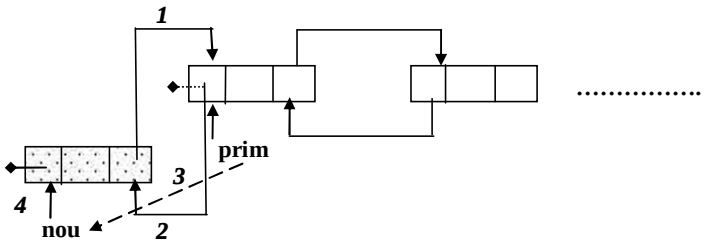
ADĂUGAREA UNUI NOU NOD

Sunt tratate în continuare diferite modalităţi de adăugare a unui nou nod într-o listă dublu înlănţuită:

1. Adăugare înaintea primului nod

Adăugarea unui nou nod înaintea primului nod ale listei presupune efectuarea următoarelor operaţii:

1. alocarea şi încărcarea noului nod:
2. stabilirea faptului că acest nou nod va adresa ca următor element chiar pe nodul prim: **nou->urm=prim** ***(1)***
3. stabilirea faptului că nodul **prim** va referi ca precedent element pe nodul nou: **prim->prec=nou** ***(2)***
4. stabilirea noului nod ca prim element al listei: **prim=nou** ***(3)***
5. marcarea nodului prim: **prim->prec=0** ***(4)***



Observaţie: În cazul în care lista este înaintea adăugării unui nod nou, efectul operaţiei constă în obţinerea unei liste cu un singur element, fapt pentru care capetelor listei *prim* şi *ultim* sunt confundate şi este necesară tratarea acestui caz particular.

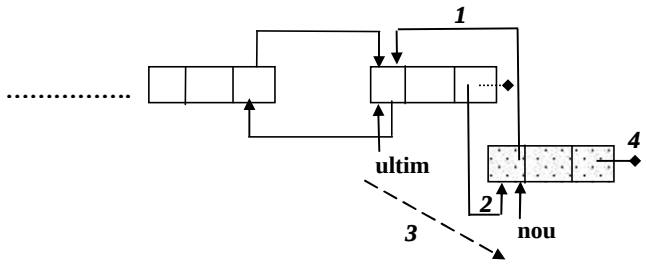
```
void adaugare_prim()
      {
      tnod *nou; p=incarca_nod();
      if (prim==0){
            prim=nou;ultim=nou;
            prim->prec=0;ultim->urm=0; }
      else{
            nou->urm=prim; //pasul 1
            prim->prec=nou; //pasul 2
            prim=nou; //pasul 3
            prim->prec=0; //pasul 4
            }}
```

2. Adăugare după ultimul nod

Adăugarea unui nou nod după ultimul nod al listei presupune efectuarea următoarelor operaţii:

- alocarea şi încărcarea noului nod:
- stabilirea faptului că acest nou nod va adresa ca precedent element chiar pe nodul ultim: **nou->prec=ultim** ***(1)***
- stabilirea faptului că nodul **ultim** va referi ca următor element pe nodul nou: **ultim->urm=nou** ***(2)***
- stabilirea noului nod ca ultim element al listei: **ultim=nou** ***(3)***
- marcarea nodului ultim: **ultim->urm=0** ***(4)***

Cazul special al procedurii (lista este vidă) se tratează în mod diferit.




```
void adaugare_ultim()
      {
      tnod *nou;nou=incarca_nod();
      if (prim==0)
            {
            prim=nou;ultim=nou;
            prim->prec=0;ultim->urm=0;
            }
      else
            {
            nou->prec=ultim; //(1)
            ultim->urm=nou; //(2)
            ultim=nou; //(3)
```

```
            ultim->urm=0; //(4)
            }
      }
```

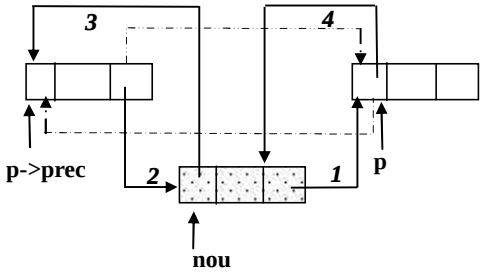
3. Adăugare înaintea uni nod specificat prin cheie

Adăugarea unui nod înaintea unui nod specificat prin valoarea cheii, se realizează prin două etape:

- căutarea nodului înaintea căruia se va face inserarea
- inserarea propriu-zisă

Reamintim că la operaţia similară pentru liste simplu înlănţuite era necesară determinarea nodului precedent nodului precizat de cheie li acest lucru se realiza printr-un pointer auxiliar în care se reţinea acea adresă. În cazul listelor dublu înlănţuite lucrurile sunt simplificate datorită adreselor *precedent* conţinute de fiecare nod, adrese utile în accesarea vecinilor (nodurile anterioare).

Dacă nodul căutat este chiar primul, problema se reduce la un caz particular tratat prin funcţia *adaugare_prim*. Dacă lista este vidă sau nodul căutat nu s-a găsit, nu se va adăuga un nou nod.




```
void adaugare_inainte(int valoare)
{
tnod *p,*nou; //p contine  adresa nodului curent in parcurgerea listei
p=prim;
while(p!=0)
  { //se parcurge direct lista, de la primul spre ultimul nod
  if (p->cheie!=valoare)
      {//nu s-a gasit inca nodul de cheie data
      p=p->urm; //trecere la urmatorul nod
      }
  else
      {
      //am gasit nodul p inaintea caruia se insereaza nou;
      if (p!=prim)
            {
            nou=incarca_nod();
            nou->urm=p; //(1)
            (p->pre)->urm=nou; //(2)
            nou->pre=p->pre; //(3)
            p->pre=nou; //(4)
            return;
            }
      else
            {
            adaugare_prim();
                return;
            }
      }
  }
}//sfarsit functie
```

Observaţie: Pentru manevrarea legăturii următoare a nodului precedent celui curent (notăm nodul curent **p**) este valabilă construcţia: **(p->pre)->urm**, unde **(p->pre)** este adresa precedentului nodului **p.**

4. Adăugare după un nod specificat prin cheie

Procedura de adăugare a unui nou nod după un nod precizat prin valoarea cheii este simalră celei prezentate la punctul **3.** Cazul particular este cel în care nodul căutat este ultimul nod al listei, în această situaţie se va apela funcţia *adăugare_ultim*. Dacă lista este vidă sau nu s-a găsit nodul căutat, nu are sens să se opereze adăugarea unui nou nod.

Inserarea propriu-zisă a nodului *nou* înaintea nodului *p* presupune refacerea legăturilor în aşa fel încât consistenţa listei să fie asigurată (să nu se piardă secvenţe de noduri prin distrugerea accesului la ele):

- nodul **nou** va avea ca legătură *urm* pe următorul nod al nodului p:
  nou->urm=p->urm; ***(1)***
- nodul **p** va fi urmat de **nou**
  p->urm=nou; ***(2)***
- precedentul nodului **nou** va fi nodul **p**
  nou->pre=p; ***(3)***
- nodul precedent al nodului următorul lui **p** devine **nou:**
  (p->urm)->pre=nou; ***(4)***

```
void adaugare_dupa(int valoare)
      {
      tnod *p,*nou;
      //caut p si inserez nod
      p=prim;
      while(p!=0)
        {
        if (p->cheie!=valoare)
            {
            p=p->urm;
            }
        else
            {
            if (p==ultim)
                  {adaugare_ultim();return;}
            else
                  {
                  nou=incarca_nod();
                  nou->urm=p->urm;
                  p->urm=nou;
                  nou->pre=p;
                  (p->urm)->pre=nou;
                  return;
                  }}  }}
```

ŞTERGEREA UNUI NOD

Ştergerea capetelor *prim* şi *ultim* ale unei liste dublu înlănţuite nu diferă prin costul de calcul precum la listele simplu înlănţuite. Am văzul că în cazul listelor simplu înlănţuite ştergerea ultimului nod necesita un efort computaţional mai mare. Prin legătura *prec* a nodurilor unei liste dublu înlănţuite putem accesa nodurile precedent, fapt pentru care, la ştergerea ultimului nod nu este necesară traversarea completă a listei.

Ştergerea unui capăt al listei presupune:

- salvarea temporară a adresei capătului respectiv într-un pointer auxiliar
- actualizarea şi marcarea noului capăt
- eliberarea memoriei

Oferim în continuare două variante pentru funcţiile de ştergere a capetelor prim şi ultim:

/*ştergere prim nod*/	**/*ştergere ultim nod*/**

`void stergere_prim()`
`{`
`tnod*primvechi;`
`if (prim==0) //lista vidă`
`  return;`
`else`
`{//mai sunt noduri`
` if(prim!=ultim)`
`  {//salvare prim`
`  primvechi=prim;`
`  //actualizare prim`
`  prim=prim->urm;`
	`void stergere_ultim()`
	`{`
	`tnod*ultimvechi;`
	`if (ultim==0) //lista vidă`
	`  return;`
	`else`
	`{`
	`if (prim!=ultim)`
	`  { //salvare ultim`
	`  ultimvechi=ultim;`
	`  //actualizare ultim`
	`  ultim=ultim->pre;`

```
   //marcare prim
   prim->pre=0;
   //eliberare memorie
   free(primvechi);
   }
  else
    prim=ultim=0;
 }
}//sfarsit functie
```

```
    //marcare ultim
    ultim->urm=0;
    //eliberare memorie
    free(ultimvechi);
  }
  else
    prim=ultim=0;
  }
}//sfarsit functie
```

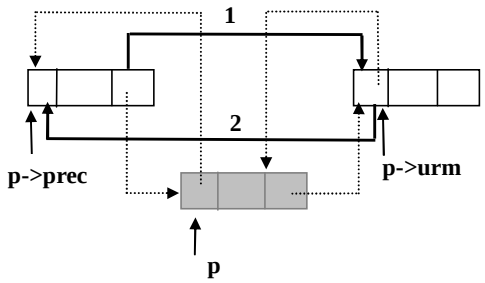
Ştergerea unui nod oarecare

Ştergerea unui nod oarecare precizat prin valoarea cheii presupune:

- căutarea nodului
- ştergerea propriu-zisă a nodului

Căutarea nodului se face prin parcurgerea într-un sens a listei şi compararea valorii cheii nodurilor curente cu valoarea dată. Dacă se găseşte un nod care verifică condiţie, se va opera etapa de ştergere propriu-zisă a nodului prin:

- salvarea adresei nodului de şters
- refacerea legăturilor pentru a asigura consistenţa listei şi posibilitatea parcurgerii ulterioare în ambele sensuri
- eliberarea memoriei alocate nodului de şters



Refacerea legăturilor constă din următoarele asignări:

- precedentul următorului lui p devine precedentul lui p
  p->urm->pre=p->pre; ***(1)***
- următorul precedentului lui p devine următorul lui p:
  p->pre->urm=p->urm; ***(2)***

Cazurile particulare se tratează separat: lista este deja vidă sau lista devine vidă după ştergerea nodului.

```
void stergere_nod(int valoare)
{
tnod *p,*pvechi;
if (prim==0) return; //lista este deja vida
if (prim==ultim && prim->cheie==valoare)
      {//lista devine vida
      prim=ultim=0;
      return;
      }
p=prim;
while(p!=0)
      {
      if (p->cheie==valoare) //gasit
            {
            if (p==prim)
                  {stergere_prim();return;}
            if (p==ultim)
                  {stergere_ultim();return;}
            pvechi=p; //salvare adresă nod curent
            p->urm->pre=p->pre; (1)
            p->pre->urm=p->urm; (2)
            free(pvechi); //eliberare memoriei- adresa pvechi
            return;
            }
      else //nu s-a gasit încă
            p=p->urm; //trecere la următorul nod
      }}
```

Ştergerea listei

Ştergerea completă listei dublu înlănţuită se poate face cu acelaşi efort de calcul prin apelarea repetată a funcţiei stergere_prim sau stergere_ultim. Ştergerea capetelor listei nu asigură eliberarea memoriei ocupate de nodurile intermediare.
Exemplu:ştergerea listei prin apelul repetat al funcţiei de ştergere a primului nod.

```
void stergere_lista(){
            while(prim!=0)
                  stergere_prim();}
```