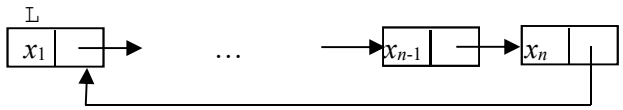
LISTELE CIRCULARE

Sunt liste cu legături în care legătura ultimului element este primul element al listei, în loc de NULL. Se pot parcurge elementele listei plecând din orice punct al listei.

(Problema lui Josephas) Se dă o listă cu elementele 1, 2, …, *n*. Începând de la primul element, cu un pas dat să se afişeze şi să se elimine toate elementele listei.

Soluţie: Se poate folosi o implementare cu listă circulară simplu înlănţuită *fără header*, primul element fiind la adresa dată de pointerul *L* şi ultimul element al listei fiind legat la adresa dată de pointerul *L* (la primul element). Funcţia care rezolvă problema este dată în continuare:




```
void Josephas(celula *&L, int pas)
{
  int i;
  if (L == NULL) { cout << "Lista vida"; return; }
  celula *p = L, *temp;
  while (p->leg != L) p = p->leg; //ultimul elem. din lista
  cout << "Elementele eliminate sunt urmatoarele: ";
  while (L->leg != L) //exista cel putin doua elem in lista
  {
    //avansez in lista de (pas-1) ori
    for (i = 1; i < pas; i++) p = p->leg;
    temp = p->leg;
    p->leg = L = temp->leg;
    cout << temp->info << " ";
    delete(temp);
  }
  cout << L->info; delete(L); //ultima celula
}
```

Observaţie: Crearea unei liste circulare cu elementele 1, 2, …, *n*, se poate face astfel:

```
void CreareLista(celula *&L) {  //lista circulara cu elem 1,…,n
 int n;
 cout<<"n="; cin>>n;
 if (n<=0) {cout<<"Eroare"; getch(); exit(1);}
 L = new(celula); L->info = 1; L->leg = L;
 //s-a creat prima celula (cu informatia 1)
 celula *ultim = L,*temp;
 for(int i=2;i<=n;i++) {
   temp = new(celula);
   temp->info = i;
   ultim->leg = temp; //ultima celula creata pana acum se leaga
   temp->leg = L;     //la noua celula, care se leaga apoi la L
   ultim = temp; //aceasta este acum ultima celula adaugata
 } }
```

ALTE APLICAȚII ALE LISTELOR SIMPLU ÎNLĂNȚUITE

1. Să se realizeze o funcţie ce numără elementele unei liste.
Soluţie: Vom considera întâi cazul implementării fără header.

```
int NumarElemente(celula *L) {
  celula *p = L; //pointer catre prima celula din lista
  //care contine efectiv un element (lista fara header)
  int contor = 0;
  while (p) //p!=NULL, se refera efectiv la o celula
  {
   contor++;      //numaram celula curenta
   p = p->leg;  //trecem la urmatoarea celula
  }
 return contor;
}
```

În cazul implementării cu header, pornim cu *p* de pe prima celula efectivă, adică:

```
celula *p = L->leg;
```

Este singura modificare necesară.

2. Să se realizeze o funcţie ce numără elementele unei liste egale cu o anumită valoare.
Soluţie: În cazul anterior număram toate celulele, acum va trebui să le contorizăm doar pe cele cu o anumită informaţie.

```
int NumarElementeI (celula *L, int i) {
  celula *p=L; //consideram cazul listei fara header
  //pointer catre prima celula din lista
  int contor = 0;
  while (p!=NULL) { //p se refera efectiv la o celula
   if (p->info == i)   contor++;
   //numaram celulele cu informatia i
   p = p->leg;  //apoi oricum trecem la celula urmatoare
  }
 return contor;
}
```

3. Să se realizeze reuniunea a două mulţimi reprezentate ca liste liniare cu legături simple.
Soluţie: Dându-se două liste simplu înlănţuite, pentru a determina reuniunea lor se consideră toate elementele din a doua listă care nu se găsesc în prima listă şi se inserează în lista finală care este iniţializată cu elementele din prima listă. Aici listele se consideră implementate fără header. Vom da în continuare funcţia care realizează lista de capăt *L* ce reprezintă reuniunea elementelor listelor de capete *L*1 şi *L*2, ştergându-se celulele ce nu intră în reuniune.

```
void Reuniune (celula *L1, celula *&L2, celula *&L)
{celula *p, *p2;
 L = L1; //initial elementele primei liste "le trecem" si in reuniune
 if (L == NULL) { L=L2; return; }
    //prima lista vida => reuniunea contine doar elemente din a doua lista
 while (L2) //se parcurg elementele din a doua lista
```

```
 {
  p2 = L2; //primul element din a doua lista
  L2 = L2->leg;  //L2 se va referi la urmatorul element din lista a doua
  for (p=L; p->info!=p2->info&&p->leg!=NULL; p=p->leg);
  //parcurg L pana gasesc o celula cu info ca p2->info sau pana la ultima celula
  if (p->info != p2->info)  //am gasit informatia din p2
    {
     p->leg = p2; //leg celula p la p2
     p2->leg = NULL; //=> in fata celulei referita de p2
    }
  else delete p2;  //element din a doua lista care-i si in prima
 }
}
```

4. Să se împartă elementele unei liste cu legături în două liste, prima conţinând elementele de ordin impar şi cea de-a doua pe cele de ordin par.

Soluţie: Rezolvarea problemei presupune schimbarea legăturilor astfel: primul element din lista iniţială va fi legat de al treilea, al doilea de al patrulea etc. Se consideră cazul listelor reprezentate fără header.

```
void PozParImpar(celula *L, celula *&L1, celula *&L2)
{ celula *p1, *p2;
  p1 = L1 = L; //p1 porneste de pe primul element
  if (L == NULL) { L2 = NULL; return; }
  p2 = L2 = L->leg; //p2 porneste de pe al 2-lea elem
  while (p2) //p2!= NULL => p2 se refera la o celula
   {
    p1->leg = p2->leg; //leg p1 la ce este dupa p2
    p1 = p2; //p1 avansaza pe urmatoarea (referita de p2)
    p2 = p2->leg; //p2 avanseaza pe urmatoarea celula
   }
  p1->leg = NULL;
  //in final leg p1 la NULL pt a fi si aceasta o lista
}
```

5. Fiind dată o listă simplu înlănţuită de numere întregi, să se creeze cu elementele ei două liste ce conţin elementele pare şi respectiv impare ale ei.

Soluţie: Se parcurge lista iniţială şi în cazul în care elementul este par se leagă la prima listă, iar dacă este impar se leagă la a doua listă. O funcţie ce poate realiza aceasta este următoarea:

```
void ValParImpar(celula *L, celula *&L1, celula *&L2)
{celula *p, *p1, *p2; //liste fara header
 p1 = p2 = L1 = L2 = NULL; //se initializeaza noile liste
 while (L) //exista elemente in lista initiala
 {
  p = L; //primul element din lista initiala
  L = L->leg; //L va fi capul restului de lista
  if (p->info%2)  //daca elementul este impar se pune in prima lista
    if (!L1) { //ca prim element
       L1 = p1 = p; //capul de lista L1 este la celula p
```

```
        p->leg = NULL;
       }
     else {
     //nu ca prim element => se leaga de ultima inserata
        p1->leg = p;
        p->leg = NULL;
        p1=p;
       }
  else //altfel se insereaza la sfarsitul celei de-a doua lista
     if (!L2) { //ca prim element
        L2 = p2 = p;
        p->leg = NULL;
       }
     else {
        p2->leg = p;
        p->leg = NULL;
        p2=p;
       }
  }
}
```

6. Să se inverseze legăturile unei liste simplu înlănţuite în aşa fel încât primul element să devină ultimul şi reciproc.

Soluţie: Presupunem cazul listei reprezentate fără header. Rezolvarea problemei presupune folosirea a trei pointeri: *p*1, *L*, *p*3 care fac referire la trei elemente consecutive din listă; *p*1 şi *L* se folosesc pentru inversarea legăturii, iar *p*3 va reţine adresa capătului părţii din lista iniţială, nemodificată încă.

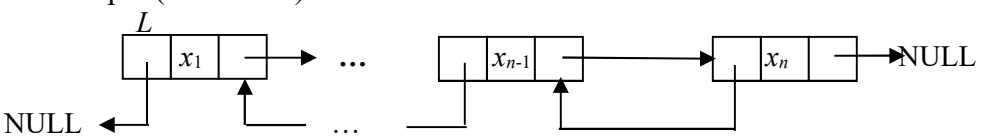
```
void Inversare(celula *&L)
{ celula *p1 = NULL, *p3;
  //p1 = pointer catre celula de dinainte de L
  //p3 = pointer catre celula de dupa L
  if (L == NULL) return; // L nu se mai refera la o celula efectiv
  while((p3 = L->leg) != NULL)
     //daca p3 care este pointer catre celula de dupa L se refera la o celula
     {
      L->leg = p1; //leg L la celula anterioara
      p1 = L; //p1 avanseaza in restul listei
      L = p3; //p3 avanseaza in restul listei
     }
  L->leg = p1; //a mai ramas aceasta legatura
}
```

LISTE CU LEGĂTURI DUBLE

Listele cu legături duble au celule cu câte două legături, una către celula predecesoare, pe care o vom numi *legătură stângă* şi o vom nota *legs*, şi una către celula succesoare, pe care o vom numi *legătură dreaptă* şi o vom nota *legd*.

Aplicaţia 7 (suplimentar). Să se implementeze în C/C++ operaţiile cu liste dublu înlănţuite.

Soluţie: Se foloseşte o reprezentare a listei *fără header* şi spre deosebire de listele simplu înlănţuite aici există două legături ale unei celule: una spre celula din stânga (anterioară) şi una spre celula din dreapta (următoare).



În acest caz, structura unei celule este:

```
struct celula
{ int info;
  celula *legs, *legd;
} *L;
```

Codul C++ pentru iniţializarea listei poate fi:

```
void Initializare() { L = NULL; }
```

Inserarea unui nou element *a* la începutul listei se poate face astfel:

```
void InserareInceput(int a){
  celula *k = new(celula);
  k->info = a;
  k->legd = L;  k->legs = NULL;  L = k;
  if(k->legd != NULL) k->legd->legs = k;
}
```

Inserarea unui nou element *a* după al *m*-lea elemnt din listă sau la sfârşitul listei, dacă *m* este mai mare decât numărul de elemente din listă se poate face astfel:

```
void InserareInterior(celula *&L, int a, int m){
  celula *k = new(celula), *s = NULL, *d = L;
  k->info = a;
  for( ; m > 0 && d != NULL; m--) { s = d; d = d->legd;}
  k->legs = s;  k->legd = d;
  if(s != NULL) s->legd = k;
  else L = k;
  if(d != NULL) d->legs = k;
}
```

Inserarea unui nou element *a* la sfarşitul listei se poate face astfel:

```
void InserareSfarsit(celula *&L, int a){
  celula *k = new(celula), *s = L;
  k->info = a;
  k->legd = NULL;
  if(L == NULL) { k->legs = NULL; L = k; return; }
  while(s->legd != NULL) s = s->legd;
  s->legd = k; k->legs = s;
```

```
}
```

Eliminarea primului element din listă se poate face astfel:

```
void StergereInceput(celula *&L, int &a){
  celula *k = L;
  if(k == NULL) { cout << "Lista vida"; return; }
  a = k->info;
  L = L->legd;
  if(L != NULL) L->legd = NULL;
  delete(k);
}
```

Eliminarea elementului de ordin *m* din listă se poate face astfel:

```
void StergereInterior(celula *&L, int &a, int m){
  celula *s = NULL, *k = L, *d;
  while(--m > 0 && k != NULL) { s = k; k = k->legd; }
  if(m) cout << "Nu exista element de ordinul dat";
  else
  { a = k->info;    d = k->legd;
    if(s != NULL) s->legd = d;  else L = d;
    if(d != NULL) d->legs = s;
    delete(k);  }
}
```

Eliminarea ultimului element din listă se poate face astfel:

```
void StergereSfarsit(celula *&L, int &a)
{ celula *s = NULL, *k = L;
  if(L == NULL) cout << "Lista vida";
  while(k->legd != NULL) { s = k; k = k->legd; }
  a = k->info;
  if(s != NULL) s->legd = NULL;  else L = NULL;
  delete(k);
}
```