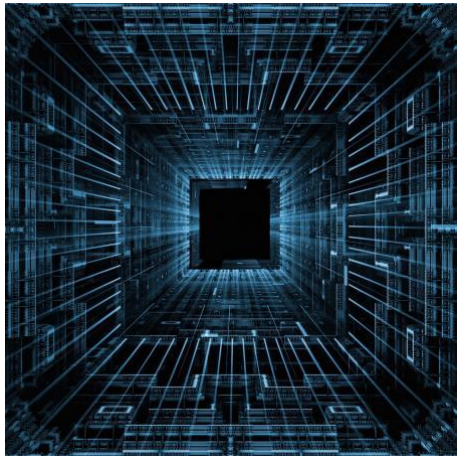
Structuri de Date

Anul universitar 2019-2020

Prof. Adina Magda Florea



Curs Nr. 3

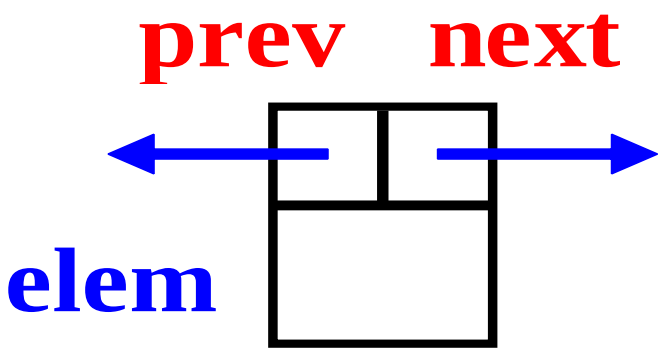
Liste înlănţuite – continuare

- Liste dublu inlantuite
- Dictionare
  - Vector de liste
  - Liste de liste

1. Liste dublu înlănţuite

Celulele din listă au două câmpuri legătură:

- *prev – adresa celulei predecesoare*
- *next – adresa celulei următoare*.



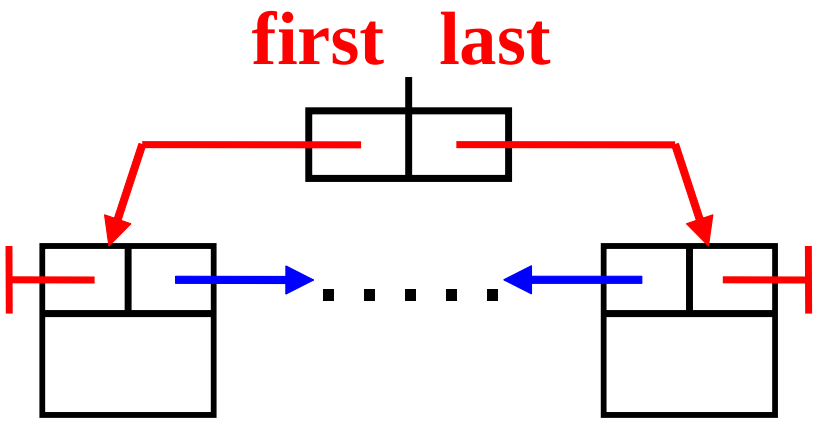
Pentru parcurgerea eficientă a listei, indiferent de sens (de la început sau de la sfârşit), este necesar accesul la cele două extremităţi.

Liste dublu înlănţuite

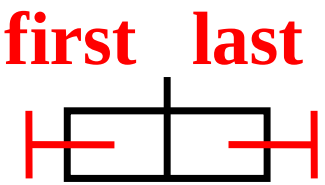
```
typedef struct ListNode {
      Item value;
      struct ListNode *next;
      struct ListNode *prev;
                  } ListNode, *TListNode;

/* lista este referita de 2 pointeri */
TListNode   first=NULL, last=NULL;
/* lista cu 1 celula */
first = (TListNode)malloc(sizeof(ListNode));
last = first;
```

Liste dublu înlănţuite deschise



lista nevida

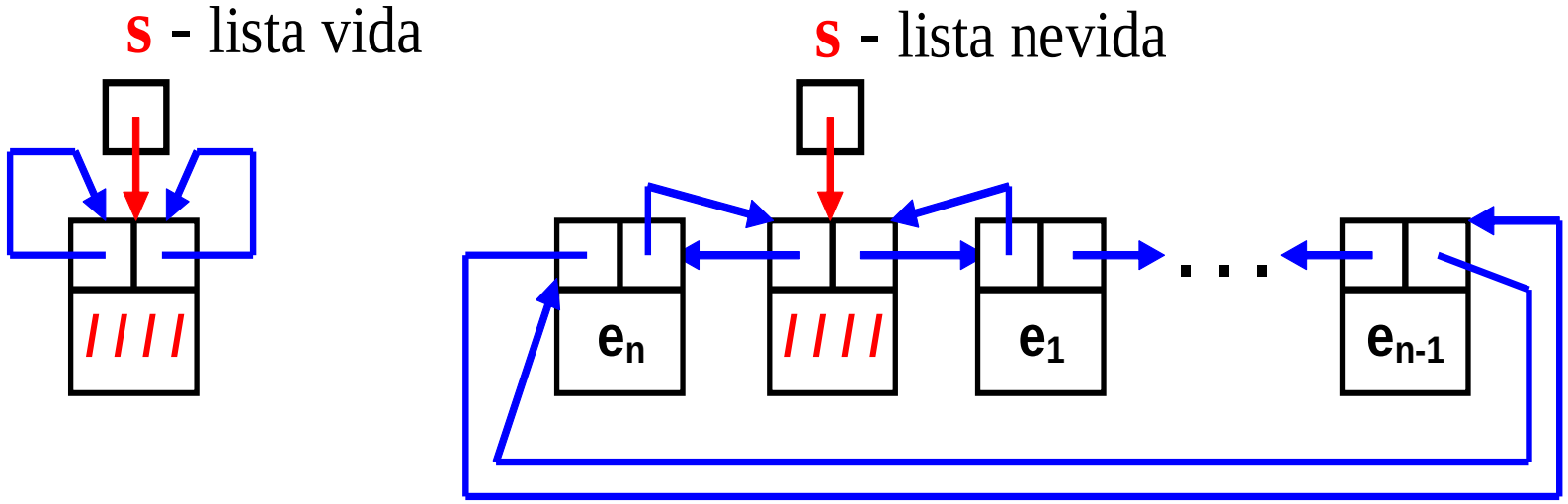


lista vida

Se observă că există diferențe de tratare în cazul modificărilor (inserare / eliminare) la extremități, deoarece acestea efectează fie câmpul *first*, fie câmpul *last*.

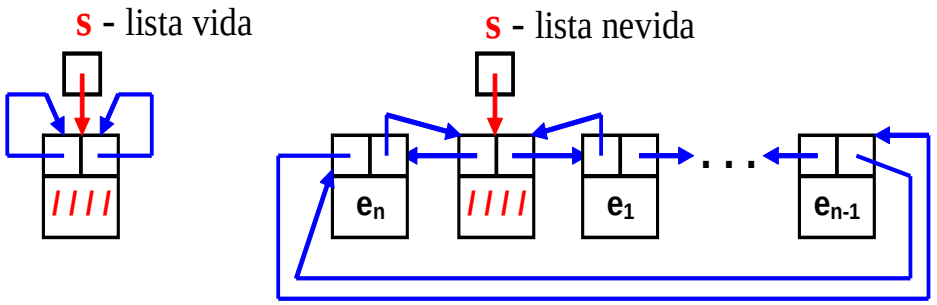
Liste dublu înlănţuite circulare (LDIC)

Pentru o tratare uniforma se poate adopta soluţia de organizare ca listă circulară cu santinelă.



Câmpul elem al celulei santinelă poate fi utilizat pentru păstrarea adresei sau valorii unor informaţii cu caracter general, cum ar fi referinţa la funcţia de verificare a relaţiei de ordine în cazul în care elementele din listă sunt ordonate.

Liste dublu înlănţuite circulare (LDIC)




```
typedef struct ListNode {
          Item value;
          struct ListNode *next;
          struct ListNode *prev;
                    } ListNode, *TListNode;

TListNode   s;
/* creare lista vida */
s = (TListNode)malloc(sizeof(ListNode));
s->next = s->prev = s;
```

Prelucrarea LDIC

Test listă vidă:

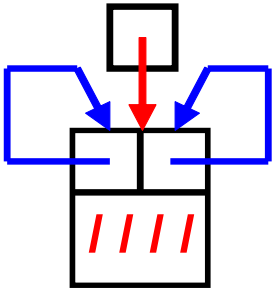
`(s->next == s)  sau  (s->prev == s)`

Atenţie! Expresia

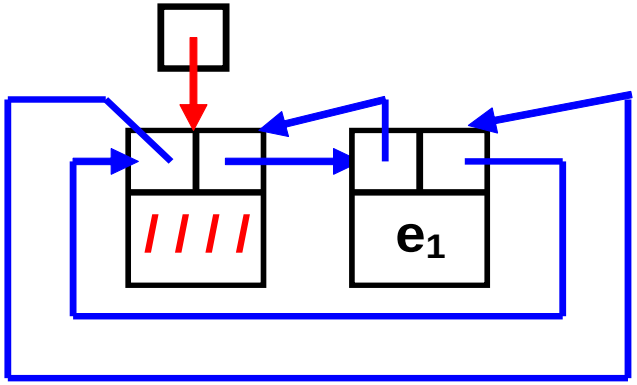
`(s->next == s->prev)`

are valoarea 1 şi în cazul listei cu un singur element.

s - lista vida



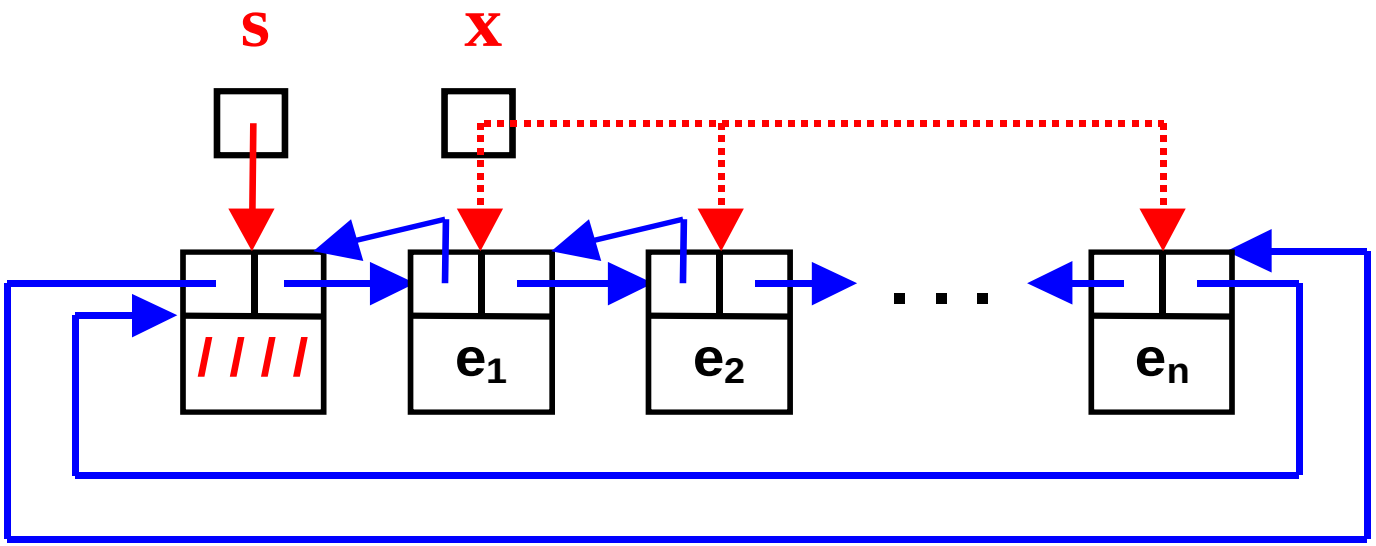
s - lista cu un element



Parcurgerea LDIC

- **de la început**

```
for (x = s->next; x != s; x = x->next)
{. . . .}
```



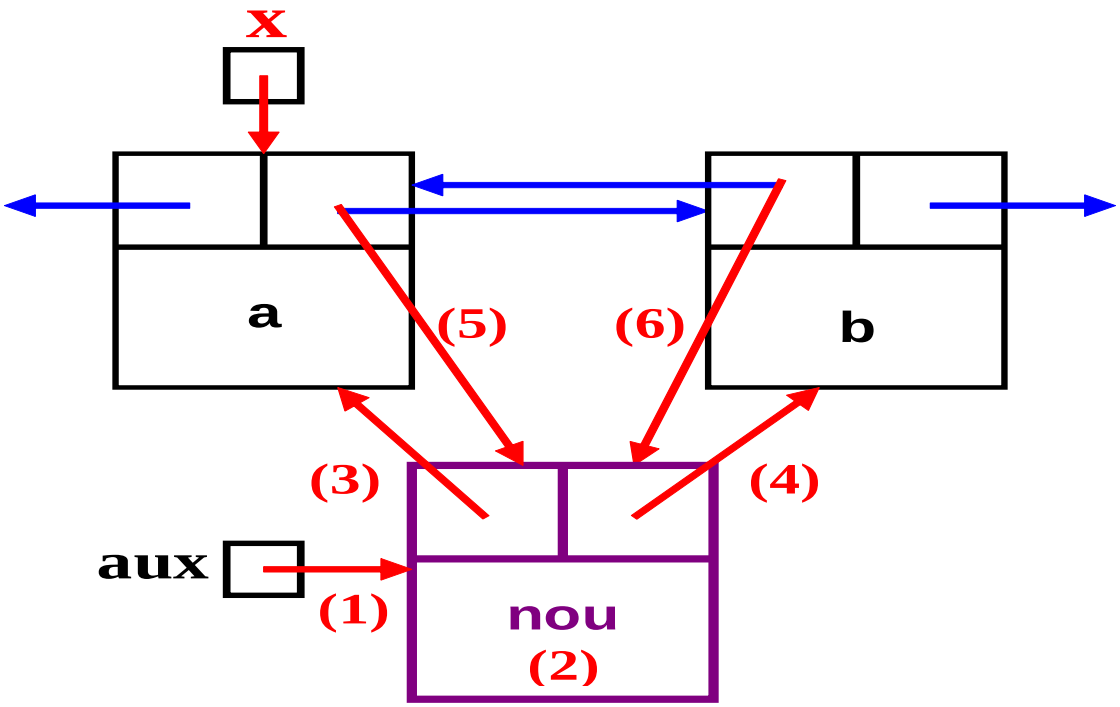
- **de la sfârşit**

```
for (x = s->prev; x != s; x = x->prev)
{. . . .}
```

Inserare după celula cu adresa x

Secvenţa de operaţii:

- alocare spaţiu pentru o noua celulă (1)
- actualizarea câmpurilor noii celule (2,3,4)
- actualizare referinţe la noua celulă (5,6)

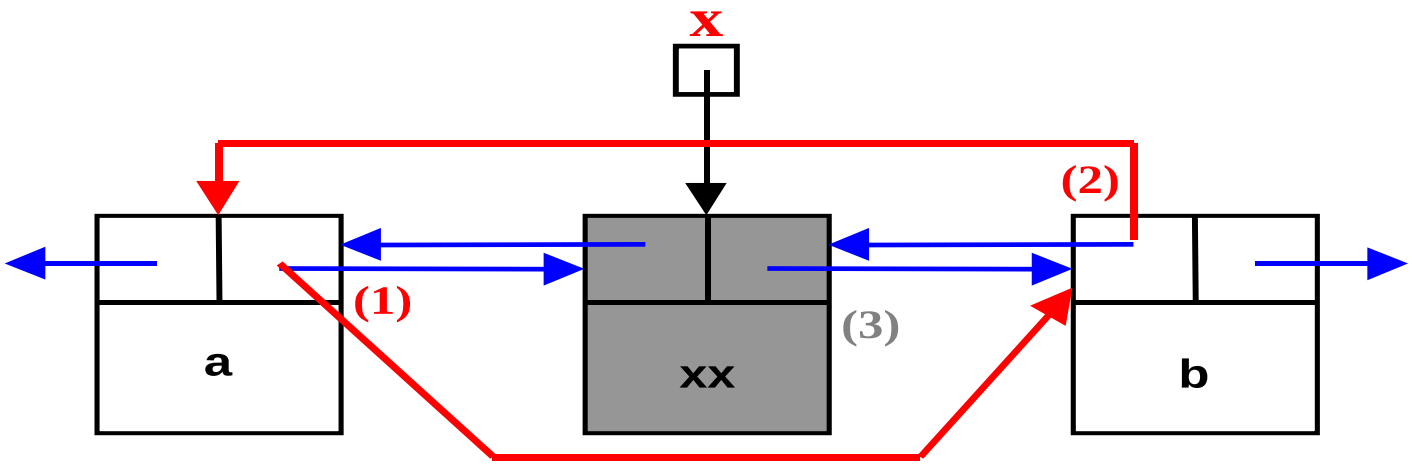


Eliminarea celulei de la adresa x (x != s)

Secvenţa de operaţii:

- actualizare legături în celulele vecine (1,2)
- eliberarea spaţiului ocupat de celulă (3)

Dacă informaţia din celulă este necesară pentru prelucrări ulterioare, atunci ea trebuie salvată înainte de eliminarea celulei.



Aplicatii ale listelor duble inlantuite

- Sisteme de operare – *thread scheduler* – ce procese se executa si cand
- Aplicatii de redat melodii de pe un CD: next, previous
- Representarea unei suite de carti dintr-un joc
- Cache-ul unui browser (back, forward)
- Functionalitati de Undo Redo

etc.

2. Dictionare

- Există un număr imens de aplicaţii în care avem nevoie să căutăm informaţii
  - Căutare clienţi într-o bază de date
  - Căutare filme preferate
  - Căutare cuvinte în pagini Web
  - Căutare calculator preferat pe un site etc.
- Avem nevoie de o structură care să permită căutarea elementelor după o **cheie de căutare**
- Baze de date
- Tabele de simboluri
- Dicţionare

Dictionare

- **ADT dictionar**
- Operatii ADT dictionar
- Presupunem elementele din dictionar organizate in ***categorii***
- O categorie poate fi: litera, clasa de produse, etc.
- O categorie contine mai multe ***inregistrari*** (record)
- ***Cheie*** (key)
- (a) Cunoastem numarul de categorii
- (b) Nu cunoastem de la inceput numarul de categorii

ADT dictionar cu vector de liste (a)

```
#define MAXCAT  27
#define MAXNAME  20
typedef struct {
    int  id;
    char *lastn;
    char *firstn;
    int   wage;    } Employee;

typedef struct RecCel {
    Employee elem;
    struct RecCel *urm;} RecList, *TRecList;

typedef struct {
    char masterk;
    TRecList  link; } Master;

typedef Master Dict[MAXCAT];
```

ADT dictionar cu vector de liste

```
TRecList readRecord(int identity)
……
void displayDict(Dict d)
……
main()
{ Dict d; char letter; int i=0, num;
  TRecList q; int index;
  char *e,*letterstring =
     "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
```

ADT dictionar cu vector de liste

```
TRecList readRecord(int identity)
{
    TRecList p;
    p = (TRecList)malloc(sizeof(RecList));
    if(p!=NULL)
    {
        p -> urm = NULL;
        p->elem.id = identity;
        p->elem.lastn = malloc(sizeof(MAXNAME));
        p->elem.firstn = malloc(sizeof(MAXNAME));
        gets(p->elem.lastn);
        gets(p->elem.firstn);
     }
        return p;
}
```

ADT dictionar cu vector de liste

```
void displayDict(Dict d)
{ int i; TRecList p;
    for(i=0;i<MAXCAT;i++)
    {
        printf("Master Key %c \n", d[i].masterk);
        p = d[i].link;
        while(p!=NULL)
        {
  printf("%d %s %s\n", p->elem.id, p->elem.lastn,
  p->elem.firstn);
            p = p->urm;
        }
    }
}
```

ADT dictionar cu vector de liste

```
main()
{ Dict d; char letter; int i=0, num;
  TRecList q; int index;
  char *e,*letterstring = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    for(i=0; i<MAXCAT;i++) {d[i].masterk = letterstring[i]; d[i].link=NULL;}
    scanf("%d", &i);
    while (i!=0)
    {
        q = readRecord(i);
        if(q==NULL) {printf("ERROR");exit(1);}
        else
        {
            letter = q->elem.lastn[0];
            e = strchr(letterstring,letter);
            index = (int)(e - letterstring);
            if(d[index].link==NULL)
                d[index].link=q;
            else { q->urm = d[index].link;
                d[index].link = q;}
            scanf("%d", &i);
        }
    }
displayDict(d);
}
```

ADT dictionar cu liste de liste (b)

```
#define MAXNAME  20
typedef struct {
    int id;
    char *lastn;
    char *firstn;
    int  wage;    } Employee;

typedef struct RecCel {
    Employee elem;
    struct RecCel *urm;} RecList, *TRecList;

typedef struct MasterCel {
    int masterk;
    TRecList  link;
    struct MasterCel *next;} MasterList, *TMasterList;
```

ADT dictionar cu liste de liste

```
TRecList readRecord(int identity)
……
void displayDict(TMasterList d)
……
TMasterList isIn(int categorie, TMasterList d)
……..
main()
{ TMasterList d = NULL, dp = NULL;
  int id=100, categorie;
  TRecList q;
```

ADT dictionar cu liste de liste

```
TRecList readRecord(int identity)
{
    TRecList p;
    p = (TRecList)malloc(sizeof(RecList));
    if(p!=NULL)
    {
        p -> urm = NULL;
        p->elem.id = identity;
        p->elem.lastn = malloc(sizeof(MAXNAME));
        p->elem.firstn = malloc(sizeof(MAXNAME));
        gets(p->elem.lastn);
        gets(p->elem.firstn);
     }
        return p;
}
```

ADT dictionar cu liste de liste (santinela)

```
void displayDict(TMasterList d)
{ TRecList p;

    d=d->next; /*sar peste santinela */
    while(d!=NULL)
    {
        printf("Master Key %d \n", d->masterk);
        p = d->link;
        while(p!=NULL)
        {
     printf("%d %s\n", p->elem.id, p->elem.lastn);
            p = p->urm;
        }
        d=d->next;
    }
}
```

ADT dictionar cu liste de liste

```
TMasterList isIn(int categorie, TMasterList d)
{
    d = d->next;
    while(d!=NULL)
          if(d->masterk == categorie)return d;
          else d = d->next;
    return NULL;
}
```

ADT dictionar cu liste de liste

```
main()
{ TMasterList d=NULL, dp=NULL;
  int id=100, categorie;
  TRecList q;

  d =(TMasterList)malloc(sizeof(MasterList));
  /*santinela*/
    d->masterk=0;
    d->link=NULL;
    d->next=NULL;
    scanf("%d", &categorie);
```

ADT dictionar cu liste de liste

```
while (categorie!=0)
    {
        q = readRecord(id);  /*creaza record */
        if((dp = isIn(categorie,d))!=NULL)
               /*dp pointer la celula care contine categoria */
        {
            q->urm = dp->link;
            dp->link = q;
        }
        else
        {
            dp=(TMasterList)malloc(sizeof(MasterList));
            dp->masterk=categorie;
            dp->link=q;
            dp->next=d->next;
            d->next=dp;
        }
    scanf("%d", &categorie);
    }
displayDict(d);
}
```