Tehnici de programare - TP

Cursul 7 – Liste liniare simplu înlănțuite

Ș.l. dr. ing. Cătălin Iapă

catalin.iapa@cs.upt.ro

De data trecută

Structuri de date

Liste simplu înlănțuite

De data trecută:
Argumente din linia de comandă

```
int main(int argc, char *argv[])
{
   int i, n;
      for (i = 0; i < argc; i++) {
            printf("Parametrul i este: %s",argv[i]);
            printf("\n");
      }
       return 0;
}
```

Dacă rulăm programul astfel:

./prg Ana 23 "Un program"

Va afișa:

Parametrul 0 este: ./prg
Parametrul 1 este: Ana
Parametrul 2 este: 23
Parametrul 3 este: Un program

De data trecută: Funcții cu număr variabil de argumente

Să se calculeze suma argumentelor variabile (întregi). Numărul argumentelor variabile va fi specificat de argumentul fix al funcției.

```
#include <stdarg.h>

int suma(int n, ...) {
  va_list va;
  int suma = 0;
  va_start(va, n);

  for(int i = 0; i < n; i++) {
    suma += va_arg(va, int);
  }
  va_end(va);
  return suma;
}
```

La apel:

suma(3, 1, 2, 3) -> rez: 6

suma(5, 5, 5, 5, 5, 5) -> rez: 25

suma(1, 100) -> rez: 100

De data trecută: Funcții cu număr variabil de argumente

Img: https://guardian.ng/wp-content/uploads/2018/06/Various-cocktails.-Photo-Getty-Images.jpg

De data trecută

Structuri de date

Liste simplu înlănțuite

Structuri de date

Structurile de date fundamentale ale limbajului C (tabloul,structura şi uniunea) sunt fixe din punct de vedere al formei şi al dimensiunii. Aceste structuri de date se numesc statice.

În practica programării însă, există multe probleme care presupun structuri ale informaţiei mult mai complicate (de ex.: liste, arbori, grafuri), a căror caracteristică principală este aceea că *se modifică structural în timpul execuţiei programului*. Rezultând o modificare dinamică atât a formei cât şi a dimensiunii lor, aceste structuri de date se numesc dinamice.

Structuri de date

În general, atât structurile de date dinamice cât şi cele statice sunt formate, în ultimă instanţă, din componente de dimensiune constantă care se încadrează într-unul din tipurile acceptate de limbaj.

În cazul structurilor dinamice aceste componente se numesc noduri. Natura dinamică a structurii rezultă atât din faptul că nodurile se alocă dinamic în maniera cunoscută (malloc) cât și datorită faptului că numărul nodurilor şi legăturile între noduri se pot modifica pe durata execuției programului.

Structuri de date

Atât structurile dinamice în ansamblu cât şi nodurile lor individuale se deosebesc de structurile statice prin faptul că ele nu se declară ca variabile şi, în consecinţă, nu se poate face referire la ele utilizând numele lor deoarece, practic, nu au nume.

Referirea unor astfel de structuri se face cu ajutorul unor variabile statice, definite special în acest scop, numite pointeri. Un pointer poate conţine la un moment dat o referinţă la un nod al unei structuri dinamice materializată prin adresa nodului respectiv.

Pointeri – ați făcut la PC – reluați informațiile din cursul de sem trecut

De data trecută

Structuri de date

Liste liniare simplu înlănțuite

Liste liniare simplu înlănțuite

Listele liniare simplu înlănțuite sunt structuri de date care permit stocarea și gestionarea unui set de elemente într-o ordine specifică.

Fiecare element este legat printr-un pointer către următorul element din listă, astfel încât elementele pot fi accesate în ordine secvențială.

Lista simplu înlănțuită este diferită de alte structuri de date, cum ar fi vectorii sau matricele, deoarece lista poate fi extinsă sau micșorată în funcție de nevoile aplicației.

Liste – unde se pot folosi?

Implementarea coșului de cumpărături online - listele simplu înlănțuite pot fi utilizate pentru a gestiona produsele din coșul de cumpărături online. Astfel, putem adăuga produse noi la coșul de cumpărături, șterge sau actualiza cantitatea produselor existente.

Implementarea unei liste de contacte într-un smartphone - un smartphone poate utiliza o listă simplu înlănțuită pentru a gestiona lista de contacte. Această listă poate fi utilizată pentru a adăuga noi contacte, pentru a șterge sau actualiza contactele existente.

Implementarea unui joc video - jocurile video pot utiliza liste simplu înlănțuite pentru a gestiona obiecte în mișcare, cum ar fi gloanțele sau inamicii, astfel încât să poată fi urmărite și actualizate în timp real.

*Exemple generate cu ajutorul chat.openai.com

Operații de bază cu liste

Operaţiile care se pot efectua asupra listelor sunt:

- crearea listei;
- parcurgerea listei;
- inserarea unui nod într-o anumită poziţie (nu neapărat prima);
- căutarea unui nod cu o anumită proprietate;
- furnizarea conţinutului unui nod dintr-o anumită poziţie sau cu o anumită proprietate;
- ștergerea unui nod dintr-o anumită poziţie sau cu o anumită proprietate.

Liste simplu înlănțuite - implementare

O listă este formată dintr-o succesiune de elemente alocate distinct, fiecare element având pe lângă informația utilă și un pointer la următorul element din listă.

Pointerul la următorul element se numește **urm**(ător, en: next). Pointerul *urm* al ultimului element din listă va fi *NULL*, pentru a marca faptul că nu pointează la nimic.

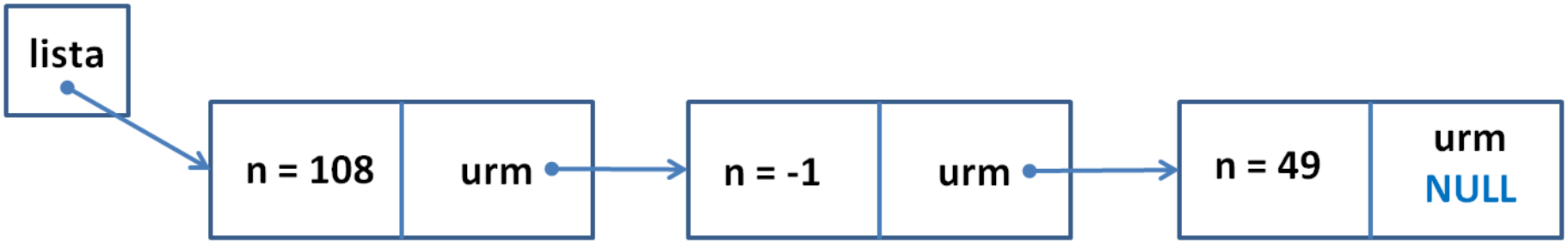
Toată lista va fi accesibilă prin intermediul unui pointer la primul element din listă.

Liste simplu înlănțuite - implementare

De exemplu, vom considera că un element din listă are ca informație utilă un număr întreg *n*.

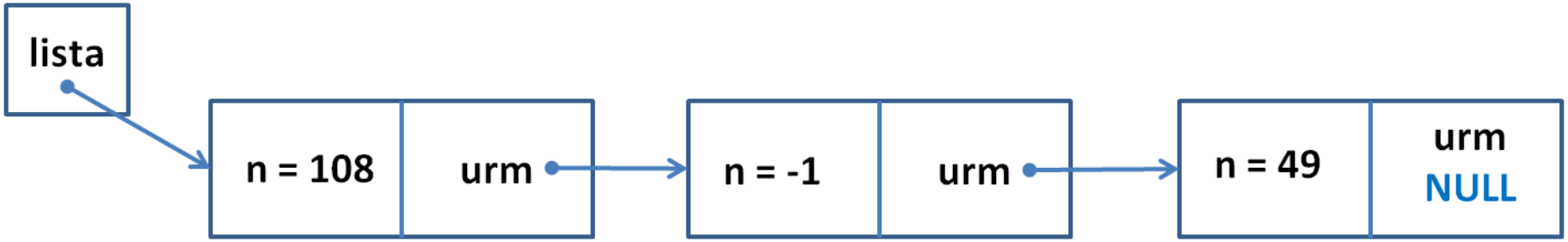
Lista are 3 elemente {108, -1, 49} și este adresată printr-un pointer numit *lista*.

Structura memoriei arată astfel:



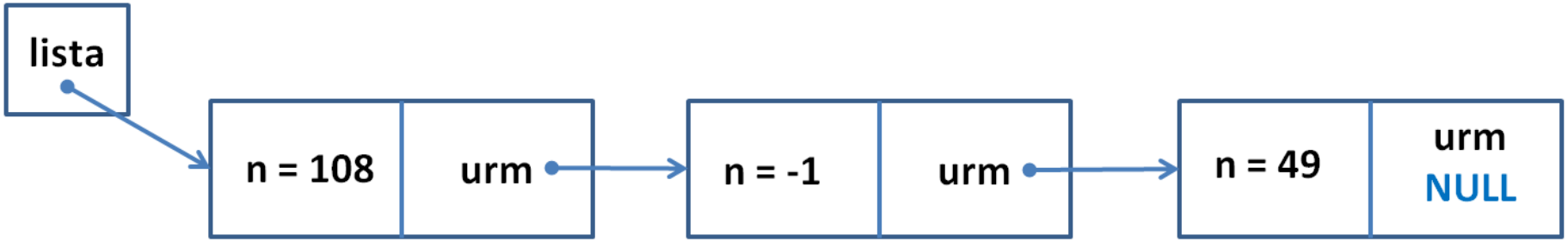
Liste simplu înlănțuite - nodul

```
// un element al listei – un nod
typedef struct elem{
    int info;                   // informatia utila
    struct elem *urm;           // urmatorul nod
    }elem;

elem *lista = NULL;
```

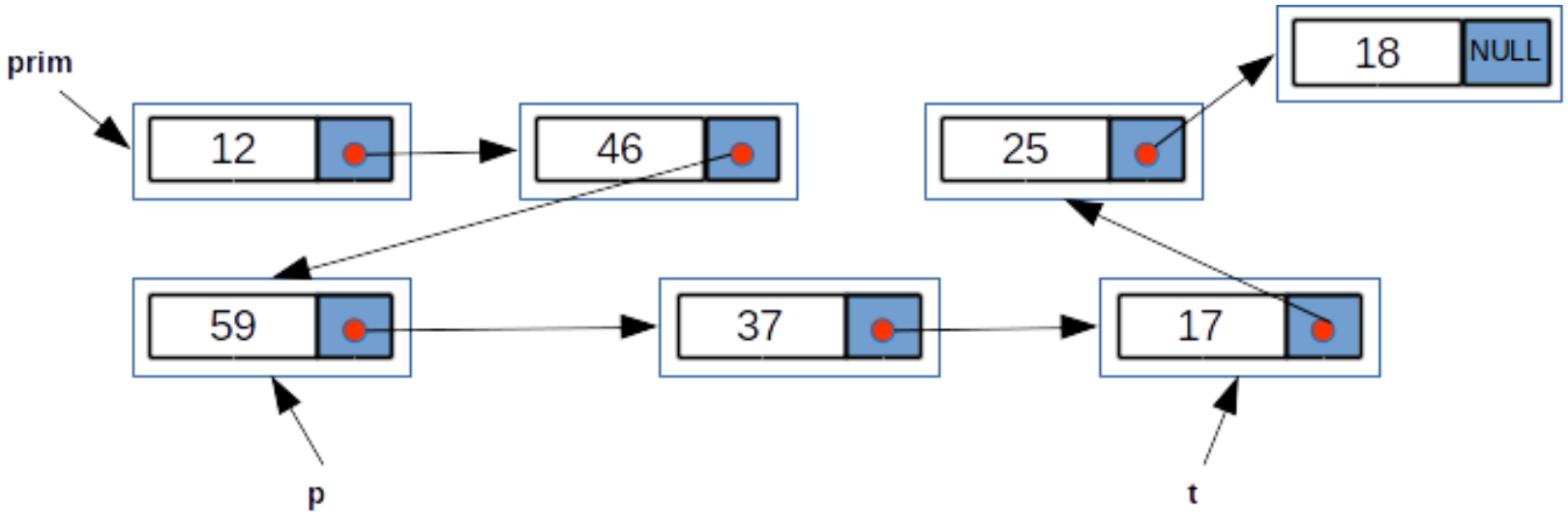


Liste simplu înlănțuite - nodul

```
// un element al listei – un nod
typedef struct elem{
    int info;                    // informatia utila
    struct elem *urm;            // urmatorul nod
    }elem;

elem *lista = NULL;
elem *ultim = NULL;
```

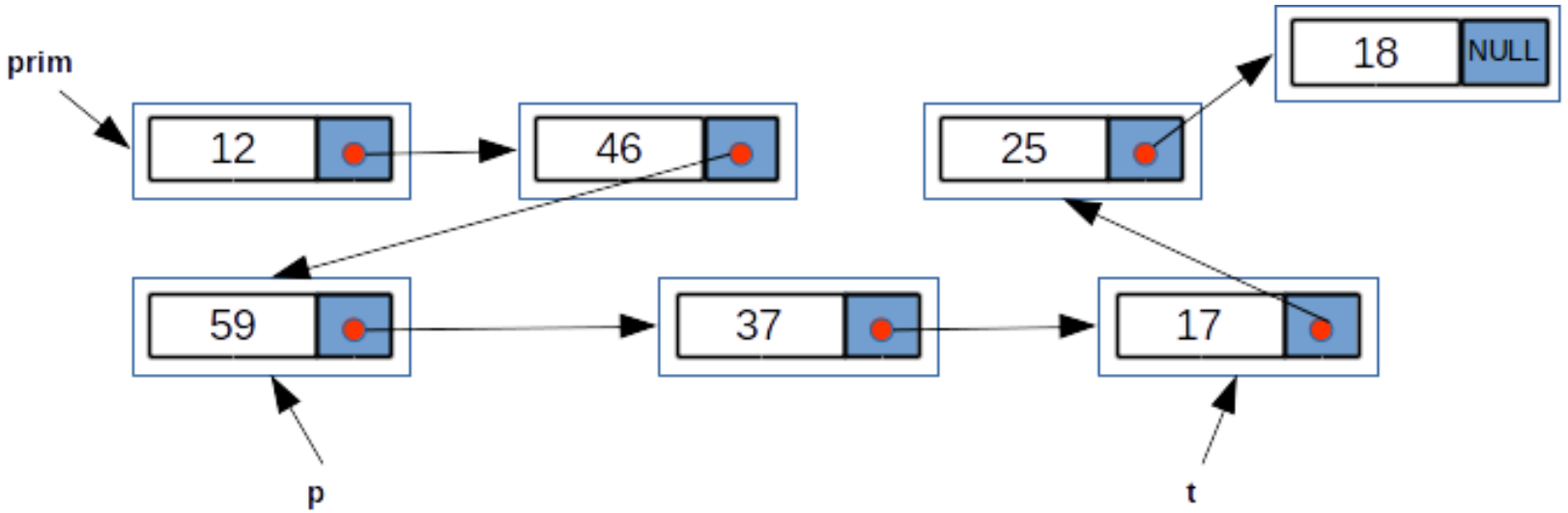


Relații între nodurile unei liste



https://www.pbinfo.ro/articole/19576/liste-liniare-simplu-inlantuite-alocate-dinamic

Relații între nodurile unei liste



prim este adresa elementului cu valoarea .....;

prim->info==.....

prim->urm->info==.....

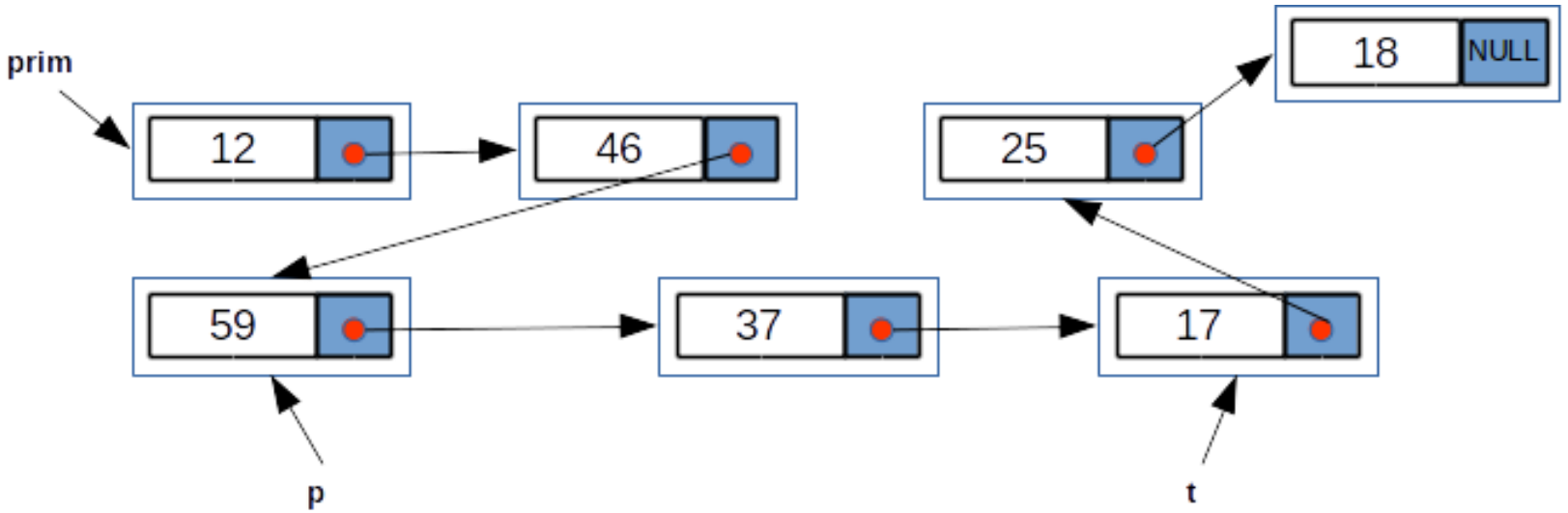
prim->urm este adresa elemenului cu valoarea .....

prim->urm->urm==.....

p->info==.....

https://www.pbinfo.ro/articole/19576/liste-liniare-simplu-inlantuite-alocate-dinamic

Relații între nodurile unei liste



prim este adresa elementului cu valoarea 12;

prim->info==12

prim->urm->info==46

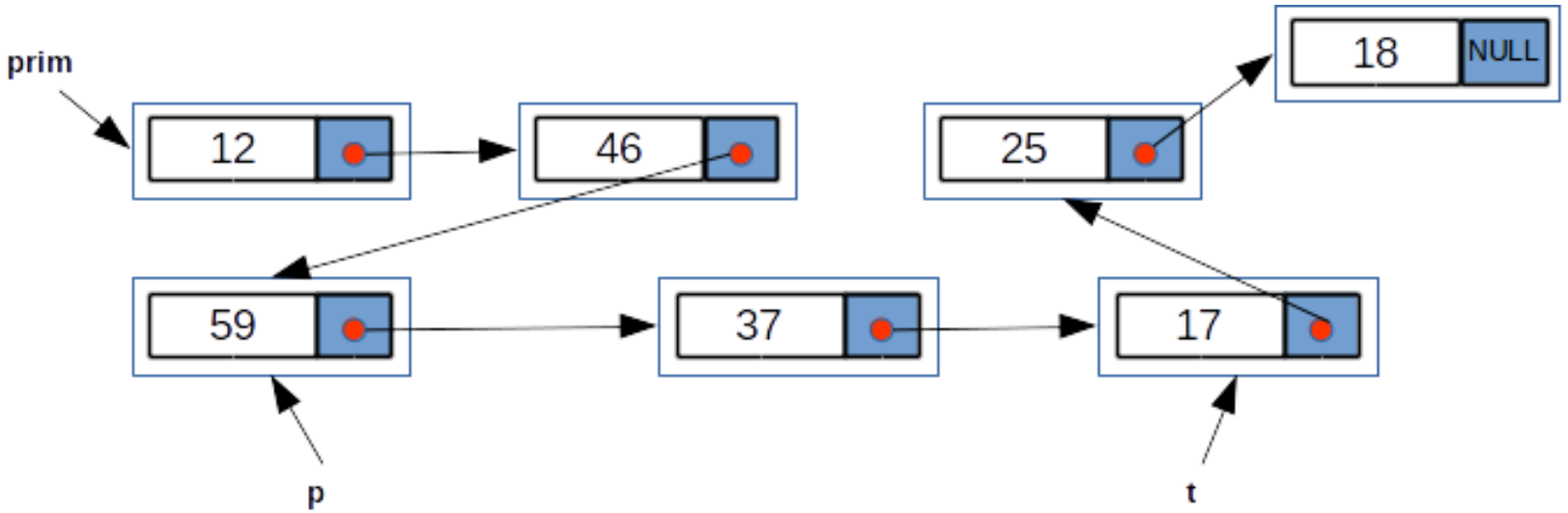
prim->urm este adresa elemenului cu valoarea 46

prim->urm->urm==p

p->info==59

https://www.pbinfo.ro/articole/19576/liste-liniare-simplu-inlantuite-alocate-dinamic

Relații între nodurile unei liste



p->urm->urm== ...

t->info== ...

t->urm->info== ...

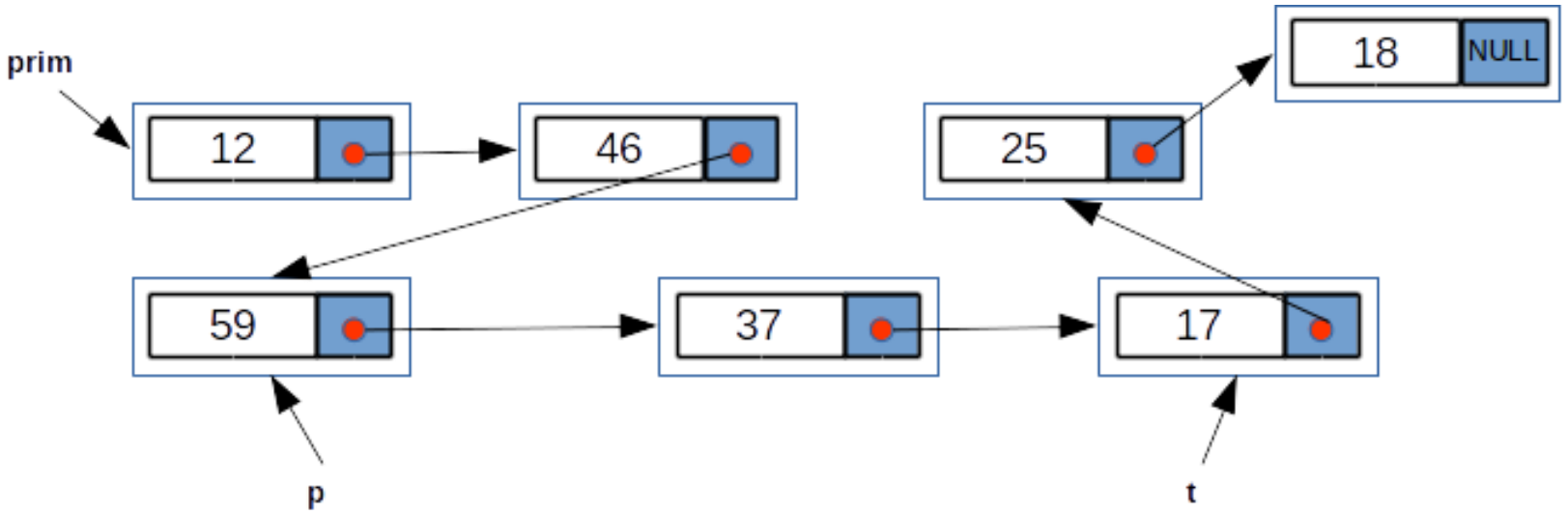
t->urm->urm->info== ...

t->urm->urm->urm== ...

t->urm->urm->urm->info ...

https://www.pbinfo.ro/articole/19576/liste-liniare-simplu-inlantuite-alocate-dinamic

Relații între nodurile unei liste



p->urm->urm==t

t->info==17

t->urm->info==25

t->urm->urm->info==18

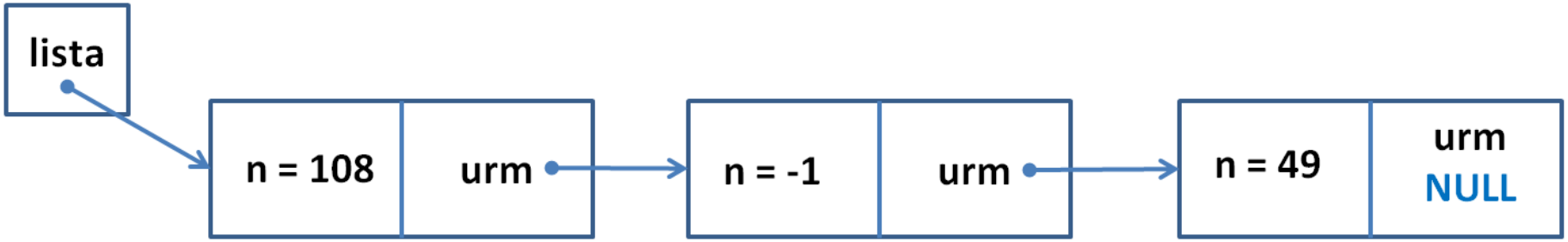
t->urm->urm->urm==NULL

t->urm->urm->urm->info nu există->Segmentation fault ☺

https://www.pbinfo.ro/articole/19576/liste-liniare-simplu-inlantuite-alocate-dinamic

Liste simplu înlănțuite- afișare

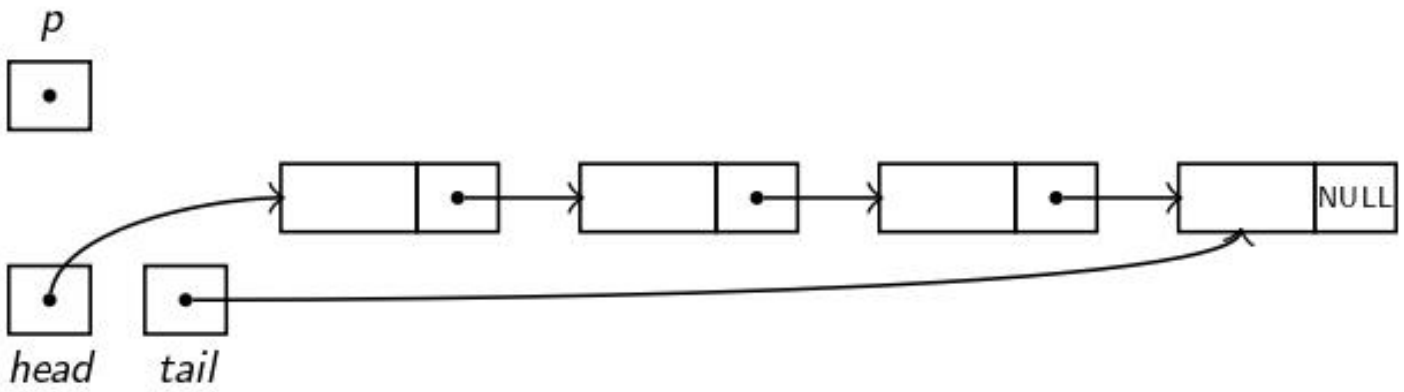
```
void afisare(elem *lista)
{  elem *p;
   for(p=lista ; p != NULL ; p=p->urm){
      printf("%d ",p->info);
      }
   printf("\n");
}
```



Singly Linked Linear Lists

Basic Operations

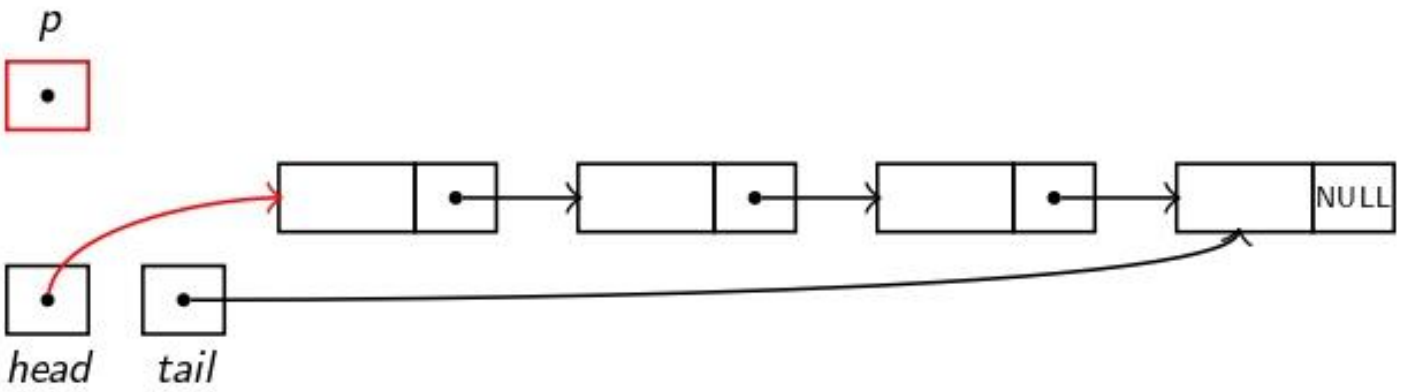
- Printing the content of a non-empty list



Singly Linked Linear Lists

Basic Operations

- Printing the content of a non-empty list

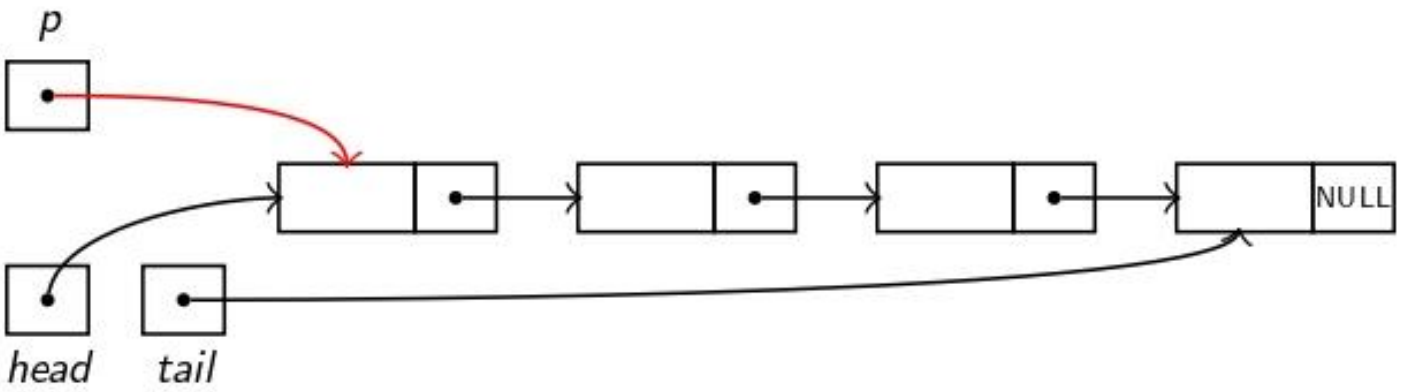



```
p = head;          /* Start with the cursor pointer
                    * from the head of the list. */
```

Singly Linked Linear Lists

Basic Operations

- Printing the content of a non-empty list

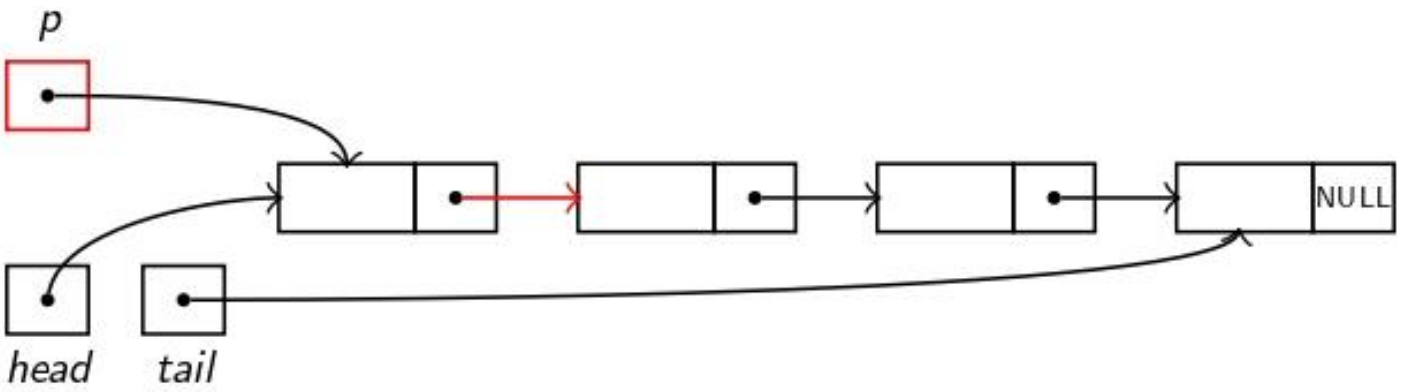



```
printf("%d ", p->info);     /* ... print the info in
                             * the node pointed by
                             * the cursor ... */
```

Singly Linked Linear Lists

Basic Operations

- Printing the content of a non-empty list

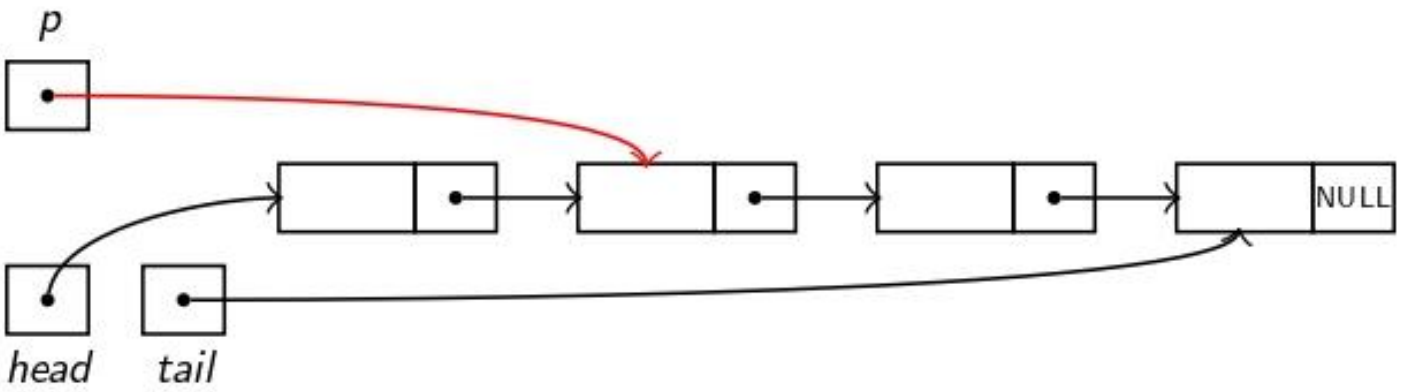



```
p = p->next;        /* ... and move the cursor ahead
                     * to the next node. */
```

Singly Linked Linear Lists

Basic Operations

- Printing the content of a non-empty list

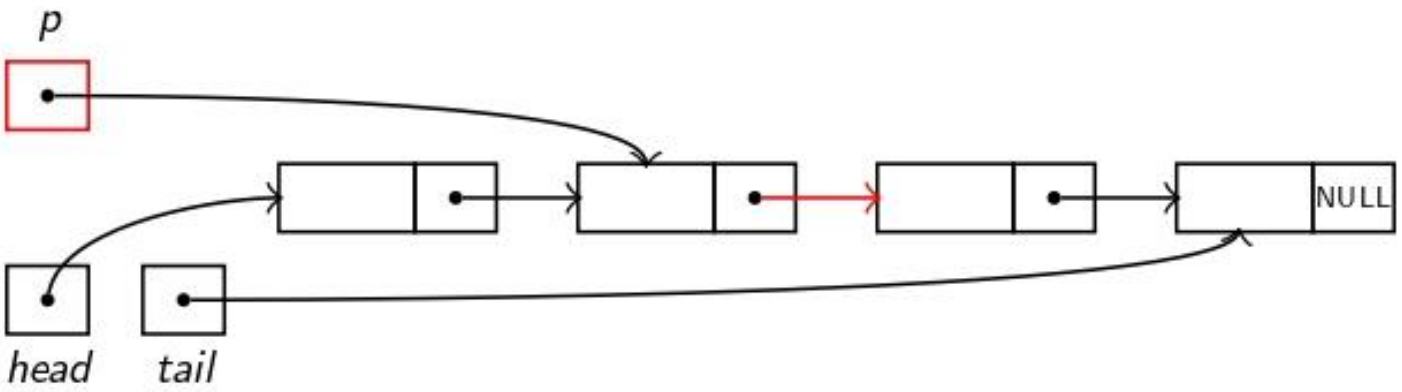



```
printf("%d ", p->info);     /* ... print the info in
                             * the node pointed by
                             * the cursor ... */
```

Singly Linked Linear Lists

Basic Operations

- Printing the content of a non-empty list

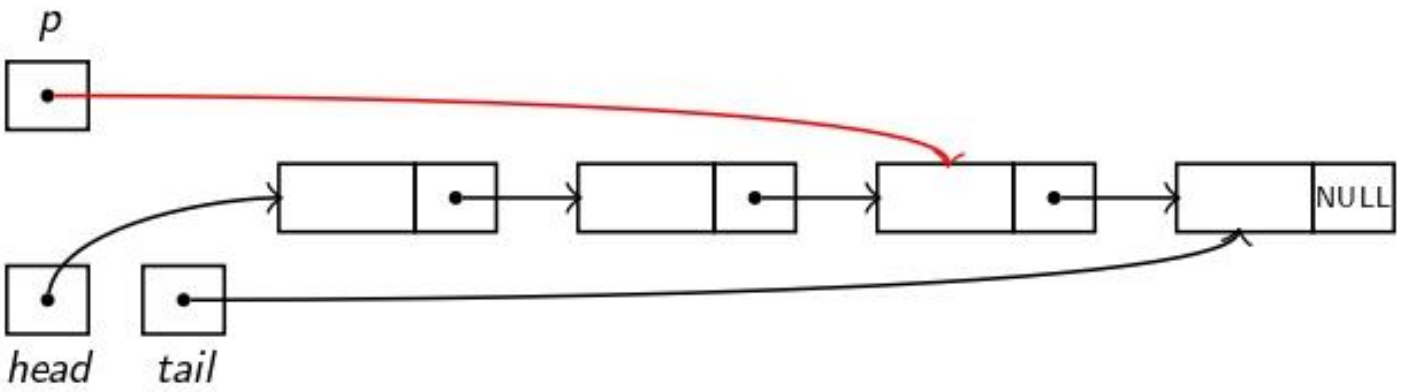



```
p = p->next;        /* ... and move the cursor ahead
                     * to the next node. */
```

Singly Linked Linear Lists

Basic Operations

- Printing the content of a non-empty list

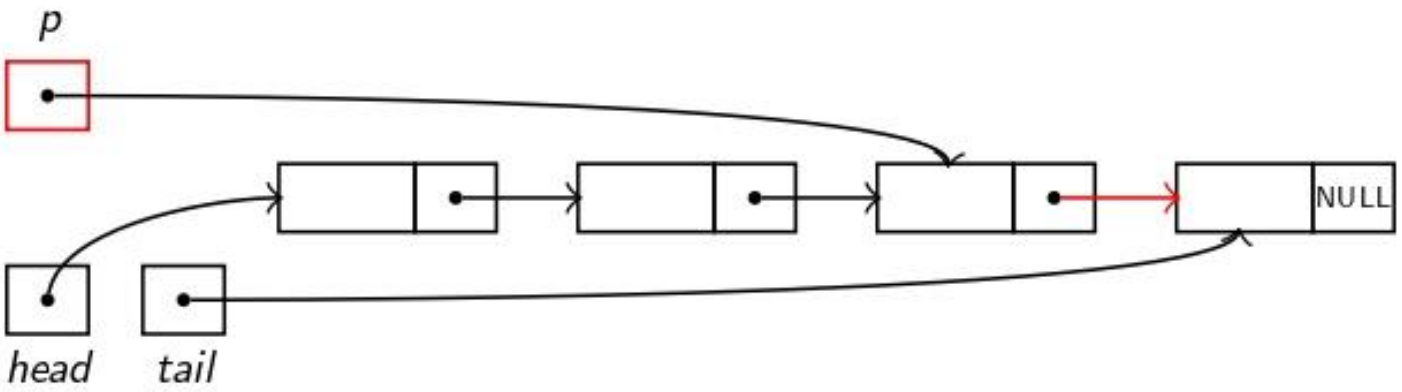



```
printf("%d ", p->info);     /* ... print the info in
                             * the node pointed by
                             * the cursor ... */
```

Singly Linked Linear Lists

Basic Operations

- Printing the content of a non-empty list

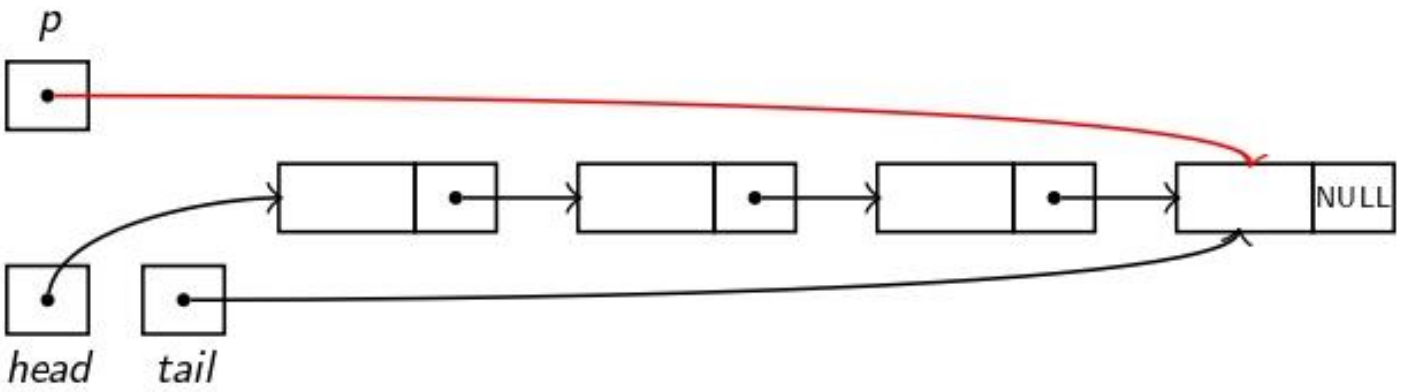



```
p = p->next;        /* ... and move the cursor ahead
                     * to the next node. */
```

Singly Linked Linear Lists

Basic Operations

- Printing the content of a non-empty list

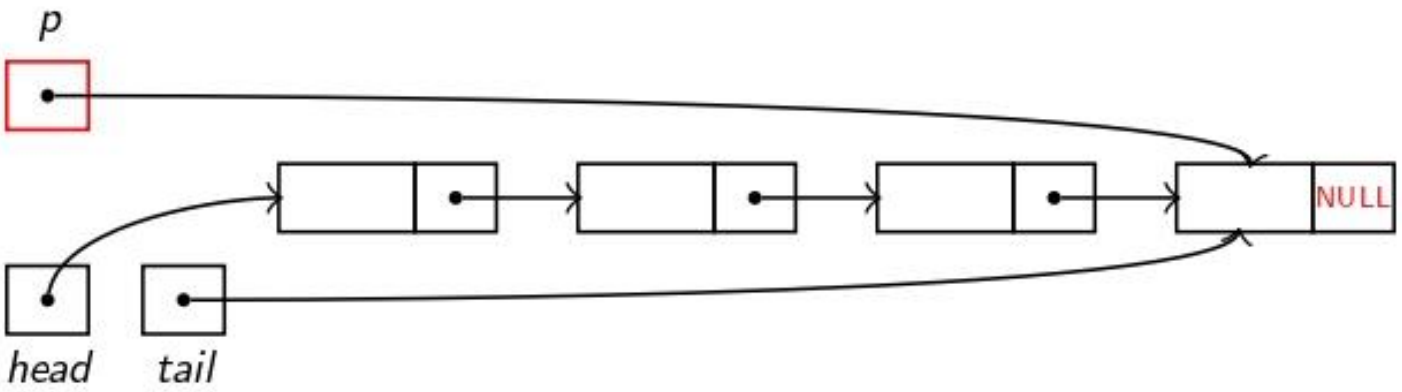



```
printf("%d ", p->info);     /* ... print the info in
                             * the node pointed by
                             * the cursor ... */
```

Singly Linked Linear Lists

Basic Operations

- Printing the content of a non-empty list

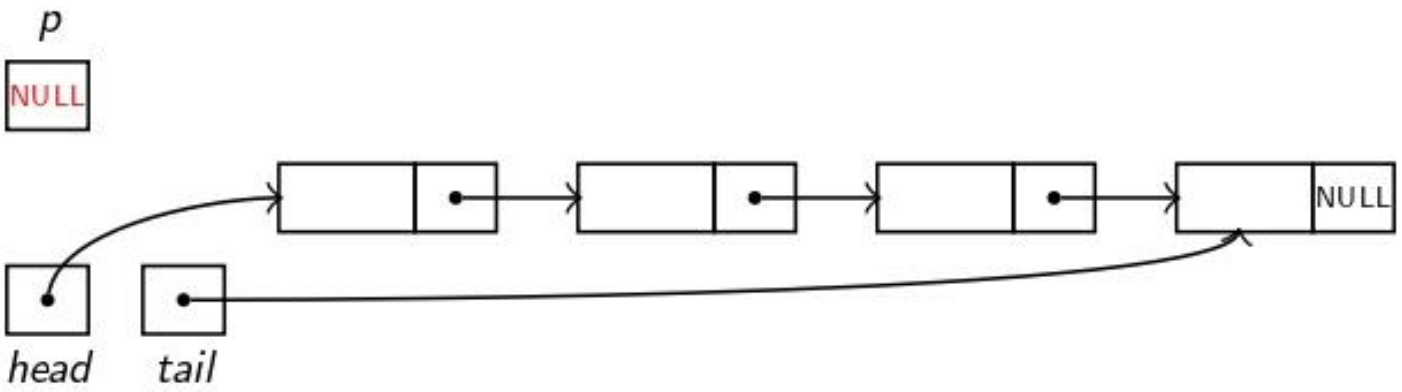



```
p = p->next;        /* ... and move the cursor ahead
                     * to the next node. */
```

Singly Linked Linear Lists

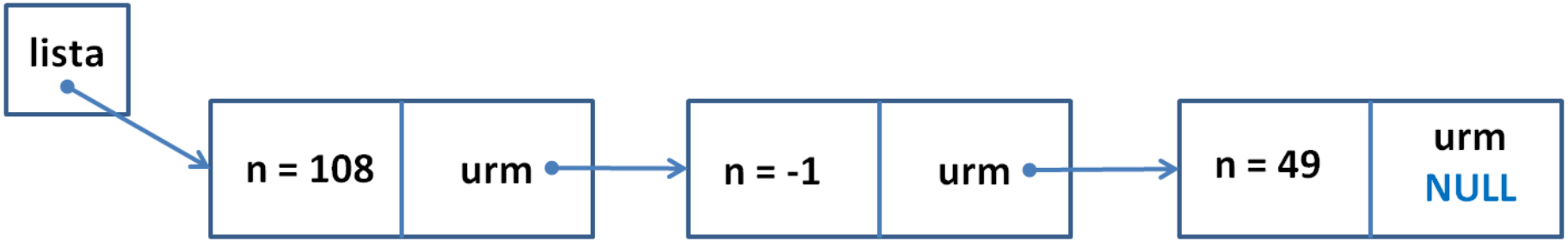
Basic Operations

- Printing the content of a non-empty list



Adăugarea unui element la început

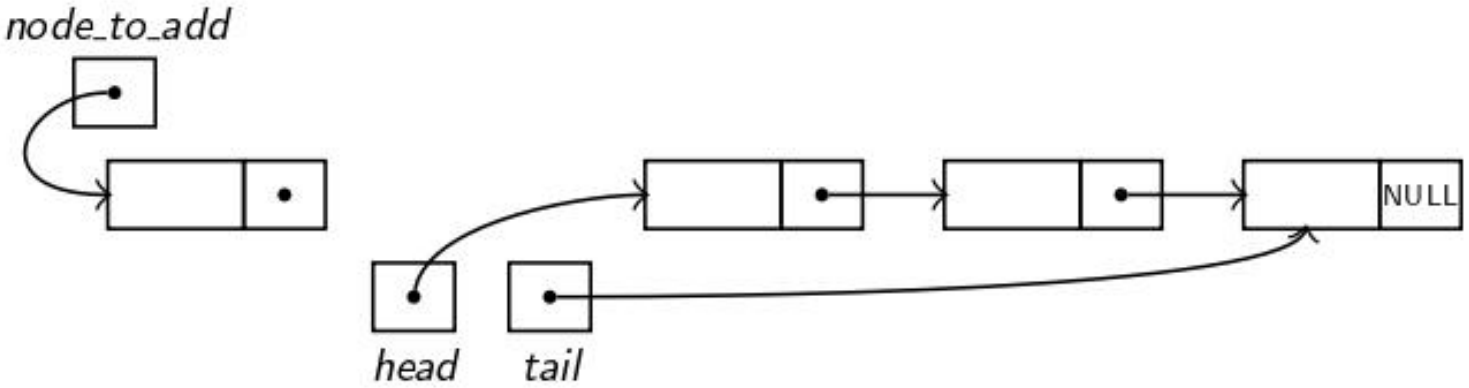
```
elem * adaugaInceput(elem *lista, int n)
{
   elem *nou;
   nou=nod_nou(n,lista);
   lista=nou;
   return lista;
}
```



Singly Linked Linear Lists

Basic Operations

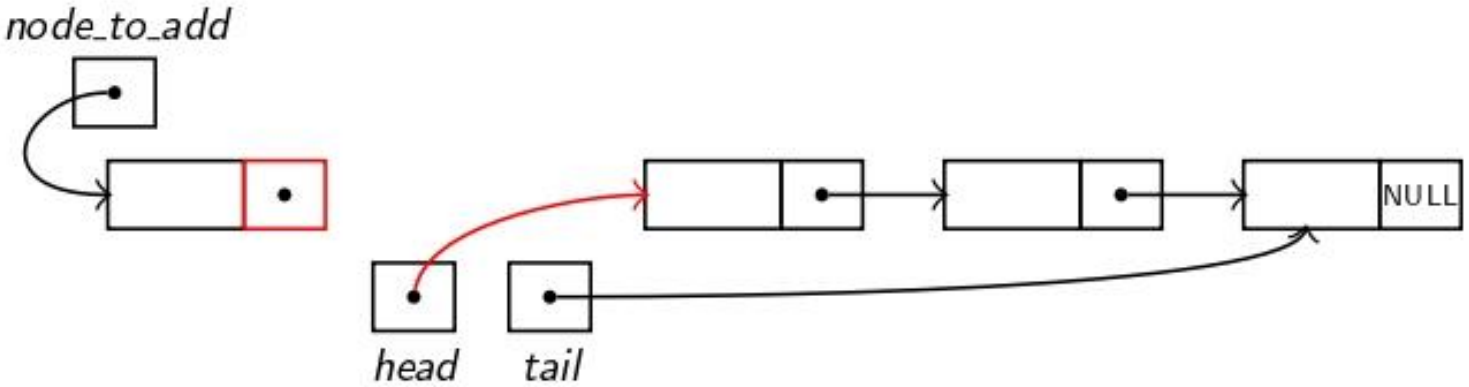
- Adding a node to the beginning of a non-empty list



Singly Linked Linear Lists

Basic Operations

- Adding a node to the beginning of a non-empty list

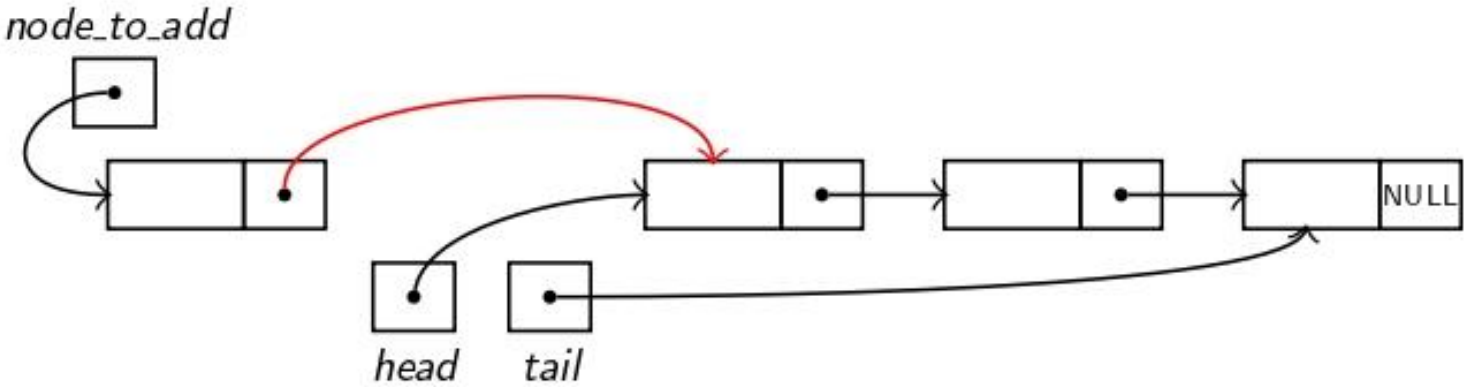



```
node_to_add->next = *head;      /* Link the head of the
                                 * list (and thus the
                                 * entire list) after the
                                 * node to be added. */
```

Singly Linked Linear Lists

Basic Operations

- Adding a node to the beginning of a non-empty list

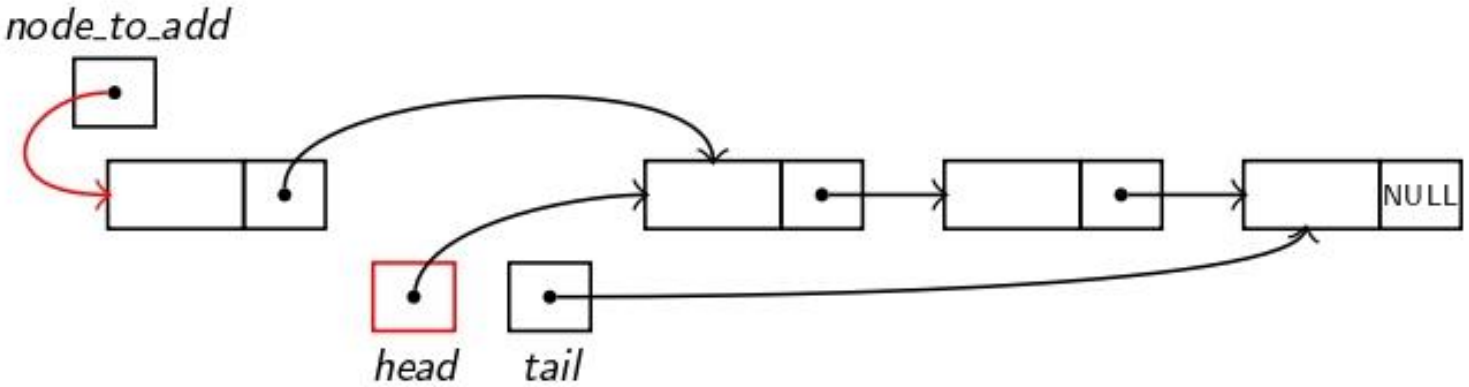



```
node_to_add->next = *head;      /* Link the head of the
                                 * list (and thus the
                                 * entire list) after the
                                 * node to be added. */
```

Singly Linked Linear Lists

Basic Operations

- Adding a node to the beginning of a non-empty list

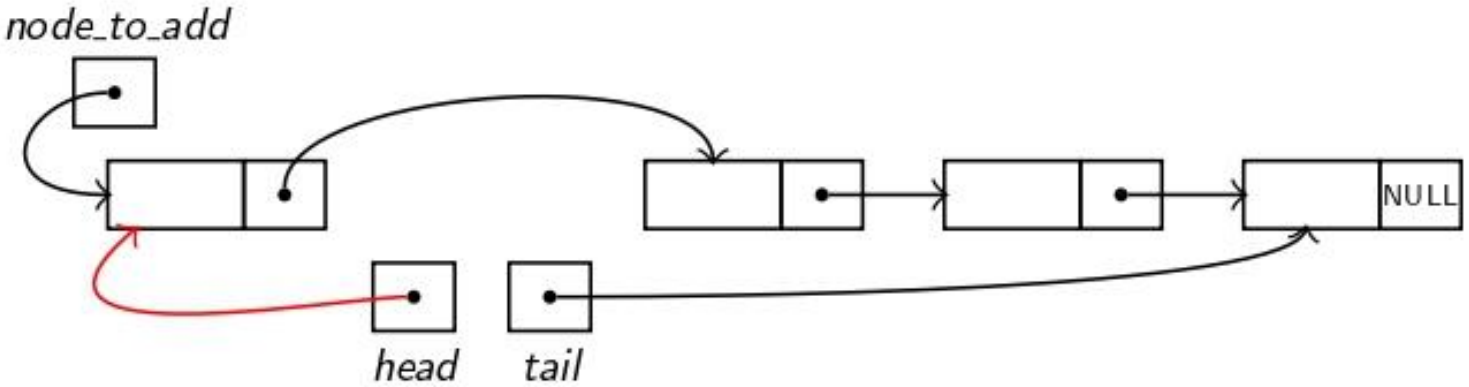



```
*head = node_to_add;   /* The head will point to the
                        * newly added node. */
```

Singly Linked Linear Lists

Basic Operations

- Adding a node to the beginning of a non-empty list

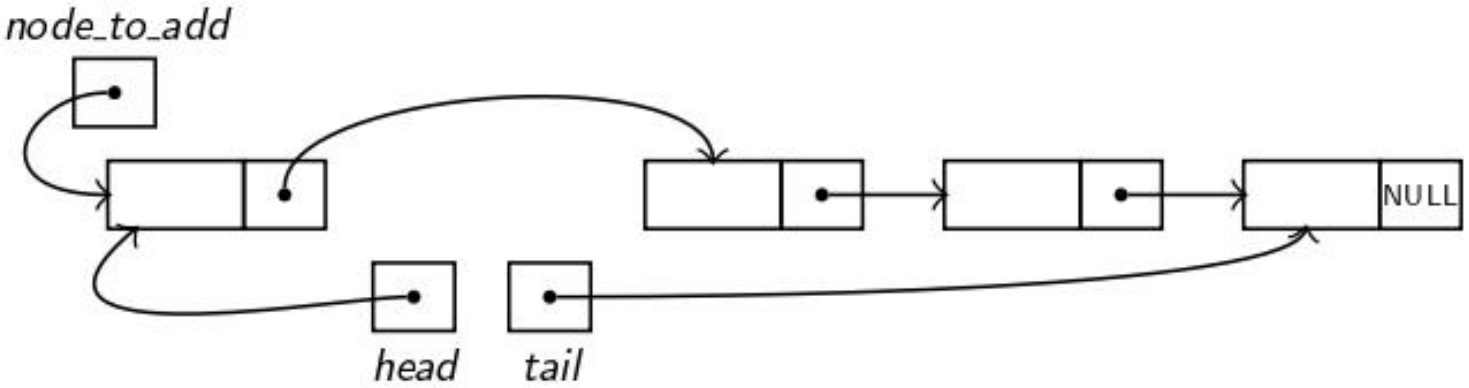



```
*head = node_to_add;   /* The head will point to the
                        * newly added node. */
```

Singly Linked Linear Lists

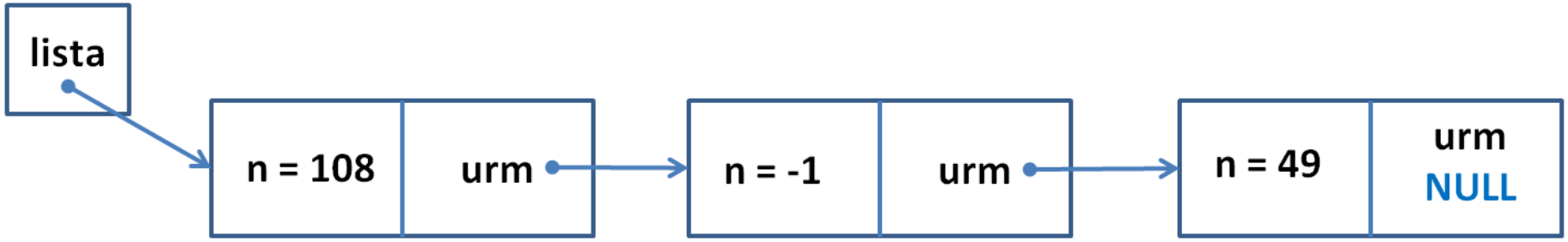
Basic Operations

- Adding a node to the beginning of a non-empty list



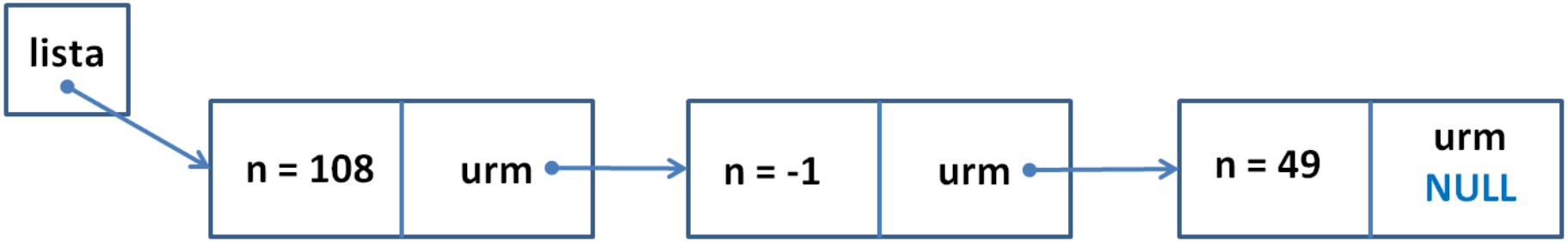
Alocarea unui nod nou și inițializarea

```
elem *nod_nou(int n,elem *urm) {
elem *p=(elem*)malloc(sizeof(elem));
   if(!p){
      printf("memorie insuficienta");
      exit(EXIT_FAILURE);  }
   p->info=n;
   p->urm=urm;
   return p;
}
```



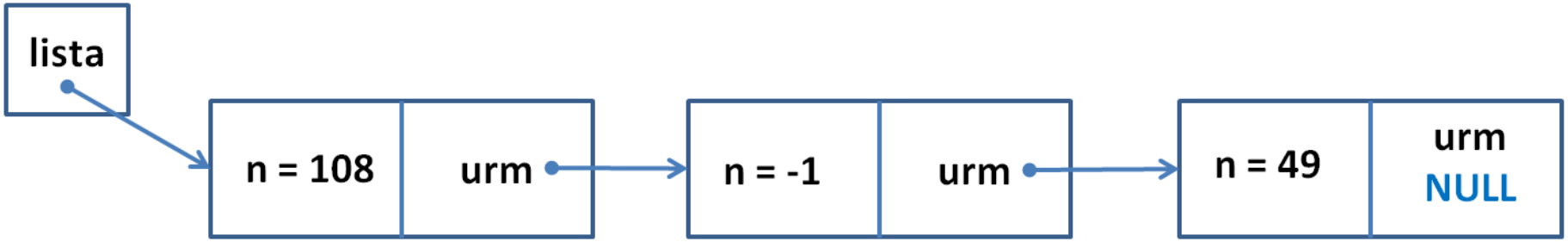
Adăugarea unui nod la sfârșitul listei

```
elem *adaugaSfarsit(elem *lista, int n) {
elem *q, *nou=nod_nou(n, NULL);
if (lista == NULL)
        lista = nou;
else {
        for(q=lista; q->urm!=NULL; q=q->urm);
        q->urm=nou;   }
return lista;
}
```



Ștergerea unui nod de la început

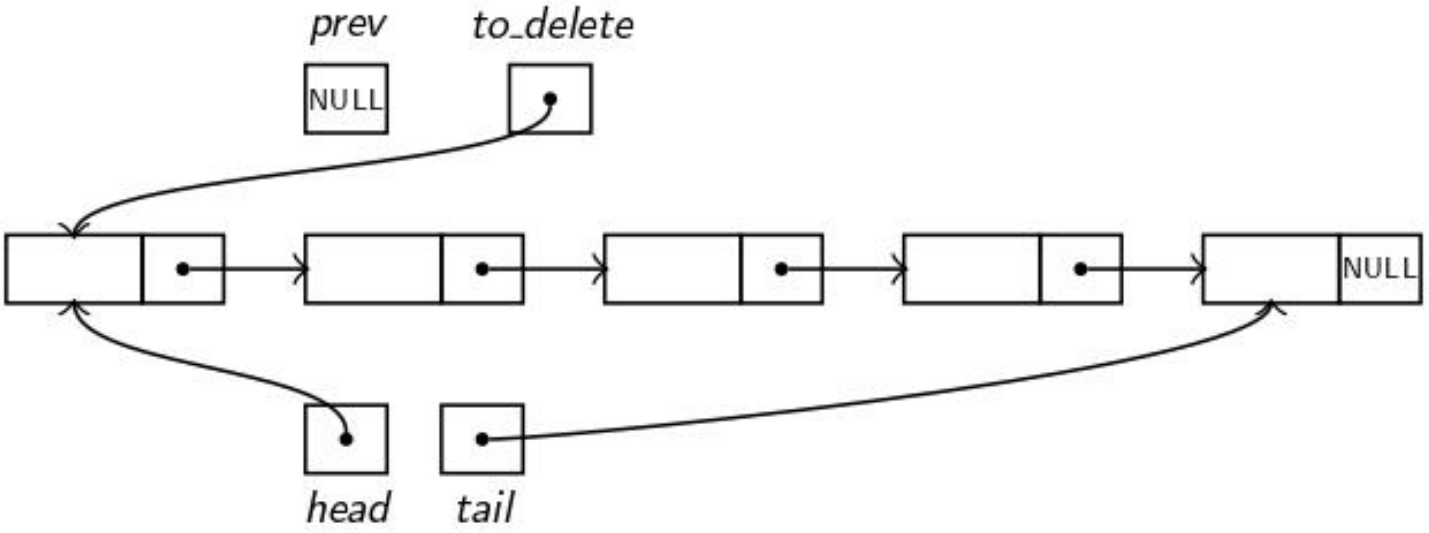
```
elem *stergeInceput(elem *lista) {
if (lista == NULL)
        return lista;
elem *p = lista;
lista = lista->urm;
free(p);
return lista;
}
```



Singly Linked Linear Lists

Basic Operations

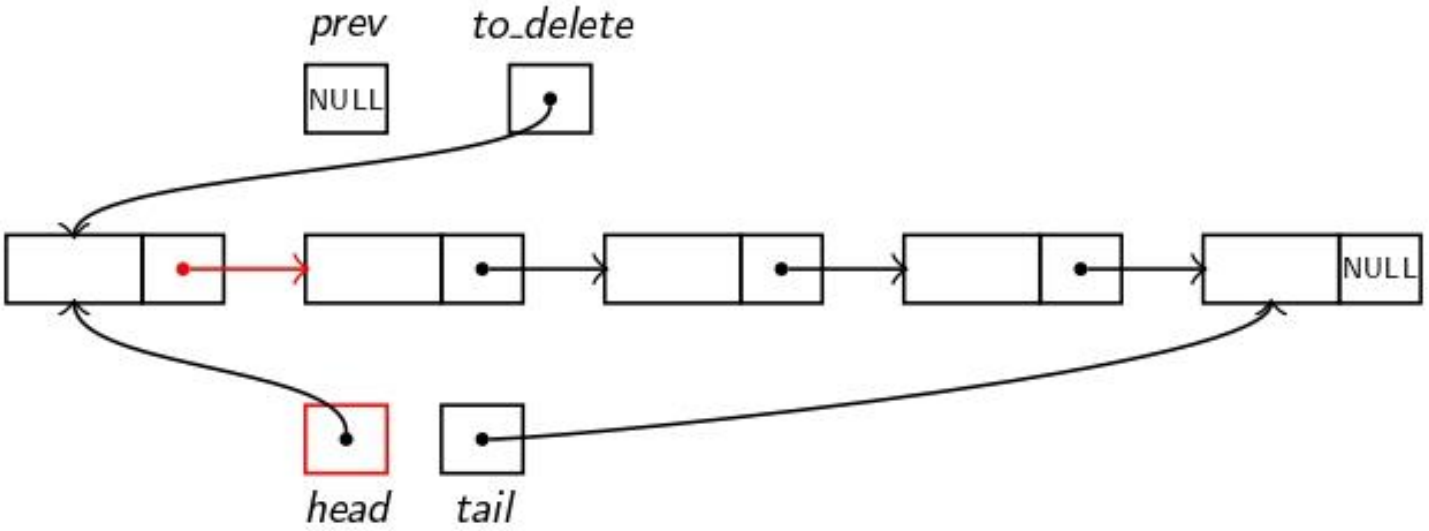
- Deleting a node from the beginning of a list



Singly Linked Linear Lists

Basic Operations

- Deleting a node from the beginning of a list

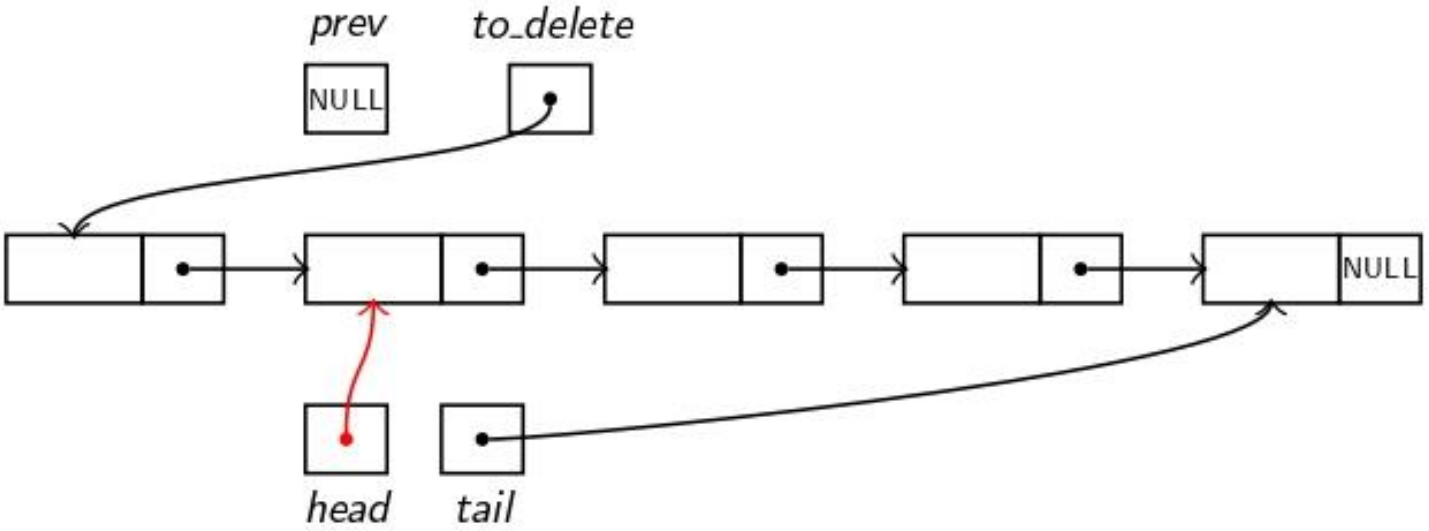



```
*head = (*head)->next;      /* Move the head to point
                             * to the next node. */
```

Singly Linked Linear Lists

Basic Operations

- Deleting a node from the beginning of a list

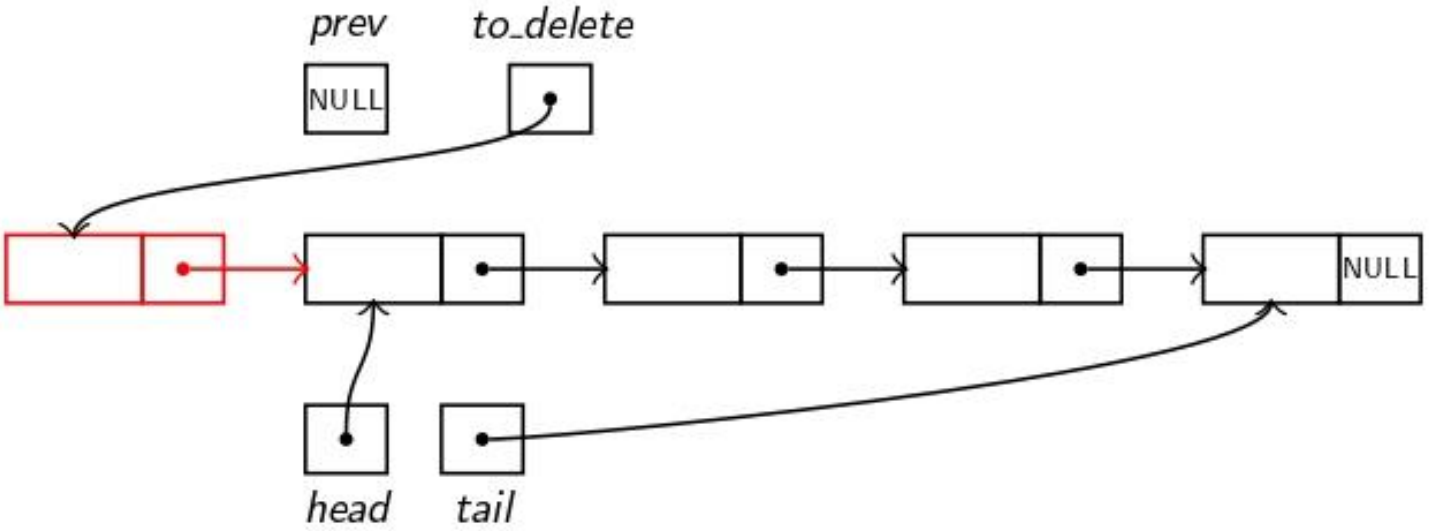



```
*head = (*head)->next;      /* Move the head to point
                             * to the next node. */
```

Singly Linked Linear Lists

Basic Operations

- Deleting a node from the beginning of a list

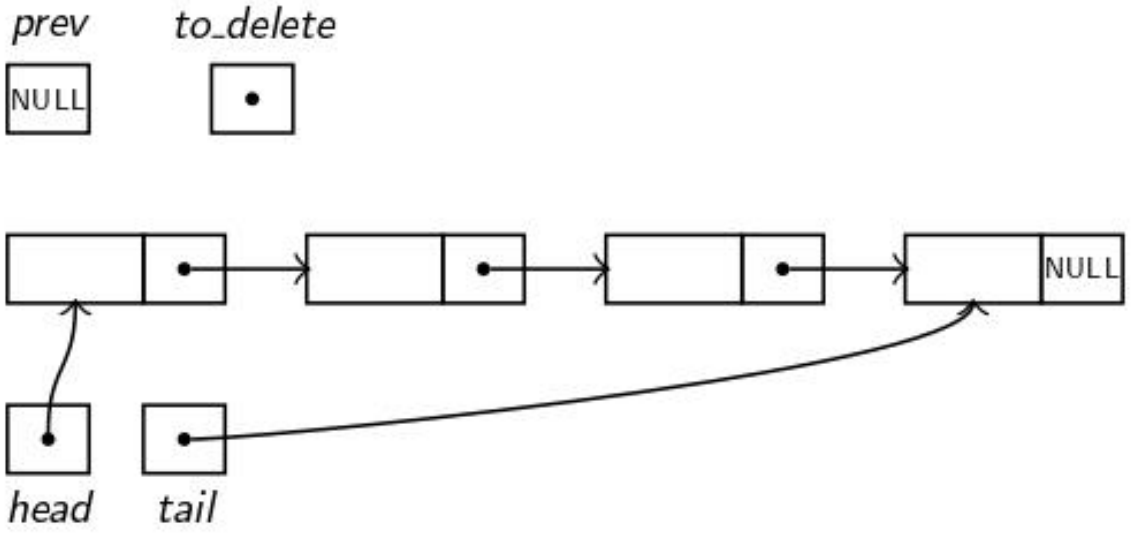



```
free(to_delete);      /* Don't forget: free the memory
                       * used by the node. */
```

Singly Linked Linear Lists

Basic Operations

- Deleting a node from the beginning of a list

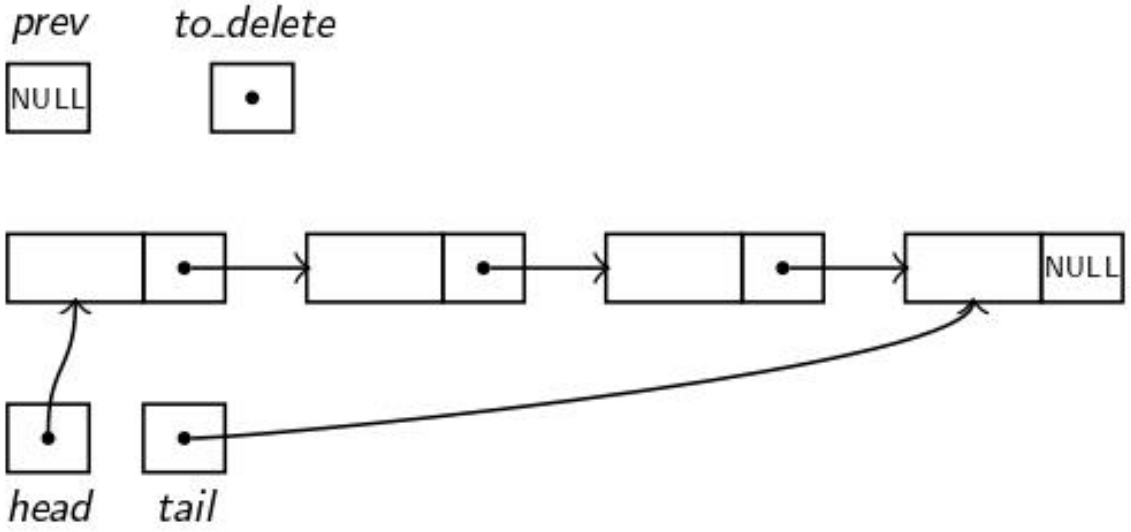



```
free(to_delete);      /* Don't forget: free the memory
                       * used by the node. */
```

Singly Linked Linear Lists

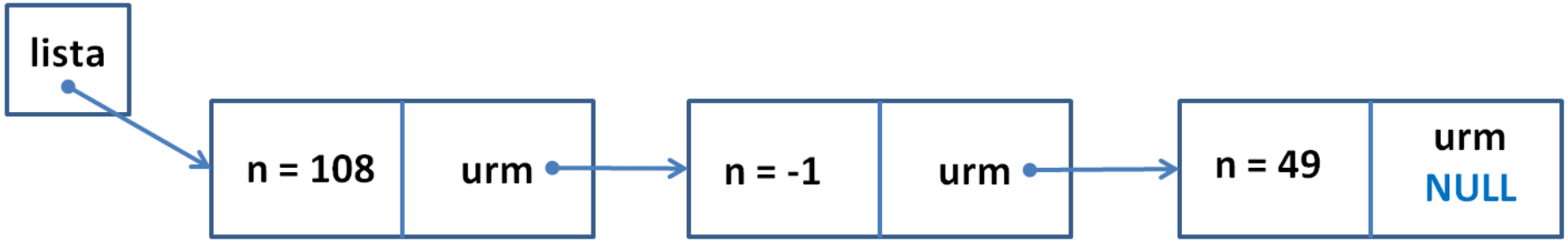
Basic Operations

- Deleting a node from the beginning of a list



Ștergerea unui nod de la sfârșitul listei

```
elem *stergeSfarsit(elem *lista) {
elem *p, *q;
if (lista == NULL)    return lista;
if(lista->urm == NULL) {
            free(lista);
            return lista = NULL; }
for (p = lista; p->urm->urm != NULL; p=p->urm);
q=p->urm;
p->urm=NULL;
free(q);
return lista;
}
```



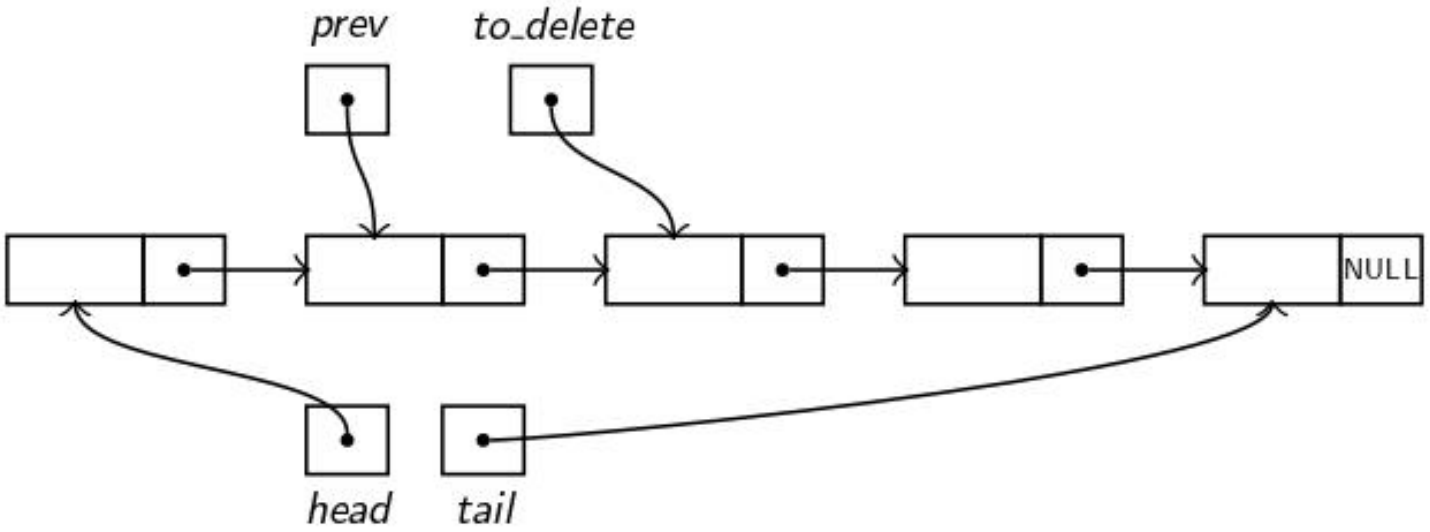
Ștergerea unui nod specific

```
elem *stergeElement(elem *lista, int n) {
elem *prev,*p;
if (lista == NULL)
        return lista;
if (lista->info==n)        {
        p=lista;
        lista=lista->urm;
        free(p);           }
else
    for (prev=lista, p=lista->urm; p!=NULL; prev=prev->urm, p=p->urm)
                if(p->info==n) {
                        prev->urm=p->urm;
                        free(p);
                        break;   }
return lista;
}
```

Singly Linked Linear Lists

Basic Operations

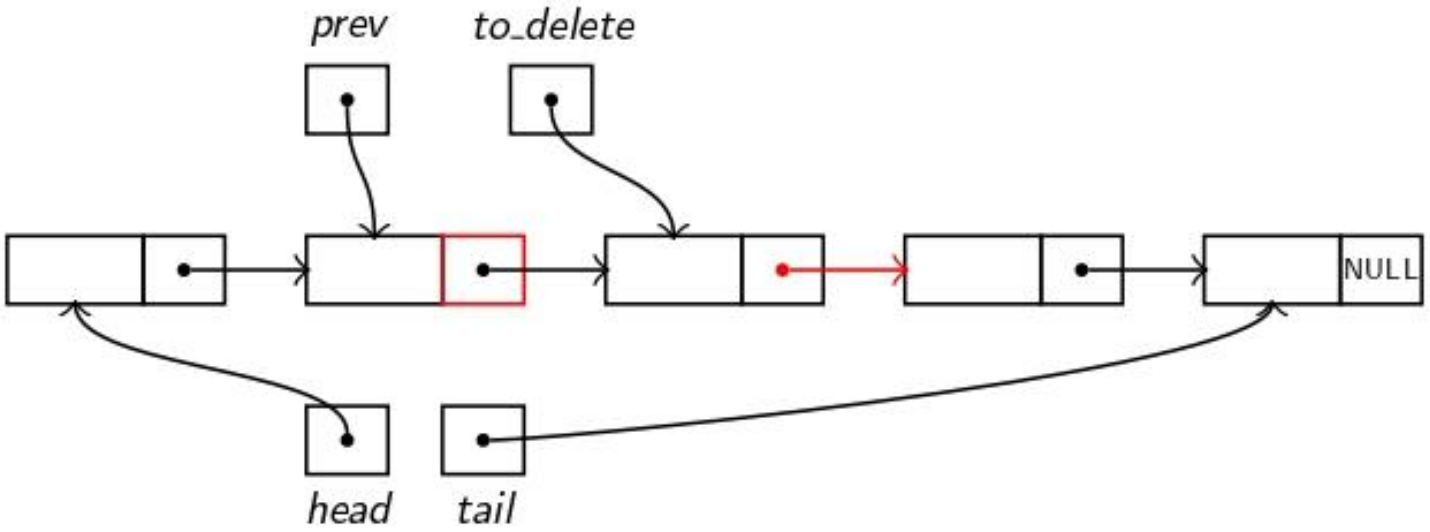
- Deleting a node from the middle of a list



Singly Linked Linear Lists

Basic Operations

- Deleting a node from the middle of a list

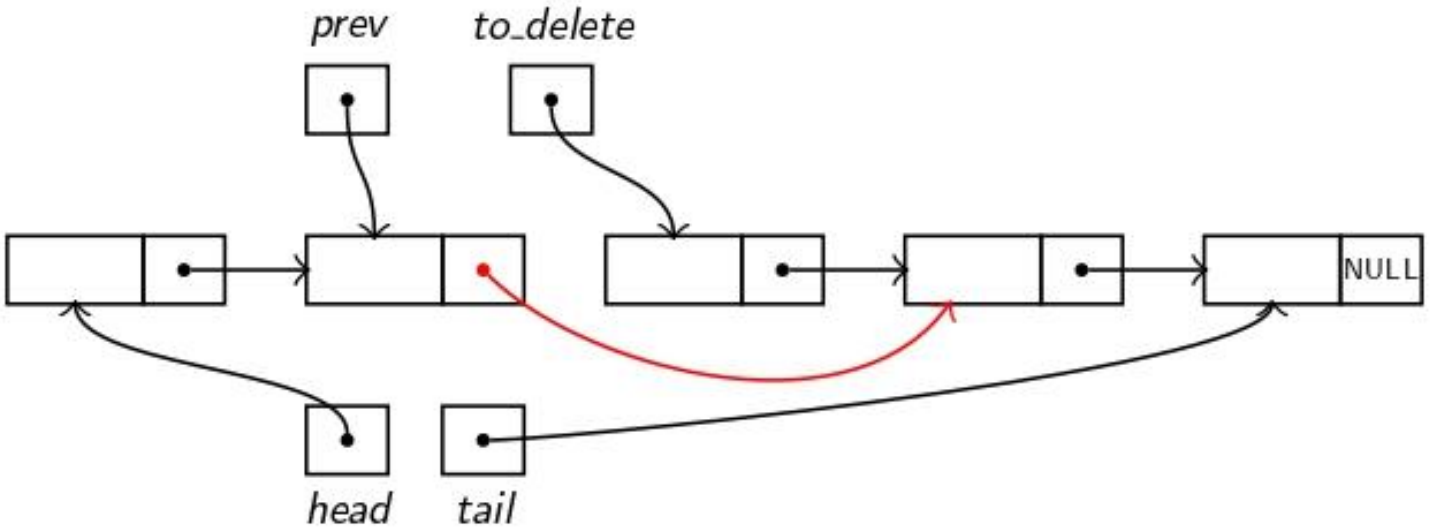



```
prev->next = to_delete->next;        /* Remove from
                                      * the list the
                                      * node to be
                                      * deleted. */
```

Singly Linked Linear Lists

Basic Operations

- Deleting a node from the middle of a list

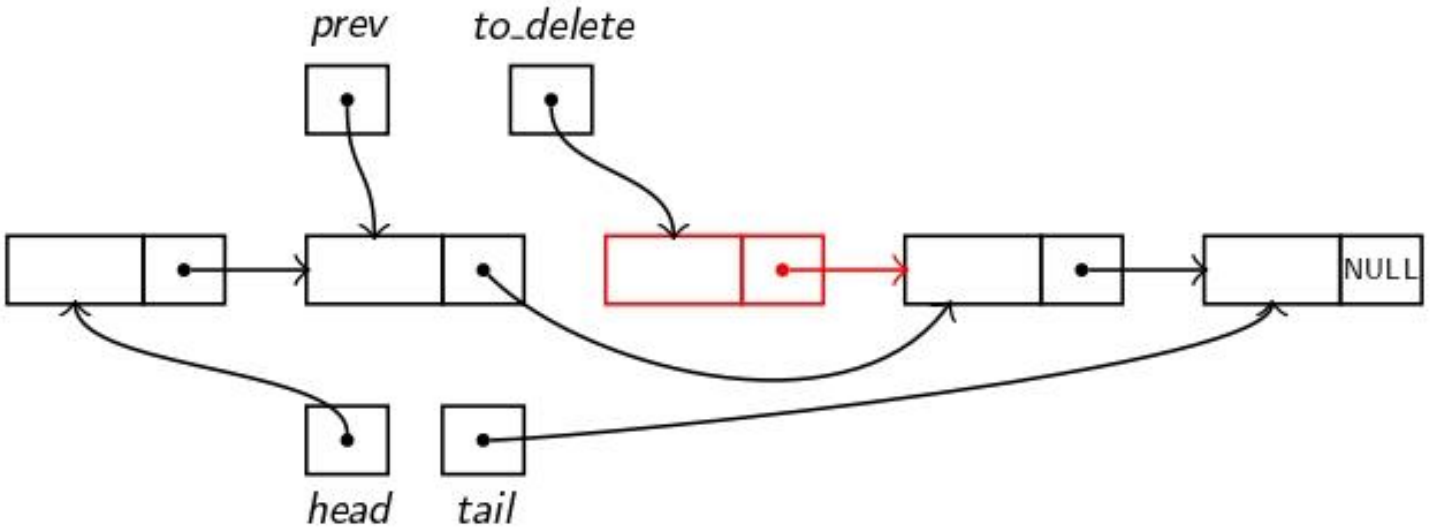



```
prev->next = to_delete->next;        /* Remove from
                                      * the list the
                                      * node to be
                                      * deleted. */
```

Singly Linked Linear Lists

Basic Operations

- Deleting a node from the middle of a list

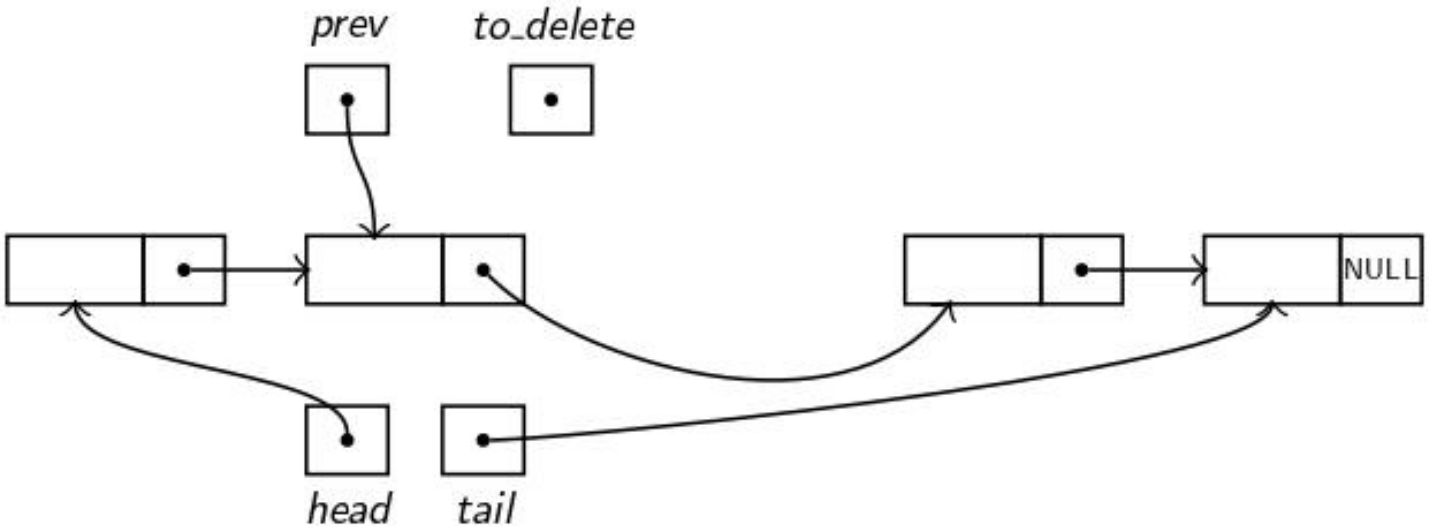



```
free(to_delete);      /* Don't forget: free the memory
                       * used by the node. */
```

Singly Linked Linear Lists

Basic Operations

- Deleting a node from the middle of a list

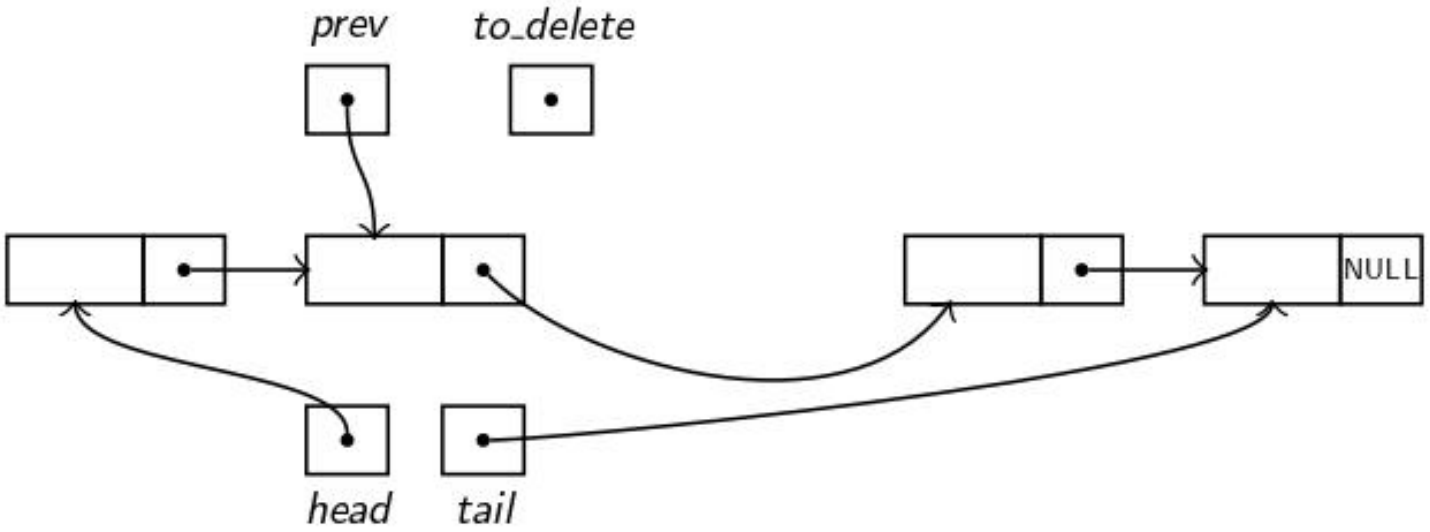



```
free(to_delete);      /* Don't forget: free the memory
                       * used by the node. */
```

Singly Linked Linear Lists

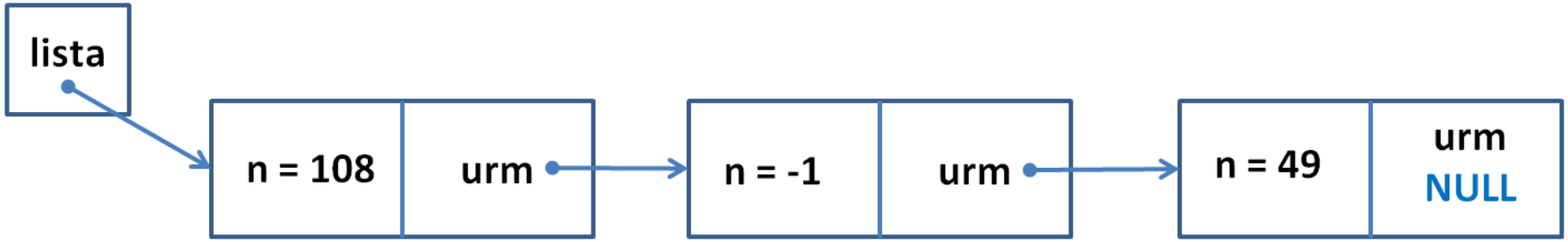
Basic Operations

- Deleting a node from the middle of a list



Eliberarea memoriei ocupate de lista

```
void eliberare(elem *lista)
{
elem *p;
while(lista != NULL)   {
        p=lista->urm;
        free(lista);
        lista=p;   }
}
```



Apelarea funcțiilor

```
7
8 7
9 8 7
9 8 7 12
9 8 7 12 15
8 7 12 15
8 7 12
8 12
8 12
8
```

```
int main(){
elem *lista = NULL;

lista = adaugaInceput(lista, 7);
afisare(lista);
lista = adaugaInceput(lista, 8);
afisare(lista);
lista = adaugaInceput(lista, 9);
afisare(lista);
lista = adaugaSfarsit(lista, 12);
afisare(lista);
lista = adaugaSfarsit(lista, 15);
afisare(lista);
lista = stergeInceput(lista);
afisare(lista);
lista = stergeSfarsit(lista);
afisare(lista);
lista = stergeElement(lista, 7);
afisare(lista);
lista = stergeElement(lista, 20);
afisare(lista);
lista = stergeElement(lista, 12);
afisare(lista);
eliberare(lista);
return 0;}
```

Exercițiu

Aplicația 7.2: Să se scrie o funcție care primește o listă și returnează lista respectivă cu elementele inversate. Funcția va acționa doar asupra listei originare, fără a folosi vectori sau alocare de noi elemente.

Exercițiu

```
elem *inversare(elem *lista){
if(lista == NULL)    {
return lista;    }
elem *prev, *q,*aux;
for(prev=lista, q=lista->urm;q->urm!= NULL; prev = q, q=aux)
{
        aux=q->urm;
        q->urm = prev;    }
q->urm=prev;
lista->urm=NULL;
lista=q;
return lista;
}
```

Vă mulțumesc!