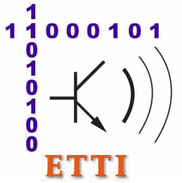
UNIVERSITATEA
POLITEHNICA
DIN BUCUREȘTI



Liste simplu înlănțuite

Cuprins

Partea I – Noțiuni teoretice

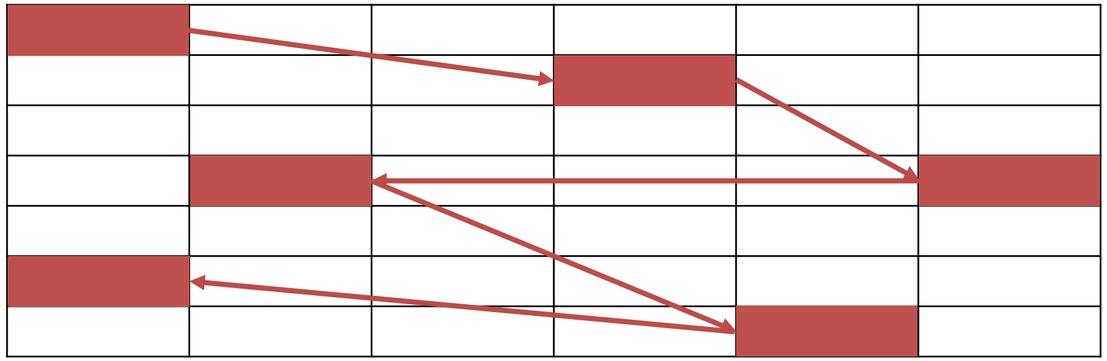
1. Noțiuni fundamentale
2. Operații de bază
3. Cazuri particulare de liste

Partea II – Aplicații Laborator

Rezolvare aplicații Moodle

Definiție: structură dinamică de date cu elemente conectate prin legături. O listă este formată din mai multe noduri care conțin datele utile și legătura către următorul nod. Nodurile consecutive ale unei liste nu sunt forțate să ocupe zone de memorie adiacente.


```
typedef struct node{
    int data;
    struct node *next;
} node_t;
```



Lista este utilizată prin intermediul unui pointer către primul nod (dacă există) – `head`
Ultimul nod nu este relaționat “următor” cu niciun alt nod, i.e. legătura va lua valoarea `NULL`

	Liste simplu înlănțuite	Tablouri multidimensionale
Dimensiune	variabilă	fixă
Acces aleator la elemente	nu	da
Zonă de memorie ocupată	mai multe blocuri mici de memorie	un singur bloc mare de memorie
Necesitate spațiu de memorie suplimentar	da, pentru stocarea legăturilor	nu
Ștergere/inserție de elemente	operație simplă, se acționează asupra unui singur nod	operație complexă, necesită deplasarea elementelor
Dificultate de implementare	crescută, necesită atenție sporită pentru a evita utilizarea improprie a memoriei (memory leaks)	simplă

2. Operații de bază

Listele simplu înlănțuite permit utilizarea unui set de operații de bază:

a) Parcurgere listă

b) Adăugare element la finalul listei

c) Adăugare element la începutul listei

d) Adăugare element pe o poziție intermediară

e) Ștergere element de la finalul listei

f) Ștergere element de la începutul listei

g) Ștergere element de pe o poziție intermediară

a) Parcurgere listă

```
typedef struct node{
    int data;
    struct node *next;
} node_t;
```

1. Se creează un pointer la nod `*ptr` care ia valoarea `head`.
2. Cât timp `ptr` e diferit de `NULL` se afișează data nodului și se avansează cu `ptr` pe poziția `ptr->next`.

```
void afisare_lista(node_t *head) {
    node_t *ptr = head;

    while(ptr != NULL){
        printf("%d ", ptr->data);
        ptr = ptr->next;
    }
}
```

b) Adăugare element la finalul listei

```
typedef struct node{
    int data;
    struct node *next;
} node_t;
```

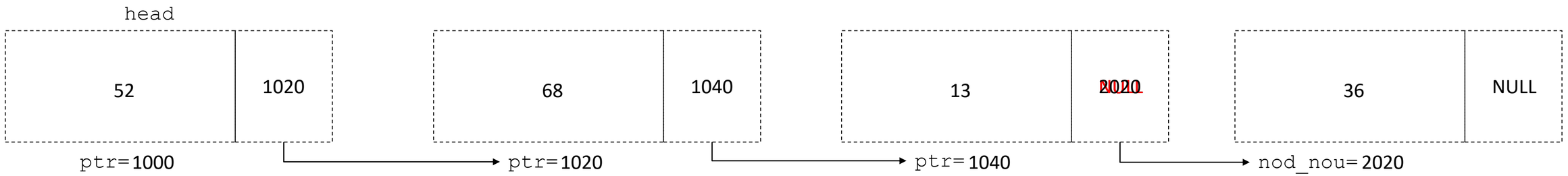
1. Se verifică dacă lista este goală.
   - Dacă lista este goală se creează un nod nou, `head`, și se returnează pointerul către acesta.
   - Dacă lista nu este goală se continuă cu pasul următor.
2. Se parcurge lista până se ajunge la finalul ei (nodul a cărui legătură are valoarea `NULL`).
3. Se creează un nod nou, a cărui legătură are valoarea `NULL`.
4. Se stabilește legătura nodului care era pe ultima poziție către nodul nou.
5. Se returnează pointerul către primul nod.

```
node_t * adaugare_nod_sfarsit_lista(node_t * head) {
  node_t * ptr, * nod_nou;
  if (head == NULL) {
    head = malloc(sizeof node_t);
    if (head != NULL){
      scanf("%d", head -> data);
      head -> next = NULL;
    }
    return head;
  }
  ptr = head;
  while (ptr -> next != NULL){
    ptr = ptr -> next;
  }
  nod_nou = malloc(sizeof node_t);
  if (nod_nou != NULL){
    scanf("%d", nod_nou -> data);
    nod_nou -> next = NULL;
  }
  ptr -> next = nod_nou;
  return head;
}
```

b) Adăugare element la finalul listei

```
typedef struct node{
    int data;
    struct node *next;
} node_t;
```

1. Se verifică dacă lista este goală.
   - Dacă lista este goală se creează un nod nou, `head`, și se returnează pointerul către acesta.
   - Dacă lista nu este goală se continuă cu pasul următor.
2. Se parcurge lista până se ajunge la finalul ei (nodul a cărui legătură are valoarea NULL).
3. Se creează un nod nou, a cărui legătură are valoarea NULL.
4. Se stabilește legătura nodului care era pe ultima poziție către nodul nou.
5. Se returnează pointerul către primul nod.



c) Adăugare element la începutul listei

```
typedef struct node{
    int data;
    struct node *next;
} node_t;
```

1. Se verifică dacă lista este goală.
   - Dacă lista este goală, atunci această operație coincide cu adăugarea primului element la listă.
   - Dacă lista nu este goală se continuă cu pasul următor.
2. Se creează un nod nou, a cărui legătură ia valoarea `head`.
3. Se returnează pointerul către noul nod.

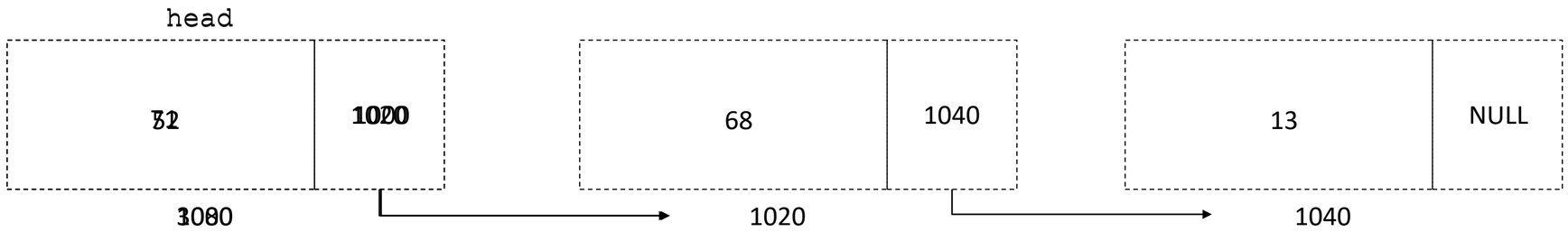
```
node_t * adaugare_nod_inceput_lista(node_t * head) {
  node_t * nod_nou;
  if (head == NULL) {
    head = malloc(sizeof node_t);
    if (head != NULL){
      scanf("%d", head -> data);
      head -> next = NULL;
    }
    return head;
  }

  nod_nou = malloc(sizeof node_t);
  if (nod_nou != NULL){
    scanf("%d", nod_nou -> data);
    nod_nou -> next = head;
  }
  return nod_nou;
}
```

c) Adăugare element la începutul listei

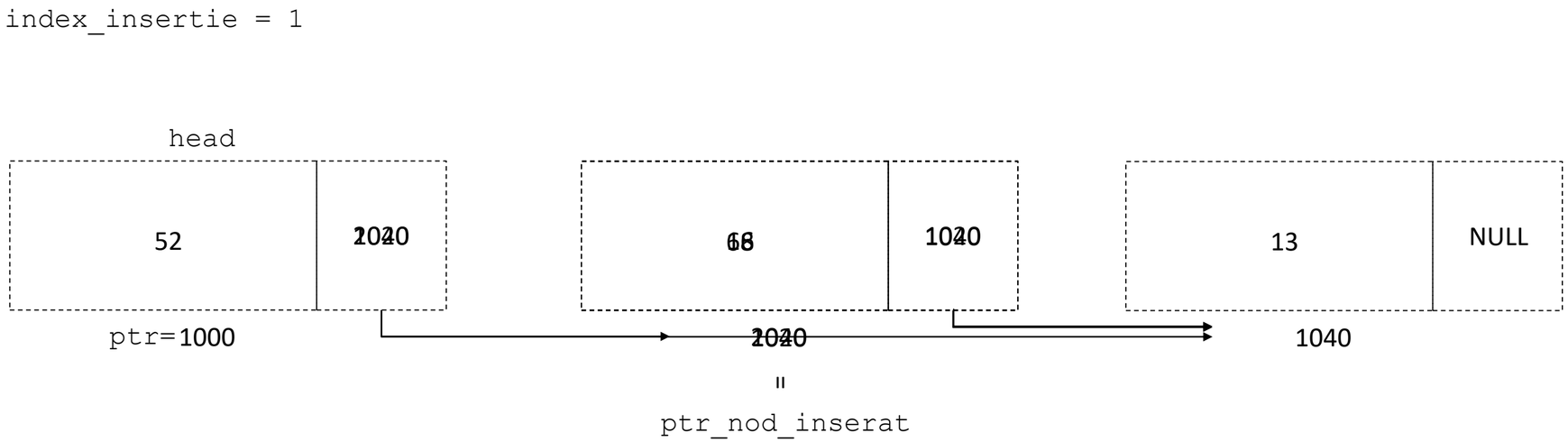
```
typedef struct node{
    int data;
    struct node *next;
} node_t;
```

1. Se verifică dacă lista este goală.
   - Dacă lista este goală, atunci această operație coincide cu adăugarea primului element la listă.
   - Dacă lista nu este goală se continuă cu pasul următor.
2. Se creează un nod nou, a cărui legătură ia valoarea `head`.
3. Se returnează pointerul către noul nod.



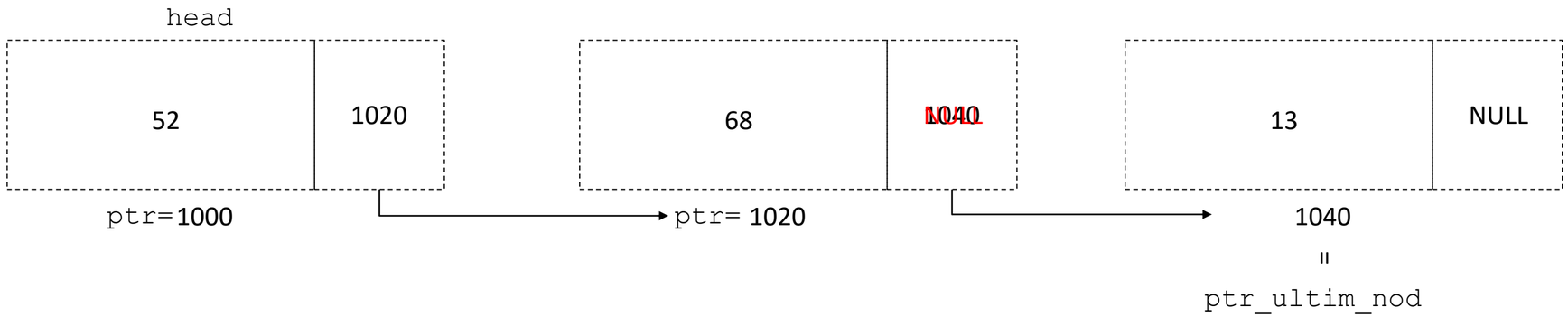
d) Adăugare element pe poziție intermediară

1. Se verifică dacă poziția selectată, `index_insertie`, este validă (≤ nr. de elemente din listă).
   - Dacă indexul este 0, operația este identică cu adăugarea unui element la începutul listei.
   - Dacă indexul este egal cu lungimea listei, operația este identică cu adăugarea unui element la sfârșitul listei.
   - Dacă indexul este valid și diferit de 0, respectiv de lungimea listei, se continuă cu următorul pas.
2. Se parcurge lista până se ajunge la elementul de index `index_insertie-1`.
3. Se creează un nod nou și se stabilește legătura sa către nodul de index `index_insertie`.
4. Se stabilește legătura nodului de index `index_insertie-1` către noul nod.
5. Se returnează pointerul către primul nod.



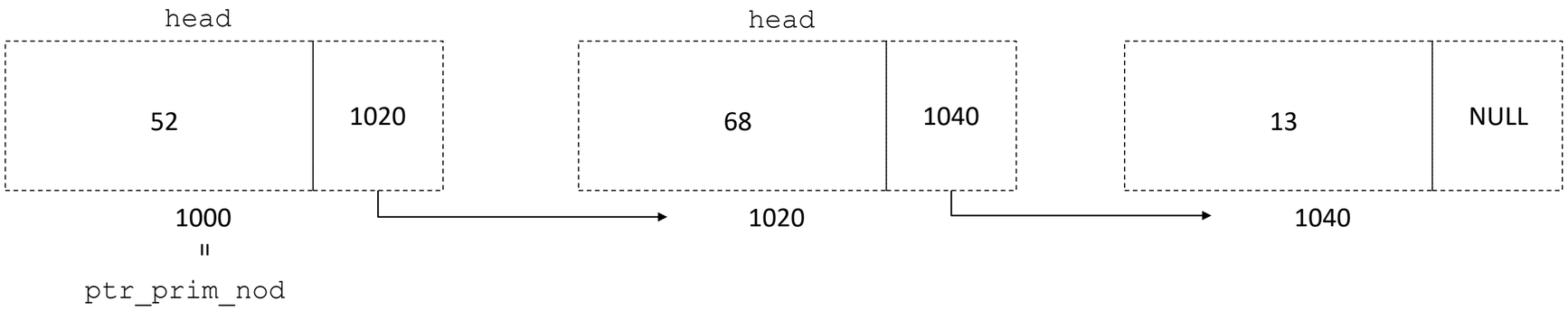
e) Ștergere element de la finalul listei

1. Se verifică dacă lista este goală.
   - Dacă lista este goală se afișează un mesaj de eroare și se returnează valoarea `NULL`.
   - Dacă lista nu este goală, se continuă cu pasul următor.
2. Se verifică dacă există un singur element în listă.
   - Dacă lista este formată dintr-un singur element, acesta se șterge cu instrucțiunea `free()` și se returnează `NULL`.
   - Dacă lista conține mai mult de un element, se continuă cu pasul următor.
3. Se parcurge lista până la penultimul element.
4. Se creează un nou pointer `ptr_ultim_nod` către ultimul nod din listă.
5. Legătura penultimului nod se modifică la valoarea `NULL`.
6. Se șterge ultimul nod din listă, către care indică `ptr_ultim_nod`.
7. Se returnează pointerul către primul nod.



f) Ștergere element de la începutul listei

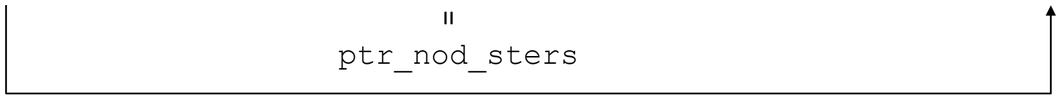
1. Se verifică dacă lista este goală.
   - Dacă lista este goală se afișează un mesaj de eroare și se returnează valoarea `NULL`.
   - Dacă lista nu este goală, se continuă cu pasul următor.
2. Se verifică dacă există un singur element în listă.
   - Dacă lista este formată dintr-un singur element, acesta se șterge cu instrucțiunea `free()` și se returnează `NULL`.
   - Dacă lista conține mai mult de un element, se continuă cu pasul următor.
3. Se creează un nou pointer, `ptr_prim_nod`, către primul nod din listă.
4. Se reține adresa celui de-al doilea nod din listă în `head`.
5. Se șterge primul nod din listă, către care indică `ptr_prim_nod`.
6. Se returnează pointerul către primul nod.



g) Ștergere element de pe poziție intermediară

1. Se verifică dacă lista este goală.
   - Dacă lista este goală se afișează un mesaj de eroare și se returnează valoarea `NULL`.
   - Dacă lista nu este goală, se continuă cu pasul următor.
2. Se verifică dacă există un singur element în listă.
   - Dacă lista este formată dintr-un singur element, acesta se șterge cu instrucțiunea `free()` și se returnează `NULL`.
   - Dacă lista conține mai mult de un element, se continuă cu pasul următor.
3. Se verifică dacă poziția selectată, `index_stergere`, este validă (≤ nr. elemente din listă).
   - Dacă indexul este 0, operația este identică cu ștergerea primului element din listă.
   - Dacă indexul este egal cu lungimea listei, operația este identică cu ștergerea ultimului element din listă.
   - Dacă indexul este valid și diferit de 0, respectiv de lungimea listei, se continuă cu pasul următor.
4. Se parcurge lista până se ajunge la elementul de `index index_stergere-1`.
5. Se reține adresa nodului de index `index_stergere` într-un pointer, `ptr_nod_sters`.
6. Se face legătura elementului de index `index_stergere-1` către nodul de index `index_stergere+1`.
7. Se șterge elementul de index `index_stergere`, prin intermediul pointerului `ptr_nod_sters`.
8. Se returnează pointerul către primul nod.

```
index_stergere = 1
```



3. Cazuri particulare

Există o serie de cazuri particulare pentru listele simplu înlănțuite, utilizate des în practică:

a) Stiva

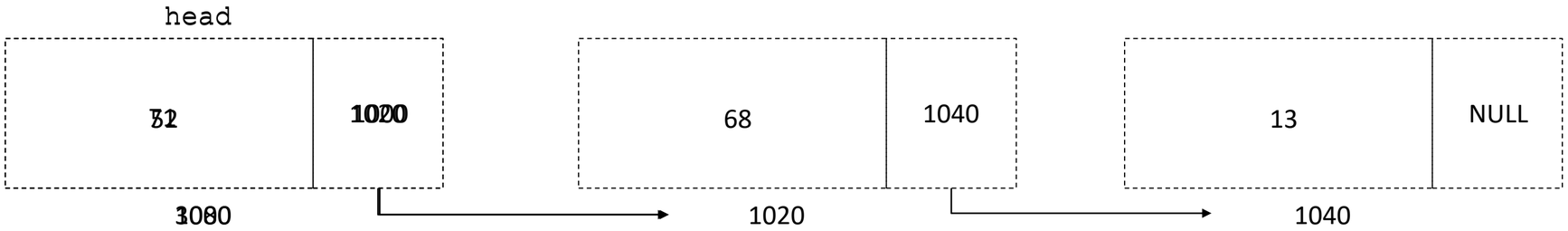
b) Coada

c) Coada circulară

a) Stiva

Listă de date simplu înlănțuită în care operațiile de inserare/extragere de elemente se fac doar pe prima poziție a listei (LIFO – Last In First Out).

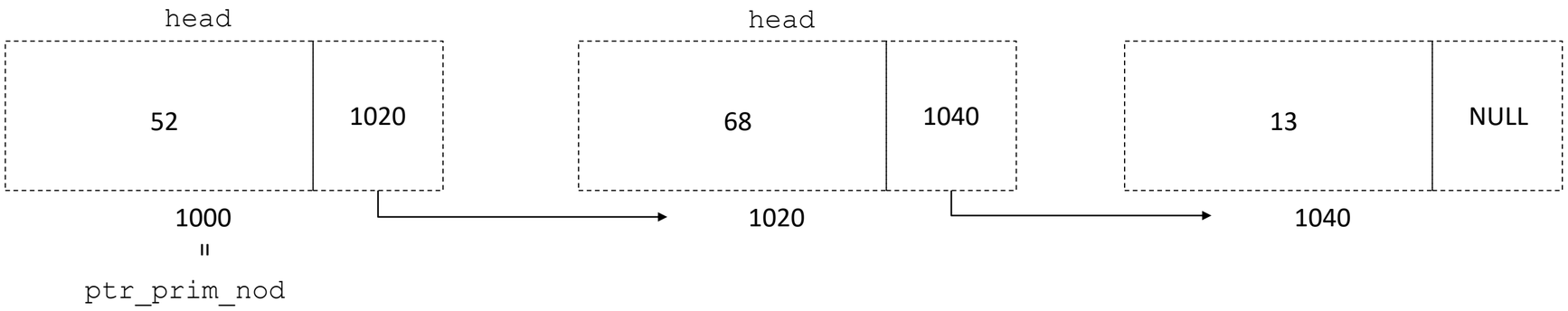
1. Adăugare element = push



a) Stiva

Listă de date simplu înlănțuită în care operațiile de inserare/extragere de elemente se fac doar pe prima poziție a listei (LIFO – Last In First Out).

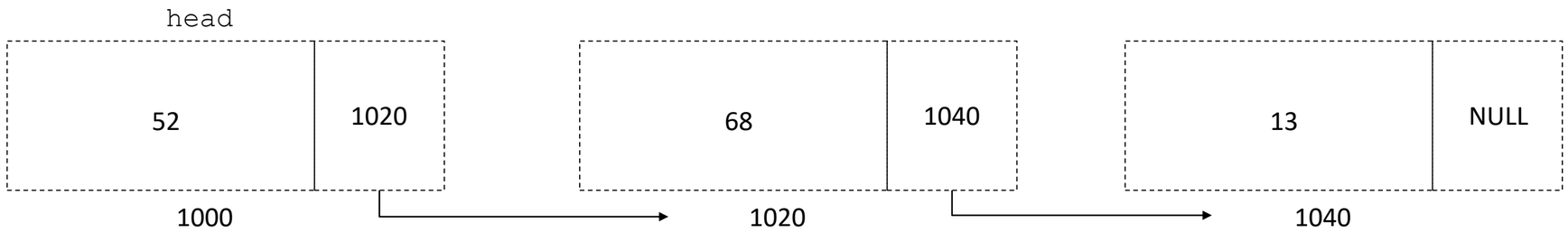
1. Adăugare element = push
2. Extragere element (și ștergere din stivă) = pop



a) Stiva

Listă de date simplu înlănțuită în care operațiile de inserare/extragere de elemente se fac doar pe prima poziție a listei (LIFO – Last In First Out).

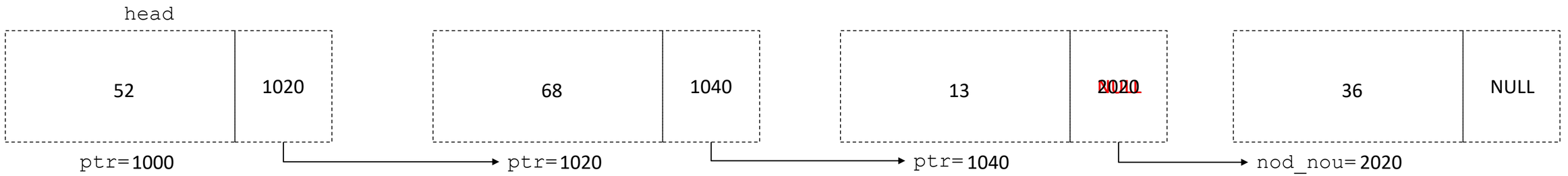
1. Adăugare element = push
2. Extragere element (și ștergere din stivă) = pop
3. Extragere element (fără ștergere din stivă) = peek



b) Coada

Listă de date simplu înlănțuită în care operația de inserare de elemente se face la sfârșitul listei, iar operația de extragere elemente se face de la începutul listei (FIFO – First In First Out).

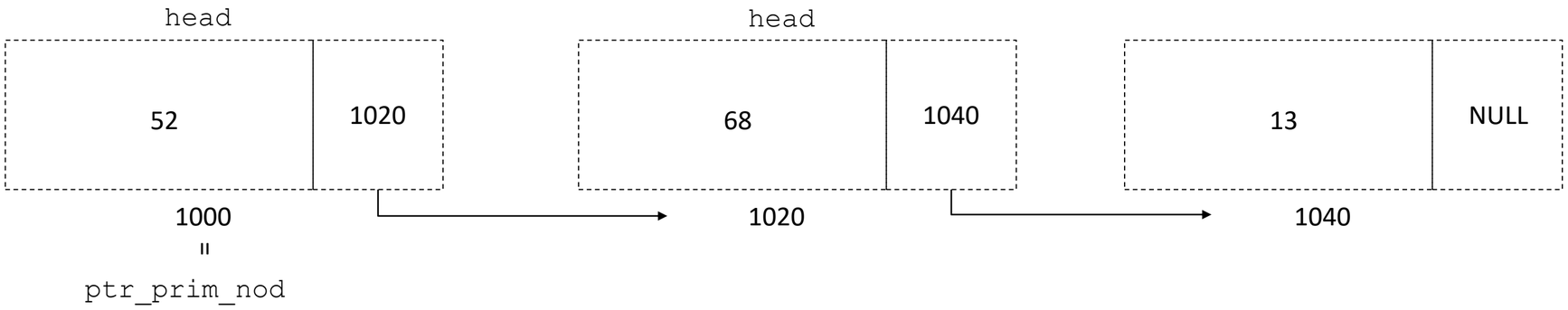
1. Adăugare element = enqueue



b) Coada

Listă de date simplu înlănțuită în care operația de inserare de elemente se face la sfârșitul listei, iar operația de extragere elemente se face de la începutul listei (FIFO – First In First Out).

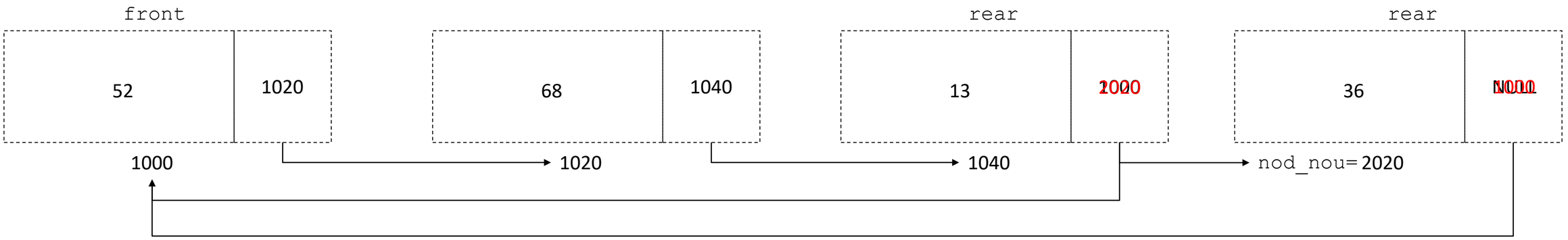
1. Adăugare element = enqueue
2. Extragere element (și ștergere din coadă) = dequeue



c) Coada circulară

Tip particular de coadă: ultimul element are legătură către primul element din coadă. Se implementează folosind un pointer către primul element (prim/front) și un pointer către ultimul element (ultim/rear).

1. Adăugare element = enqueue



c) Coada circulară

Tip particular de coadă: ultimul element are legătură către primul element din coadă. Se implementează folosind un pointer către primul element (prim/front) și un pointer către ultimul element (ultim/rear).

1. Adăugare element = enqueue
2. Extragere element = dequeue