Structuri de Date și Algoritmi
Tipuri particulare de liste

Mihai Nan

Departamentul de Calculatoare
Facultatea de Automatică și Calculatoare
Universitatea POLITEHNICA din București

Anul Universitar 2022–2023

Conținutul cursului

Listă cu santinelă

- În implementările de până acum a fost nevoie să tratăm separat cazurile limită (adăugarea la început / ștergerea de la început).
- Acest lucru se întâmpla, deoarece aceste operații modificau pointerul pe care îl foloseam pentru a accesa elementele din listă.

Listă cu santinelă

- În implementările de până acum a fost nevoie să tratăm separat cazurile limită (adăugarea la început / ștergerea de la început).
- Acest lucru se întâmpla, deoarece aceste operații modificau pointerul pe care îl foloseam pentru a accesa elementele din listă.

Pentru a scăpa de acest inconvenient, consider că lista are un nod suplimentar la început care nu conține informație utilă și pe care îl folosesc pentru a nu mai fi nevoie de modificarea lui `head`.

Listă cu santinelă

- În implementările de până acum a fost nevoie să tratăm separat cazurile limită (adăugarea la început / ștergerea de la început).
- Acest lucru se întâmpla, deoarece aceste operații modificau pointerul pe care îl foloseam pentru a accesa elementele din listă.

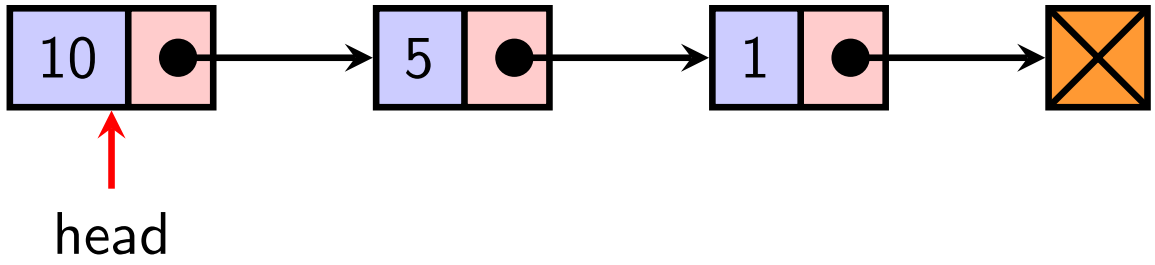
Pentru a scăpa de acest inconvenient, consider că lista are un nod suplimentar la început care nu conține informație utilă și pe care îl folosesc pentru a nu mai fi nevoie de modificarea lui `head`.

O santinelă este un nod fictiv care ne permite să simplificăm condițiile de la extreme.

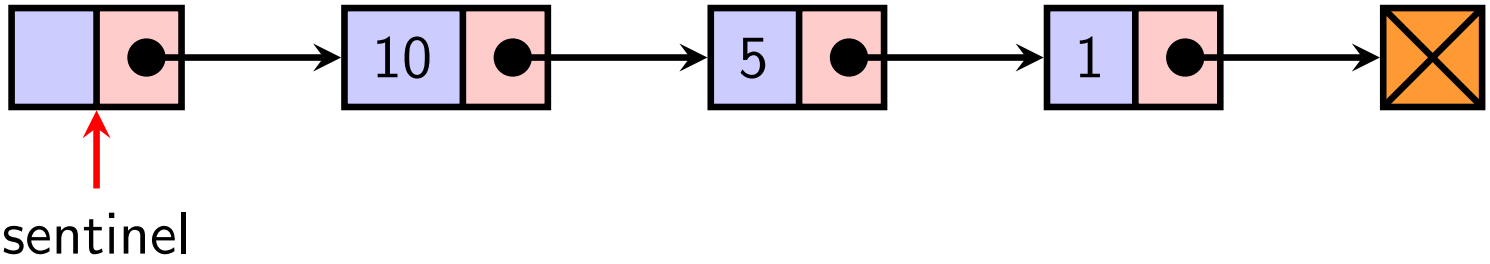
Listă cu santinelă

Varianta simplu înlănțuită

1. Exemplu de listă fără santinelă



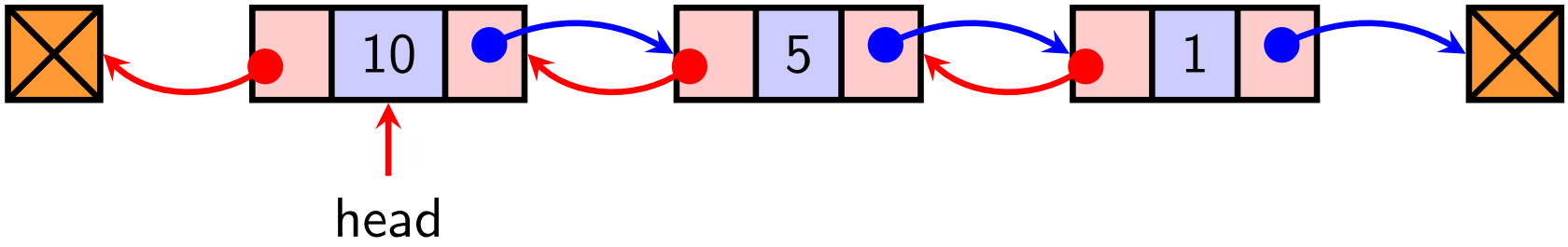
2. Exemplu de listă cu santinelă



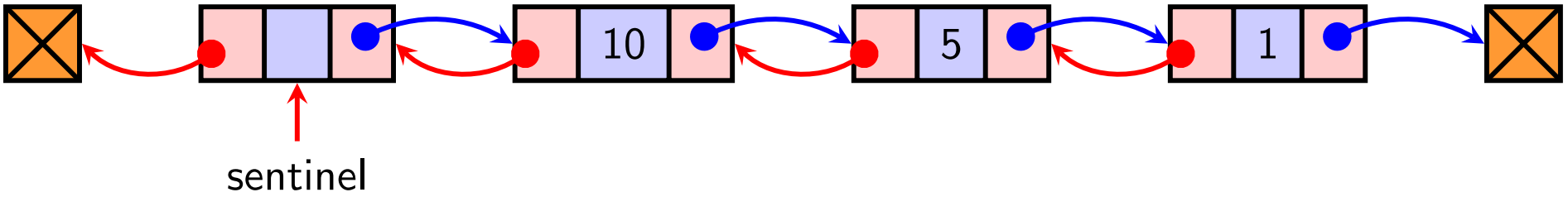
Listă cu santinelă

Varianta dublu înlănțuită

1. Exemplu de listă fără santinelă



2. Exemplu de listă cu santinelă



Listă cu santinelă – Definiția structurii

```
1  typedef int T;
2
3  typedef struct list {
4      T value;
5      struct list *next;
6  } *TList;
```

Listă cu santinelă – Definiția structurii

```
1  typedef int T;
2
3  typedef struct list {
4      T value;
5      struct list *next;
6  } *TList;
```

Cum putem inițializa o listă simplu înlănțuită cu santinelă?

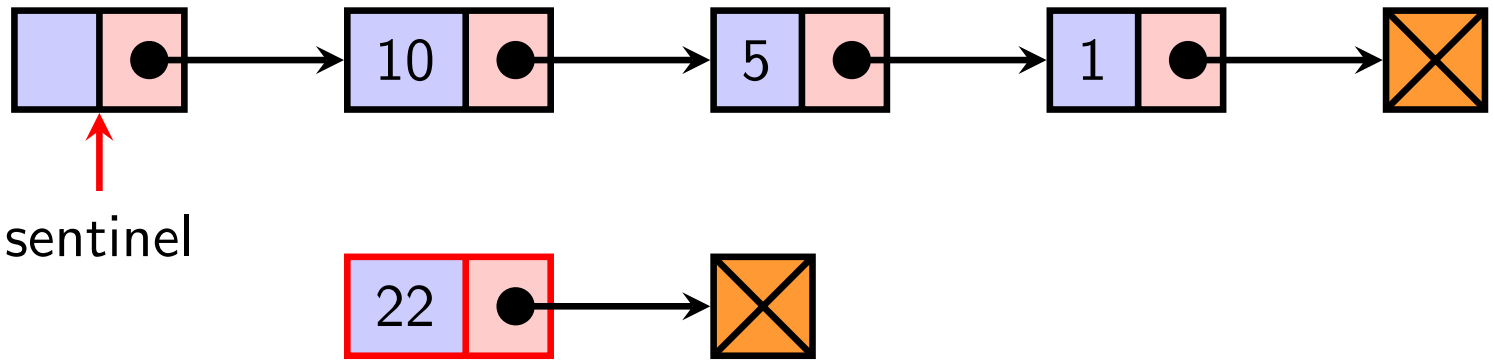
Listă cu santinelă – Definiția structurii

```
1  typedef int T;
2
3  typedef struct list {
4      T value;
5      struct list *next;
6  } *TList;
```

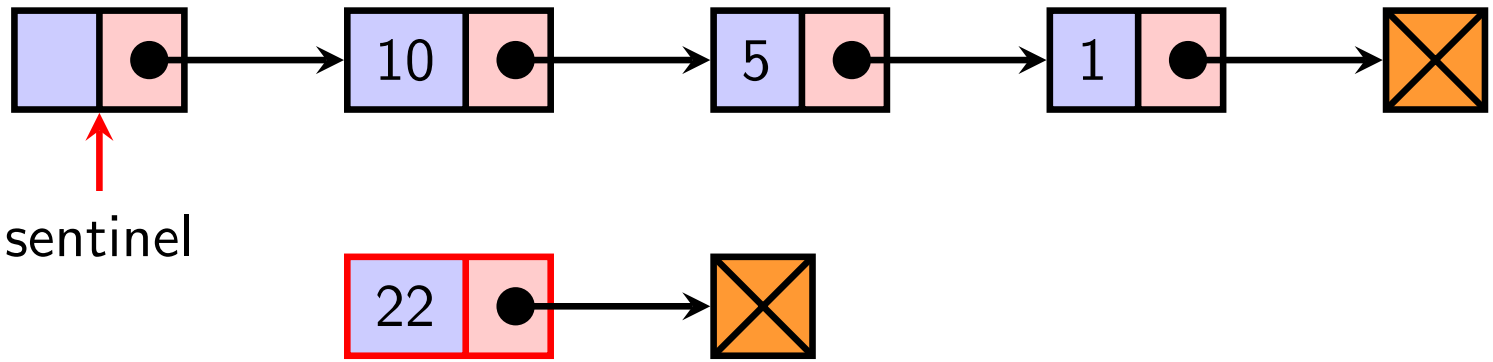
Cum putem inițializa o listă simplu înlănțuită cu santinelă?

```
7  TList initList() {
8      TList sentinel = calloc(1, sizeof(struct list));
9      return sentinel;
10 }
```

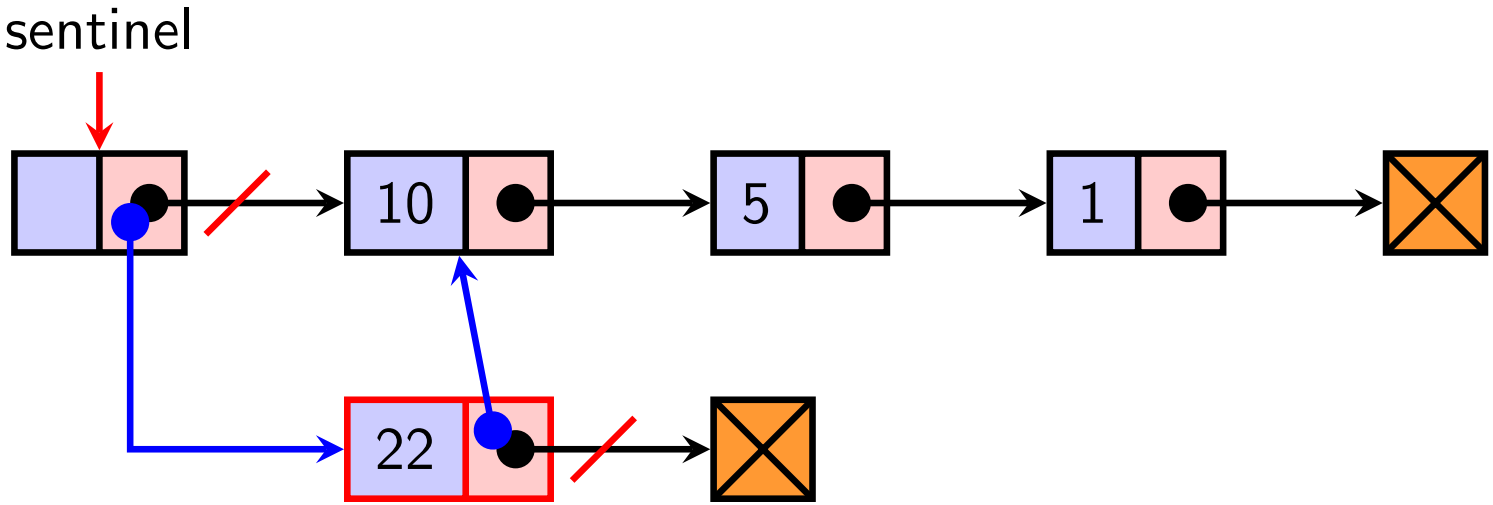
Listă cu santinelă – Adăugarea la început



Listă cu santinelă – Adăugarea la început



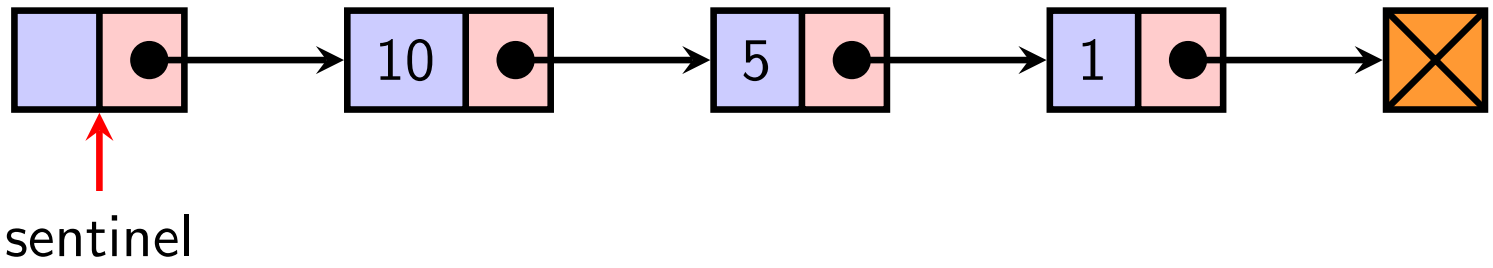
Refacerea legăturilor



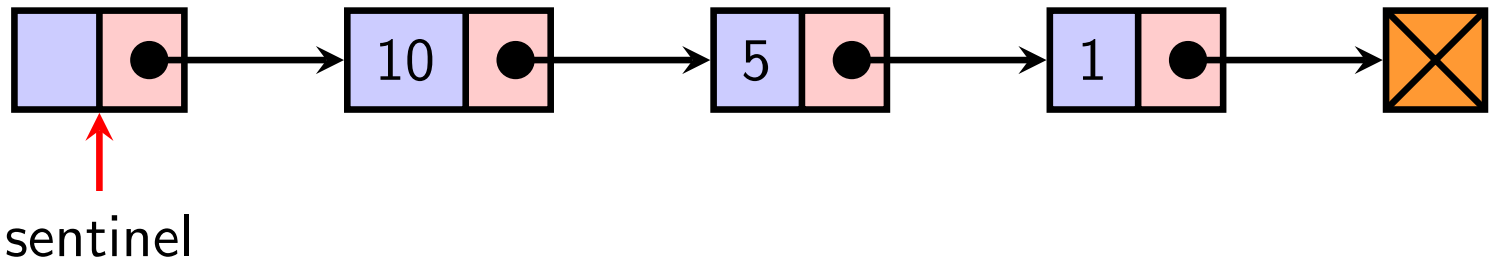
Listă cu santinelă – Adăugarea la început

```
11  TList createNode(T value) {
12      TList node = malloc(sizeof(struct list));
13      node->value = value;
14      node->next = NULL;
15      return node;
16  }
17
18  TList insertFront(TList sentinel, T value) {
19      TList node = createNode(value);
20      node->next = sentinel->next;
21      sentinel->next = node;
22      return sentinel;
23  }
```

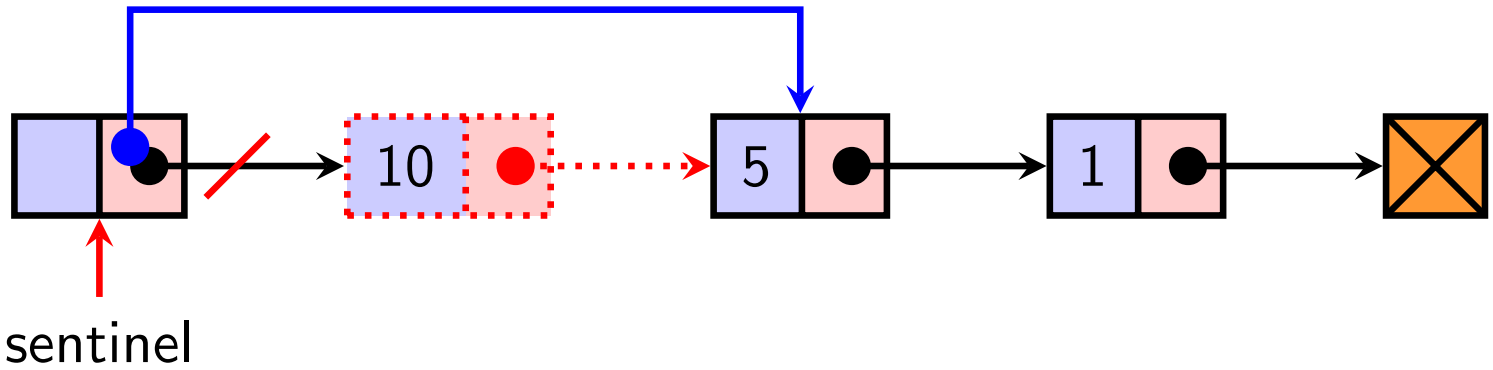
Listă cu santinelă – Ștergerea de la început



Listă cu santinelă – Ștergerea de la început



Ștergerea nodului și refacerea legăturilor



Listă cu santinelă – Ștergerea de la început

```
24  TList removeFront(TList sentinel) {
25      TList temp = sentinel->next;
26      if (temp != NULL)
27          sentinel->next = temp->next;
28      free(temp);
29      return sentinel;
30  }
```

Listă cu santinelă – Ștergerea de la început

```
24  TList removeFront(TList sentinel) {
25      TList temp = sentinel->next;
26      if (temp != NULL)
27          sentinel->next = temp->next;
28      free(temp);
29      return sentinel;
30  }
```

Mai este nevoie ca aceste două funcții să întoarcă un rezultat?

Listă cu santinelă – Ștergerea de la început

```
24  TList removeFront(TList sentinel) {
25      TList temp = sentinel->next;
26      if (temp != NULL)
27          sentinel->next = temp->next;
28      free(temp);
29      return sentinel;
30  }
```

Mai este nevoie ca aceste două funcții să întoarcă un rezultat?

Nu mai este nevoie pentru că ele nu vor ajunge să modifice `sentinel`.

Listă cu santinelă – Adăugare & ștergere

Putem să le rescriem astfel încât amândouă să fie de tip `void`.

```
18  void insertFront(TList sentinel, T value) {
19      TList node = createNode(value);
20      node->next = sentinel->next;
21      sentinel->next = node;
22  }
23  void removeFront(TList sentinel) {
24      TList temp = sentinel->next;
25      if (temp != NULL)
26          sentinel->next = temp->next;
27      free(temp);
28  }
```

Listă cu santinelă – Adăugare & ștergere

Putem să le rescriem astfel încât amândouă să fie de tip `void`.

```
18  void insertFront(TList sentinel, T value) {
19      TList node = createNode(value);
20      node->next = sentinel->next;
21      sentinel->next = node;
22  }
23  void removeFront(TList sentinel) {
24      TList temp = sentinel->next;
25      if (temp != NULL)
26          sentinel->next = temp->next;
27      free(temp);
28  }
```

Am scăpat de dublu pointer la structură!

Listă cu santinelă – Afișarea

```
31  void print(TList sentinel) {
32      TList iter = sentinel->next;
33      while (iter != NULL) {
34          printf("%d ", iter->value);
35          iter = iter->next;
36      }
37      printf("\n");
38  }
```

Listă cu santinelă – Dealocarea memoriei

```
39  TList freeList(TList sentinel) {
40      TList temp, iter = sentinel->next;
41      while (iter != NULL) {
42          temp = iter;
43          iter = iter->next;
44          free(temp);
45      }
46      free(sentinel);
47      return NULL;
48  }
```

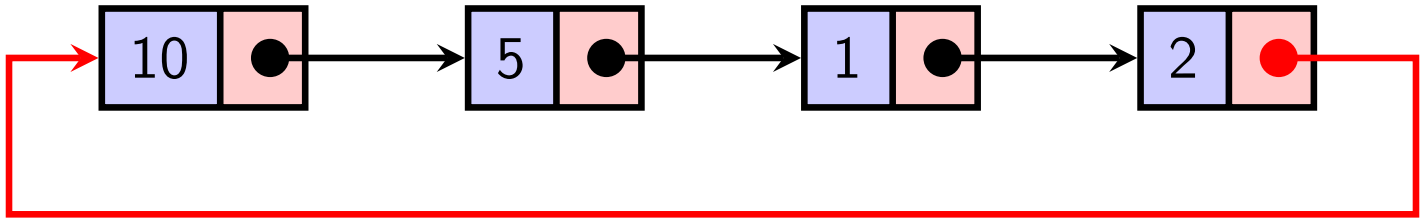
Listă cu santinelă – Exemplu

```
41  int main() {
42      TList sentinel = initList();
43      sentinel = insertFront(sentinel, 1);
44      sentinel = insertFront(sentinel, 5);
45      sentinel = insertFront(sentinel, 10);
46      print(sentinel);
47      sentinel = removeFront(sentinel);
48      print(sentinel);
49      sentinel = freeList(sentinel);
50      return 0;
51  }
```

Listă circulară

- Câmpul `next` al ultimului nod din listă o să pointeze către primul nod.
- Trebuie să avem grijă când parcurgem lista pentru a nu traversa lista la infinit.

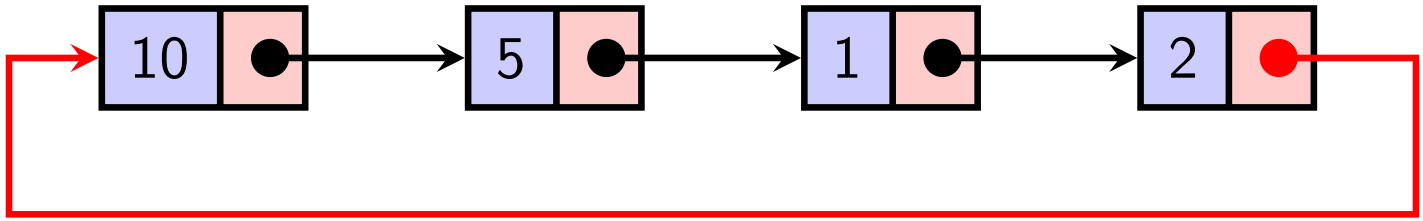
Reprezentare grafică



Listă circulară

- Câmpul `next` al ultimului nod din listă o să pointeze către primul nod.
- Trebuie să avem grijă când parcurgem lista pentru a nu traversa lista la infinit.

Reprezentare grafică



Definiția structurii

```
1  typedef int T;
2
3  typedef struct node {
4      T value;
5      struct node* next;
6  } *List;
```

Listă circulară – Inițializarea listei

Cum putem inițializa lista vidă?

Listă circulară – Inițializarea listei

Cum putem inițializa lista vidă?

```
7  List init() {
8      return NULL;
9  }
```

Cum putem crea lista care conține un singur element?

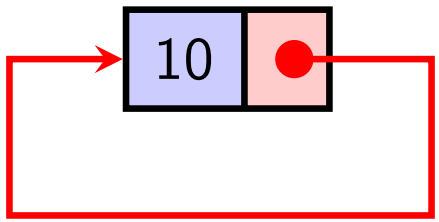
```
10 List createList(T data) {
11     List node = malloc(sizeof(struct node));
12     node->value = data;
13     node->next = node;
14     return node;
15 }
```

Listă circulară – Inițializarea listei

Cum putem crea lista care conține un singur element?

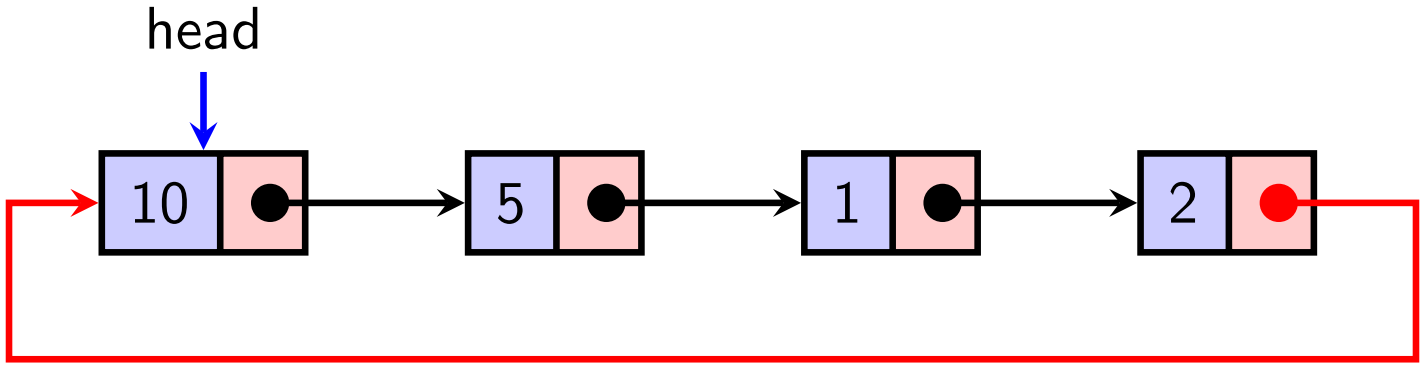
```
10  List createList(T data) {
11      List node = malloc(sizeof(struct node));
12      node->value = data;
13      node->next = node;
14      return node;
15  }
```

Reprezentare grafică



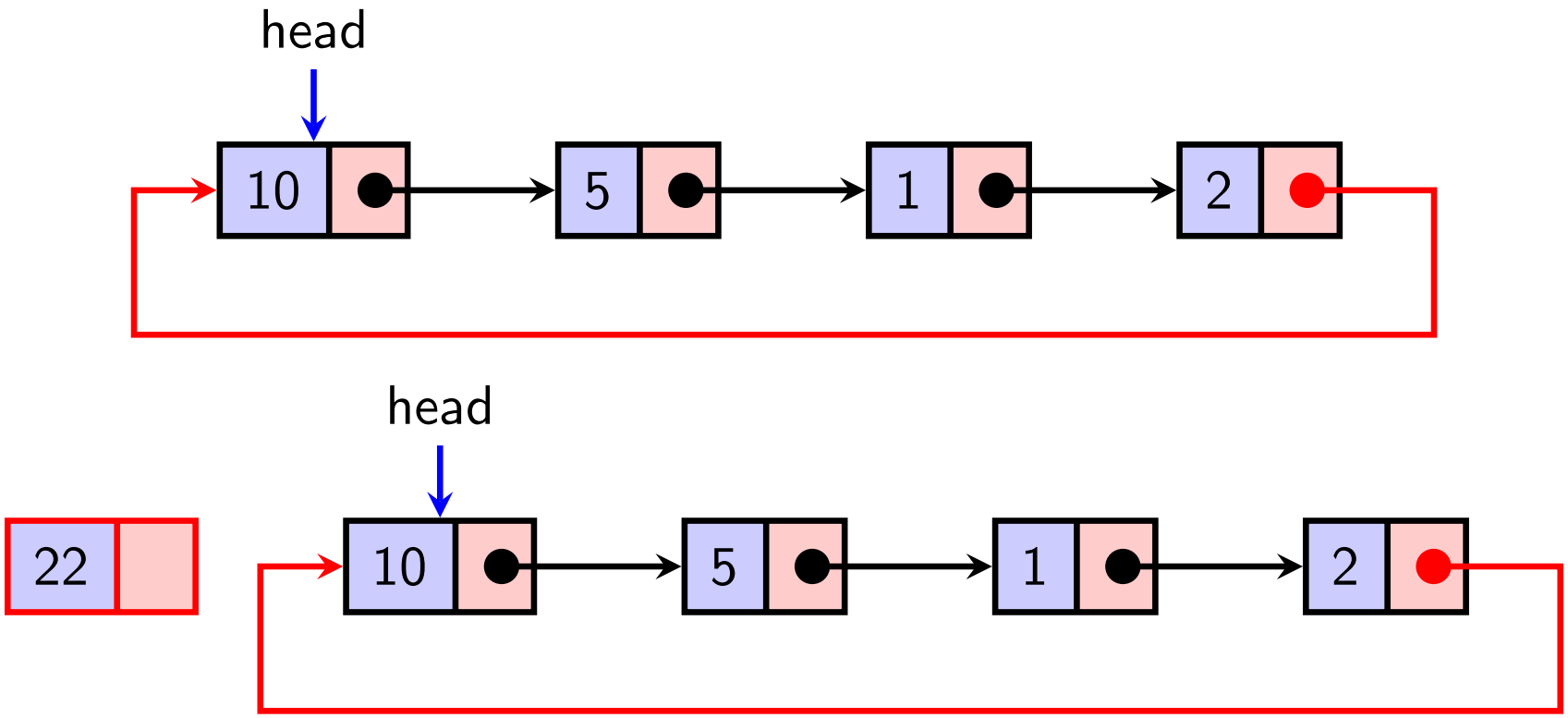
Listă circulară – Inserarea la începutul listei

Pornesc de la următoarea listă și vreau să inserez valoarea 22 la început.



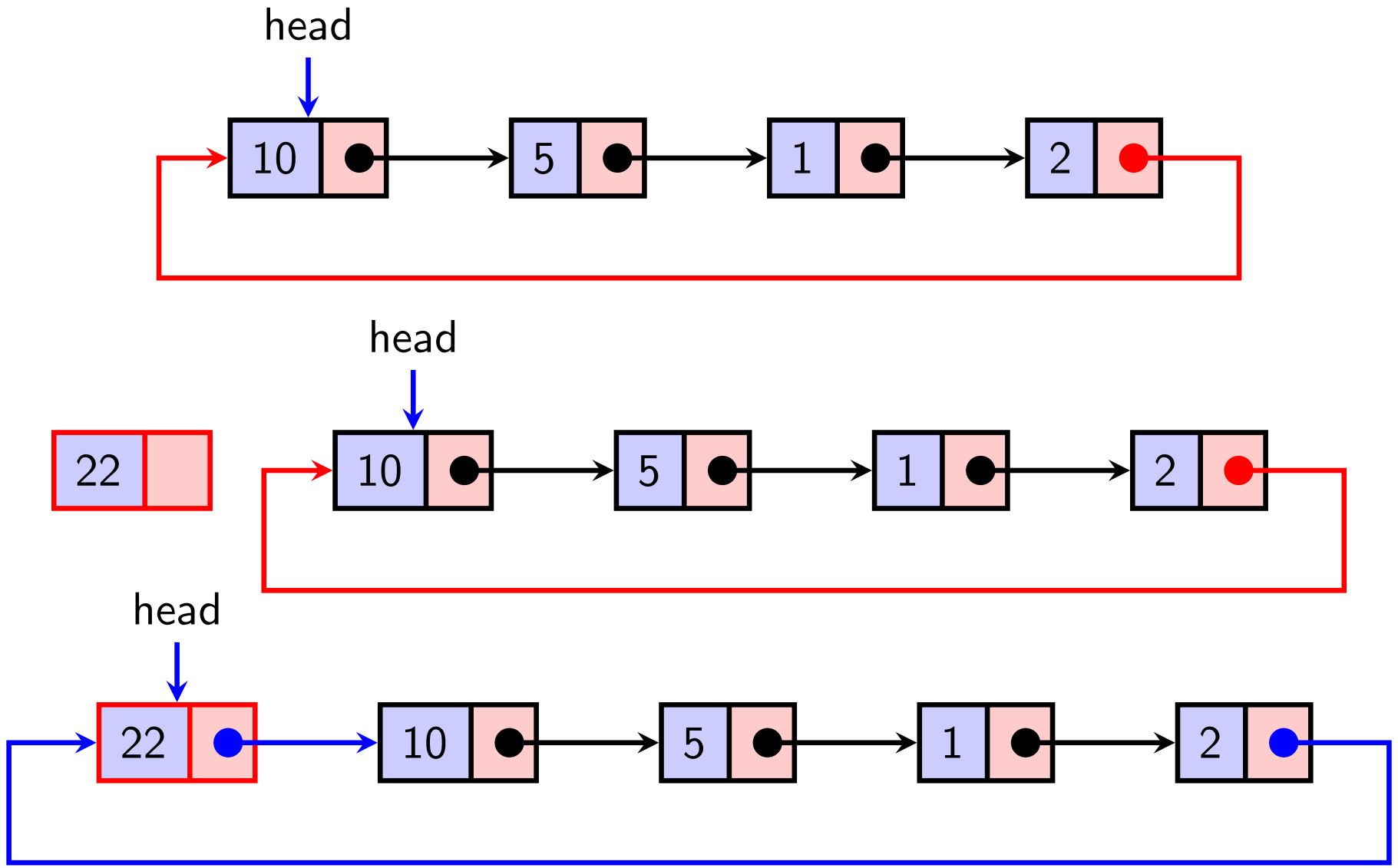
Listă circulară – Inserarea la începutul listei

Pornesc de la următoarea listă și vreau să inserez valoarea 22 la început.



Listă circulară – Inserarea la începutul listei

Pornesc de la următoarea listă și vreau să inserez valoarea 22 la început.



Listă circulară – Inserarea la începutul listei

Am văzut anterior că avem de modificat două legături? Cum le putem modifica?

Listă circulară – Inserarea la începutul listei

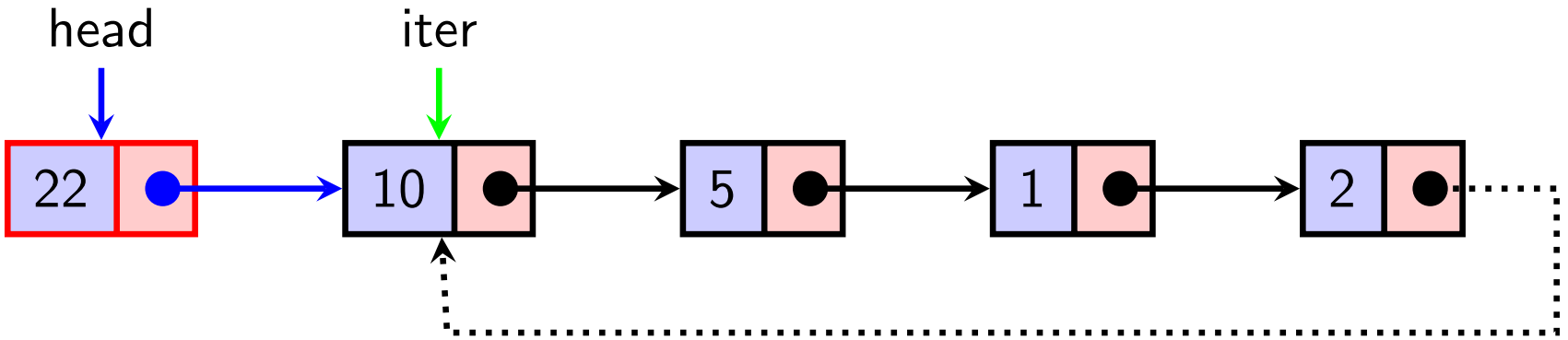
Am văzut anterior că avem de modificat două legături? Cum le putem modifica?

O să fie nevoie să parcurgem lista pentru a identifica *ultimul* nod!

Listă circulară – Inserarea la începutul listei

Am văzut anterior că avem de modificat două legături? Cum le putem modifica?

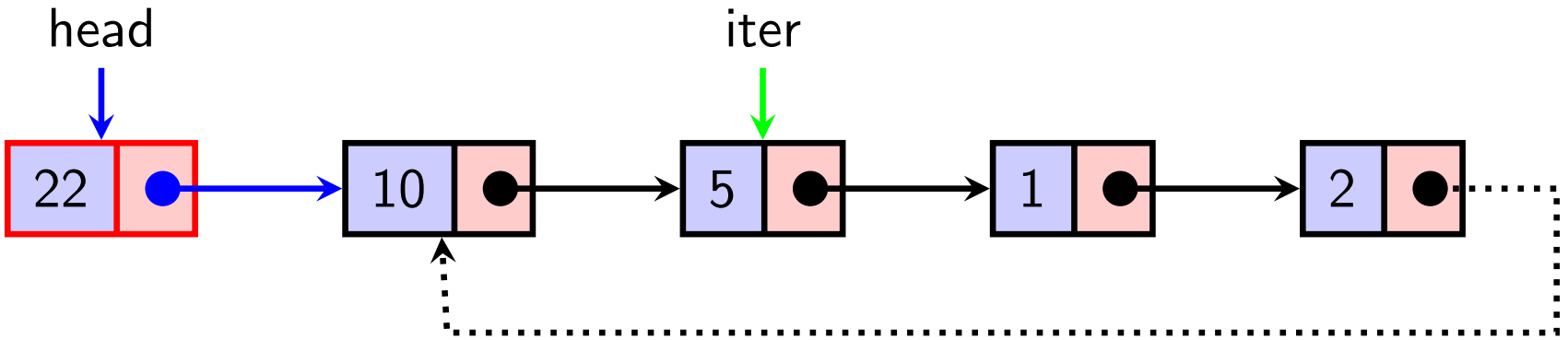
O să fie nevoie să parcurgem lista pentru a identifica *ultimul* nod!



Listă circulară – Inserarea la începutul listei

Am văzut anterior că avem de modificat două legături? Cum le putem modifica?

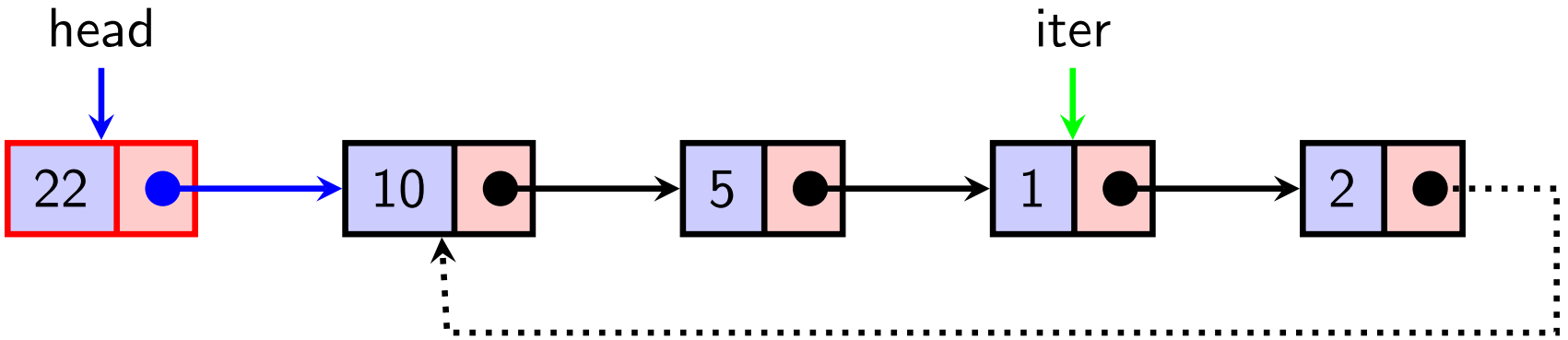
O să fie nevoie să parcurgem lista pentru a identifica *ultimul* nod!



Listă circulară – Inserarea la începutul listei

Am văzut anterior că avem de modificat două legături? Cum le putem modifica?

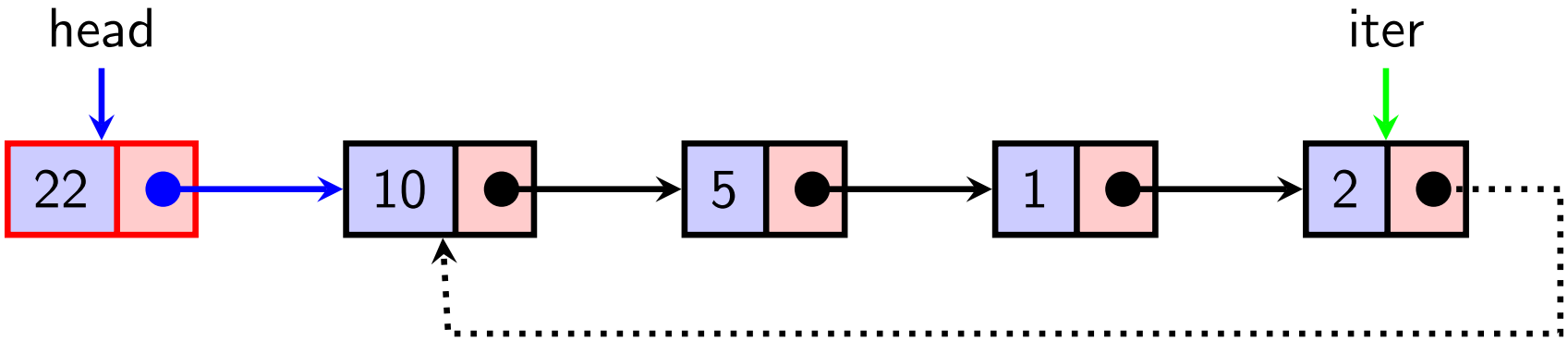
O să fie nevoie să parcurgem lista pentru a identifica *ultimul* nod!



Listă circulară – Inserarea la începutul listei

Am văzut anterior că avem de modificat două legături? Cum le putem modifica?

O să fie nevoie să parcurgem lista pentru a identifica *ultimul* nod!



Listă circulară – Inserarea la începutul listei

Când trebuie să mă opresc?

Listă circulară – Inserarea la începutul listei

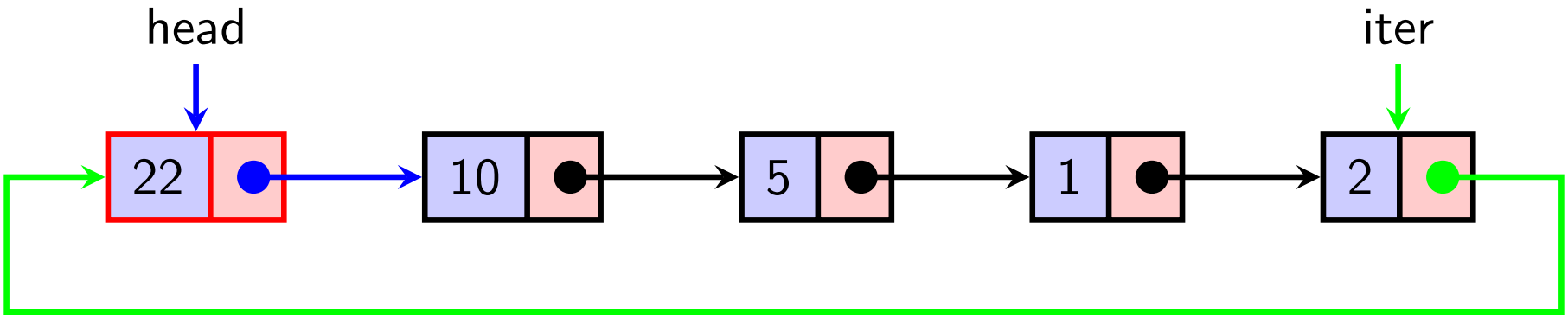
Când trebuie să mă opresc?

Atunci când `iter->next` o să fie egal cu adresa de început a listei în care vreau să inserez!

Listă circulară – Inserarea la începutul listei

Când trebuie să mă opresc?

Atunci când `iter->next` o să fie egal cu adresa de început a listei în care vreau să inserez!

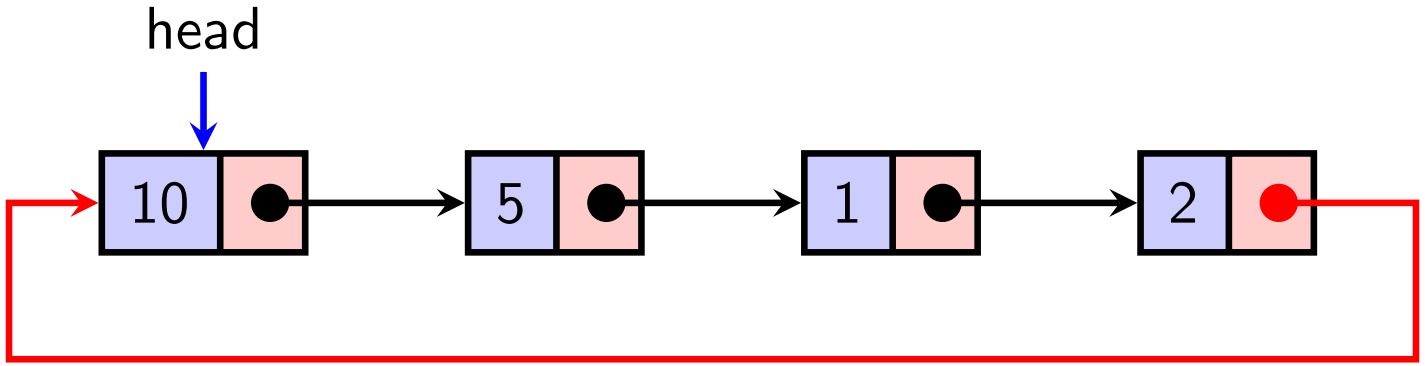


Listă circulară – Inserarea la începutul listei

```
16  List insertFront(List head, T data) {
17      List node = createList(data);
18      if (head == NULL)
19          return node;
20      node->next = head;
21      List iter = head;
22      while (iter->next != head)
23          iter = iter->next;
24      iter->next = node;
25      return node;
26  }
```

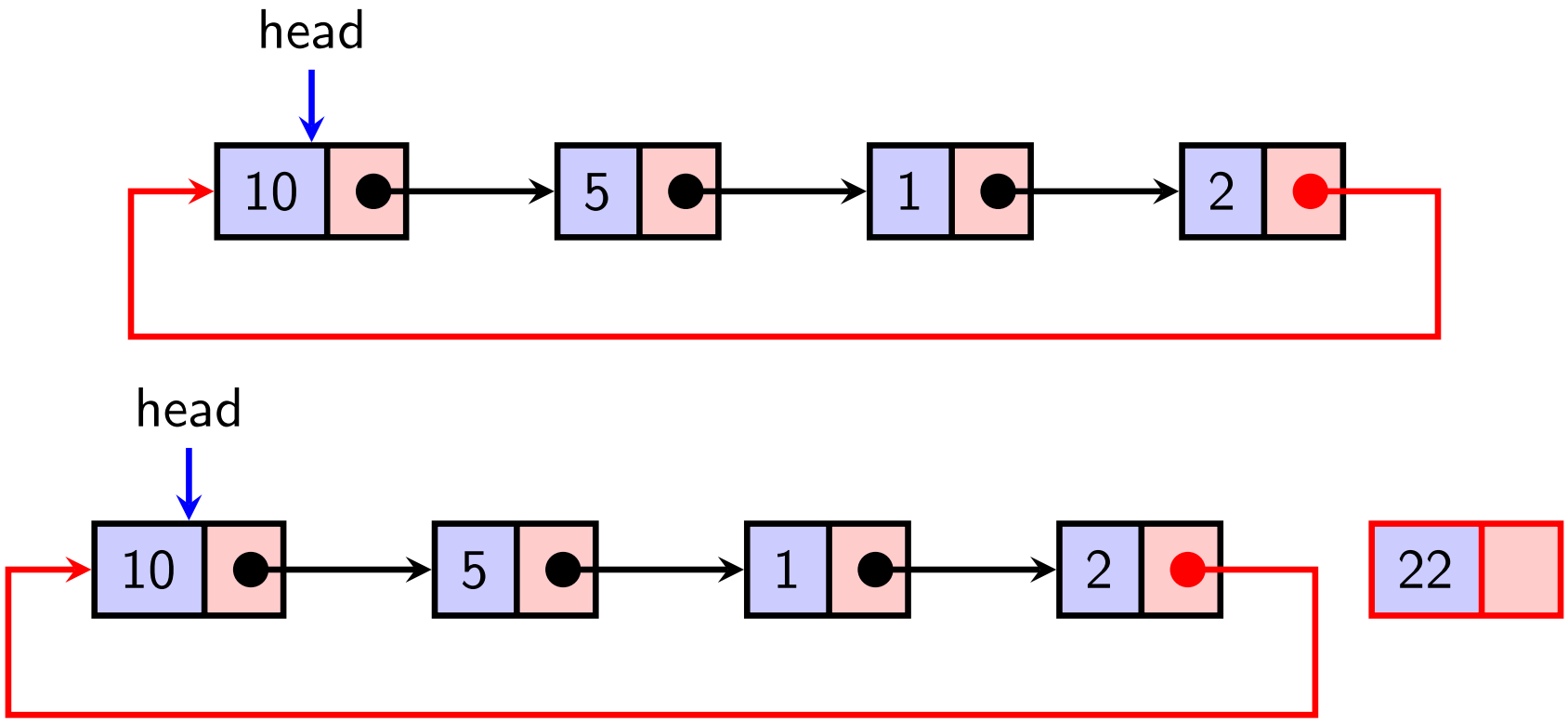
Listă circulară – Inserarea la finalul listei

Pornesc de la următoarea listă și vreau să inserez valoarea 22 la final.



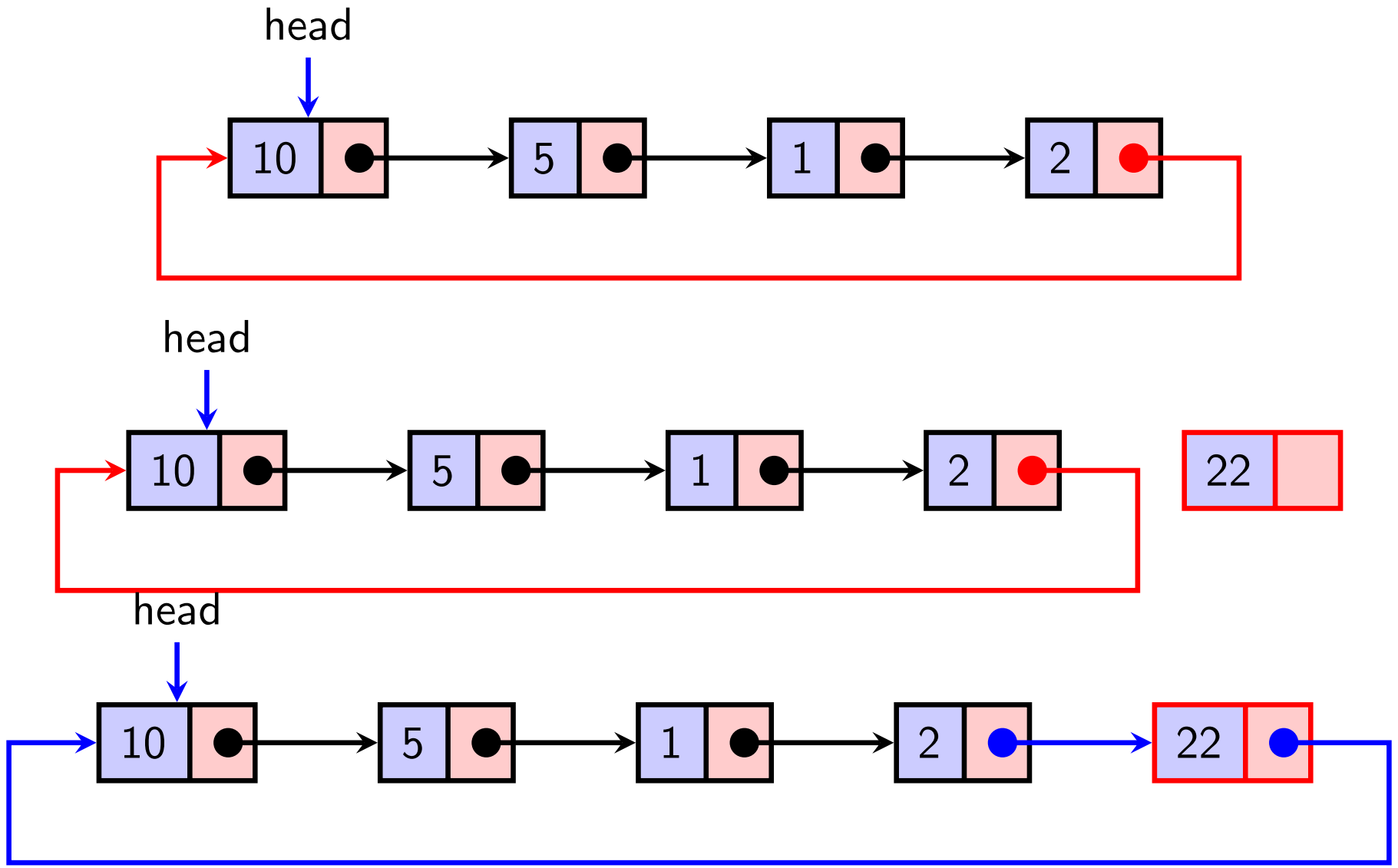
Listă circulară – Inserarea la finalul listei

Pornesc de la următoarea listă și vreau să inserez valoarea 22 la final.



Listă circulară – Inserarea la finalul listei

Pornesc de la următoarea listă și vreau să inserez valoarea 22 la final.



Listă circulară – Inserarea la finalul listei

```
27  List insertRear(List head, T data) {
28      List node = createList(data);
29      if (head == NULL)
30          return node;
31      node->next = head;
32      List iter = head;
33      while (iter->next != head)
34          iter = iter->next;
35      iter->next = node;
36      return head;
37  }
```

Listă circulară – Inserare la început / final

Care sunt diferențele între cele două tipuri de inserare?

```
16  List insertFront(List head, T data){
17      List node = createList(data);
18      if (head == NULL)
19          return node;
20      node->next = head;
21      List iter = head;
22      while (iter->next != head)
23          iter = iter->next;
24      iter->next = node;
25      return node;
26  }
```

```
27  List insertRear(List head, T data){
28      List node = createList(data);
29      if (head == NULL)
30          return node;
31      node->next = head;
32      List iter = head;
33      while (iter->next != head)
34          iter = iter->next;
35      iter->next = node;
36      return head;
37  }
```

Listă circulară – Inserare la început / final

Care sunt diferențele între cele două tipuri de inserare?

```
16  List insertFront(List head, T data){
17      List node = createList(data);
18      if (head == NULL)
19          return node;
20      node->next = head;
21      List iter = head;
22      while (iter->next != head)
23          iter = iter->next;
24      iter->next = node;
25      return node;
26  }
```

```
27  List insertRear(List head, T data){
28      List node = createList(data);
29      if (head == NULL)
30          return node;
31      node->next = head;
32      List iter = head;
33      while (iter->next != head)
34          iter = iter->next;
35      iter->next = node;
36      return head;
37  }
```

Diferă doar nodul pe care îl returnăm!

Listă circulară – Afișarea elementelor

```
38  void printList(List head) {
39      if (head == NULL) {
40          printf("NULL\n");
41          return;
42      }
43      List iter = head;
44      do {
45          printf("%d ", iter->value);
46          iter = iter->next;
47      } while (iter != head);
48      printf("\n");
49  }
```

Listă circulară – Dealocarea memoriei

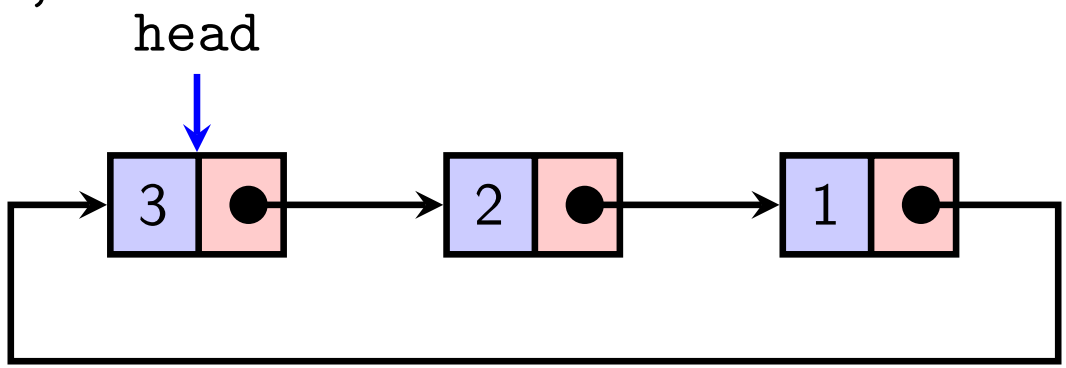
```
50  List freeList(List head) {
51      if (head == NULL)
52          return head;
53      List temp, iter = head;
54      do {
55          temp = iter;
56          iter = iter->next;
57          free(temp);
58      } while (iter != head);
59      return NULL;
60  }
```

Listă circulară – Exemplu

```
61  int main() {
62      List head = init();
63      head = insertFront(head, 1);
64      head = insertFront(head, 2);
65      head = insertFront(head, 3);
66      printList(head);
67      printList(head->next);
68      printList(head->next->next);
69      head = freeList(head);
70      return 0;
71  }
```

Listă circulară – Exemplu

```
61  int main() {
62      List head = init();
63      head = insertFront(head, 1);
64      head = insertFront(head, 2);
65      head = insertFront(head, 3);
66      printList(head);
67      printList(head->next);
68      printList(head->next->next);
69      head = freeList(head);
70      return 0;
71  }
```



Listă circulară – Recapitulare

- Operația de adăugare la începutul listei are complexitatea $O(N)$
- Operația de adăugare la finalul listei are complexitatea $O(N)$

Listă circulară – Recapitulare

- Operația de adăugare la începutul listei are complexitatea $O(N)$
- Operația de adăugare la finalul listei are complexitatea $O(N)$

Putem optimiza cumva aceste două operații?

Listă circulară – Recapitulare

- Operația de adăugare la începutul listei are complexitatea $O(N)$
- Operația de adăugare la finalul listei are complexitatea $O(N)$

Putem optimiza cumva aceste două operații?

Problema apare, deoarece trebuie să determinăm *ultimul* nod din listă.

Listă circulară – Recapitulare

- Operația de adăugare la începutul listei are complexitatea $O(N)$
- Operația de adăugare la finalul listei are complexitatea $O(N)$

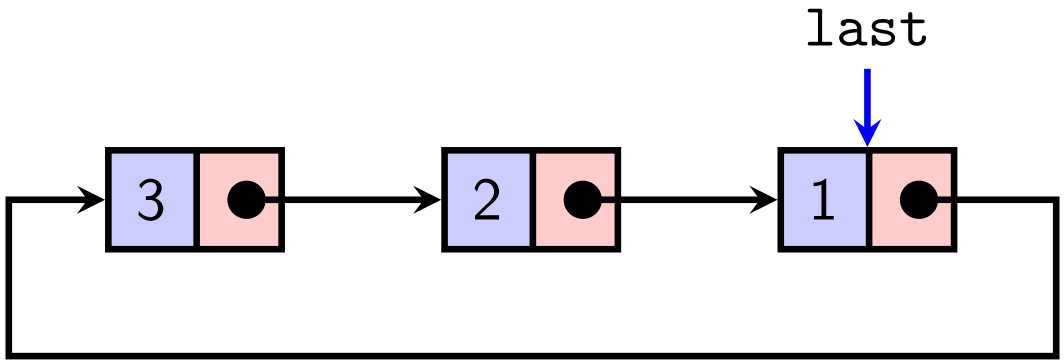
Putem optimiza cumva aceste două operații?

Problema apare, deoarece trebuie să determinăm *ultimul* nod din listă.

În loc să reținem primul nod din listă, îl reținem pe ultimul!

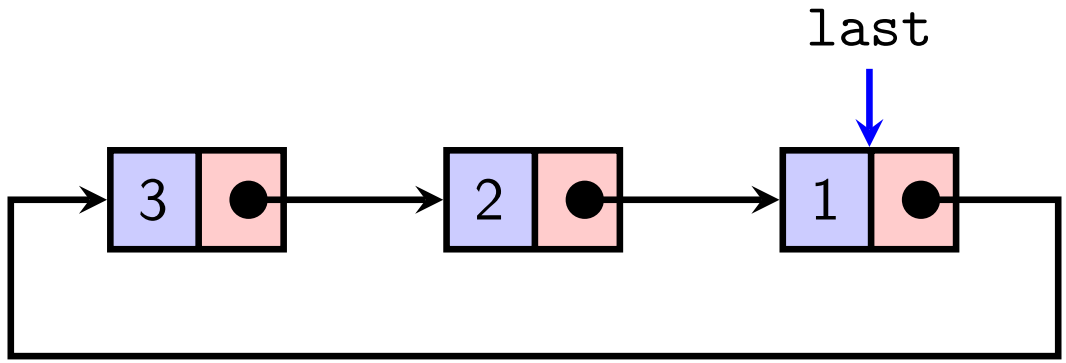
Listă circulară – Inserarea la începutul listei

Pornim de la următoarea listă și inserăm la început.

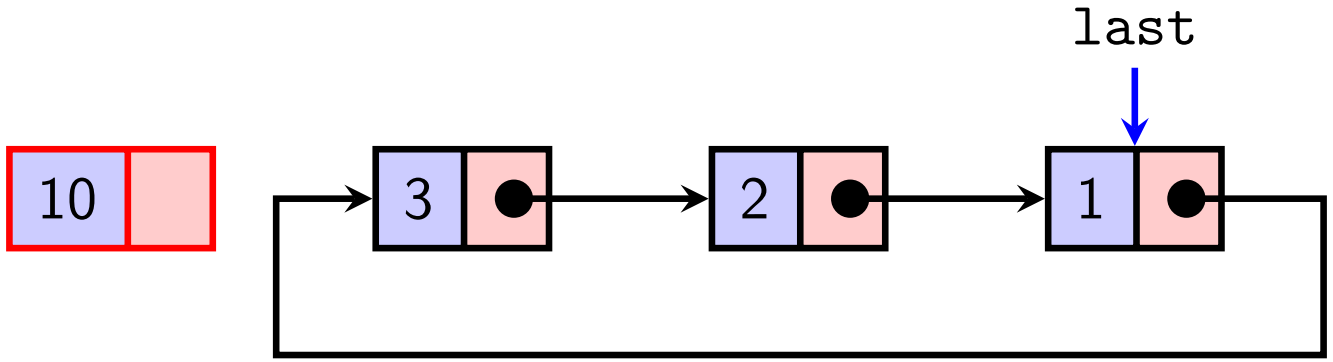


Listă circulară – Inserarea la începutul listei

Pornim de la următoarea listă și inserăm la început.

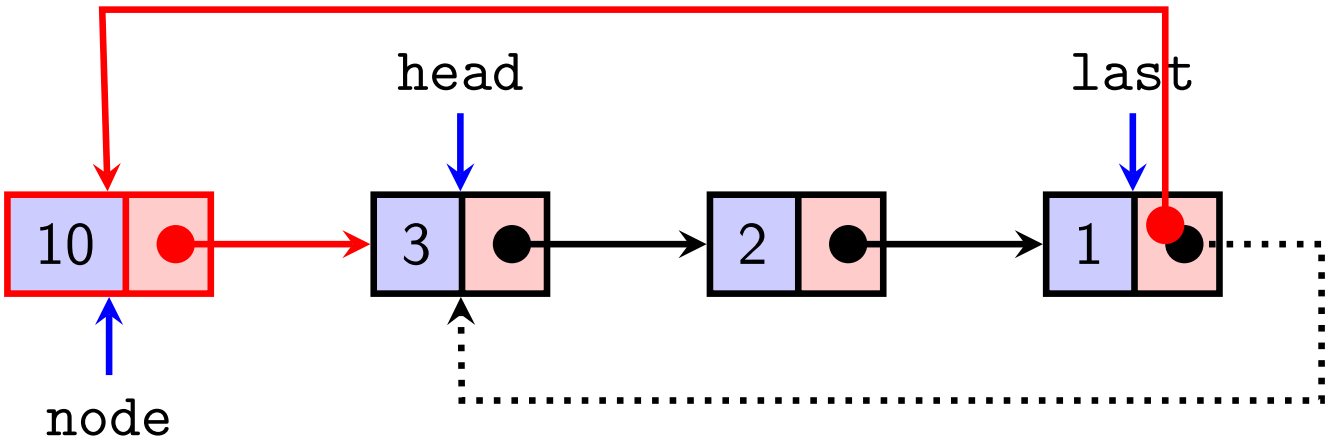


Construim un nod nou



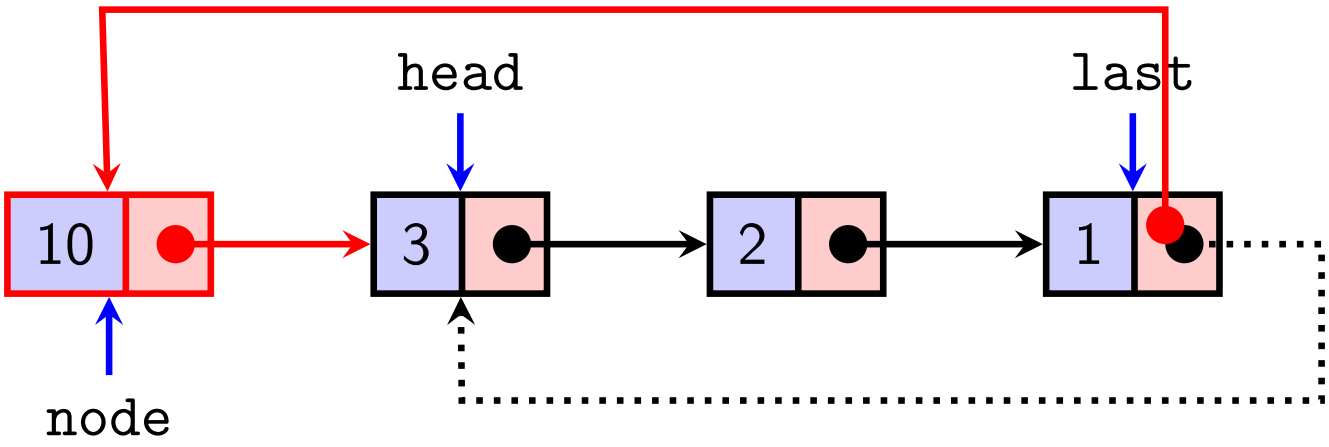
Listă circulară – Inserarea la începutul listei

Refacem legăturile



Listă circulară – Inserarea la începutul listei

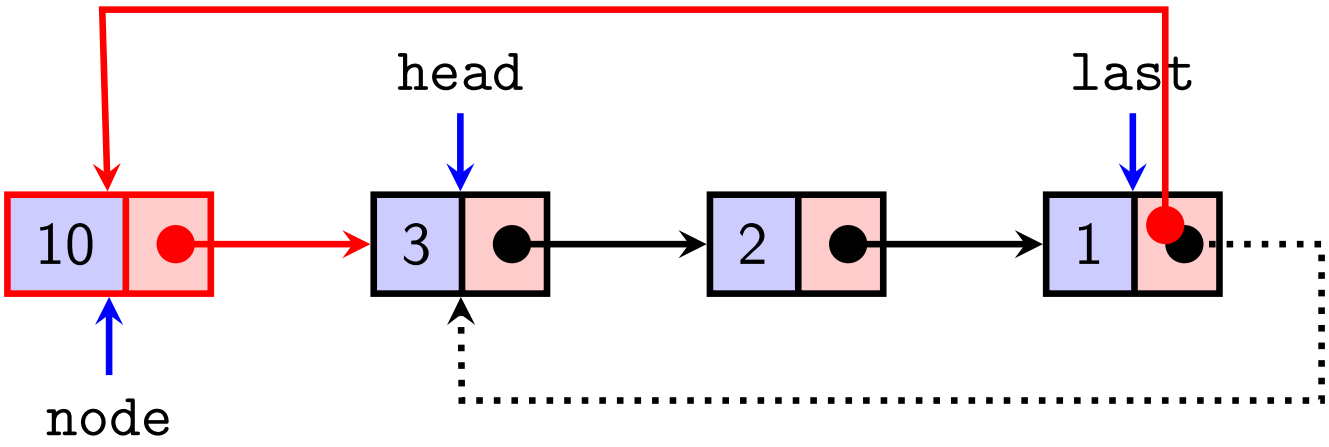
Refacem legăturile



Ce legături trebuie să refacem?

Listă circulară – Inserarea la începutul listei

Refacem legăturile



Ce legături trebuie să refacem?

```
node = createList(data);
head = last->next;
node->next = head;
last->next = node;
```

Listă circulară – Inserarea la începutul listei

```
1  List insertFront(List last, T data) {
2      List node = createList(data);
3      if (last == NULL)
4          return node;
5      List head = last->next;
6      node->next = head;
7      last->next = node;
8      return last;
9  }
```

Listă circulară – Inserarea la începutul listei

```
1  List insertFront(List last, T data) {
2      List node = createList(data);
3      if (last == NULL)
4          return node;
5      List head = last->next;
6      node->next = head;
7      last->next = node;
8      return last;
9  }
```

Ce complexitate avem acum?

Listă circulară – Inserarea la începutul listei

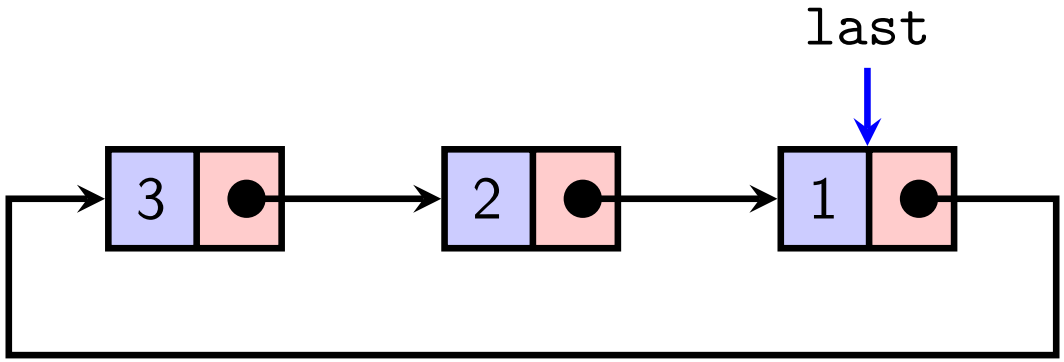
```
1  List insertFront(List last, T data) {
2      List node = createList(data);
3      if (last == NULL)
4          return node;
5      List head = last->next;
6      node->next = head;
7      last->next = node;
8      return last;
9  }
```

Ce complexitate avem acum?

Complexitatea nu mai depinde de numărul de elemente: $O(1)$

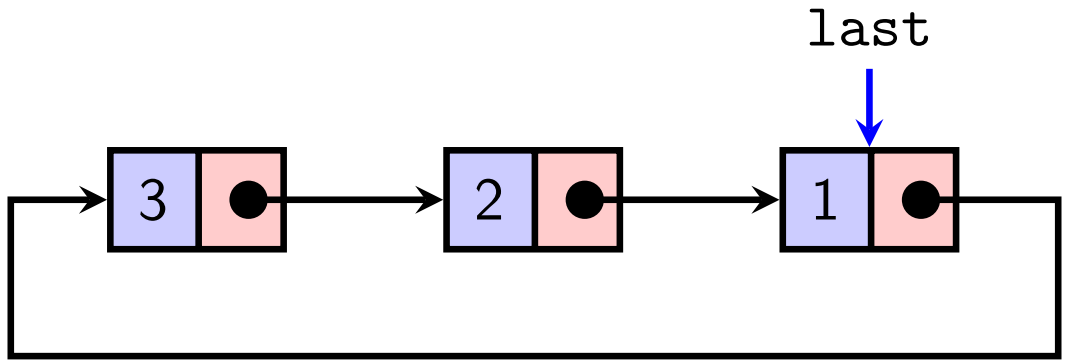
Listă circulară – Inserarea la finalul listei

Pornim de la următoarea listă și inserăm la început.

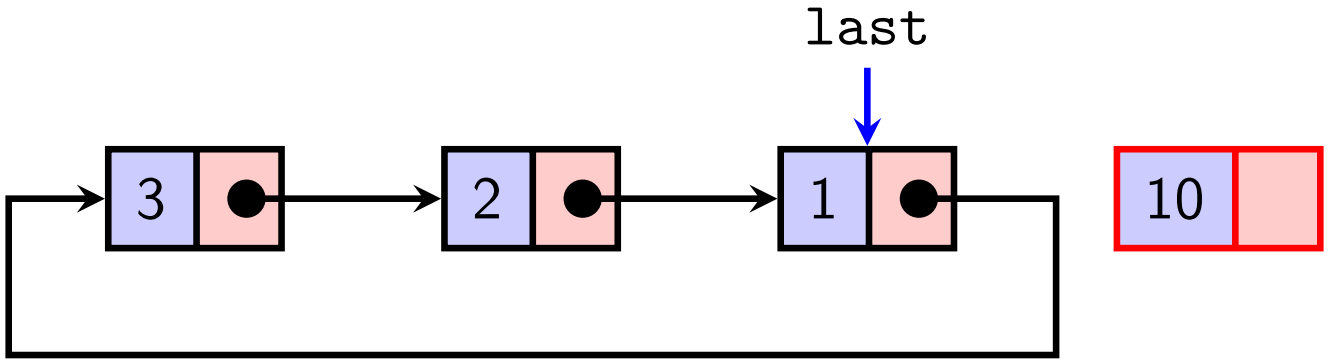


Listă circulară – Inserarea la finalul listei

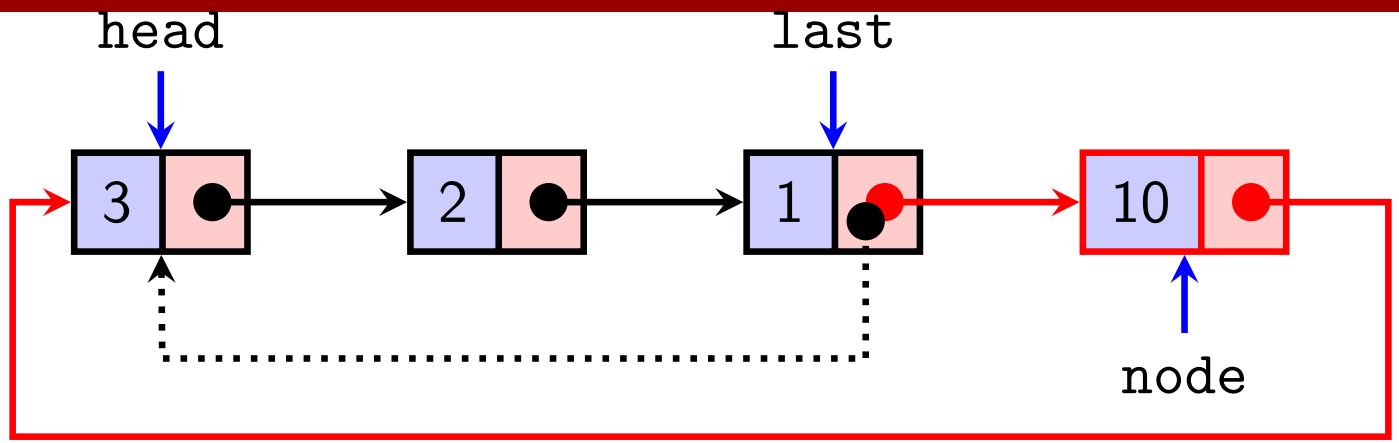
Pornim de la următoarea listă și inserăm la început.



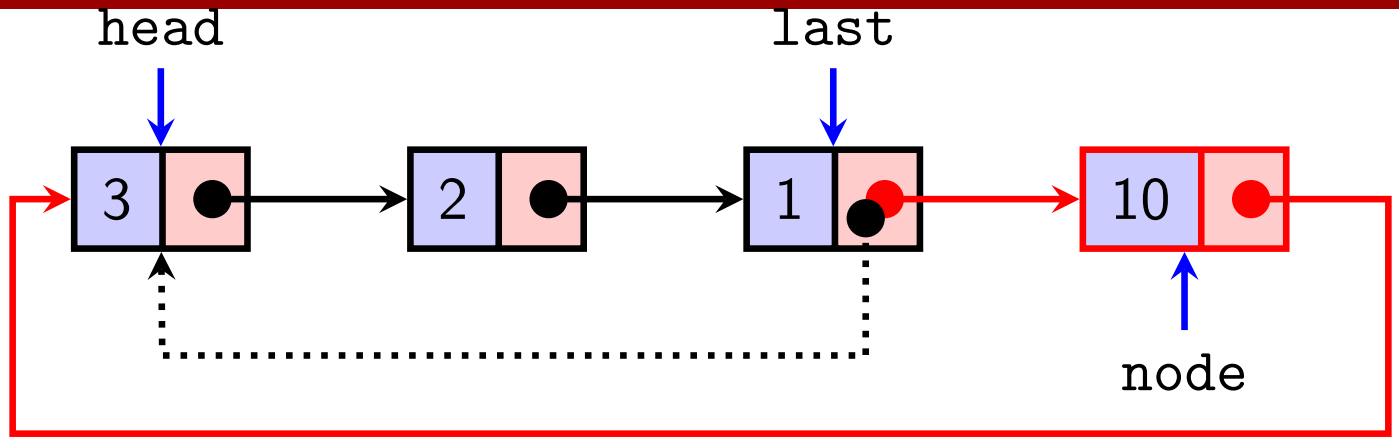
Construim un nod nou



Listă circulară – Inserarea la finalul listei

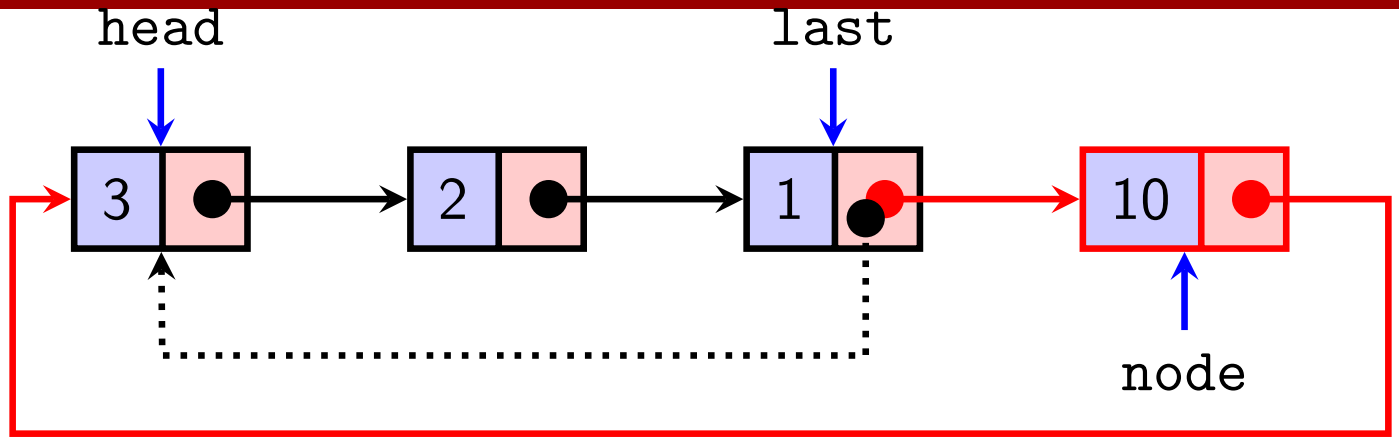


Listă circulară – Inserarea la finalul listei



Ce legături trebuie să refacem?

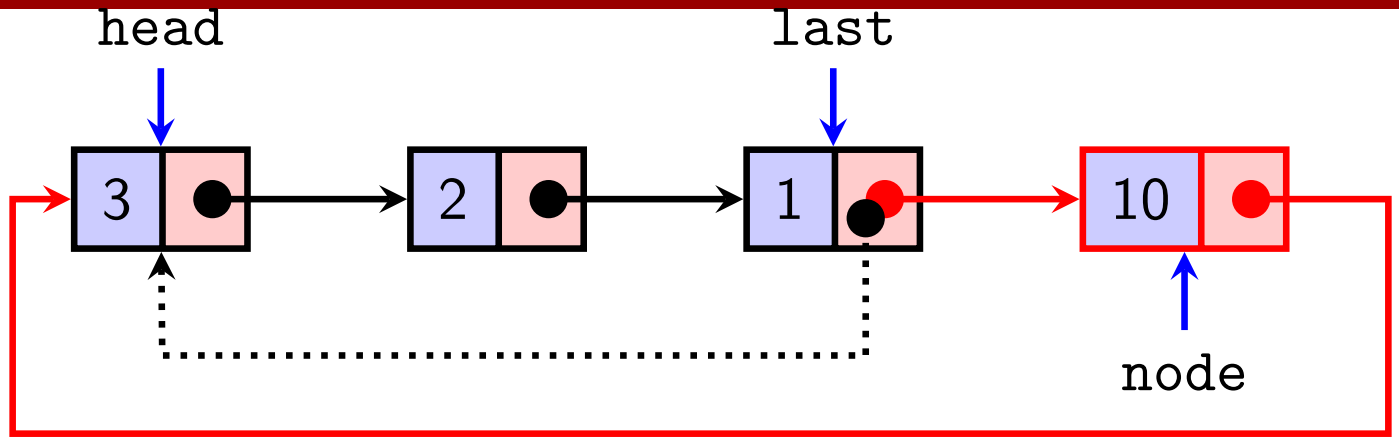
Listă circulară – Inserarea la finalul listei



Ce legături trebuie să refacem?

```
node = createList(data);
head = last->next;
node->next = head;
last->next = node;
```

Listă circulară – Inserarea la finalul listei



Ce legături trebuie să refacem?

```
node = createList(data);
head = last->next;
node->next = head;
last->next = node;
```

Către cine trebuie să pointeze `last`?

Listă circulară – Inserarea la finalul listei

```
10  List insertRear(List last, T data) {
11      List node = createList(data);
12      if (last == NULL)
13          return node;
14      List head = last->next;
15      node->next = head;
16      last->next = node;
17      return node;
18  }
```

Listă circulară – Inserarea la finalul listei

```
10  List insertRear(List last, T data) {
11      List node = createList(data);
12      if (last == NULL)
13          return node;
14      List head = last->next;
15      node->next = head;
16      last->next = node;
17      return node;
18  }
```

Ce complexitate avem acum?

Listă circulară – Inserarea la finalul listei

```
10  List insertRear(List last, T data) {
11      List node = createList(data);
12      if (last == NULL)
13          return node;
14      List head = last->next;
15      node->next = head;
16      last->next = node;
17      return node;
18  }
```

Ce complexitate avem acum?

Complexitatea nu mai depinde de numărul de elemente: $O(1)$

Listă circulară – Afișarea elementelor

```
19  void printList(List last) {
20      if (last == NULL) {
21          printf("NULL\n");
22          return;
23      }
24      List head = last->next, iter = head;
25      do {
26          printf("%d ", iter->value);
27          iter = iter->next;
28      } while (iter != head);
29      printf("\n");
30  }
```

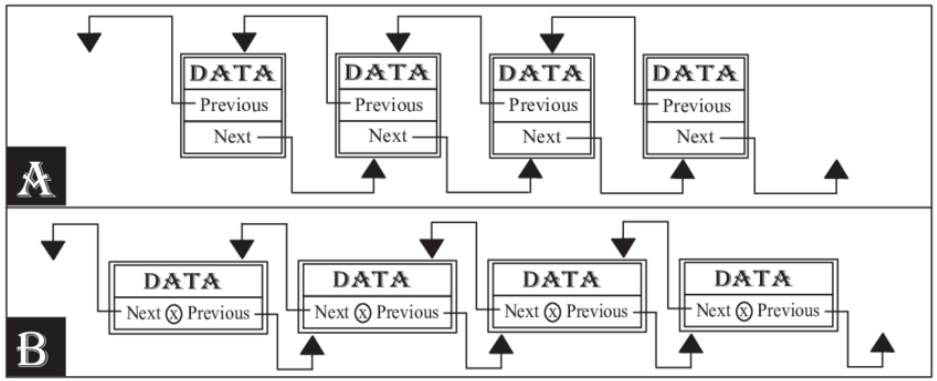
Listă circulară – Dealocarea memoriei

```
19  List freeList(List last) {
20      if (last == NULL)
21          return last;
22      List temp, head = last->next, iter = head;
23      do {
24          temp = iter;
25          iter = iter->next;
26          free(temp);
27      } while (iter != head);
28      return NULL;
29  }
```

Proprietățile operației XOR

1. **Comutativitate**
   `A ^ B = B ^ A`
2. **Asociativitate**
   `A ^ (B ^ C) = (A ^ B) ^ C`
3. **Element neutru**
   `A ^ 0 = A`
4. **Propriul invers**
   `A ^ A = 0`

Listă dublu înlănțuită cu legături XOR




```
1  #define XOR(a, b) (List)((intptr_t)(a)^(intptr_t)(b))
2  typedef struct node {
3      int val;
4      struct node *link;
5  } *List;
6  List initList(int val) {
7      List list = malloc(sizeof(struct node));
8      list->val = val;
9      list->link = NULL;
10     return list;
11 }
```

Listă dublu înlănțuită cu legături XOR

Inserarea la început

```
12  List insertHead(List head, int data) {
13      List newNode = initList(data);
14      if (head == NULL) {
15          return newNode;
16      } else {
17          /* Update original link of head node */
18          head->link = XOR(head->link, newNode);
19          newNode->link = head;
20          return newNode;
21      }
22  }
```

Listă dublu înlănțuită cu legături XOR

Ștergerea de la început

```
23  List deleteHead(List head) {
24      if (!head)
25          return head;
26      List tmp = head->link;
27      /* Update the link of new head */
28      if (tmp)
29          tmp->link = XOR(head, tmp->link);
30      free(head);
31      return tmp;
32  }
```

Listă dublu înlănțuită cu legături XOR

Afișarea listei

```
33  void printXorList(List head) {
34      printf("NULL ");
35      if (!head)
36          return;
37      List prev = NULL;
38      while (head) {
39          List tmp = head;
40          printf("<- %d -> ", head->val);
41          head = XOR(prev, head->link);
42          prev = tmp;
43      }
44      printf("NULL\n");
45  }
```

Listă dublu înlănțuită cu legături XOR

Dealocarea memoriei

```
46  List freeList(List head) {
47      if (!head)
48          return head;
49      List prev = NULL;
50      while (head) {
51          List tmp = head;
52          head = XOR(prev, head->link);
53          if (prev != NULL) {
54              free(prev);
55          }
56          prev = tmp;
57      }
58      if (prev != NULL)
59          free(prev);
60  }
```

Listă dublu înlănțuită cu legături XOR

```
61  int main() {
62      int keys[] = { 1, 2, 3, 4, 5 };
63      int n = sizeof(keys)/sizeof(keys[0]);
64
65      List head = NULL;
66      for (int i = n - 1; i >=0; i--)
67          head = insertHead(head, keys[i]);
68
69      printXorList(head);
70
71      head = freeList(head);
72      return 0;
73  }
```

Probleme de tip interviu

Enunț: Dându-se o listă simplu înlănțuită, determinați valoarea nodului din mijlocului listei cât mai eficient posibil. **NU** ne putem folosi de lungime în implementare.

Probleme de tip interviu

Enunț: Dându-se o listă simplu înlănțuită, determinați valoarea nodului din mijlocului listei cât mai eficient posibil. **NU** ne putem folosi de lungime în implementare.

Ne vom folosi de doi pointeri pentru a parcurge lista: `slow` și `fast`.

Când știm că am ajuns la mijloc?

Când `fast` ajunge la final, `slow` este la mijloc.

Probleme de tip interviu

Enunț: Dându-se o listă simplu înlănțuită, determinați valoarea nodului din mijlocului listei cât mai eficient posibil. **NU** ne putem folosi de lungime în implementare.

Ne vom folosi de doi pointeri pentru a parcurge lista: `slow` și `fast`. Primul pointer sare peste un element la fiecare iterație, iar cel de-al doilea pointer sare peste 2 elemente la fiecare iterație.

Probleme de tip interviu

Enunț: Dându-se o listă simplu înlănțuită, determinați valoarea nodului din mijlocului listei cât mai eficient posibil. **NU** ne putem folosi de lungime în implementare.

Ne vom folosi de doi pointeri pentru a parcurge lista: `slow` și `fast`. Primul pointer sare peste un element la fiecare iterație, iar cel de-al doilea pointer sare peste 2 elemente la fiecare iterație.

Când știm că am ajuns la mijloc?

Probleme de tip interviu

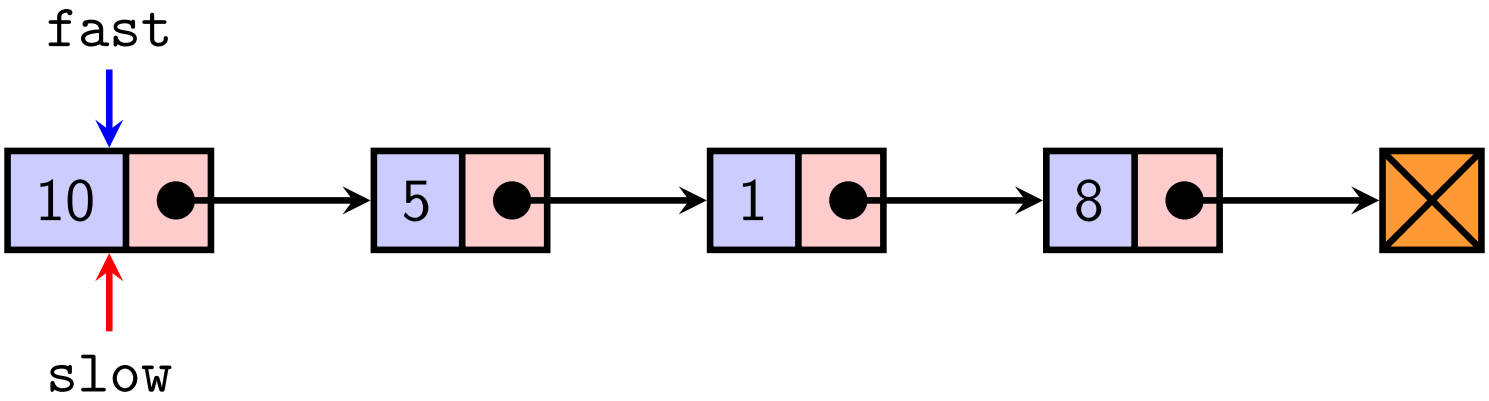
Enunț: Dându-se o listă simplu înlănțuită, determinați valoarea nodului din mijlocului listei cât mai eficient posibil. **NU** ne putem folosi de lungime în implementare.

Ne vom folosi de doi pointeri pentru a parcurge lista: `slow` și `fast`. Primul pointer sare peste un element la fiecare iterație, iar cel de-al doilea pointer sare peste 2 elemente la fiecare iterație.

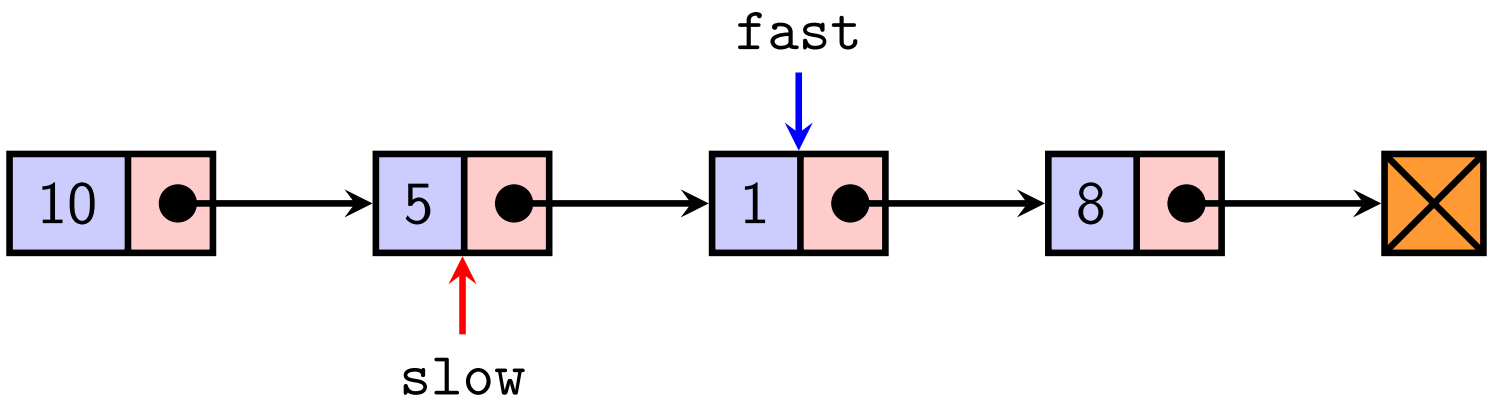
Când știm că am ajuns la mijloc?

Când `fast` ajunge la final, `slow` este la mijloc.

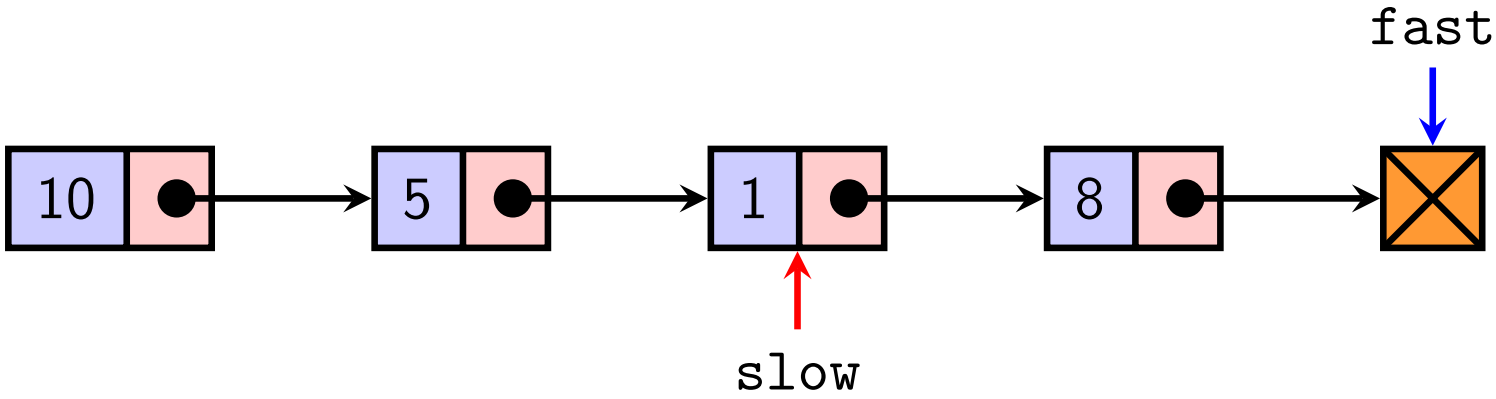
Probleme de tip interviu



Probleme de tip interviu



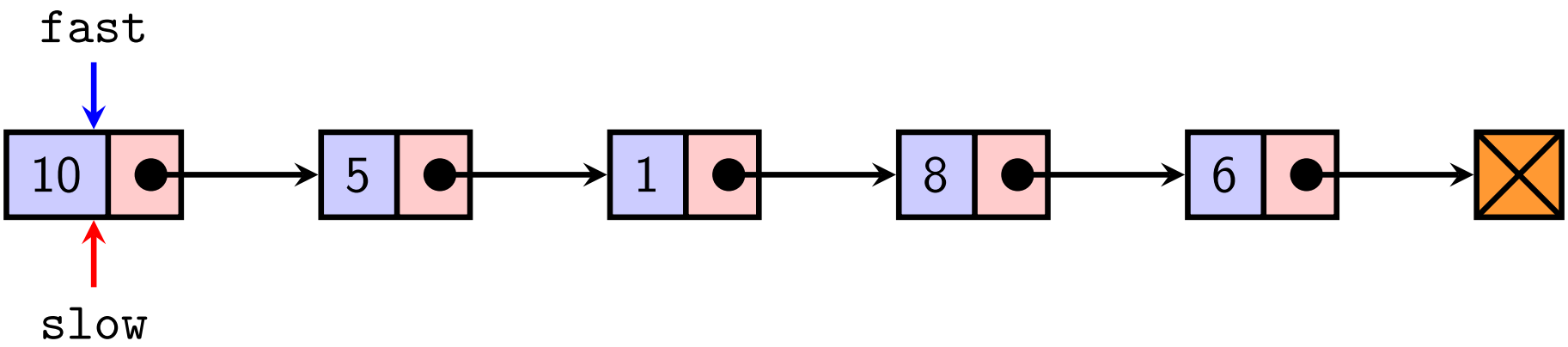
Probleme de tip interviu



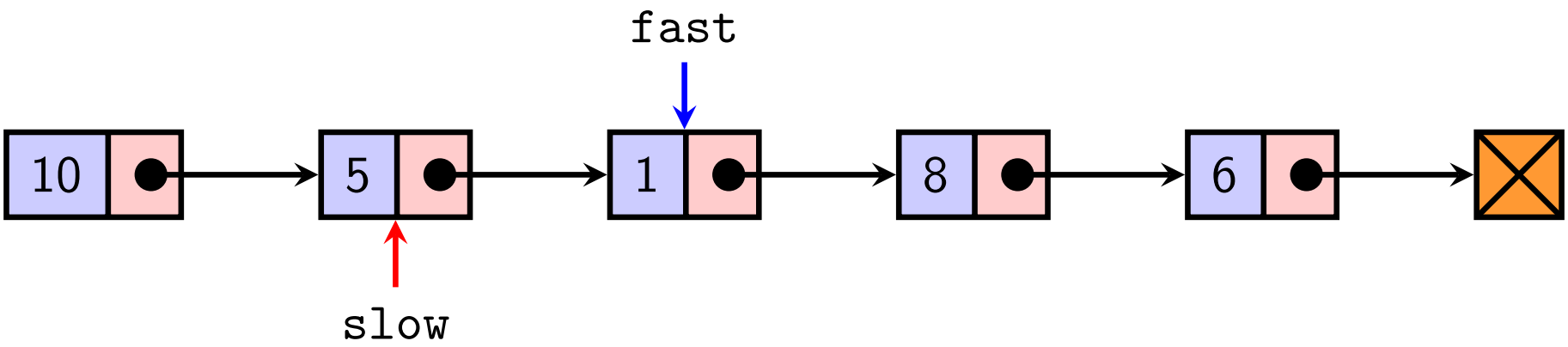
Observație

O să considerăm că în cazul în care lista are număr par de elemente acesta este elementul din *mijloc*.

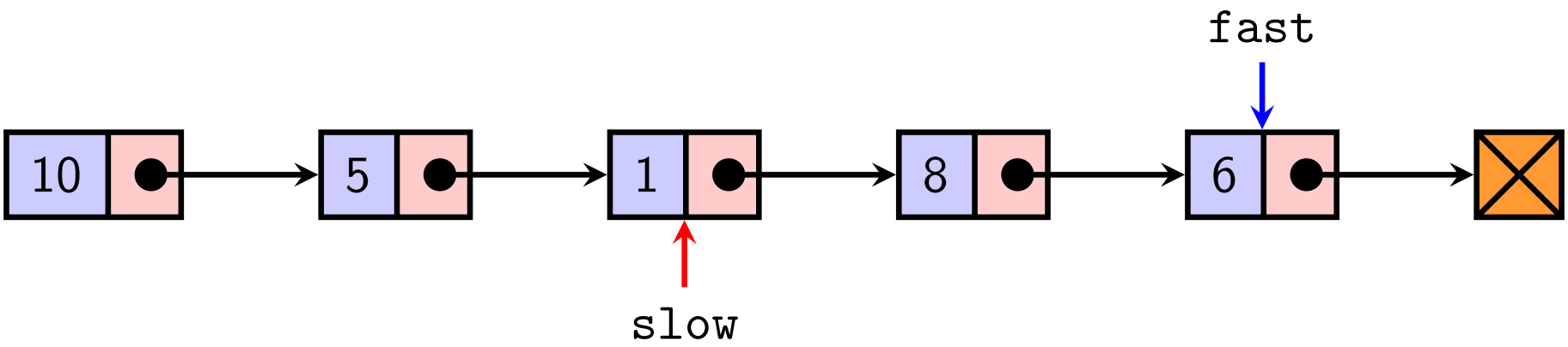
Probleme de tip interviu



Probleme de tip interviu



Probleme de tip interviu



Observație

În cazul în care lista are număr impar de elemente, ne oprim când `fast->next == NULL`.

Probleme de tip interviu

Enunț: Implementați o funcție care primește o listă simplu înlănțuită cu N noduri și un număr natural k, ce se garantează că este mai mic sau egal cu N, și afișează ultimele k valori din listă în ordine inversă.

Exemple:

```
list: 1->2->3->4->5, k = 2
output: 5 4

list: 3->10->6->9->12->2->8, k = 4
output: 8 2 12 9
```

Probleme de tip interviu

Enunț: Implementați o funcție care primește o listă simplu înlănțuită cu N noduri și un număr natural k, ce se garantează că este mai mic sau egal cu N, și afișează ultimele k valori din listă în ordine inversă.

Exemple:

```
list: 1->2->3->4->5, k = 2
output: 5 4

list: 3->10->6->9->12->2->8, k = 4
output: 8 2 12 9
```

Ce concept ne-ar ajuta să implementăm această funcție?

Probleme de tip interviu

Enunț: Implementați o funcție care primește o listă simplu înlănțuită cu N noduri și un număr natural k, ce se garantează că este mai mic sau egal cu N, și afișează ultimele k valori din listă în ordine inversă.

Exemple:

```
list: 1->2->3->4->5, k = 2
output: 5 4

list: 3->10->6->9->12->2->8, k = 4
output: 8 2 12 9
```

Ce concept ne-ar ajuta să implementăm această funcție?

Recursivitatea

Probleme de tip interviu

```
1  void printLastKRev(TList head, int *count, int k) {
2      if (head == NULL)
3          return;
4      printLastKRev(head->next, count, k);
5      (*count)++;
6      if (*count <= k)
7          printf("%d ", head->value);
8  }
```

Probleme de tip interviu

```
1  void printLastKRev(TList head, int *count, int k) {
2      if (head == NULL)
3          return;
4      printLastKRev(head->next, count, k);
5      (*count)++;
6      if (*count <= k)
7          printf("%d ", head->value);
8  }
```

Cum pot apela această funcție?

Probleme de tip interviu

```
1  void printLastKRev(TList head, int *count, int k) {
2      if (head == NULL)
3          return;
4      printLastKRev(head->next, count, k);
5      (*count)++;
6      if (*count <= k)
7          printf("%d ", head->value);
8  }
```

Cum pot apela această funcție?

```
int count = 0;
printLastKRev(head, &count, k);
```

Probleme de tip interviu

Enunț: Implementați o funcție care inversează o listă dublu înlănțuită fără a folosi memorie suplimentară.

Probleme de tip interviu

Enunț: Implementați o funcție care inversează o listă dublu înlănțuită fără a folosi memorie suplimentară.

Algorithm 1 Reverse

```
1:  procedure REVERSE(list)
2:      temp ← NULL
3:      current ← list
4:      while current ≠ NULL do
5:          temp ← current → prev
6:          current → prev ← current → next
7:          current → next ← temp
8:          current ← current → prev
9:      if temp ≠ NULL then
10:         list ← temp → prev
        return list
```

Vă mulțumesc pentru atenție!