Partie 1: Listes

Marie-Pierre Béal, Laurent Bulteau

Université Paris-Est Marne-la-Vallée

Plan

1. Préambule
2. Présentation des listes
   - Motivation
   - Définition
3. Implémentation
   - Type et constructeur
   - Exemple
4. Fonctions utiles
5. Piles et Files
   - Piles
   - Listes doublement chaînées
   - Files

Préambule

- Nous utiliserons une syntaxe semblable à celle de C pour présenter les exemples ou les signatures de fonctions. Néanmoins, le contenu de ce cours n'est pas propre à C.
- Nous passons sous silence certains aspects pratiques du C (nettoyage de la mémoire, gestion des erreurs, etc.) pour se focaliser sur les aspects algorithmiques.
- Complexité d'un algorithme : compter le nombre d'opérations effectuées dans le pire cas en fonction de la taille de l'entrée.
- Notation $O()$: borne supérieure asymptotique "à un facteur près". Quelques exemples :
  $\frac{n}{3} \in O(n) \quad 5n + \sqrt{n} + 3 \in O(n) \quad n^2 \notin O(n) \quad 18 - \frac{1}{n} \in O(1)$

Exercice 1 – Complexité

```
entrées: tableau d'entiers tab et sa longueur n
s:=0;
pour i allant de 1 à n faire
    pour j allant de 1 à tab[i] faire
        s:=i*j;
```

Quelle est la complexité de cet algorithme...

1. ... si tous les entiers contenus dans `tab` sont plus petits que 10 ?
2. ... si tous les entiers contenus dans `tab` sont plus petits que n ?
3. ... si la valeur maximale m des entiers contenus dans `tab` est un paramètre ?

Exercice 1 – Complexité

```
entrées: tableau d'entiers tab et sa longueur n
s:=0;
pour i allant de 1 à n faire
    pour j allant de 1 à tab[i] faire
        s:=i*j;
```

Quelle est la complexité de cet algorithme...

1. ... si tous les entiers contenus dans `tab` sont plus petits que 10 ?
   $O(n)$
2. ... si tous les entiers contenus dans `tab` sont plus petits que n ?

3. ... si la valeur maximale m des entiers contenus dans `tab` est un paramètre ?

Exercice 1 – Complexité

```
entrées: tableau d'entiers tab et sa longueur n
s:=0;
pour i allant de 1 à n faire
    pour j allant de 1 à tab[i] faire
        s:=i*j;
```

Quelle est la complexité de cet algorithme...

1. ... si tous les entiers contenus dans `tab` sont plus petits que 10 ?
   $O(n)$
2. ... si tous les entiers contenus dans `tab` sont plus petits que n ?
   $O(n^2)$
3. ... si la valeur maximale m des entiers contenus dans `tab` est un paramètre ?

Exercice 1 – Complexité

```
entrées: tableau d'entiers tab et sa longueur n
s:=0;
pour i allant de 1 à n faire
    pour j allant de 1 à tab[i] faire
        s:=i*j;
```

Quelle est la complexité de cet algorithme...

1. ... si tous les entiers contenus dans `tab` sont plus petits que 10 ?
   $O(n)$
2. ... si tous les entiers contenus dans `tab` sont plus petits que n ?
   $O(n^2)$
3. ... si la valeur maximale m des entiers contenus dans `tab` est un paramètre ? $O(nm)$

Exercice 1 – Complexité

```
entrées: tableau d'entiers tab et sa longueur n
s:=0;
pour i allant de 1 à n faire
    pour j allant de 1 à tab[i] faire
        s:=i*j;
```

Quelle est la complexité de cet algorithme...

4. ... si n est fixé et m est un paramètre ?
5. ... si tous les entiers contenus dans `tab` sont plus petits que n et la multiplication de deux entiers a et b coûte $O(M)$, avec $M = \max(a, b)$?

Exercice 1 – Complexité

```
entrées: tableau d'entiers tab et sa longueur n
s:=0;
pour i allant de 1 à n faire
    pour j allant de 1 à tab[i] faire
        s:=i*j;
```

Quelle est la complexité de cet algorithme...

4. ... si n est fixé et m est un paramètre ? $\boxed{O(m)}$
5. ... si tous les entiers contenus dans `tab` sont plus petits que n et la multiplication de deux entiers a et b coûte $O(M)$, avec $M = \max(a, b)$?

Exercice 1 – Complexité

```
entrées: tableau d'entiers tab et sa longueur n
s:=0;
pour i allant de 1 à n faire
    pour j allant de 1 à tab[i] faire
        s:=i*j;
```

Quelle est la complexité de cet algorithme...

4. ... si n est fixé et m est un paramètre ? $\boxed{O(m)}$
5. ... si tous les entiers contenus dans `tab` sont plus petits que n et la multiplication de deux entiers a et b coûte $O(M)$, avec $M = \max(a, b)$? $\boxed{O(n^3)}$

Motivation

La liste est une *structure de donnée* permettant de stocker des collections ordonnées d'objets (en général de même type).

Motivation

La liste est une *structure de donnée* permettant de stocker des collections ordonnées d'objets (en général de même type).

Par rapport à un tableau classique :

+ Faciliter les insertions/suppressions
+ Taille dynamique

Motivation

La liste est une *structure de donnée* permettant de stocker des collections ordonnées d'objets (en général de même type).

Par rapport à un tableau classique :

+ Faciliter les insertions/suppressions
+ Taille dynamique

Définition

Soit T un type quelconque.

Définition (récursive)

Une *liste de* T est

- soit la *liste vide*,

$$\emptyset$$

- soit composée d'une *tête de liste* de type T avec une *queue de liste*, de type *liste de* T.

Définition

Soit T un type quelconque.

Définition (récursive)

Une *liste de* T est

- soit la *liste vide*,

$$\emptyset$$

- soit composée d'une *tête de liste* de type T avec une *queue de liste*, de type *liste de* T.

Exemple : $(12, 42, -3, 42)$

$T =$ ensemble de nombres entiers,

Définition

Soit T un type quelconque.

Définition (récursive)

Une *liste de* T est

- soit la *liste vide*,

$$\emptyset$$

- soit composée d'une *tête de liste* de type T avec une *queue de liste*, de type *liste de* T.

Exemple : $(12, 42, -3, 42)$

$T =$ ensemble de nombres entiers,

Définition

Soit T un type quelconque.

Définition (récursive)

Une *liste de* T est

- soit la *liste vide*,

$$\emptyset$$

- soit composée d'une *tête de liste* de type T avec une *queue de liste*, de type *liste de* T.

Exemple : $(12, 42, -3, 42)$

$T =$ ensemble de nombres entiers,

Définition

Soit T un type quelconque.

Définition (récursive)

Une *liste de* T est

- soit la *liste vide*,

$$\emptyset$$

- soit composée d'une *tête de liste* de type T avec une *queue de liste*, de type *liste de* T.

Exemple : $(12, 42, -3, 42)$

$T =$ ensemble de nombres entiers,

Définition

Soit T un type quelconque.

Définition (récursive)

Une *liste de* T est

- soit la *liste vide*,

$$\emptyset$$

- soit composée d'une *tête de liste* de type T avec une *queue de liste*, de type *liste de* T.

Exemple : $(12, 42, -3, 42)$

$T =$ ensemble de nombres entiers,

Implémentation par pointeurs

Déclaration du type Liste

```
typedef int Element;
typedef struct cellule{
    Element elem;
    struct cellule* succ;
} Cellule;
typedef Cellule* Liste;
```

Une liste de T est **représentée par un pointeur**

- pour la liste vide : on utilise la valeur **NULL**
- sinon le pointeur désigne une **structure** ("cellule") comprenant un élément de type T (la tête de liste) et un nouveau pointeur (la queue de liste).

Implémentation par pointeurs

Déclaration du type Liste

```
typedef int Element;
typedef struct cellule{
    Element elem;
    struct cellule* succ;
} Cellule;
typedef Cellule* Liste;
```

Construction d'une liste

```
Liste cons(Liste queue, Element elem){
    Liste l = malloc(sizeof(Cellule));
    l->elem = elem;
    l->succ = queue;
    return l;
}
```

Exemple : $(12, 42, -3)$

Exemple

```
Liste l=NULL;
l=cons(l, -3);
l=cons(l, 42);
l=cons(l, 12);
```

`l` : *NULL*

Exemple : $(12, 42, -3)$

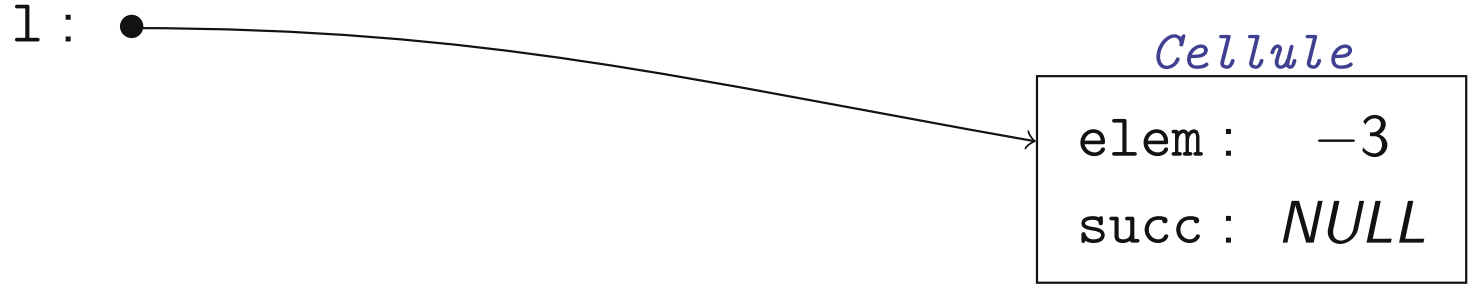
Exemple

```
Liste l=NULL;
l=cons(l, -3);
l=cons(l, 42);
l=cons(l, 12);
```

Exemple : $(12, 42, -3)$

Exemple

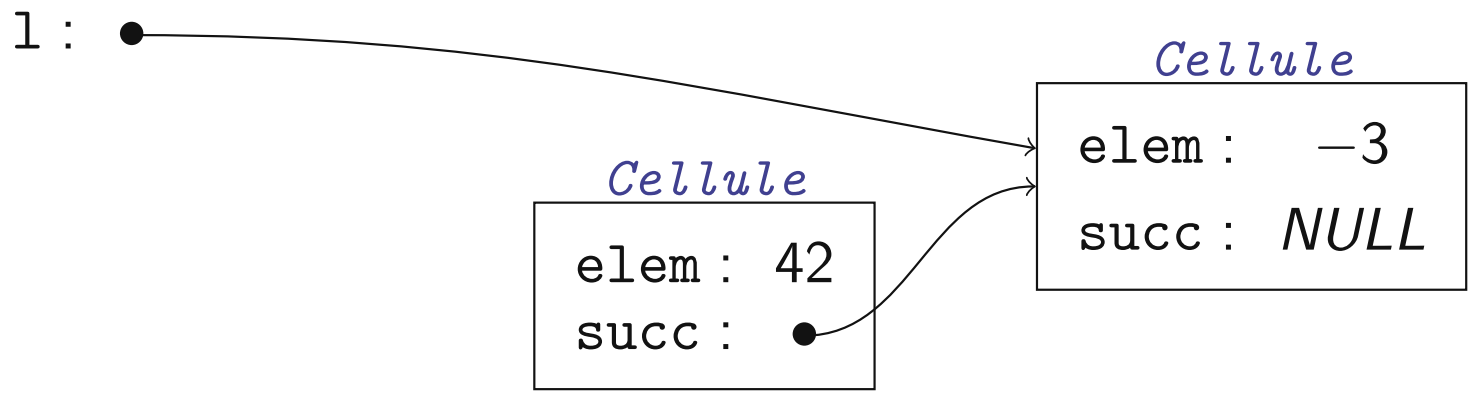
```
Liste l=NULL;
l=cons(l, -3);
l=cons(l, 42);
l=cons(l, 12);
```



Exemple : $(12, 42, -3)$

Exemple

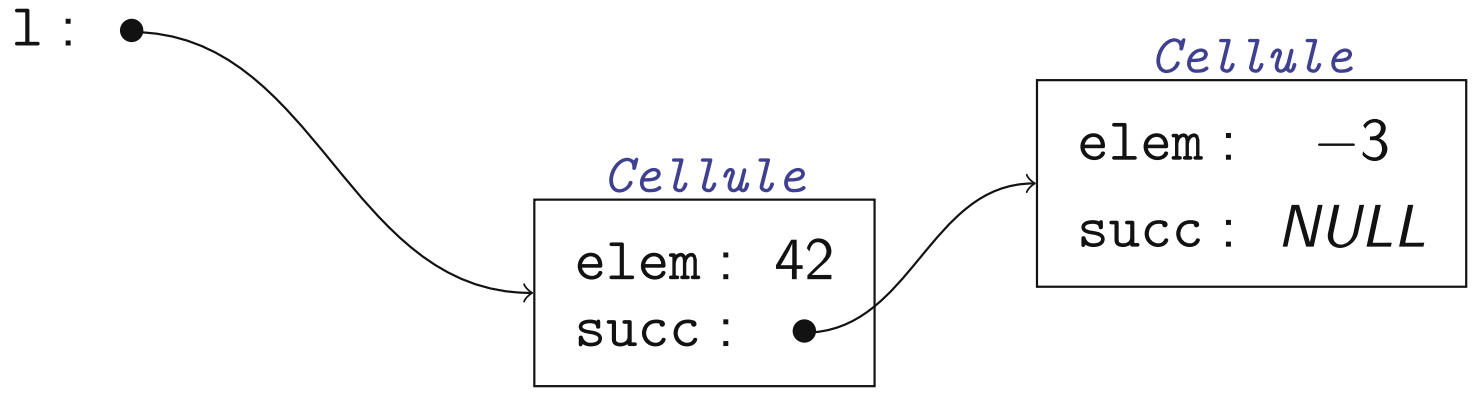
```
Liste l=NULL;
l=cons(l, -3);
l=cons(l, 42);
l=cons(l, 12);
```



Exemple : $(12, 42, -3)$

Exemple

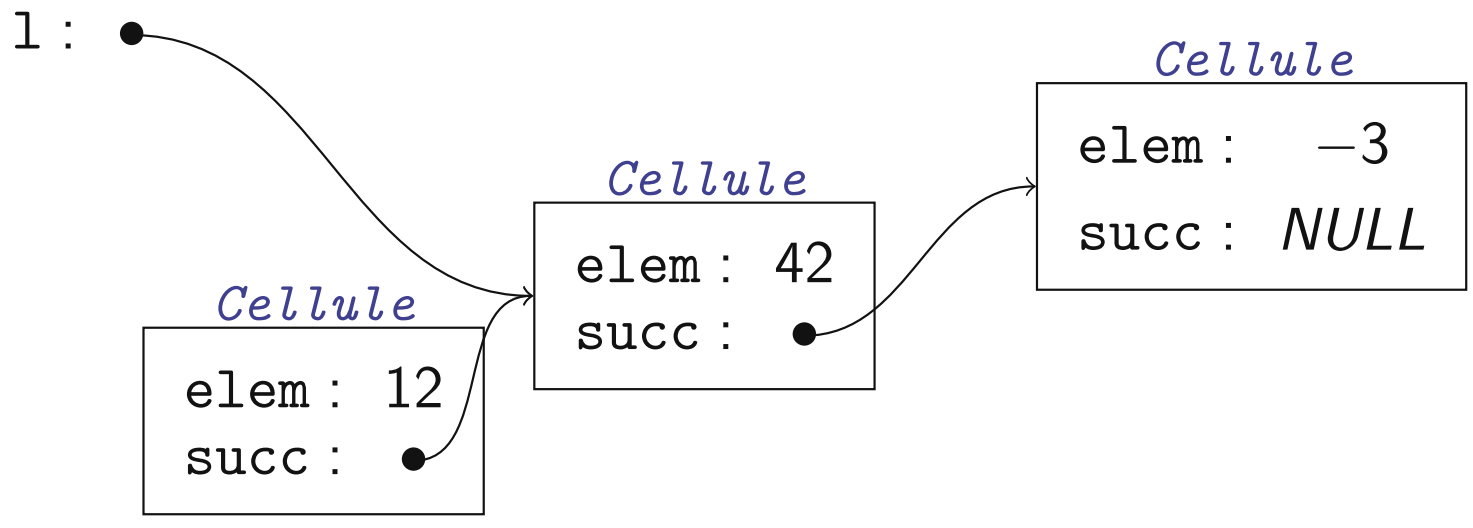
```
Liste l=NULL;
l=cons(l, -3);
l=cons(l, 42);
l=cons(l, 12);
```



Exemple : $(12, 42, -3)$

Exemple

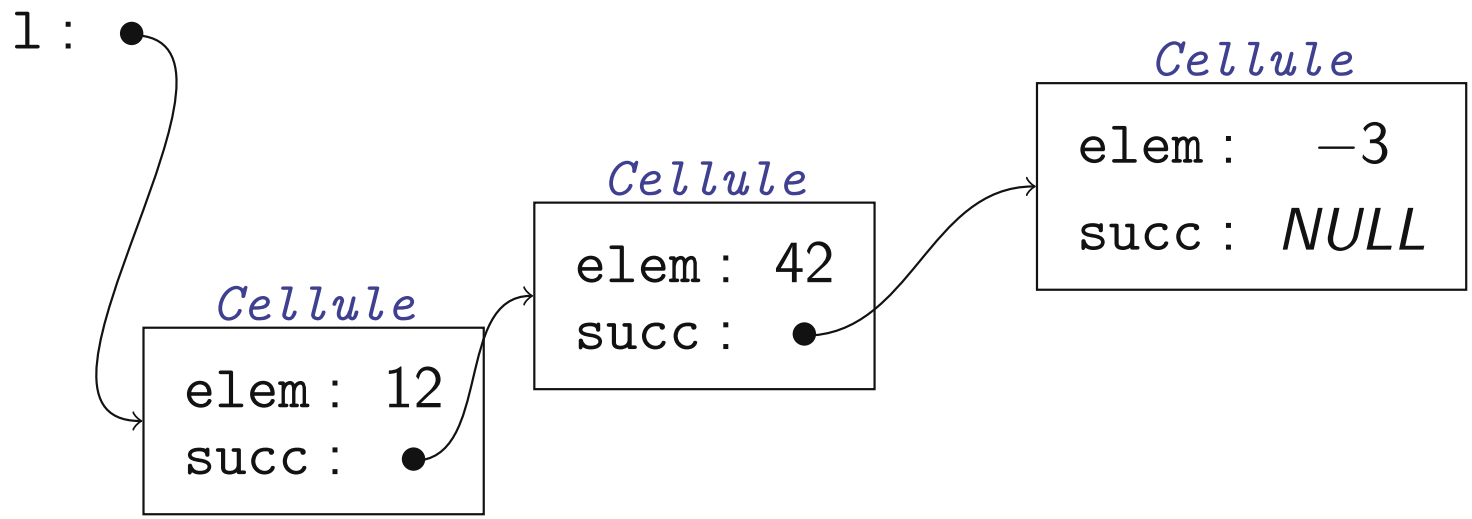
```
Liste l=NULL;
l=cons(l, -3);
l=cons(l, 42);
l=cons(l, 12);
```



Exemple : $(12, 42, -3)$

Exemple

```
Liste l=NULL;
l=cons(l, -3);
l=cons(l, 42);
l=cons(l, 12);
```



Rappel sur les pointeurs en C

Déclaration du type Liste (rappel)

```
typedef struct cellule{
    Element elem;
    struct cellule* succ;
} Cellule;
typedef Cellule* Liste;
```

Si `l` est du type `Liste` et non-NULL, alors :

- `*l` est du type `Cellule`
- `(*l).elem` est du type `Element`
- `(*l).succ` est du type `Liste`

Rappel sur les pointeurs en C

Déclaration du type Liste (rappel)

```
typedef struct cellule{
    Element elem;
    struct cellule* succ;
} Cellule;
typedef Cellule* Liste;
```

Si `l` est du type `Liste` et non-NULL, alors :

- `*l` est du type `Cellule`
- `(*l).elem` = `l->elem` est du type `Element`
- `(*l).succ` = `l->succ` est du type `Liste`

Calculer la taille d'une liste

Calculer la taille d'une liste

```
int longueur(Liste l) {
    if (l==NULL) {
        return 0;
    } else {
        return 1 + longueur(l -> succ);
    }
}
```

Complexité : $O(n)$ (où n est la taille de la liste)

Calculer la taille d'une liste

Calculer la taille d'une liste

```
int longueur(Liste l) {
    if (l==NULL) {
        return 0;
    } else {
        return 1 + longueur(l -> succ);
    }
}
```

Complexité : $O(n)$ (où n est la taille de la liste) *Les listes peuvent être parcourues soit de manière récursive, soit de manière itérative. Dans ce cours on privilégie la première méthode.*

```
while (l!=NULL) {
    [...]
    l=l->succ;
}
```

Exercice 2 – Fonctions utiles

Écrire les fonctions suivantes, et donner leur complexité.

1. `Element thirdElement(Liste l);` Renvoie le 3^e élément de `l`.
2. `Element ithElement(int i, Liste l);` Renvoie le i^e élément de `l`.
3. `int contains(Element e, Liste l);` Teste si la liste `l` contient l'élément `e`.
4. `Liste deleteAll(Element e, Liste l);` Renvoie une copie de `l` sans `e`.
5. `Liste mirror(Liste l);` Renvoie une copie de `l` en ordre inversé.

Exercice 3 – Fonctions sur les entiers

On suppose dans cet exercice que Element est le type int. Écrire les fonctions suivantes, et donner leur complexité.

1. `Element maxList(Liste l);` Renvoie le maximum de `l`.
2. `int isIncreasing(Liste l);` Teste si `l` est croissante.
3. `Liste makeRange(int a, int b);` Renvoie la liste des éléments de l'intervalle [a..b].

Piles et Files

Les piles et les files désignent des structures de donnée disposant des fonctions suivantes :

Pile ("LIFO" : Last In, First Out)

- `int empty(pile p)` teste si la pile est vide
- `void push(Element e, pile* p)` ajoute `e` en tête de `p`
- `Element pop(pile* p)` retire et renvoie le premier élément de `p`

File ("FIFO" : First In, First Out)

- `int empty(file f)` teste si la file est vide
- `void append(Element e, file* f)` ajoute `e` à la fin de `f`
- `Element pop(file* f)` retire et renvoie le premier élément de `f`

Exercice 4 – Fonctions de pile (LIFO)

1. Écrire les fonctions nécessaires pour utiliser le type Liste comme une pile.
2. Quelles sont leur complexité ?
3. Quelle serait la complexité de la fonction `append` sur ce type Liste ?

Listes doublement chaînées

Extension classique des listes :

- Conserver un pointeur vers le **dernier** élément
- Chaque élément connait son successeur **et son prédécesseur**

Avantages :

- Faible sur-coût
- Ajout et suppression en temps constant à la fin de la liste
- Parcours possible "à l'envers"

Implémentation des listes doublement chaînées

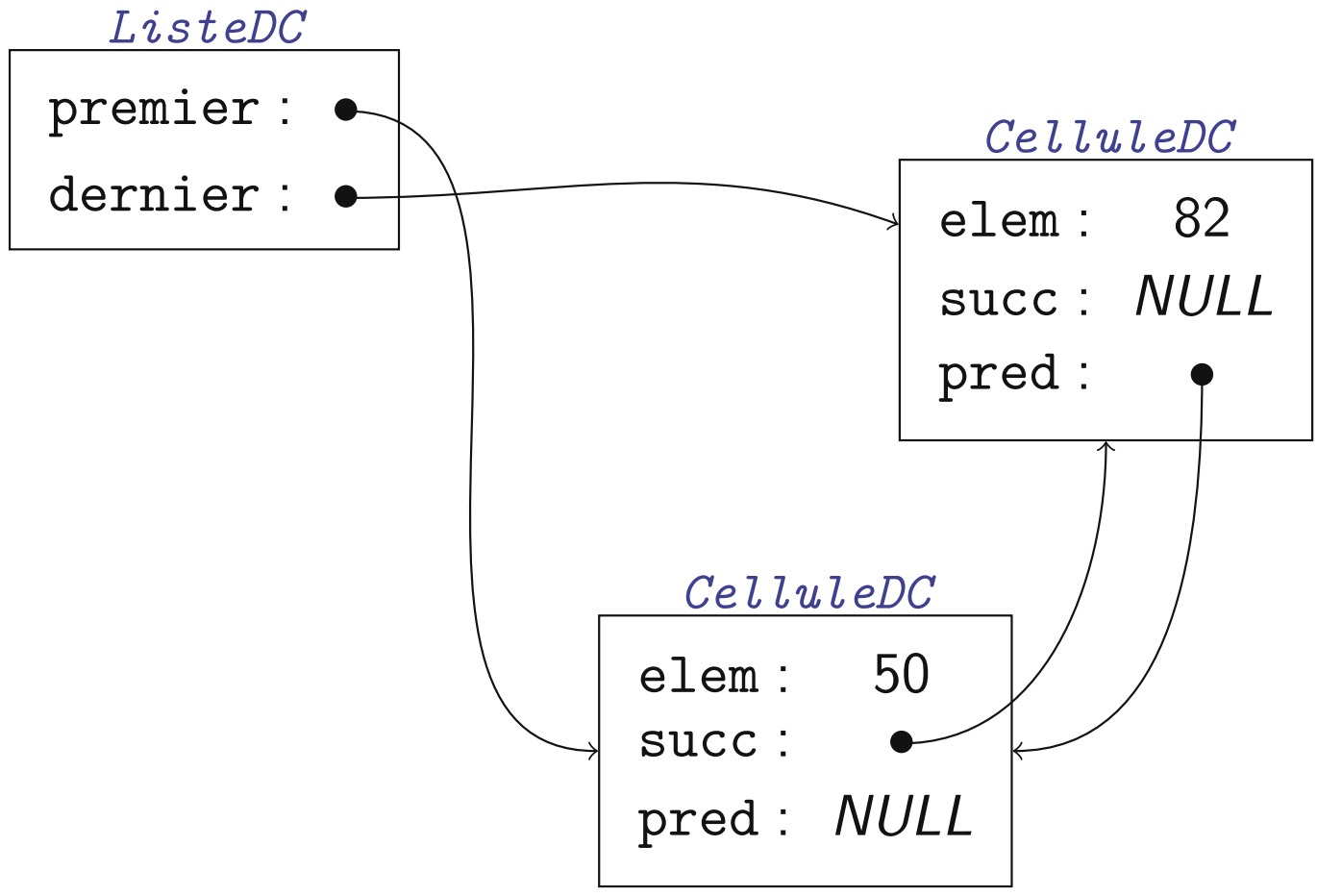
Déclaration du type "liste doublement chaînée"

```
typedef int Element;

typedef struct celluleDC{
    Element elem;
    struct celluleDC* pred;
    struct celluleDC* succ;
} CelluleDC;

typedef struct {
    CelluleDC* premier;
    CelluleDC* dernier;
} ListeDC;
```

Attention : "NULL" n'est plus synonyme de la liste vide.

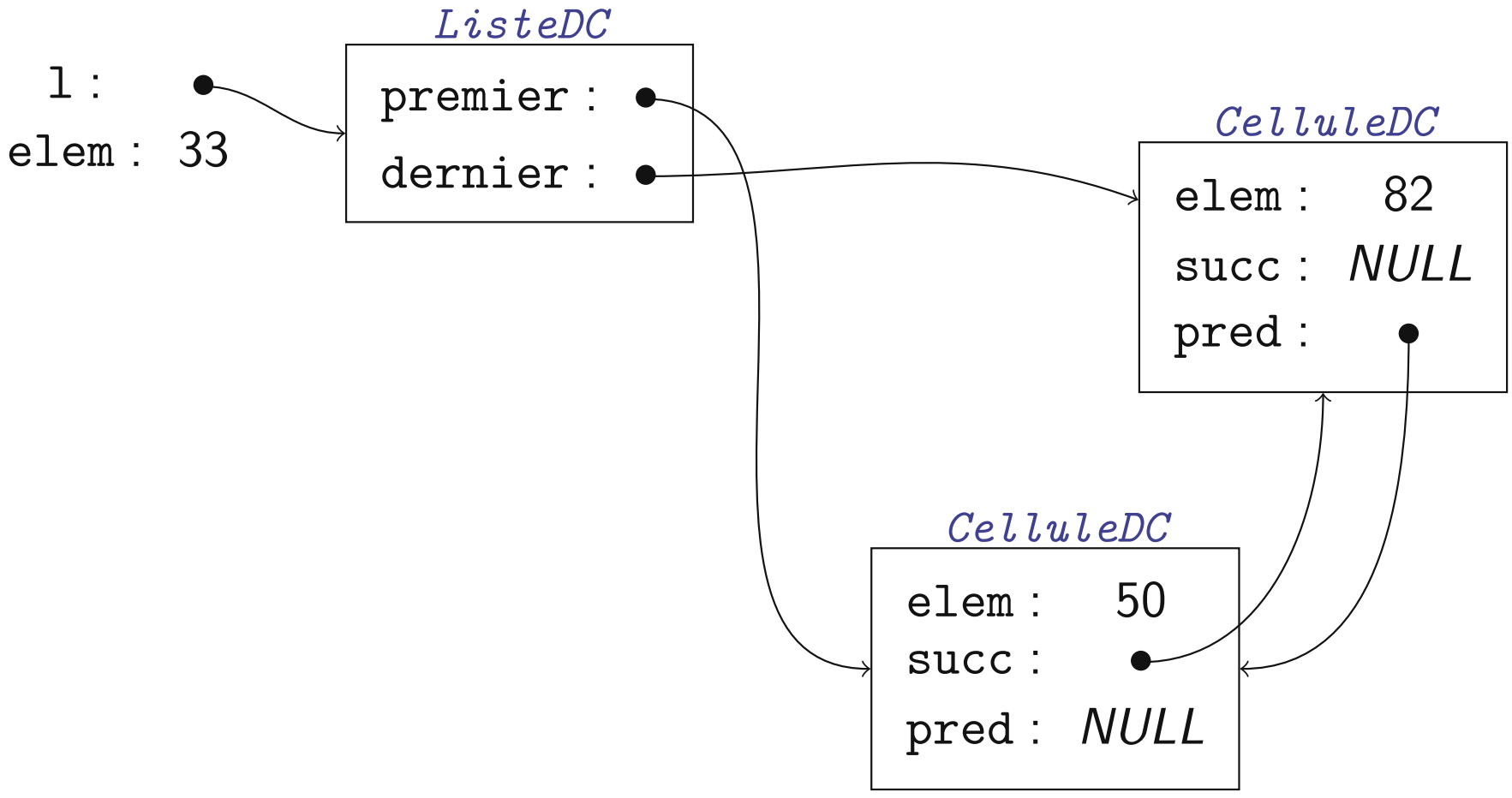
Implémentation des listes doublement chaînées

Exemple : la liste (50, 82)



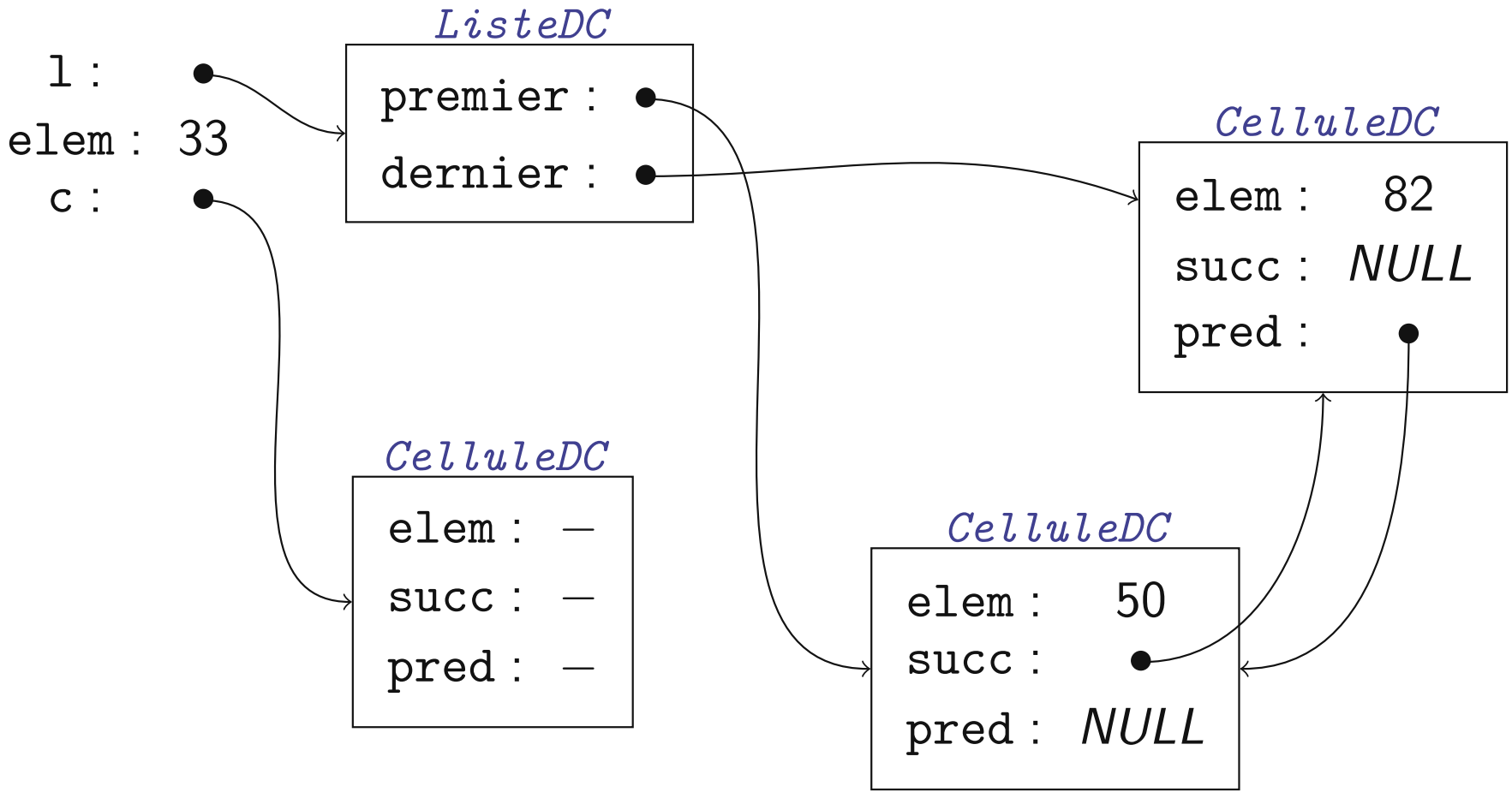
Implémentation des listes doublement chaînées

```
void push(Element elem, ListeDC* l){
```



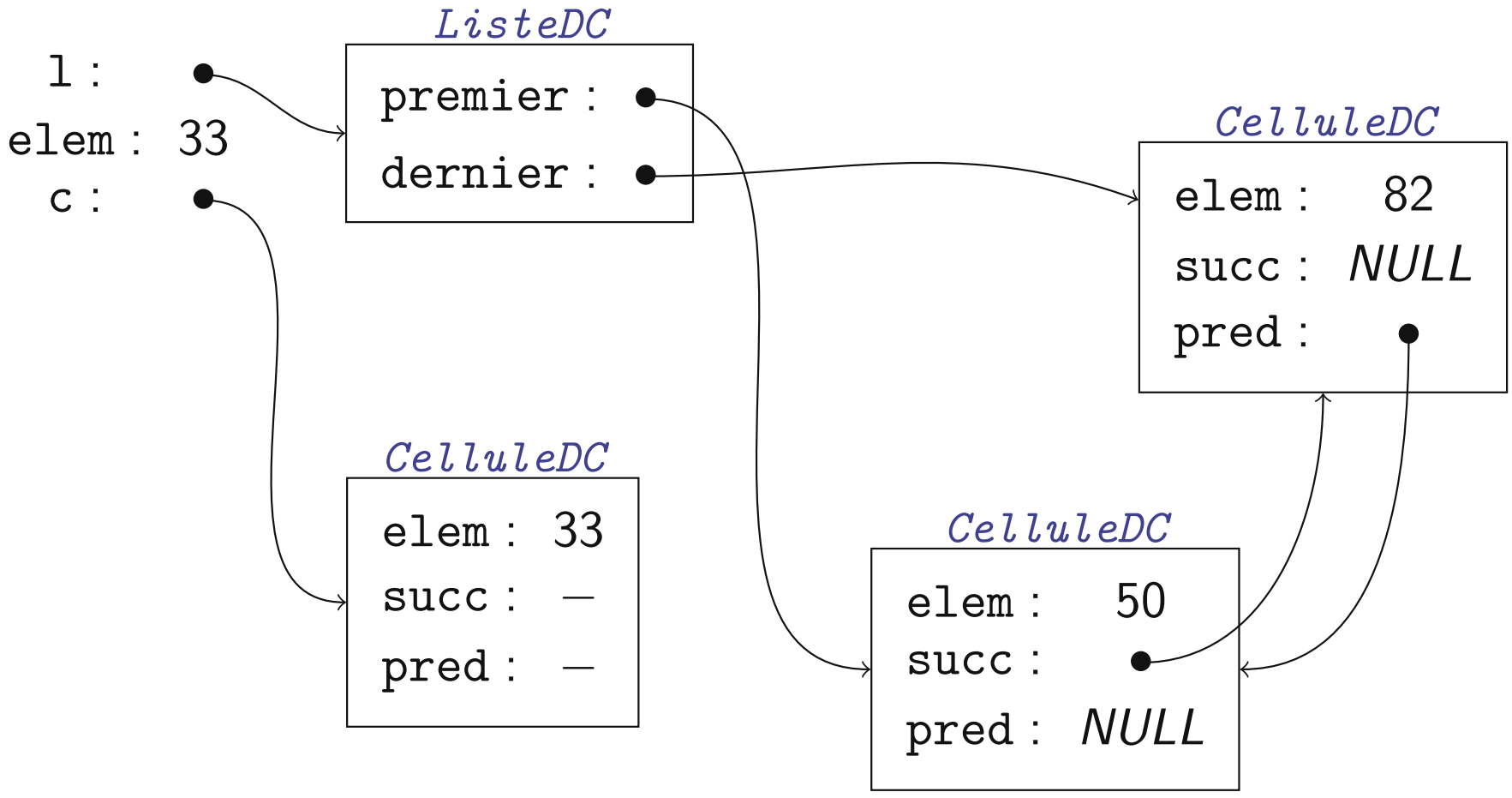
Implémentation des listes doublement chaînées

```
CelluleDC* c = malloc(sizeof(CelluleDC));
```



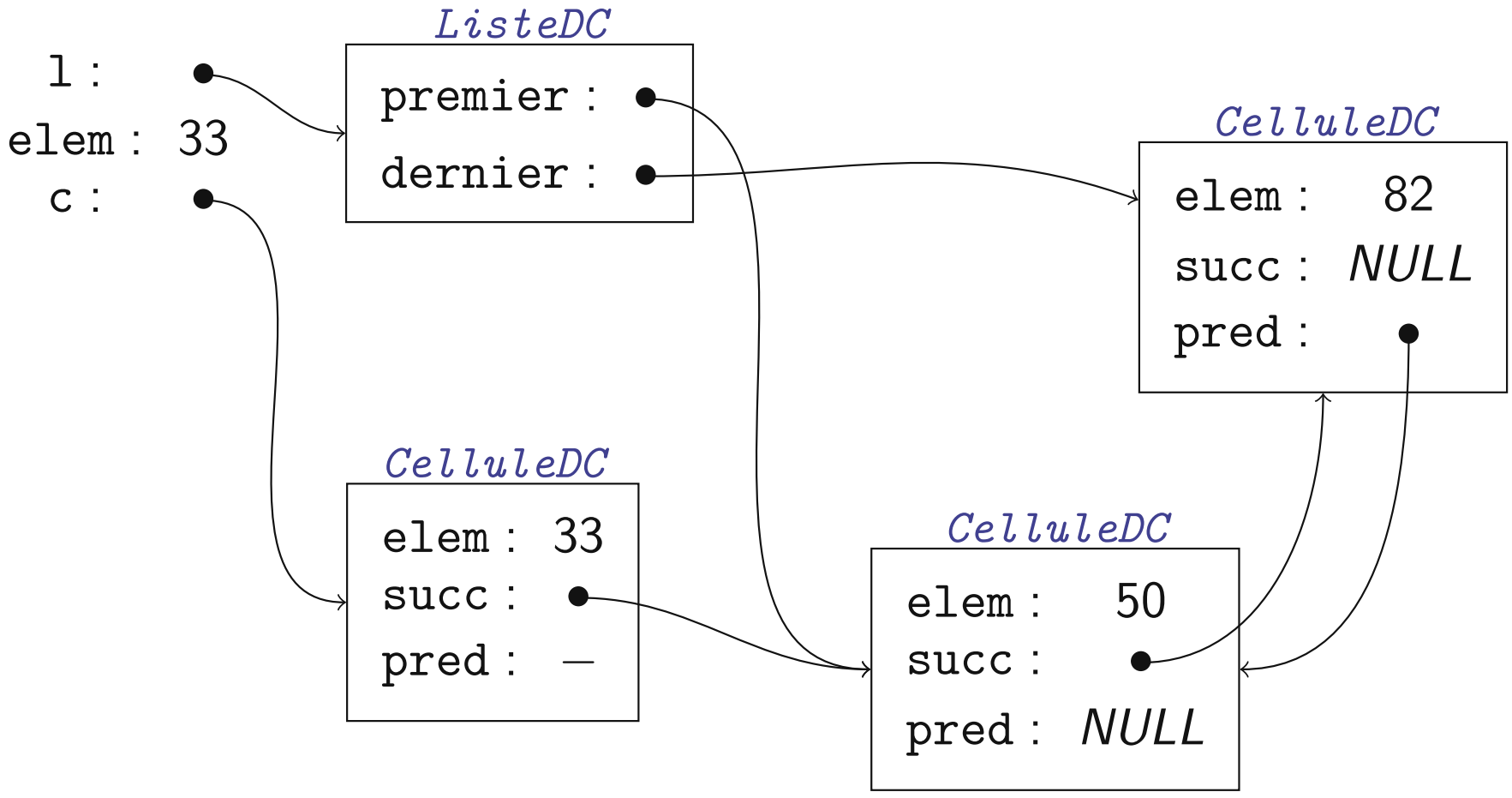
Implémentation des listes doublement chaînées

```
c->elem = elem;
```



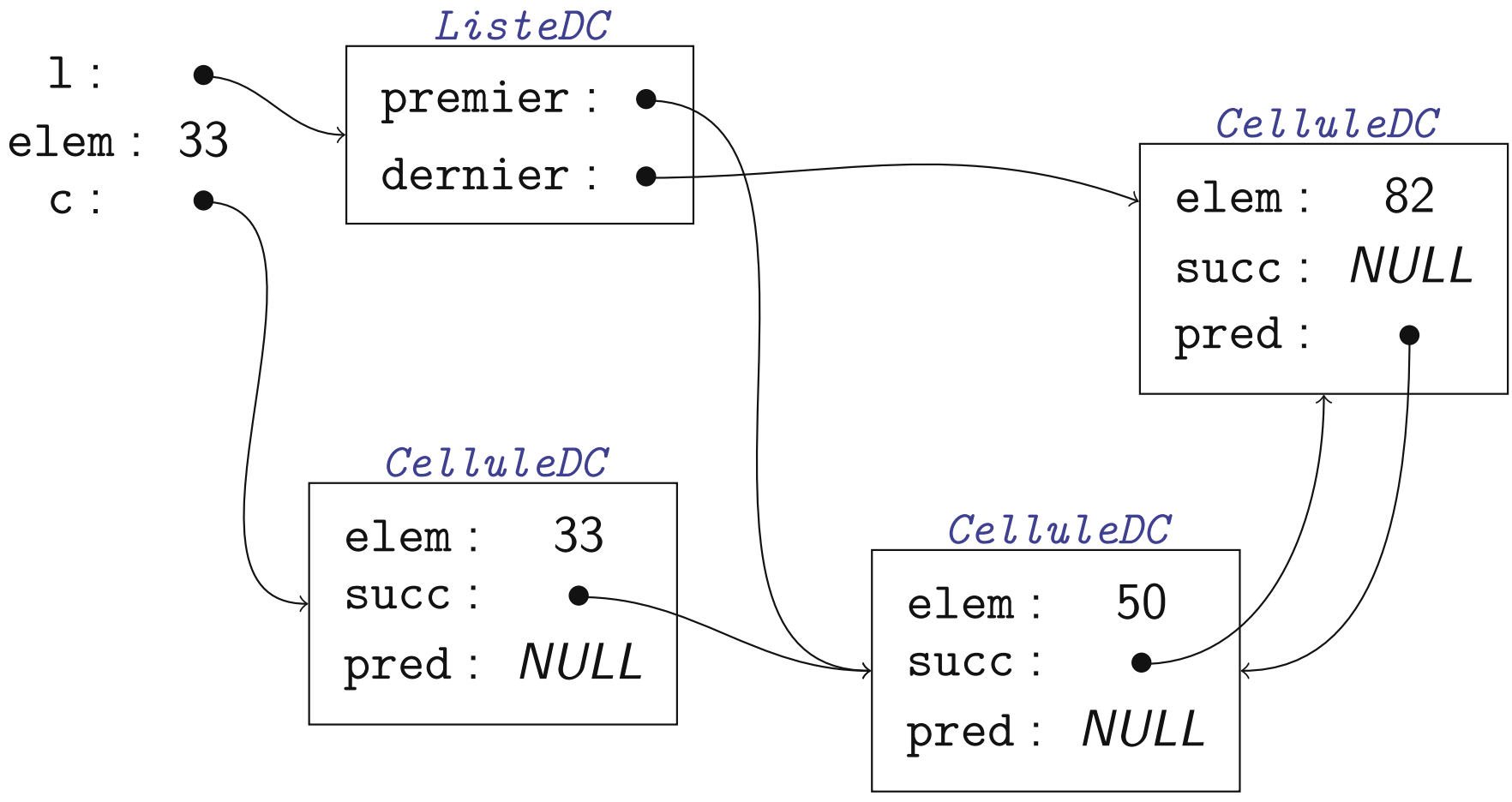
Implémentation des listes doublement chaînées

```
c->succ = l->premier;
```



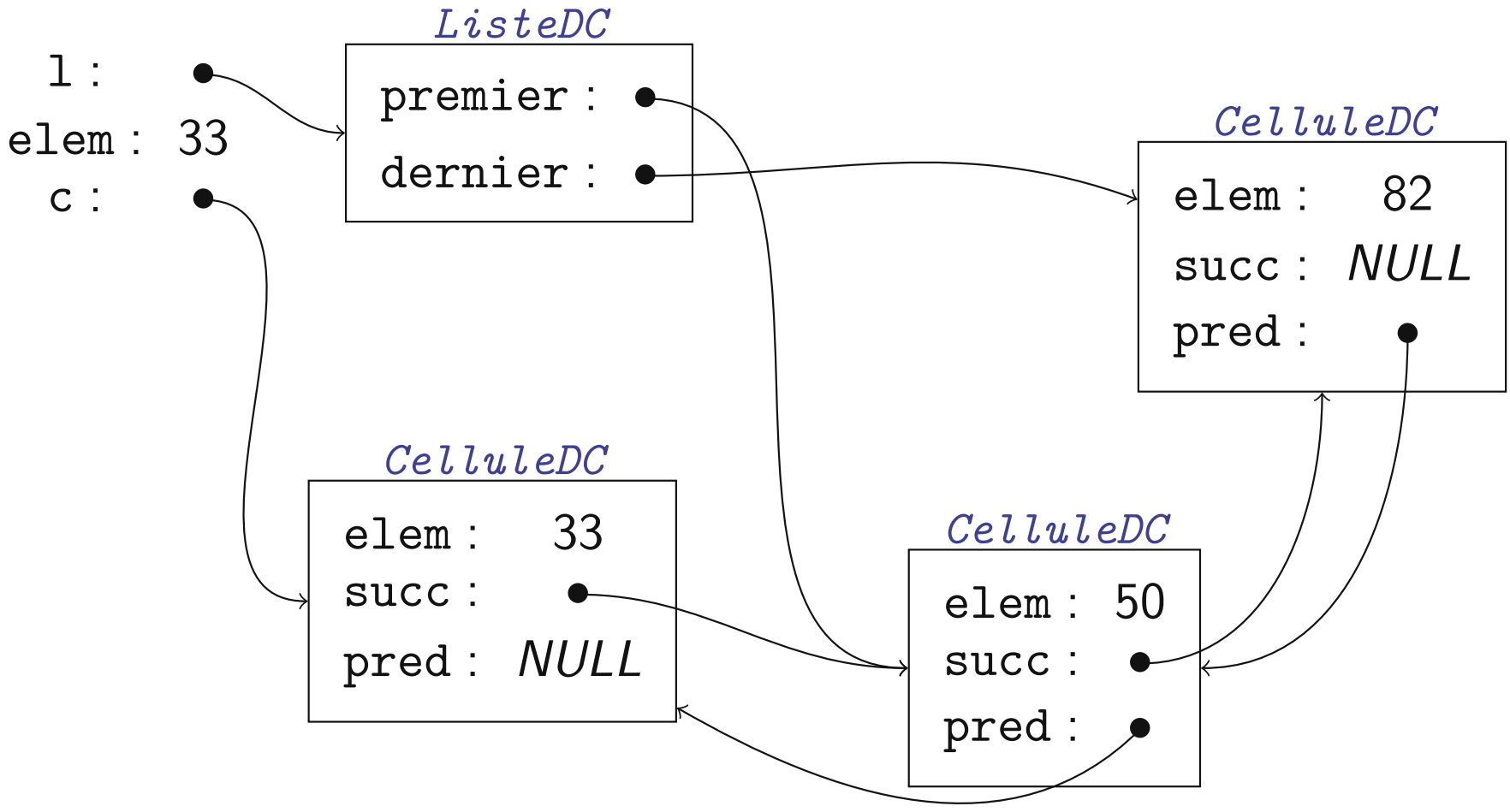
Implémentation des listes doublement chaînées

```
c->pred = NULL;
```



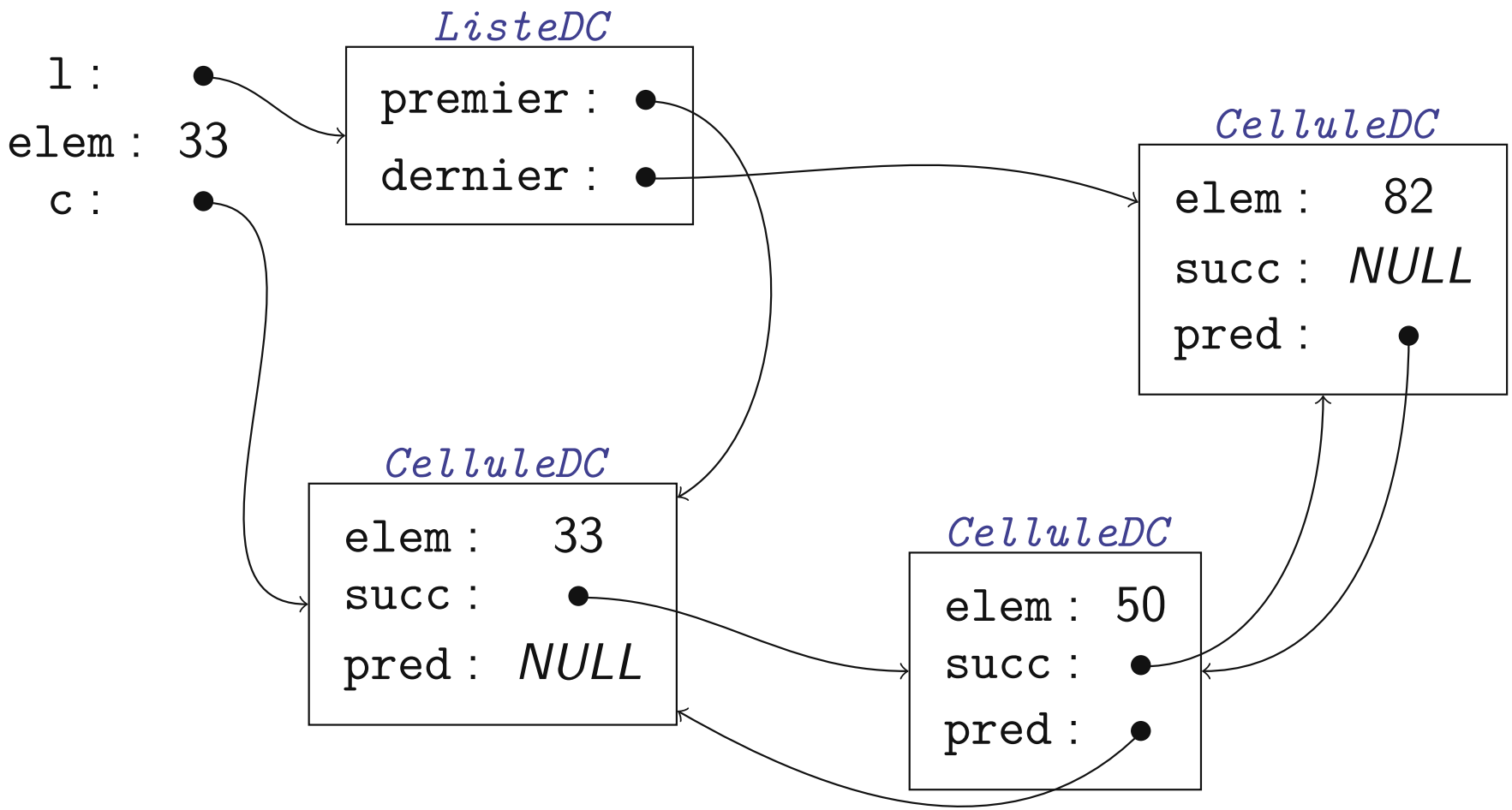
Implémentation des listes doublement chaînées

```
if (l->premier != NULL) l->premier->pred = c;
```



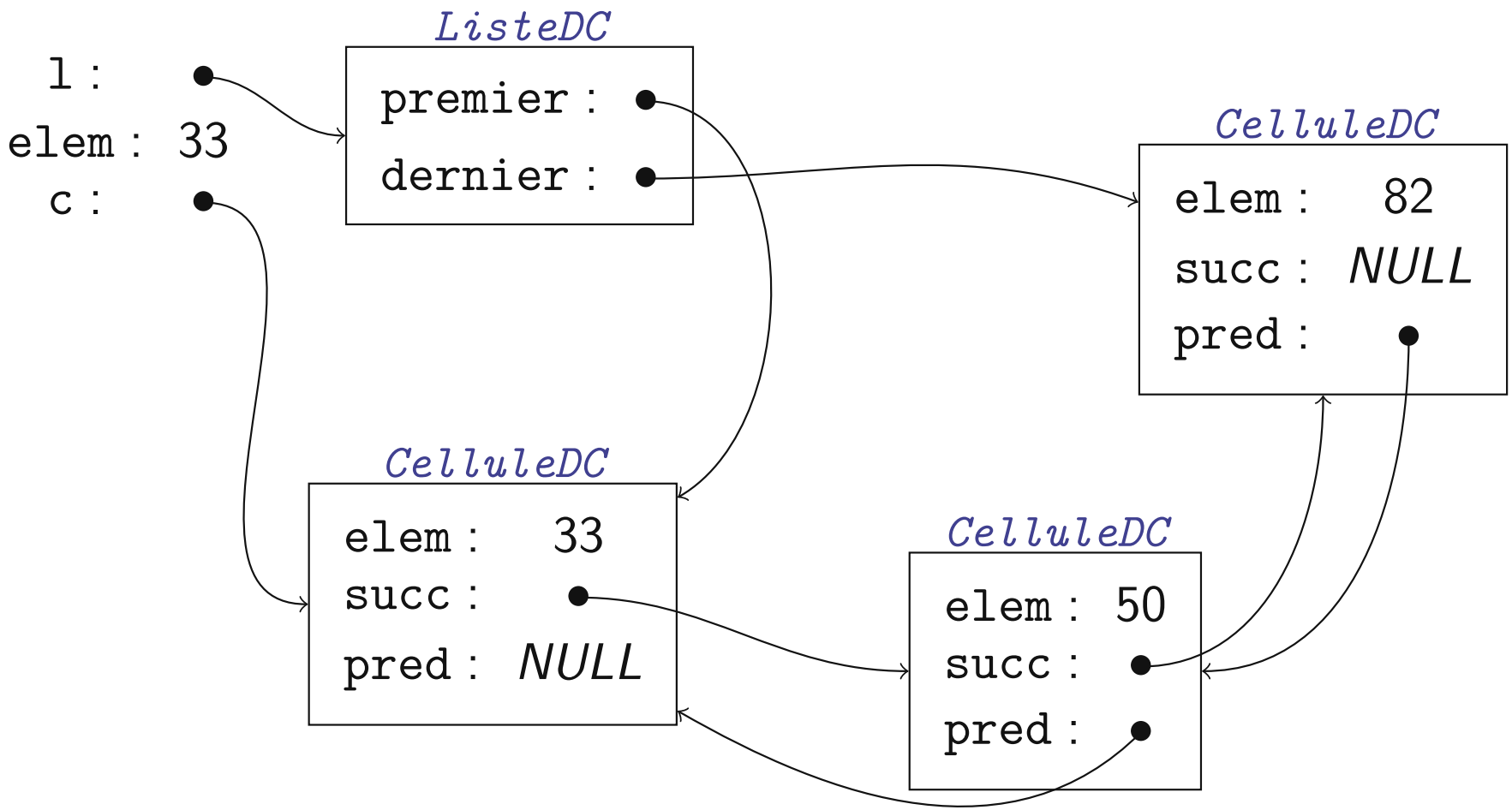
Implémentation des listes doublement chaînées

```
l->premier = c;
```



Implémentation des listes doublement chaînées

```
if (l->dernier == NULL) l->dernier = c;
```



Implémentation des listes doublement chaînées

Insertion d'un élément en tête de liste D.C.

```
void push(Element elem, ListeDC* l){
    CelluleDC* c = malloc(sizeof(CelluleDC));
    c->elem = elem;
    c->succ = l->premier;
    c->pred = NULL;
    if (l->premier != NULL) l->premier->pred = c;
    l->premier = c;
    if (l->dernier == NULL) l->dernier = c;
}
```

Exercice 5 – Fonctions de File (FIFO)

1. Écrire les fonctions nécessaires pour utiliser le type ListeDC comme une file.
2. Quelles sont leur complexité ?