



DR B. DIOP

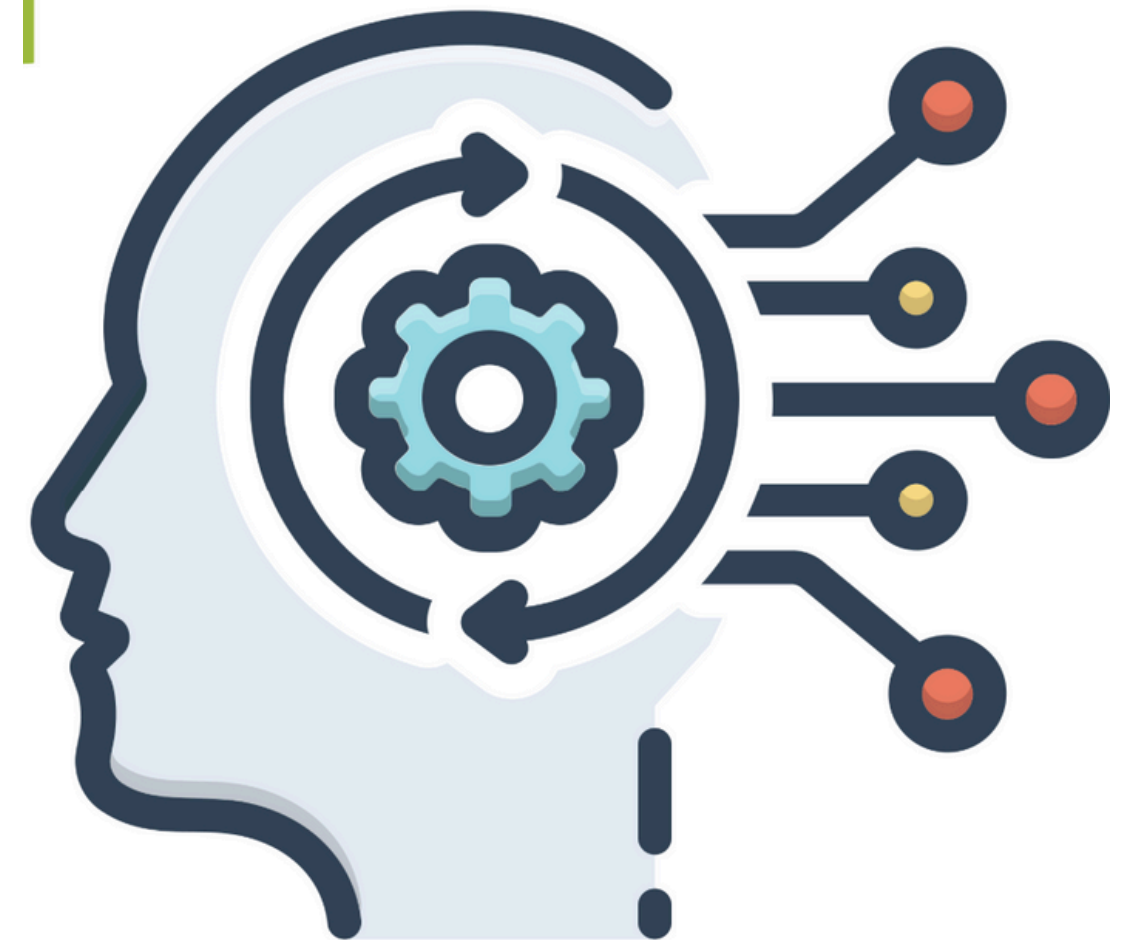
babacar.diop@ussein.edu.sn

Algorithmes et Structures de données

Listes doublement chaînées

INFO – AGROTIC

Last update: oct. 2025



**Qu'est-ce
qu'une liste
doublement
chaînée ?**

La liste doublement
chaînée est une variante de
la liste chaînée dans
laquelle le parcours est
possible dans les deux sens.

Qu'est-ce
qu'une liste
doublement
chaînée ?

Une liste doublement chaînée
comporte une **tête** et une
queue, représentant
respectivement le **premier** et
le **dernier** élément de la liste.

Qu'est-ce qu'une liste doublement chaînée ?

Chaque nœud de la liste contient
trois champs:

- un champ de donnée
 - un pointeur vers le noeud suivant
 - un pointeur vers le noeud précédent
-

Qu'est-ce qu'une liste doublement chaînée ?

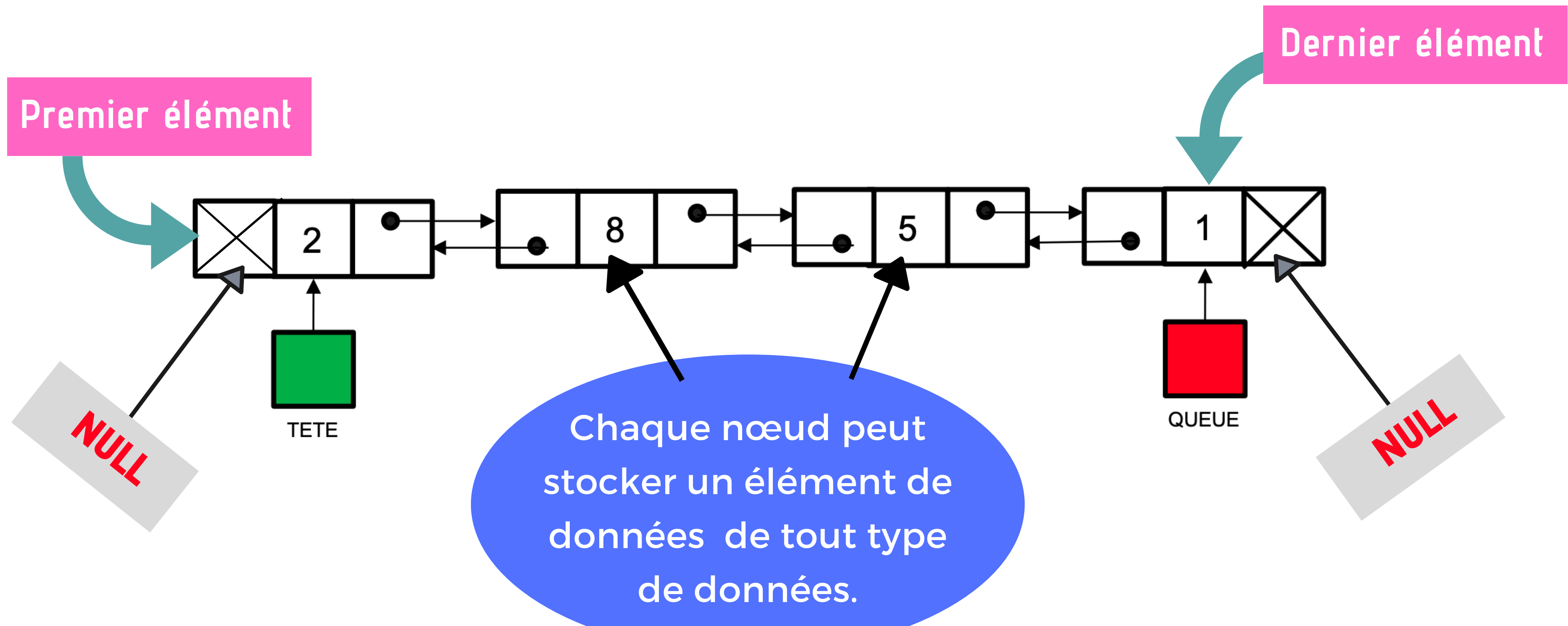
Le pointeur précédent du premier élément doit pointer vers NULL (le début de la liste).
Le pointeur suivant du dernier élément doit pointer vers NULL (la fin de la liste).

Comment définir une liste chaînée en C?

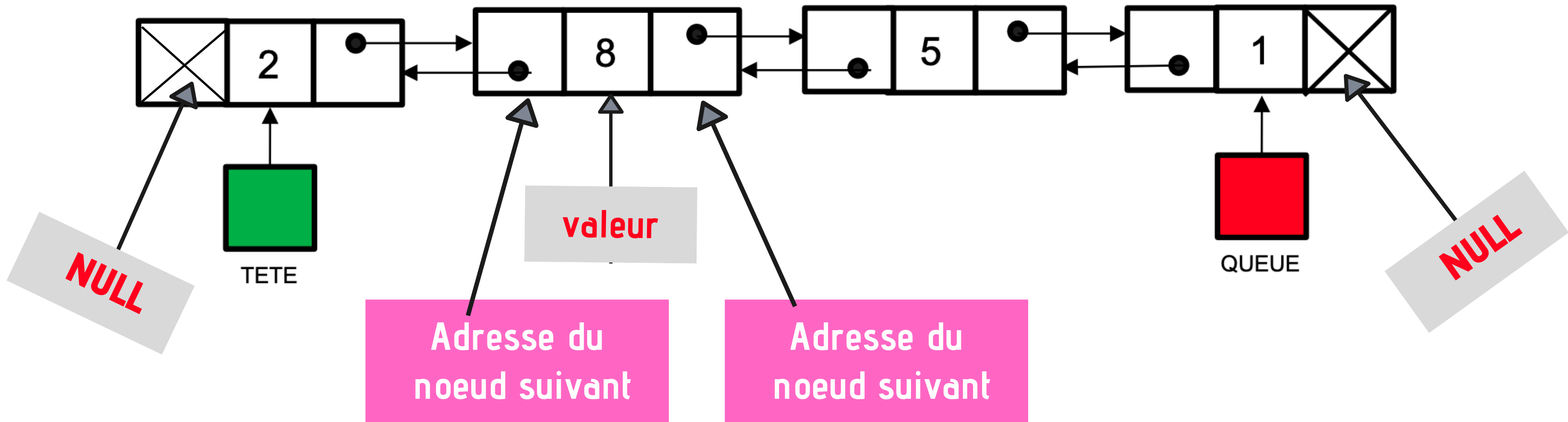
Une liste doublement chaînée peut être parcourue dans les deux sens :

- en commençant avec la tête, et se déplacer avec les pointeurs suivants.
 - en commençant à la queue, et reculer avec les éléments précédents.
-

Exemple



Représentation



**Comment
définir une
liste
doublement
chaînée en C?**

1. Structure des noeuds

```
struct noeud {  
    int valeur;  
    struct noeud *precedent;  
    struct noeud *suivant;  
}  
Typedef struct noeud Noeud;
```

**Comment
définir une
liste
doublement
chaînée en C?**

2. Structure de la liste

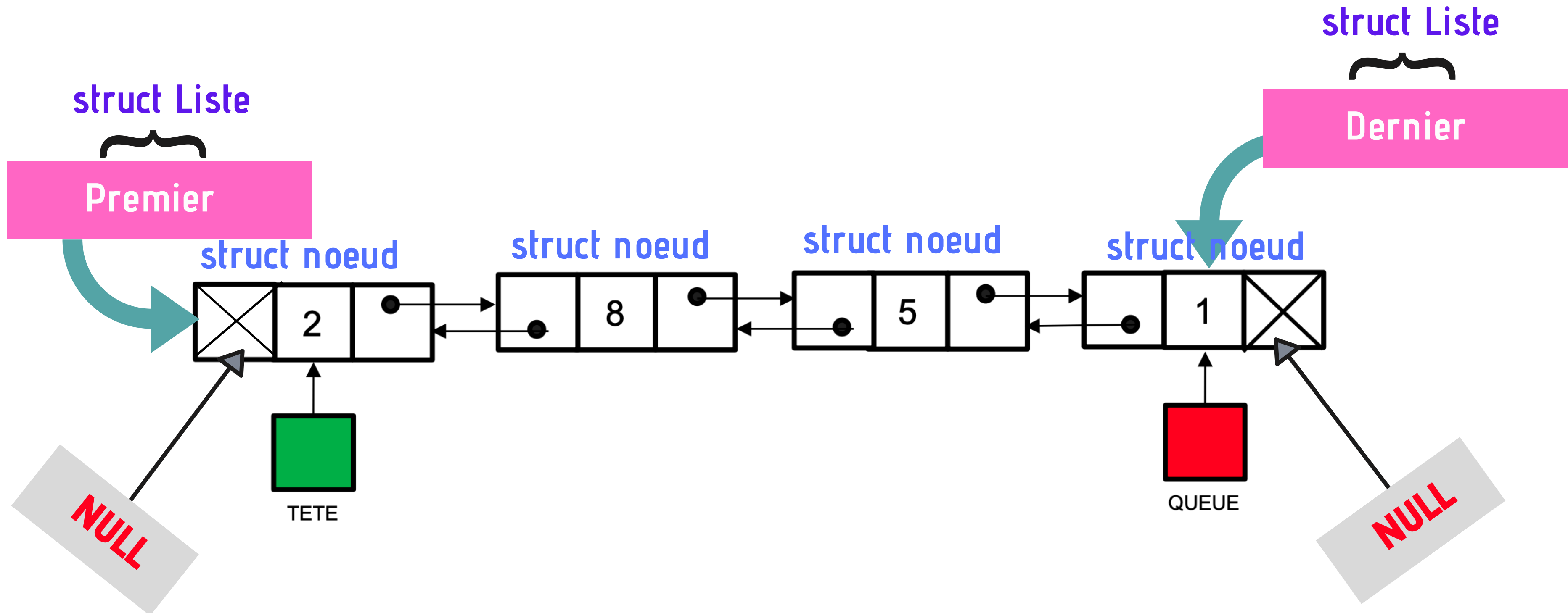
```
struct Liste {  
    Noeud* premier;  
    Noeud* dernier;  
    int nbElements;  
}
```



La structure **Liste** contient un pointeur vers le premier et le dernier élément.

Il faut conserver l'adresse du premier/dernier élément pour savoir où commence/termine la liste. Si on connaît le premier/dernier élément, on peut retrouver tous les autres en sautant/reculant d'élément en élément à l'aide des pointeurs suivant/precedent.

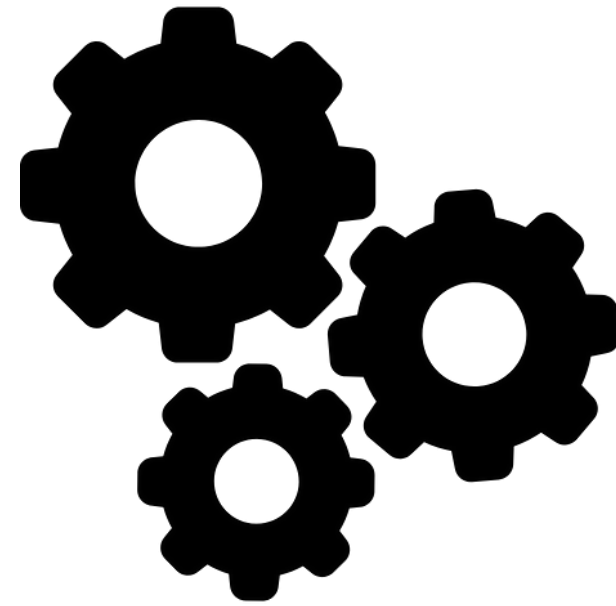
Illustration



Sommaire des éléments de contrôle?

En résumé, un schéma complet de structure de liste chaînée en C comprendra:

- la définition des **noeuds**,
- la définition de la **liste** (pour gérer la tête de liste),
- l'identification du **dernier élément** (dont le suivant pointe sur NULL)



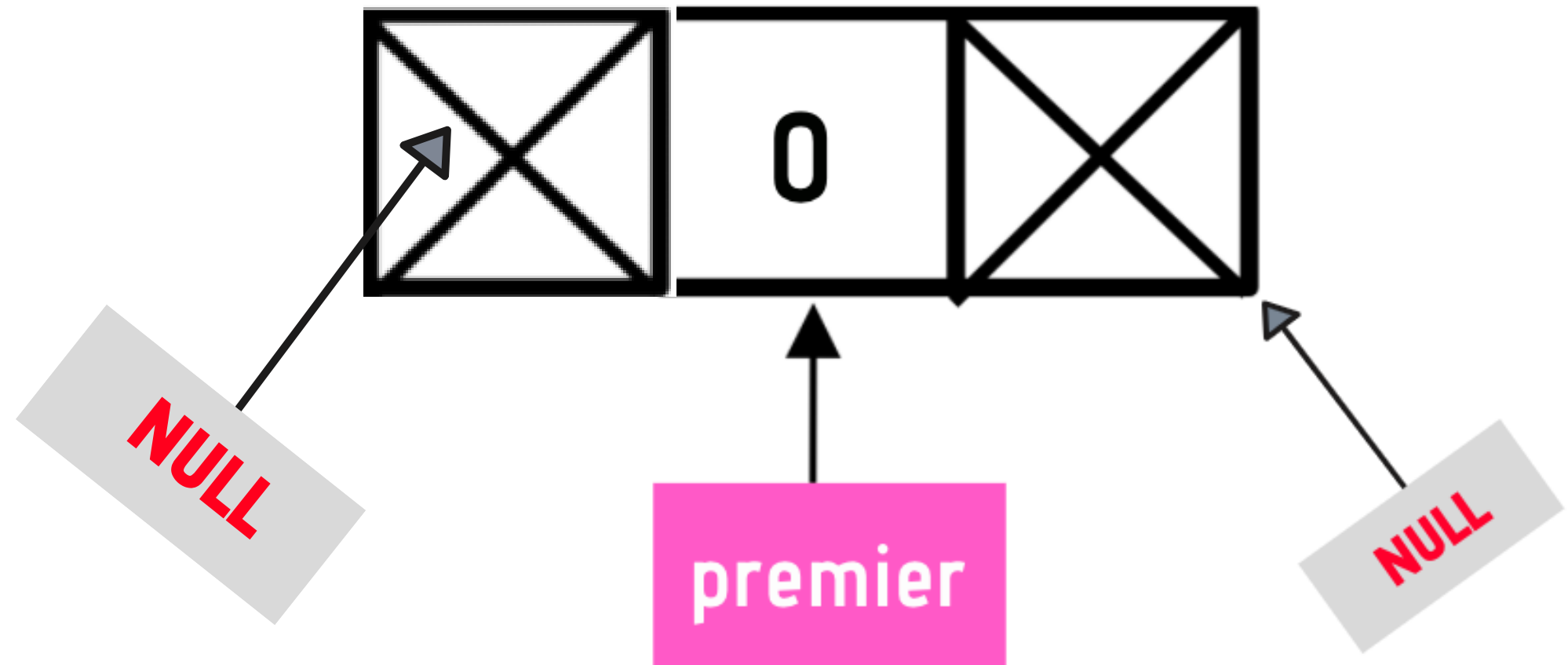
OPÉRATIONS DE BASE

& Fonctions de manipulation

Initialiser une liste

Initialiser une liste

Cette opération doit être faite avant toute autre opération sur la liste. Elle initialise le pointeur **premier** et le pointeur **dernier** avec le pointeur **NULL**, et la taille à 0.

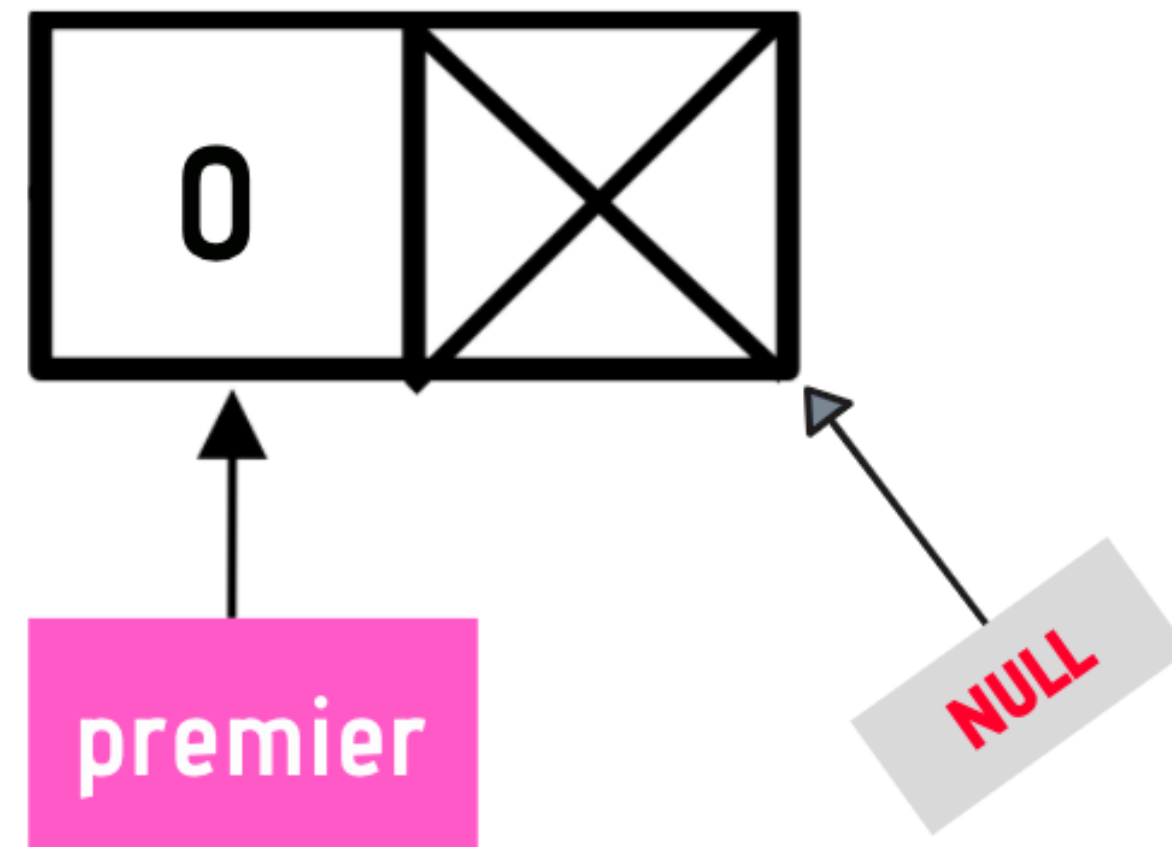


Initialiser une liste

```
void initialisation (Liste *liste)
{
    liste->premier = NULL;
    liste->dernier = NULL;
    liste->nbElements = 0;
}
```

Initialiser une liste

On obtient une liste composée d'un seul élément et ayant une forme semblable à la fig. ci-dessous.



Insérer dans une liste

Insérer dans une liste vide: étapes

- allouer de la mémoire pour le new élément
- remplir le champ de données du new élément
- le pointeur precedent du new élément pointera vers NULL
- le pointeur suivant du new élément pointera vers NULL
- les pointeurs premier et dernier pointeront vers l'élément
- la taille est mise à jour

Insérer dans une liste vide: étapes

```
int insertionDansListeVide (Liste * liste, int valeur){
    Noeud *nvElement = (struct noeud*)malloc(sizeof(struct noeud));
    if (nvElement == NULL) return -1;
    nvElement->valeur = valeur;
    nvElement->precedent = liste->premier;
    nvElement->suivant = liste->dernier;
    liste->premier = nvElement;
    liste->dernier = nvElement;
    liste->nbElement++;
    return 0;
}
```

Insérer en tête de liste: étapes

- allouer de la mémoire pour le nouvel élément
- remplir le champ de données du nouvel élément
- le pointeur précédent du nouvel élément pointe vers NULL
- le pointeur suivant du nouvel élément pointe vers le 1er élément
- le pointeur précédent du 1er élément pointe vers le nouvel élément
- le pointeur premier de Liste pointe vers le nouvel élément
- le pointeur dernier de Liste ne change pas
- la taille est mise à jour

Insérer en tête de liste: étapes

```
int insertionDebutListe (dl_Liste * liste, int valeur){
    Noeud *nvElement = (struct noeud*)malloc(sizeof(struct noeud));
    if (nvElement == NULL) return -1;
    nvElement->valeur=valeur;
    nvElement->precedent = NULL;
    nvElement->suivant = liste->premier;
    liste->premier->precedent = nvElement;
    liste->premier = nvElement;
    liste->nbElement++;
    return 0;
}
```



Attention à l'ordre des étapes !

Supprimer en tête de liste

Supprimer en tête de liste

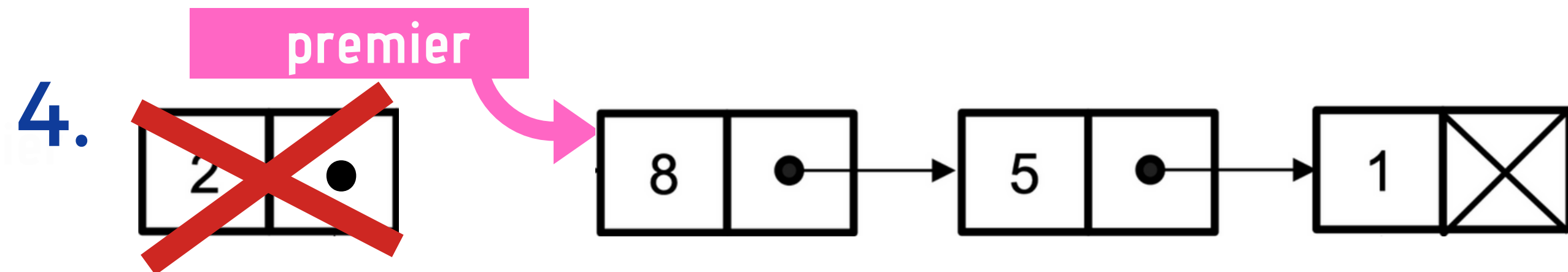
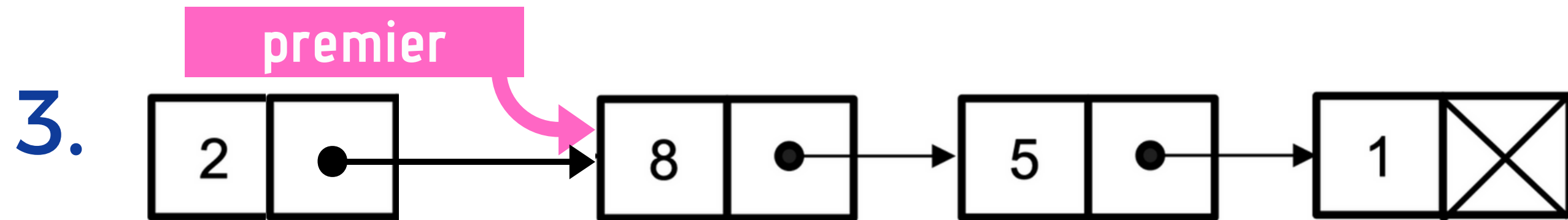
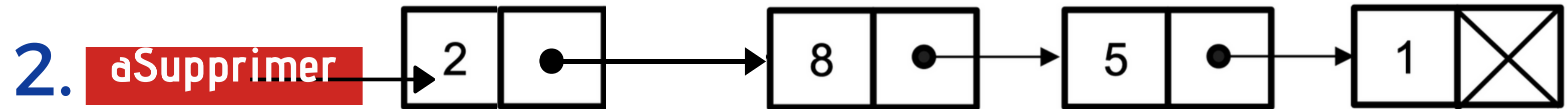
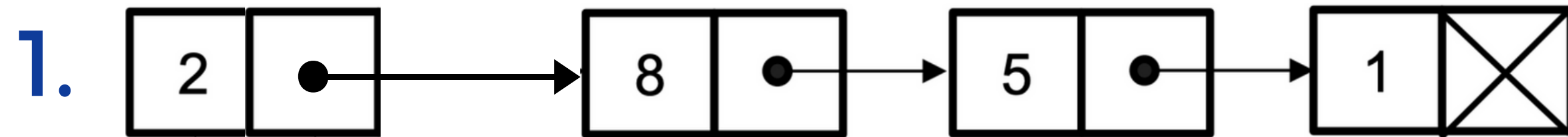
La fonction `suppression()` prend en paramètre la liste (qui contient l'adresse du premier élément).

```
void suppression (Liste *liste)
```

Supprimer en tête de liste

```
void suppression(Liste *liste)
{
    if (liste == NULL)
    {
        exit(EXIT_FAILURE);
    }
    if (liste->premier != NULL)
    {
        Element *aSupprimer = liste->premier;
        liste->premier = liste->premier->suivant;
        free(aSupprimer);
    }
}
```

Exemple



Afficher une liste

Afficher une liste

La fonction `affichage()` prend en paramètre la liste (qui contient l'adresse du premier élément

```
void afficherListe (Liste *liste)
```

Afficher une liste

```
void afficherListe(Liste *liste)
{
    if (liste == NULL)
    {exit(EXIT_FAILURE);}

    Element *actuel = liste->premier;
    while (actuel != NULL)
    {
        printf("%d -> ", actuel->nombre);
        actuel = actuel->suivant;
    }
    printf("NULL\n");
}
```

Exercices d'application

Ecrire un programme avec les fonctions suivantes :

- **listeVide()** pour tester si une liste est vide
- **listTri()** qui trie une liste par ordre croissant
- **renverserListe()** qui renverse une liste