# Présentation

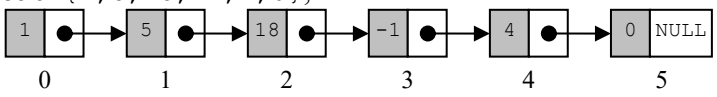
Le but de ce TP est d'utiliser les listes chaînées vues en cours. Les deux bibliothèques `liste_s` (listes simples) et `liste_d` (listes doubles) correspondant aux structures de données et fonctions étudiées en cours seront utilisées dans ce sujet. Vous devez ajouter écrire vos propres fonctions dans le fichier `main.c`.

# 1 Opérations sur une liste simple

## Exercice 1

Écrivez une fonction `int remplis_liste(int tab[], int taille, liste_s *l)` qui reçoit l'adresse d'une liste vide déjà initialisée et la remplit avec les valeurs contenues dans le tableau passé en paramètre. La fonction renvoie $-1$ si une erreur se produit, et 0 sinon.

*exemple :* pour `tab={1,5,18,-1,4,0}`, on obtient la liste suivante :



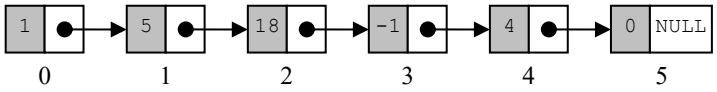

## Exercice 2

Écrivez une fonction `int calcule_longueur(liste_s l)` qui renvoie le nombre d'éléments présents dans la liste `l` passée en paramètre.

## Exercice 3

Écrivez une fonction `int recherche_valeur(liste_s l, int v)` qui recherche une valeur `v` dans une liste simplement chaînée `l` (qui n'est pas ordonnée). La fonction doit renvoyer la position de la première occurrence de la valeur dans la liste, ou $-1$ si la liste ne contient pas d'éléments de valeur `v`.

*exemple :*



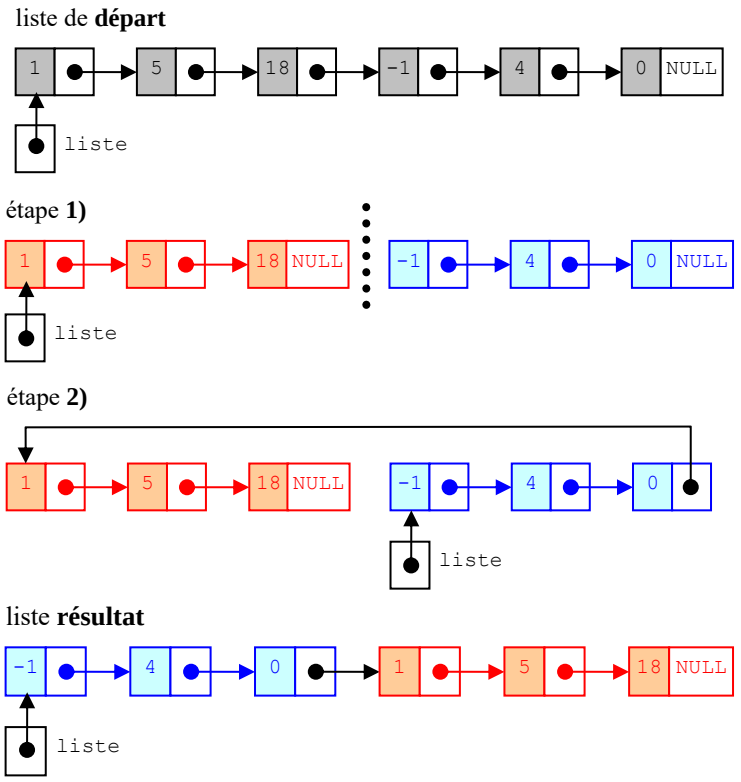

Si on recherche la valeur $-1$ dans la liste ci-dessus, la fonction doit renvoyer 3. Si on recherche la valeur 99, elle doit renvoyer $-1$.

## Exercice 4

Écrivez une fonction `void melange_liste(liste_s *l)` qui prend un pointeur sur une liste simplement chaînée en paramètre, et dont le rôle est de :

1. couper la liste en deux, à la moitié (à un élément près si la taille est impaire)
2. intervertir les deux moitiés.

*exemple :*



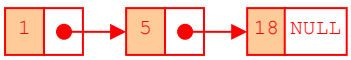

# 2 Opérations sur plusieurs listes simples

## Exercice 5

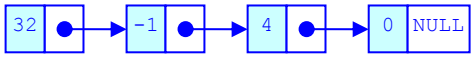
Écrivez une fonction `void fusionne_listes(liste_s *l1, liste_s l2)` qui prend en paramètres un pointeur sur une liste simplement chaînée `l1`, et une liste simplement chaînée `l2`. La fonction doit rajouter la liste `l2` à la fin de la liste `l1`.

*exemple :*

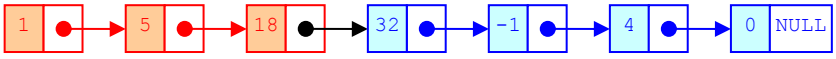
liste **1** avant la fusion




liste **2**




liste **1** après la fusion




## Exercice 6

On dit qu'une liste `l2` préfixe une liste `l1` si `l2` apparaît entièrement au début de `l1`.

*exemples :*

| **l1** | **l2** | **préfixe ?** |
|---|---|---|
| {**1, 2, 4,** 8, 0, 3} | {**1, 2, 4**} | oui |
| {**1,** 2, 4, 8, 0, 3} | {**1**} | oui |
| {**1, 2, 4, 8**} | {**1, 2, 4, 8**} | oui |
| {} | {} | oui |
| {1, 2, 4, 8, 0, 3} | {} | oui |
| {1, 2, 4, 8, 0, 3} | {5, 6} | non |
| {1, 2, **4, 8, 0,** 3} | {**4, 8, 0**} | non |
| {**1, 2, 4, 8, 0, 3**} | {**1, 2, 4, 8, 0, 3,** 7} | non |

Écrivez une fonction récursive `int est_prefixe(liste_s l1, liste_s l2)` qui prend deux listes `l1` et `l2` en paramètres, et renvoie un entier indiquant si `l2` est préfixe de `l1`. L'entier doit valoir 1 pour oui et 0 pour non.

# 3 Opérations sur une liste double

## Exercice 7

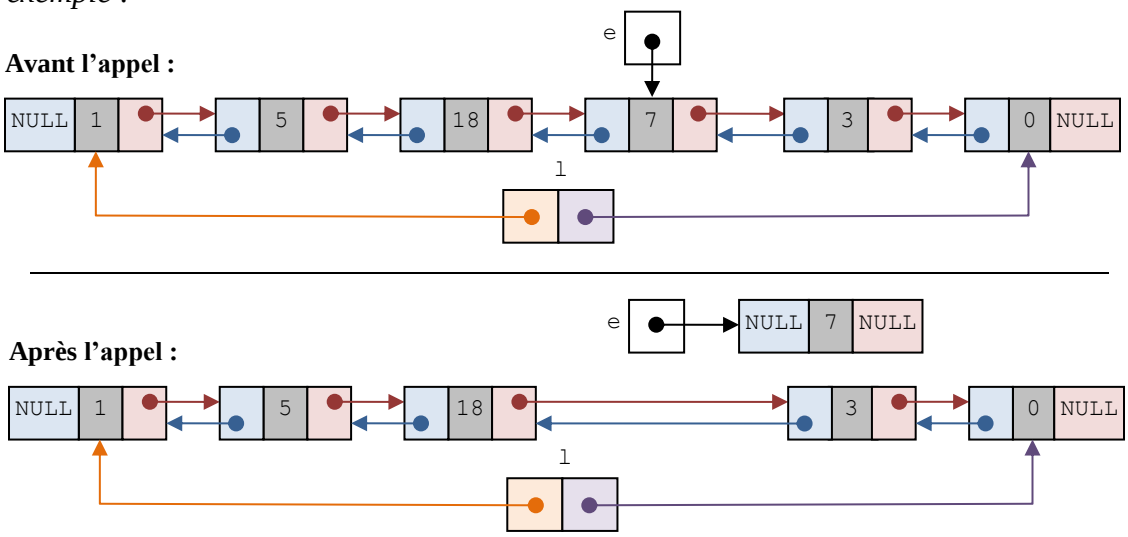
Écrivez une fonction `int echange_elements(liste_d *l, int p, int q)` qui intervertit les éléments d'une liste `l` doublement chaînée situés aux positions `p` et `q`. La fonction renvoie 0 en cas de succès ou $-1$ en cas d'erreur.

*exemple :* si on applique la fonction à la liste `1,25,34,59,2,6` et aux éléments situés aux positions `2` et `4`, on obtient la liste `1,25,2,59,34,6`.

## Exercice 8

Complétez la fonction `int detache_element(liste_d *l, element_d *e)` qui enlève l'élément `e` de la liste `l`. Attention, l'élément ne doit pas être supprimé : il s'agit seulement de modifier les pointeurs des éléments de `l` de manière à ce que `e` n'en fasse plus partie.

*exemple :*




Si la liste `l` ne contient pas l'élément `e`, la fonction doit renvoyer $-1$, et sinon elle doit renvoyer 0. Il est alors recommandé de distinguer les 4 cas suivants :

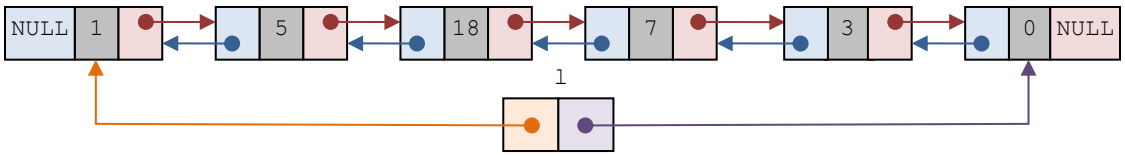
- La liste contient un seul élément (qui est bien sûr `e`) ;
- La liste contient plusieurs éléments et `e` est au début ;
- La liste contient plusieurs éléments et `e` est à la fin ;
- La liste contient plusieurs éléments et `e` n'est ni au début, ni à la fin.

## Exercice 9

Écrivez une fonction `void inverse_liste(liste_d *l)` qui inverse l'ordre des éléments d'une liste `l`. Votre fonction devra utiliser la fonction `detache_element_d`.

*exemple :*

**Avant l'appel :**




**Après l'appel :**