# Examen final de programmation impérative

**ÉNSIIE, semestre 1**

mercredi 18 janvier 2023

Durée : 1h45.
Tout document papier autorisé. Aucun appareil électronique autorisé (sauf dérogation par la scolarité).
Ce sujet comporte 3 exercices indépendants, qui peuvent être traités dans l'ordre voulu. Il contient 5 pages. Le barème est donné à titre indicatif, il est susceptible d'être modifié. Le total est sur 20 points. Il va de soi que toute réponse devra être justifiée, et que toute fonction devra être commentée (au minimum `require`, `assigns` et `ensures`).

## Exercice 1 : Fonctions simples (6 points)

1. Proposer une procédure dont l'effet est de permuter les valeurs de deux variables de type `double` définies auparavant. (Exemple de cas d'utilisation : la fonction est définie *avant* `main`. À l'intérieur de `main`, deux variables sont définies et la fonction appelée avec ces variables. On n'écrira pas la fonction `main`.)
2. Proposer une fonction qui prend en paramètre un tableau de `int` et sa taille et qui retourne la valeur minimum du tableau.
3. Proposer une fonction qui ne prend pas de paramètre et qui retourne un tableau de `char` contenant les codes ASCII des 26 caractères entre A et Z. Il sera judicieux d'utiliser les constantes `'A'` et `'Z'`.
4. Proposer une fonction qui prend en paramètre une chaîne de caractères et qui teste si cette chaîne contient deux fois de suite le même caractère.
5. Proposer une fonction qui prend en paramètre deux chaînes de caractères et qui teste si ces deux chaînes ont des caractères en commun.
6. Proposer une fonction qui prend en paramètre une liste chaînée sur des `double`, et qui retourne la moyenne des valeurs contenues dans la liste.

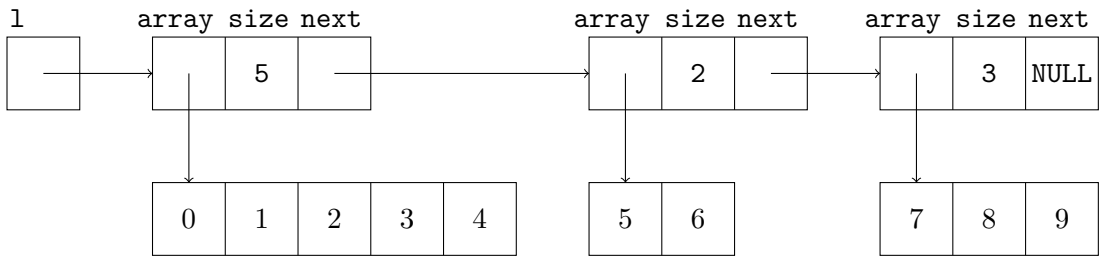
## Exercice 2 : Cordes linéaires (9 points)

Insérer un tableau au milieu d'un autre (par exemple [1, 2, 3] dans [4, 5, 6, 7] à la position 2 pour donner le tableau [4, 5, 1, 2, 3, 6, 7]) nécessite de créer un nouveau tableau dont la taille est la somme des deux, puis de recopier les éléments aux bons endroits. Pour améliorer la complexité de cette opération, nous allons plutôt utiliser une autre structure de données : les cordes linéaires [1].

1. Normalement, les cordes sont une structure formée avec un arbre binaire, mais nous considérons ici une version restreinte pour avoir des structures semblables à celles vues en cours.

Une corde linéaire est une liste chaînée dans laquelle chaque maillon contient un tableau (ici d'entiers) avec sa taille. Une corde linéaire est une représentation du tableau formé par la concaténation des tableaux de ses maillons successifs. On supposera l'invariant de type suivant : une corde linéaire contiendra toujours au moins un maillon, et dans chaque maillon la taille du tableau sera strictement positive. Il ne sera donc pas possible de représenter le tableau vide avec une corde linéaire.

Par exemple, la corde linéaire `l` suivante



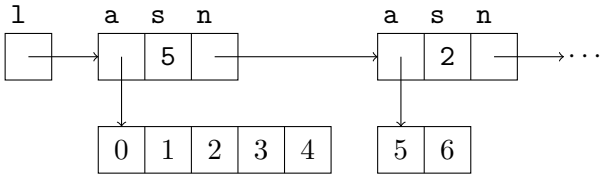

est une corde (parmi d'autres) qui permet de représenter le tableau qui contient les entiers de 0 à 9 dans l'ordre.

1. Définir le type `lcord` des cordes linéaires. Puisqu'il s'agit de listes chaînées dont les maillons contiennent des tableaux avec leur taille, on sera amené, comme vu en cours, à définir un type enregistrement pour les maillons.
2. Écrire une procédure `print_lcord` qui prend une corde linéaire en paramètre, et qui affiche sur la sortie standard ses éléments dans l'ordre, séparés par des espaces, avec un `;` après chaque maillon de la liste. Ainsi, la corde linéaire donnée en exemple ci-dessus sera affichée

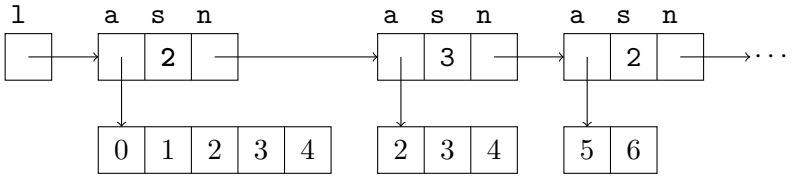
   ```
   0 1 2 3 4 ; 5 6 ; 7 8 9 ;
   ```
3. Écrire une fonction `create_from_array` qui prend en paramètre un tableau d'entiers avec sa taille, et qui retourne une corde linéaire composée d'un seul maillon contenant **une copie** du tableau avec sa taille.
4. Écrire une procédure `split_cell` qui prend en paramètres une corde linéaire dont le tableau du premier maillon est de taille $s$, et un entier $i$, tels que $0 < i < s - 1$ ; cette procédure va couper le premier maillon en deux maillons, le premier contenant le sous-tableau de 0 à $i - 1$, et le second celui de $i$ à $s - 1$.

   On fera une allocation et une copie pour le tableau du second maillon, mais on gardera le même tableau pour le premier maillon en se contentant de changer la taille indiquée dans le maillon.

   Ainsi, avec la corde linéaire suivante

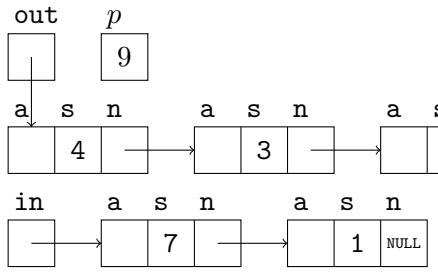
   et la position 2, la corde se transformera en



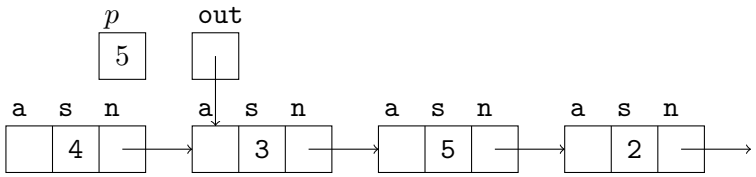

5. Écrire une procédure `insert` qui prend en paramètres deux cordes linéaires `out` et `in`, et un entier $p > 0$ ; cette procédure insère la deuxième corde dans la première à la position $p$.

   L'idée est la suivante : tant que la taille du tableau de `out` est strictement plus petite que la position où insérer, on avance dans `out` en retrachant la taille du tableau de l'ancien premier maillon à $p$. Ensuite, si la taille du tableau du premier maillon est strictement plus petite que $p$, on coupe le premier maillon en deux à la position $p$. Dans tous les cas, la taille du tableau dans le premier maillon est alors $p$. On insère donc la liste chaînée `in` entre le premier maillon et son successeur.

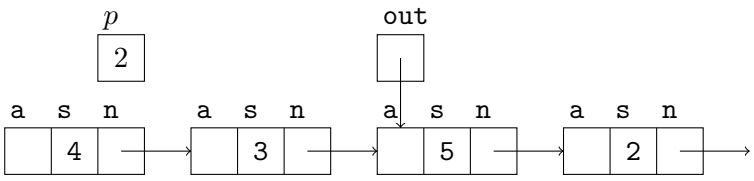
   Par exemple, sans représenter le contenu des tableaux, si on a initialement



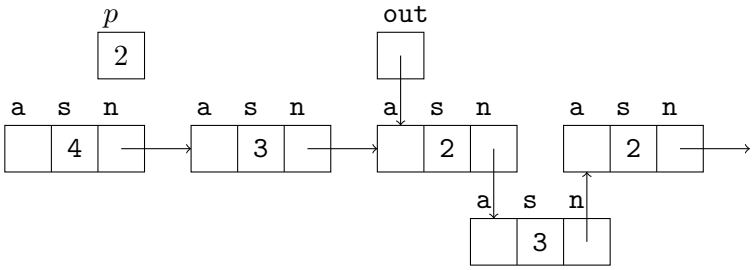

on avance dans `out`



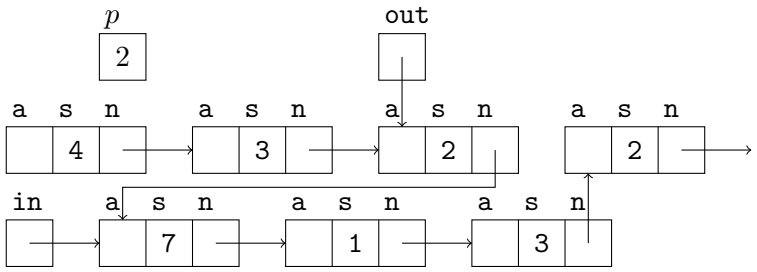

puis



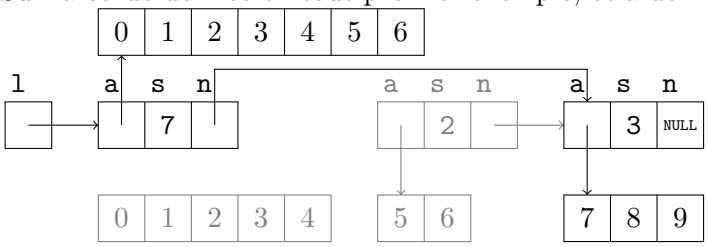

on coupe



et on insère `in` :




6. Écrire une procédure `merge_two_first` qui prend en paramètre une corde linéaire et qui fusionne les deux premiers maillons en un seul ; sans effet s'il n'y a qu'un seul maillon. On allouera un nouveau tableau qui contiendra la copie des deux tableaux, on utilisera ce nouveau tableau dans le premier maillon et on supprimera le deuxième maillon. On fera attention à ce qu'il n'y ait aucune fuite mémoire.
   Sur la corde donnée en tout premier exemple, cela donne :




7. Écrire une procédure `flatten` qui prend en paramètre une corde linéaire et qui la transforme jusqu'à ce qu'elle n'ait plus qu'un seul maillon. On itérera la procédure `merge_two_first`.
8. Écrire une procédure `free_lcord` qui prend en paramètre une corde linéaire et qui libère toute la mémoire allouée dans cette corde.

## Exercice 3 : Jeu des 5 erreurs (5 points)

Les procédures et fonctions suivantes ne sont pas correctes par rapport à la spécification donnée. Les corriger en expliquant pourquoi il y a une erreur. Le cas échéant, on suppose que les bons `#include` ont été écrits.

1.
```
/*@requires nothing
  @assigns nothing
  @ensures increments x by 1 */
void incr(int x) {
  x += 1;
}
```

2.
```
/*@requires n >= 0 and m >= 0
  @assigns nothing
  @ensures return a newly allocated matrix of dimension n * m */
```

```
int **create_matrix(int n, int m) {
  int **r = malloc(n * sizeof (int));
  for (int i = 0; i < n; i += 1)
    r[i] = malloc(m * sizeof (int));
  return r;
}
```

3. 
```
/*@requires t has dimension n * m
  @assigns nothing
  @ensures prints the content of t on the standard output */
void print_matrix(int **t, int n, int m) {
  for (int i = 0; i < n; i += 1) {
    for (int j = 0; j < m; i += 1)
      printf("%4d␣", t[i][j]);
    printf("\n");
  }
}
```

4. 
```
/*@requires n > 0
  @assigns nothing
  @ensures return the sum for i from 1 to n of the inverse of i */
double serie(int n) {
  double s = 0;
  for (int i = 1; i <= n; i += 1)
    s += 1 / i;
  return s;
}
```

5. On considère la définition des listes chaînées vue en cours.

```
/*@requires nothing
  @assigns all cells of l
  @ensures free all cells of l */
void free_list(list l) {
  if (l == NULL);
    return;
  list next = l->next;
  free(l);
  free_list(next);
}
```