# TD n° 1

Paradigmes et interprétation
Licence Informatique
Université Côte d'Azur

## Arbres

Le type `(Arbre 'a)` défini ci-dessous permet de représenter un arbre d'arité quelconque dont les nœuds sont étiquetés par des éléments de type `'a`.

```
(define-type (Arbre 'a)
  [noeud (valeur : 'a) (fils : (Listof (Arbre 'a)))])
```

1. Définissez une fonction `arbre-map` telle que `(arbre-map f a)` renvoie l'arbre obtenu à partir de `a` en appliquant la fonction `f` à toutes ses étiquettes.
2. On définit maintenant le type `(ArbreBinaire 'a)` par

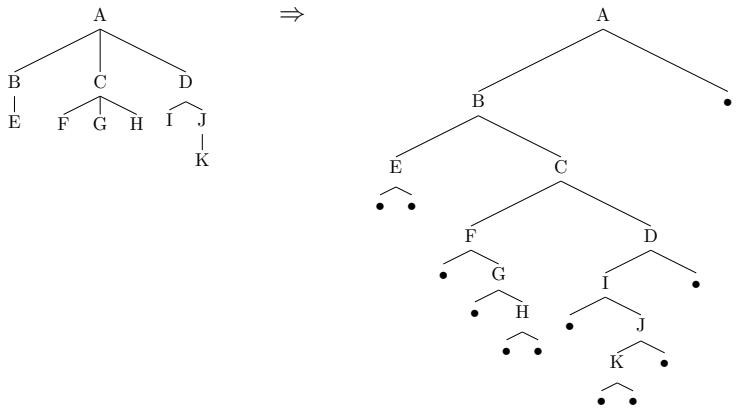
   ```
   (define-type (ArbreBinaire 'a)
     [noeud-binaire (valeur : 'a)
                    (fg : (ArbreBinaire 'a))
                    (fd : (ArbreBinaire 'a))]
     [feuille])
   ```

   On peut représenter un arbre quelconque par un arbre binaire. Tout nœud de l'arbre initial est transformé en nœud binaire de sorte que :

   — le fils gauche est la représentation du premier fils du nœud initial (s'il en a un),

   — le fils gauche est la représentation du prochain frère du nœud initial (s'il en a un).

   Exemple :



Définissez la fonction `arbre->binaire` qui réalise cette transformation.

3. Définissez la fonction `binaire->arbre` qui réalise la transformation inverse. Dans quel cas celle-ci peut échouer ?
4. On se donne le type `Atome` suivant.

```
(define-type Atome
  [symbole (symb : Symbol)]
  [nombre (n : Number)]
  [chaine (str : String)]
  [booleen (bool : Boolean)])
```

   Définissez une fonction `est-atome?` qui indique si une s-expression est un atome.

5. Définissez une fonction `s-exp->atome` qui transforme une s-expression en un `Atome`. La fonction renvoie une erreur si ce n'est pas possible.
6. Une s-expression a naturellement une structure d'arbre (en terme de listes imbriquées). Définissez une fonction `s-exp->arbre` qui prend en paramètre une s-expression et renvoie un `(Arbre Atome)` qui la représente. L'étiquette d'un nœud est le premier élément de la s-expression associée, les fils sont les autres éléments.
7. Que se passe-t-il pour la s-expression `` `((A) B C) `` ? Modifiez la fonction précédente pour qu'elle renvoie un `(Arbre (Optionof Atome))`. Si le premier élément n'est pas un atome, on suppose que le nœud associé n'a pas d'étiquette et que tous ses éléments sont des fils.

## Code Morse

Le code Morse est un codage qui permet de faire passer des messages à l'aide d'impulsions (électriques, sonores ou bien visuelles). Chaque lettre est codée par une série d'impulsions plus ou moins longues. On en distingue trois sortes : les impulsions courtes, les impulsions longues, l'absence d'impulsion. On les représente par le type `Impulsion`.

```
(define-type Impulsion
  [courte]
  [longue]
  [pause])
```

Le codage des caractères alphabétiques majuscules est donné par une liste associative nommée `dict` de type `(Listof (Char * (Listof Impulsion)))` [1].

1. Définissez la fonction `letter->morse` qui renvoie la liste d'impulsions associée à un caractère.
2. Définissez la fonction `code-impulsions` qui renvoie la liste d'impulsions associée à une chaîne de caractères : il s'agit de la concaténation des codages de chaque caractère de la chaîne séparés par une absence d'impulsion. Utilisez la fonction `string->list` qui transforme une chaîne en la liste de ses caractères.
3. Définissez la fonction `impulsions->string` qui prend en paramètre une liste d'impulsions quelconque et renvoie une chaîne équivalente plus facilement lisible. Les impulsions courtes sont représentées par un point (.), les longues par un tiret (-) et les pauses par une espace ( ). Utilisez la fonction `list->string` qui renvoie une chaîne à partir de la liste de ses caractères.

1. La notation `#\c` est la représentation du caractère `c` en Racket

4. Déduisez-en la fonction `code` qui transforme une chaîne de caractères en celle qui représente son codage.

```
(test (code "CODE MORSE") "-.-. --- -.. .   -- --- .-. ... .")
```

5. Définissez la fonction `string->impulsions` qui est la fonction réciproque de `impulsions->string`.
6. En une ligne, inversez les couples clef-valeur de la liste `dict` pour créer la liste associative `dict-inverse`.
7. Définissez la fonction `morse->letter` qui renvoie le caractère associé à une liste d'impulsions. Une erreur est lancée si la liste n'est pas un codage valide.
8. Définissez la fonction `detecter-lettres` qui découpe une liste d'impulsions pour en séparer les différents codages de lettres. Son type est donc

```
((Listof Impulsion) -> (Listof (Listof Impulsion)))
```

   Il s'agit principalement de répérer les pauses qui séparent les lettres. Prenez garde au fait que trois pauses successives correpondent au codage d'un espace. On suppose la liste d'entrée valide.

9. Déduisez-en la fonction `decode` qui est la réciproque de la fonction `code`.

```
(test (decode "-.-. --- -.. .   -- --- .-. ... .") "CODE MORSE")
```