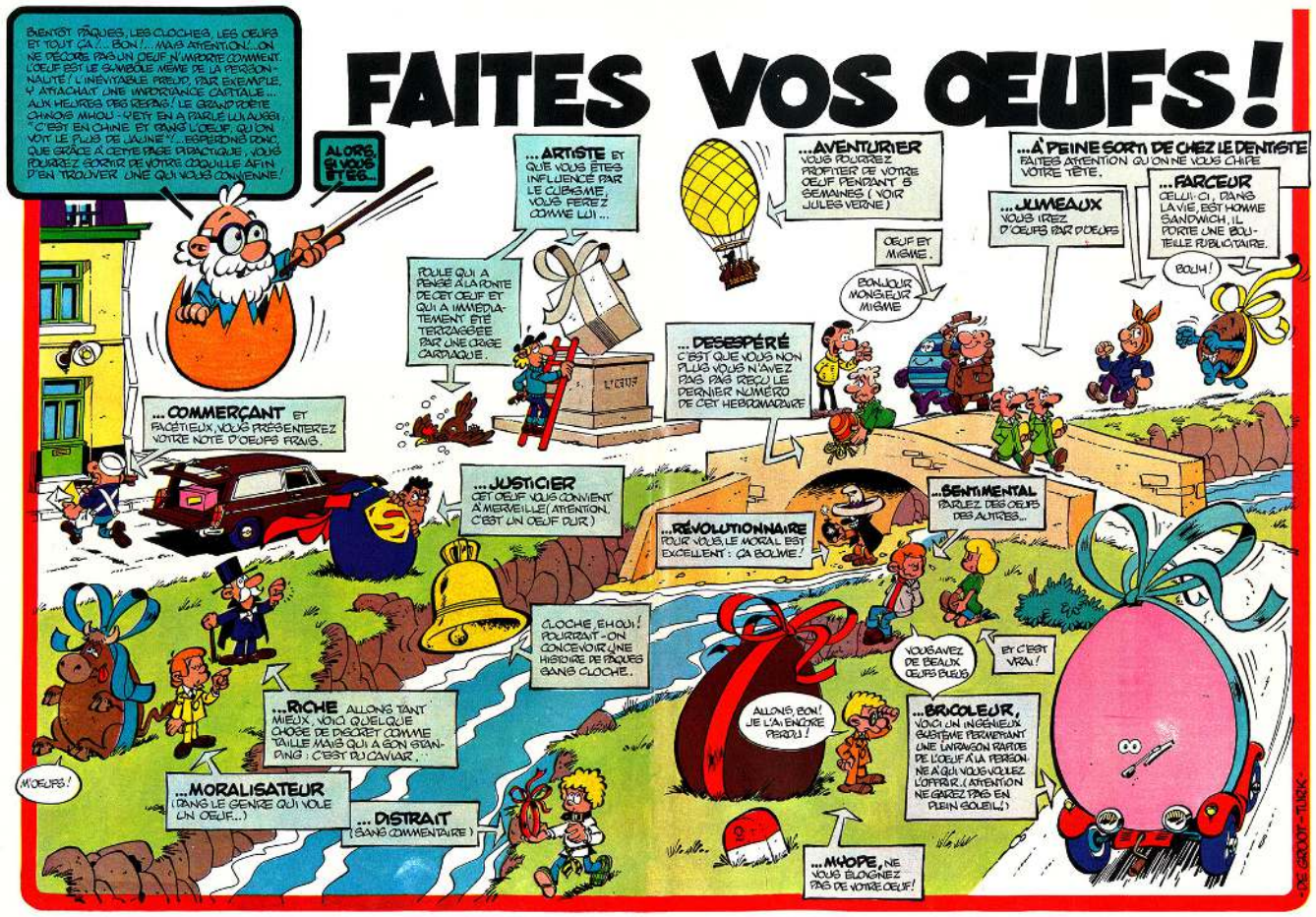
Algorithmique
Contrôle n° 1

Info-sup S1#
Epita

23 avril 2019 - 13 :30

Remarques (à lire !) :

- ☐ Vous devez répondre sur **les feuilles de réponses prévues à cet effet**. Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
  Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
- ☐ La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) peuvent être retirés de cette note.
- ☐ Tout code Caml non indenté ne sera pas corrigé.
- ☐ Tout code Caml doit être suivi de son évaluation : la réponse de Caml .
- ☐ En l'absence d'indication dans l'énoncé, les seules fonctions que vous pouvez utiliser sont `failwith` et `invalid_arg` (aucune autre fonction prédéfinie de Caml).
- ☐ Aucune réponse au crayon de papier ne sera corrigée.
- ☐ Durée : 2h00 (May the force. . .)



Exercice 1 (Types Abstraits : Vecteur (erreurs et extension) – *6 points*)

Supposons le type abstrait algébrique *Vecteur* vu en cours rappelé ci-dessous.

TYPES

vecteur

UTILISE

entier, élément, booléen

OPÉRATIONS

vect	:	entier × entier → vecteur
modifième	:	vecteur × entier × élément → vecteur
ième	:	vecteur × entier → élément
estinitialisé	:	vecteur × entier → booléen
borneinf	:	vecteur → entier
bornesup	:	vecteur → entier

PRÉCONDITIONS

ième(v,i) est-défini-ssi borneinf(v) $\leqslant$ i $\leqslant$ bornesup(v) & estinitialisé(v,i)=vrai

AXIOMES

borneinf(v) $\leqslant$ i $\leqslant$ bornesup(v) $\Rightarrow$ ième(modifième(v,i,e),i) = e
borneinf(v) $\leqslant$ i $\leqslant$ bornesup(v) & borneinf(v) $\leqslant$ j $\leqslant$ bornesup(v) & i$\neq$j
$\Rightarrow$ ième(modifième(v,i,e),j) = ième(v,j)

borneinf(v) $\leqslant$ i $\leqslant$ bornesup(v) $\Rightarrow$ estinitialisé(modifième(v,i,e),i)=vrai
borneinf(v) $\leqslant$ i $\leqslant$ bornesup(v) & borneinf(v) $\leqslant$ j $\leqslant$ bornesup(v)
$\Rightarrow$ estinitialisé(modifième(v,i,e),j)=estinitialisé(v, j)

borneinf(vect(i,j))=i
borneinf(v) $\leqslant$ i $\leqslant$ bornesup(v) $\Rightarrow$ borneinf(modifième(v,i,e))=borneinf(v)

bornesup(vect(i,j))=j
borneinf(v) $\leqslant$ i $\leqslant$ bornesup(v) $\Rightarrow$ bornesup(modifième(v,i,e))=bornesup(v)

AVEC

vecteur v
entier i, j, k
element e

1. Cette définition est incorrecte. En effet l'ensemble des axiomes donné présente deux problèmes. Pour chacun d'eux, vous préciserez sa nature, vous le détaillerez et vous proposerez ensuite une solution pour le régler.

2. Maintenant que les problèmes sont réglés, nous sommes à nouveau en possession du type abstrait algébrique *Vecteur*. On se propose alors de faire une extension à ce type en définissant une nouvelle opération : *réinitialise*. Celle-ci nous permettra de remettre un emplacement du vecteur en état initial (*à savoir : non initialisé*). Son profil est le suivant :

   OPÉRATIONS

   réinitialise : vecteur × entier → vecteur

   (a) Précisez l'éventuel domaine de définition de cette opération (les préconditions).

   (b) Donner les axiomes définissant de manière suffisamment complète cette opération.

Exercice 2 (Tri par insertion – *7 points*)

1. Écrire la fonction `insert` qui ajoute un élément à sa place dans une liste triée selon un ordre donné par une fonction de comparaison.

   Exemples d'applications :

   ```
   # insert 12 [1;5;9;13;15;28] (function x -> function y -> x <= y) ;;
    - : int list = [1; 5; 9; 12; 13; 15; 28]
   # insert 12 [28;15;13;9;5;1] (function a -> function b -> a >= b) ;;
    - : int list = [28; 15; 13; 12; 9; 5; 1]
   # insert 12 [] (function a -> function b -> a >= b) ;;
    - : int list = [12]
   ```

2. Utiliser la fonction `insert` pour écrire une fonction qui trie une liste selon un ordre donné par une fonction de comparaison.

   Exemples d'applications :

   ```
   # insertion_sort (function x -> function y -> x >= y) [12;5;47;1;23;0;48;35;3;14;9;11;8;7;65] ;;
    - : int list = [65; 48; 47; 35; 23; 14; 12; 11; 9; 8; 7; 5; 3; 1; 0]

   # let longer s1 s2 = String.length s1 > String.length s2 ;;
    val longer : string -> string -> bool = <fun>
   # insertion_sort longer ["Caml"; "C#"; "Python"; "C"; "Javascript"];;
    - : string list = ["Javascript"; "Python"; "Caml"; "C#"; "C"]
   ```

Exercice 3 (Association – 5 points)

Écrire la fonction `assoc k list` où `list` est une liste de couples (*key*, *value*) triés par clés (*key*) croissantes. Les clés sont des entiers naturels non nuls. Elle retourne la valeur (*value*) associée à la clé (*key*) `k`. Si `k` n'est pas valide ou si aucun couple n'a pour clé `k`, elle déclenche une exception.

Exemples d'applications :

```
# assoc 5 [(1, "one"); (2, "two"); (3, "three"); (5, "five"); (8, "eight")];;
 - : string = "five"

# assoc 4 [(1, "one"); (2, "two"); (3, "three"); (5, "five"); (8, "eight")];;
 Exception: Failure "not found".

# assoc (-1) [(1, "one"); (2, "two"); (3, "three"); (5, "five"); (8, "eight")];;
 Exception: Invalid_argument "k not a natural".
```

Exercice 4 (Mystery – 2 points)

Soit la fonction `mystery` définie ci-dessous

```
# let mystery = function
      [] -> failwith "..."
    | e::f::l -> (let rec aux_mystery m1 m2 = function
                    [] -> m2
                  | e::l -> if e < m1 then aux_mystery e m1 l
                            else if e < m2 then aux_mystery m1 e l
                            else aux_mystery m1 m2 l
                in if e < f then aux_mystery e f l else aux_mystery f e l);;
```

1. Donner les résultats des évaluations successives des phrases données sur les feuilles de réponses.
2. Que retourne la fonction `mystery` ?