Les listes

Mardi 4 Décembre 2012

Option Informatique

Ecole Alsacienne

Plan

1. Avant de nous lancer…
2. Définitions
3. Caractéristiques
4. Les listes en Caml
5. Premières fonctions
6. Exercices

AVANT DE NOUS LANCER

Petits conseils de programmation : Recyclage

Une bonne pratique : le recyclage !

Si votre code fonctionne, autant vous en resservir !

PETITS CONSEILS DE PROGRAMMATION : RECYCLAGE

Une bonne pratique : le recyclage !

Exemple :

- Question 3. Ecrivez une fonction `est_premier`
  - prenant en argument un nombre entier
  - renvoyant un boolén
  - telle que `est_premier p` renvoie `true` si `p` est un nombre premier, et `false` sinon.

- Question 4. Ecrivez une fonction `plus_grand_nombre_premier`
  - prenant en argument un nombre entier (supposé supérieur ou égal à 2)
  - renvoyant un boolén
  - telle que `plus_grand_nombre_premier n` renvoie le plus grand nombre premier inférieur ou égal à `n`.

- Réponse à la question 4 :

```
let rec plus_grand_nombre_premier n =
        if (est_premier n)
        then n
        else plus_grand_nombre_premier (n-1);;
```

Petits conseils de programmation : Booléens

- *Ecrire un booléen*
  - `"true"` est une chaîne de caractères
  - `(true)` et `true` sont des booléens

- *Tester un booléen*
  - Si b est une booléen
  - `(b = true)` est équivalent à `b`
  - `(b = false)` est équivalent à `(not b)`
  - Exemple : les deux lignes suivantes sont équivalentes
    - `if (est_premier n)`
    - `if ((est_premier n) = true)`

Évaluation paresseuse

- Caml est un flemmard : il en fait le moins possible

- Exemples
  - Avec une conjonction ("et")

    ```
    TrucFaux && TrucSuperLongACalculer
    ```
  - Avec une disjonction ("ou")

    ```
    TrucVrai || TrucSuperLongACalculer
    ```

- Conséquence : Attention à l'ordre de vos conditions !

- Exemple : `((!i < longueur) && (v.(!i) = x))`

AVANT D'ENVOYER UN FICHIER...

Un grand classique en informatique : éteindre et redémarrer

- Pour vérifier que votre code fonctionne bien, tuez Caml, relancez-le, et réévaluer tout votre code depuis le début.

- Exemple :

```
let minimum v j =
  let min_courant = ref v.(j) in
  let n = Array.length (v) in
  for i = j to (n-1)
  do
    if (v.(i) < !min_courant)
    then
        min_courant := v.(i);
    done;
  !min_courant;;

minimum [| 3 ; 1 ; 4 |];;
```

A PROPOS DES TD

- Des travaux satisfaisants : continuez comme ça !

- Pour le premier trimestre
  - + 1 à la moyenne si au moins un TD fini rendu

- Pour les autres trimestres
  - +1 à la moyenne par TD fini rendu
  - Dans une limite de 3 par trimestre

DÉFINITIONS

POUR COMMENCER…

Qu'est-ce qu'une liste ?

En quelque sorte… une pile de crêpes !

PLUS FORMELLEMENT

Définition récursive

Une *liste chaînée* l d'éléments de type T est

- soit la liste vide : $l = [\,]$
- soit un élément t de type T suivi d'une liste l_2 d'éléments de type T : $l = t :: l_2$

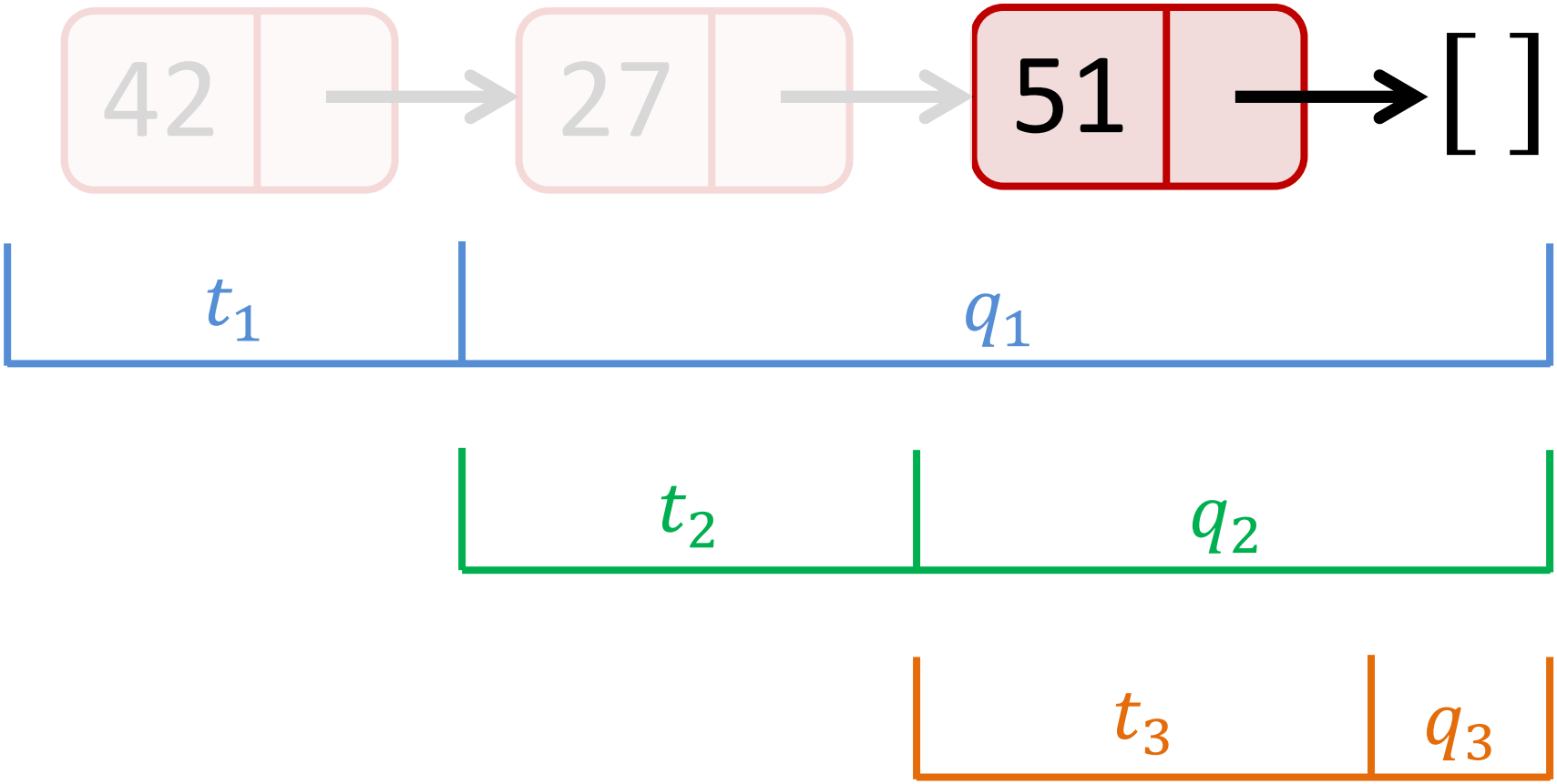
Caractérisation d'une liste non vide

- Si l est une liste non vide $(l \neq [\,])$, alors l s'écrit nécessairement sous la forme $l = t :: q$

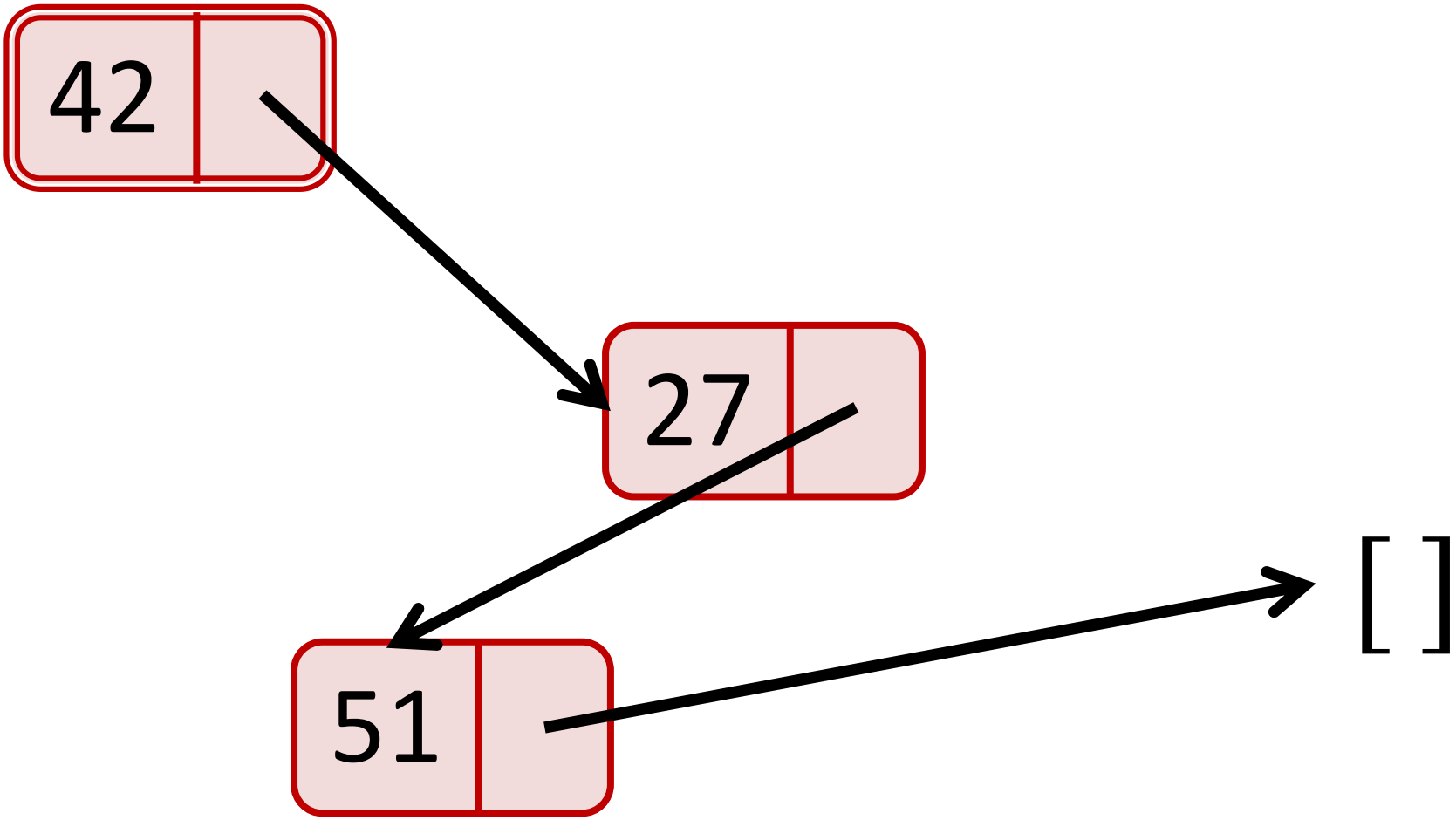
- Le premier élément t est alors appelé la tête de la liste l

- Le reste q est appelé queue de la liste

GRAPHIQUEMENT



Dans la mémoire



Caractéristiques

Mise en situation

- Que pouvez-vous faire face à cette structure ?
- Que ne pouvez-vous pas faire face à cette structure ?

L'ASSIETTE EST-ELLE VIDE ?

- La première chose qu'on vérifie en général avec une liste est sa **vacuité** : cette liste est-elle **vide** ou non ?

- On obtient ainsi un booléen qui permet un traitement au cas par cas :

```
Si EstVide(l)
   Alors
         ...
   Sinon
         ...
Fin Si
```

Ajouter une crèpe

- On a vu qu'on on utilise le **symbole `::`** pour distinguer la tête et la queue d'une liste non vide ($l = t :: q$)
  - ➔ On utilise le même symbole pour ajouter un élément à une liste

- En effet, ajouter un élément `x` à une liste `l` revient à
  1. Créer une nouvelle liste dont **`x` est la tête et `l` la queue**.
  2. Faire de cette nouvelle liste la nouvelle valeur de `l`

Ajouter une crèpe

- La **syntaxe** est donc la suivante :
  - Si `l` est la liste actuelle
  - Si `x` est l'élément à ajouter
  - Alors la nouvelle valeur à affecter à `l` est `l :: x`

- En Caml, comme il s'agit modifier la variable `l`, cette dernière devra être une **référence**.

- Dans certains langages de programmation (mais pas en Caml), ces ajouts se font grâce à des fonctions dédiés (ex : `push_front` en C++)

Accéder à la tête et à la queue

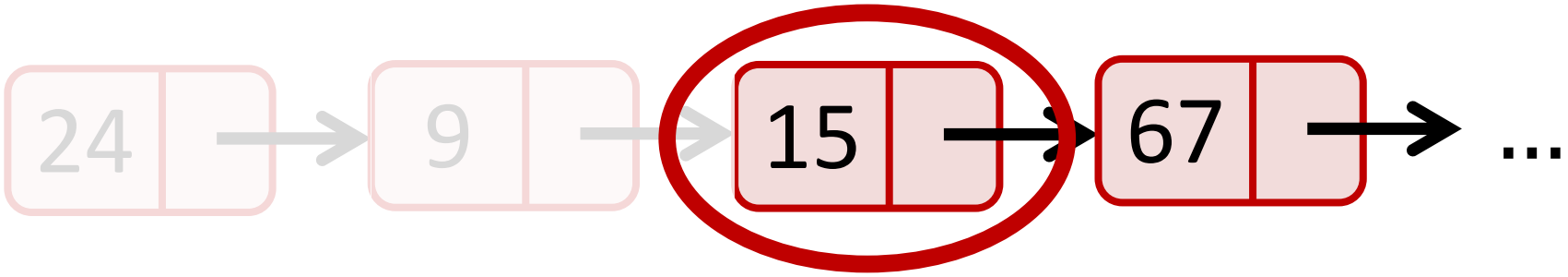
- Si une liste l est non vide, elle est constitué d'une tête t et d'une queue q, ce qu'on note $l = t :: q$

- Des **fonctions explicites** permettent alors d'accéder à ces deux composants :
  - Tête d'une liste non vide : `Tête(l)`
  - Queue d'une liste non vide : `Queue(l)`

Accès au i^{ème} élément d'une liste

- **Question** : *comment accéder au 3^e élément d'une liste ?*

- **Réponse :** on regarde la tête de la queue de la queue de cette liste.



- **Conséquences :** On est obligé de parcourir une liste dans l'ordre, élément par élément

La taille d'une liste

- La taille d'une liste n'est **pas fixée** (ni même bornée)

- Pour connaître la taille d'une liste, il est nécessaire de la **parcourir en entier**

- Cette opération ne peut donc se faire en temps constant. La durée de ce calcul est proportionnelle à la longueur de la liste : on parle de complexité **linéaire**.

Les listes en Caml

Le module `List`

- Pour les vecteurs, on avait utilisé le module `Array`

- Pour les listes, on utilise le **module** `List`.

  Ex : `List.length`

- Remarque : ces deux modules sont chargés automatiquement au lancement de Ocaml, donc aucune manipulation n'est nécessaire pour les utiliser.

DÉFINIR UNE LISTE

- `[ … ]` : Elément par élément
  - Syntaxe : `let nom_liste = [ e0 ; e1 ; … ; en ] ;;`
  - Exemple : `let prenoms = [ "Pierre" ; "Paul" ; "Jacques" ] ;;`

- `[]` : La liste vide
  - Exemple : `let liste_a_remplir = [] ;;`

- Remarque : en général, on crée une liste vide et on la ajoute progressivement du contenu élément par élément

Ajouter un élement

- **Rappel** : ajouter un élément `x` à une liste `l` revient à
  1. Créer une nouvelle liste dont `x` est la tête et `l` la queue.
  2. Faire de cette nouvelle liste la nouvelle valeur de `l`

- Comme on veut modifier la valeur d'une variable (de type liste), on utilise une **référence**.

- **Syntaxe** en Caml :

```
let l1 = ref [] in
l1 := "Hello" :: !l1
```

Accéder à la tête et à la queue

- Il existe des **méthodes dédiées** en Caml pour accéder à la tête et à la queue d'une liste non vide.

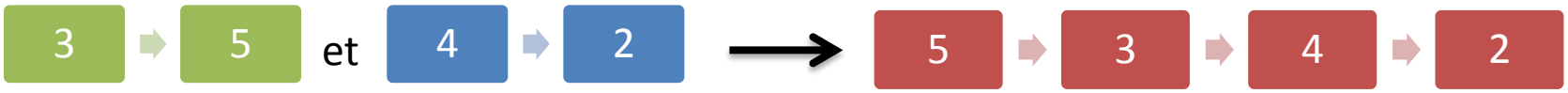
- Le module List fournit deux fonctions pour ces usages :
  - `List.hd` (pour "head") : `'a list -> 'a`
  - `List.tl` (pour "tail") : `'a list -> 'a list`

- Si vous tentez d'utiliser une de ces fonctions sur une liste vide, Caml se plaindra :

```
List.tl([]);;
    Exception: Failure "tl".
```

Concaténation de listes

- **Concaténer** deux listes, c'est les juxtaposer, les mettre l'une derrière l'autre dans une nouvelle grande liste.



- En Caml, le module `List` fournit la fonction `concat` :

```
List.concat [ [3 ; 5] ; [4 ; 2] ];;
- : int list = [3; 5; 4; 2]
```

- Plus simplement, on peut utiliser l'opérateur binaire `@`

```
[3 ; 5] @ [4 ; 2];;
- : int list = [3; 5; 4; 2]
```

Quelques fonctions du modèle `List`

- `List.length` : calculer la longueur d'une liste
- `List.nth` : récupérer le $i^{ème}$ élément d'une liste
- `List.rev` : "renverser" une liste (effet miroir)
- `List.mem` : tester si un élément appartient à une liste
- `List.sort` : trier une liste

Premières fonctions

Calculer la taille d'une liste

- Question : *Comment calculer la taille d'une liste ?*

- Solution :

```
LongueurListe(l) =
    Si EstVide(l)
          Alors
                Renvoyer 0
          Sinon
                q = Queue(l)
                Renvoyer 1 + LongueurListe(q)
    Fin Si
```

TESTER L'APPARTENANCE

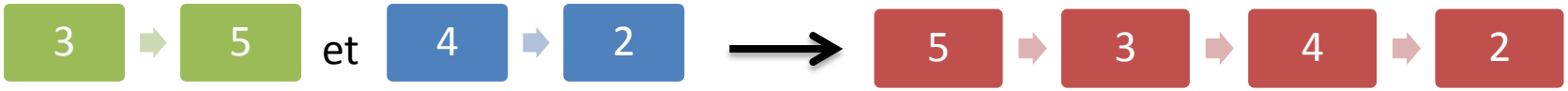
- Question : *Comment savoir si un élément* `x` *appartient à une liste* `l` *?*

- Solution :

```
Appartient(x,l) =
    Si EstVide(l)
            Alors
                    Renvoyer Faux
            Sinon
                    t = Tete(l)
                    q = Queue(l)
                    Si (t=x)
                            Alors
                                    Renvoyer Vrai
                            Sinon
                                    Renvoyer Appartient(x,q)
                    Fin Si
    Fin si
```

Concaténer deux listes

- Question : *Comment concaténer deux listes ?*



- Indice : *Avec les listes, toujours penser récursif !*

- Solution :

```
Concat(l1,l2) =
    Si EstVide(l1)
          Alors
                Renvoyer l2
          Sinon
                t = Tete(l1)
                q = Queue(l1)
                Renvoyer t :: Concat(q, l2)
    Fin si
```

Exercices

TROUVER LE MAXIMUM D'UNE LISTE D'ENTIER

- Question : *Comment le maximum parmi les éléments d'une liste d'entiers ?*

- Remarque : Vous avez à votre disposition une fonction `max` qui vous renvoie le maximum entre deux entiers.

- Solution :

```
MaxListe(l) =
    Si EstVide(l)
    Alors
        Renvoyer −∞
    Sinon
        t = Tete(l)
        q = Queue(l)
        Renvoyer max(t, MaxListe(q))
    Fin Si
```

TESTER L'APPARTENANCE DANS UN VECTEUR TRIÉ

- Question : *Comment savoir si un élément* `x` *appartient à une liste* `l` *triée par ordre croissant?*

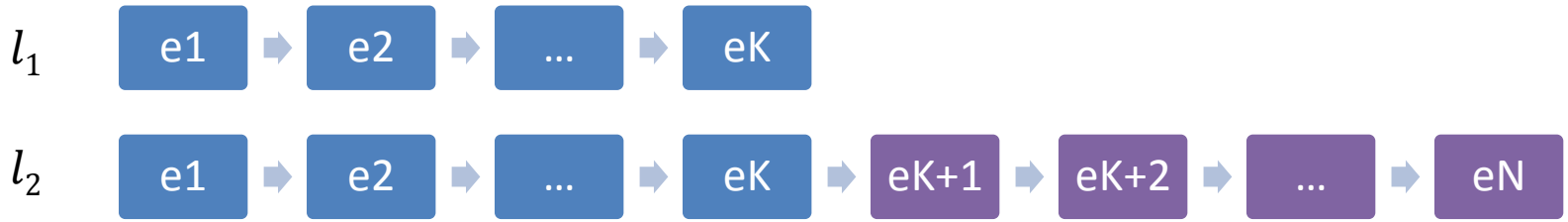
- Solution :

```
Appartient2(x,l) =
    Si EstVide(l)
       Alors
          Renvoyer Faux
       Sinon
          t = Tete(l)
          q = Queue(l)
          Si (t=x)
             Alors
                Renvoyer Vrai
             Sinon
                Si (t>x)
                   Alors Renvoyer Faux
                   Sinon Renvoyer Appartient2(x,l)
                Fin Si
          Fin Si
    Fin si
```

TESTER SI UNE LISTE EST PRÉFIXE D'UNE AUTRE

- **Définition** : : *On dit qu'une liste l_1 est préfixe d'une autre liste l_2* ssi *les premiers éléments de l_2 sont exactement ceux de l_1 (dans le même ordre)*

- **Question** : Comment tester si une liste l_1 est préfixe d'une liste l_2 ?



TESTER SI UNE LISTE EST PRÉFIXE D'UNE AUTRE

- **Question** : Comment tester si une liste l_1 est préfixe d'une liste l_2 ?

- **Solution** :

```
EstPrefixe(l1,l2) =
        Si (l1 = []) (* Cas ou l1 est vide *)
        Alors
                Renvoyer Vrai
        Sinon (* Cas ou l1 est non vide *)
                Si (l2 = []) (* Cas ou l2 est vide *)
                Alors
                        Renvoyer Faux
                Sinon (* Cas ou l2 est non vide *)
                        t1 = Tete(l1)
                        q1 = Queue(l1)
                        t2 = Tete(l2)
                        q2 = Queue(l2)
                        Renvoyer ((t1=t2) && EstPrefixe(q1,q2))
                Fin Si
   Fin Si
```

RENVERSER UNE LISTE

- Question : *Comment renverser une liste ?*

- Solution naïve (qui ne marche pas) :

```
Miroir(l) =
    Si EstVide(l)
          Alors
                Renvoyer l
          Sinon
                l = t::q
                Renvoyer (Miroir(q))::t
    Fin Si
```

RENVERSER UNE LISTE

- **Problème :** On veut stocker les éléments au fur et à mesure qu'on les rencontre et les ranger dans le bon ordre

- **Indice :** Généraliser pour mieux résoudre

  ➔ Encapsuler !

- **Idée :** On va écrire une fonction auxiliaire qui renvoie la concaténation
  - du miroir de l_1
  - et de l_2

RENVERSER UNE LISTE

- **Idée** : On va écrire une fonction auxiliaire qui renvoie la concaténation du miroir de l_1 et de l_2

- **Fonction auxiliaire** :

```
MiroirAux(l1,l2) =
      Si EstVide(l1)
      Alors
            Renvoyer l2
      Sinon
            t1 = Tete(l1)
            q1 = Queue(l1)
            Renvoyer MiroirAux(q1, t1::l2)
      Fin Si
```

RENVERSER UNE LISTE

- Solution :

```
Miroir(l) =

    MiroirAux(l1,l2) =
          Si EstVide(l1)
          Alors
                Renvoyer l2
          Sinon
                l = t1::q1
                Renvoyer MiroirAux(q1, t1::l2)
          Fin Si

    Renvoyer MiroirAux(l, [])
```

Prochaine séance

Mardi 11 Décembre 2012

[TD] Les listes – Mise en pratique