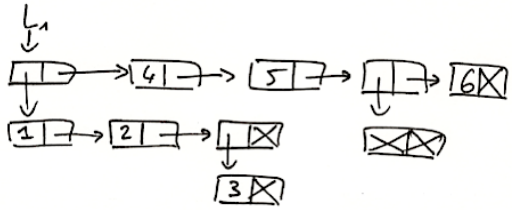
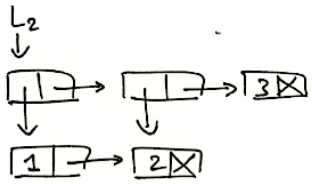
Programmation Fonctionnelle I, Printemps 2017 – TD7

http://deptinfo.unice.fr/~roy

RAPPEL : Les **listes de Scheme** [*origine : le langage **Lisp***] sont chaînées, provoquant une *distribution en spaghetti* (non contiguë) des éléments dans la mémoire. L'élément de base est un **doublet** [en anglais : *a pair*] qui est un couple (x,L) formé d'un élément x [de type quelconque] et d'une liste L, éventuellement vide. Un doublet se nomme aussi une **cellule de liste** [en anglais : *a list cell*]. Un doublet est une entité en mémoire représentée par deux cases côte à côte. La case de gauche se nomme le *first* du doublet, la case de droite le *rest*. La case de droite étant une liste, elle contient une croix pour la liste vide, et une flèche pointant vers la cellule suivante sinon. Lorsque la case *first* contient elle-même un doublet, la flèche sortante est cette fois verticale par convention.

Exercice 7.1 a) Définissez la liste `L = (a (b c) (d ()) e)` uniquement avec des appels à `cons`, puis avec `list`, puis sans aucun des deux.
b) Dessinez l'architecture de doublets de cette liste `L`.
c) Comment s'affichent les listes `(cons 1 empty)`, `(cons empty empty)`, `(cons empty 1)`, `(list empty 1)` ?
d) Quelles sont les représentations parenthésées des listes `L1` et `L2` dont les architectures de doublets sont les suivantes :





e) La liste `L2` est définie ci-dessus. A côté de son dessin, dessinez la liste `L3` définie par :

```
(define L3 (append (rest L2) (first L2)))
```

Exercice 7.2 a) Programmez naïvement une fonction `(nb-doublets L)` prenant une liste `L` et retournant le nombre de ses doublets [calculez séparément sur les doublets du *first* et ceux du *rest*...] :

`(nb-doublets '()) → 0`, `(nb-doublets '(a (b c) (d ()) e)) → 8` *(vérifiez : dessinez-le !)*

b) Pourquoi « naïvement » ? Testez donc votre algorithme sur la liste `L2` dessinée plus haut !...

Exercice 7.3 Programmez la fonction `(ppdiv>= n k)` du cours 7 page 12, retournant le plus petit diviseur de `n` qui est ≥ `k`. Exemple : `(ppdiv>= 924 50) → 66`. Testez si le prédicat `premier?` fonctionne bien. Que donne-t-il sur de grands entiers ?

Exercice 7.4 Vous connaissez peut-être le jeu des **Tours de Hanoï** ? On dispose de trois tiges `P1`, `P2`, `P3` sur lesquelles on peut enfiler des disques troués de tailles différentes. Mais attention aux **règles** suivantes :
- (R1) on ne peut poser un disque que sur un disque plus grand que lui.
- (R2) on ne peut déplacer qu'un seul disque à la fois pour le placer sur autre piquet.

But du jeu : on pose `N` disques sur le piler `P1` (donc le plus grand à la base et le plus petit tout en haut). On souhaite connaître la liste des coups à jouer pour déplacer les `N` disques sur le pilier `P3` en respectant les règles ci-dessus. Le pilier `P2` servira donc de pilier intermédiaire entre l'état initial et l'état final des piliers, ok ?

a) Programmez par récurrence sur `N` la fonction `(hanoi N P1 P2 P3)` retournant la liste des mouvements, par exemple :

`(hanoi 3 1 2 3) → ((1 vers 3) (1 vers 2) (3 vers 2) (1 vers 3) (2 vers 1) (2 vers 3) (1 vers 3))`

Pour avoir une photo du jeu (et l'algorithme au passage...), jetez un oeil à l'article suivant :

http://deptinfo.unice.fr/~roy/PF1/hanoi.pdf

b) Combien de mouvements sont nécessaires pour déplacer `N` disques ? On peut démontrer que l'algorithme par récurrence donné dans l'article cité est *optimal*.

Programmation Fonctionnelle I, Printemps 2017 – TP7

http://deptinfo.unice.fr/~roy

Exercice 7.1 **a)** Programmez la fonction `(fusion LT1 LT2)` qui est une composante du « tri-fusion » (cours 7 page 7). Les listes `LT1` et `LT2` sont supposées triées, le résultat sera aussi trié. C'est le « mélange trié ».

`(fusion '(1 5 7 8) '(2 3 5 9))` → `(1 2 3 5 5 7 8 9)`

b) Programmez complètement le **tri par fusion**, et vérifiez avec `time` que son temps de calcul est bien en O(n log n). Pour cela vous définirez deux listes `L5000` et `L10000` contenant respectivement 5000 et 10000 entiers aléatoires dans [0,100], et vous comparerez les temps de calcul pour les trier. Le temps de calcul pour trier `L10000` doit être légèrement supérieur au double du temps obtenu pour `L5000`. *Alors qu'avec le tri par insertion qui était en $O(n^2)$ c'était 4 fois plus !*

Exercice 7.2 **Le Crible d'Eratosthène** [cours 7 page 18].

a) Programmez la fonction `(suppr-mult k L)` prenant une liste d'entiers `L` et retournant une copie de `L` dans laquelle tous les multiples de `k` auront été supprimés. Exemple : `(suppr-mult 3 '(2 6 8 3 1 0 9 5))` → `(2 8 1 5)`.

b) Programmez en une ligne [avec `range` ou `build-list`] la fonction `(intervalle a b)` prenant deux entiers $a \leq b$, et retournant la liste des entiers de `[a, b]`. Par exemple `(intervalle 2 10)` → `(2 3 4 5 6 7 8 9 10)`.

c) Programmez la fonction `(crible n)` retournant la liste des nombres premiers de l'intervalle `[2,n]` en utilisant l'algorithme du crible vu en cours.

Exercice 7.3 *Examen***.** Programmez l'animation d'un **système de particules**. Regardez la video sur :

`http://deptinfo.unice.fr/~roy/PF1/sys-part-aimant.mp4`

Des billes métalliques (petits disques noirs de rayon 3) sont expulsées du milieu gauche du canvas. Une bille est une structure *(x,y,dx,dy)* contenant la position et la vitesse. Une bille part avec une vitesse de composante *dx* aléatoire dans [5,10] donc vers la droite, et de composante *dy* aléatoire dans [-10,0] donc vers le haut. Un aimant circulaire de rayon 20 est placé au centre du canvas. Le monde est une liste de 100 billes. Chaque bille poursuit une trajectoire sous l'effet de la gravitation (g = 0.5) et renaît à son point de départ dès qu'elle sort du canvas. Si elle passe au-dessus de l'aimant, elle est capturée et on la supprime du monde. L'animation stoppe dès que toutes les billes auront été capturées. A vous !

Exercices supplémentaires

Exercice 7.4 **IMPORTANT**. **a)** En utilisant le cours 7 page 18, programmez la fonction `(factor n)`. Exemple :

`(factor 6760)` → `(2 2 2 5 13 13)` `; la décomposition en facteurs premiers !`

Vous pouvez ensuite utiliser `compacter` (exo TP6.3b) pour regrouper les facteurs premiers égaux en `((2 3) (5 1) (13 2))`.

b) Récupérez la liste des nombres premiers de `[2,1000000]` sur le site :

`http://nombrespremiersliste.free.fr/listes/1-1000000.txt`

et sauvez-la sur le disque dur dans un fichier de nom `"1-1000000.txt"`. Il existe une fonction `(read-from-file f)` dans `valrose.rkt`, prenant le nom d'un fichier `f` sur disque dur contenant seulement une liste Scheme (même sur plusieurs lignes), et retournant cette liste. Utilisez-la pour définir la liste croissante `LP` de ces nombres premiers, puis reprogrammez la décomposition en facteurs premiers d'un entier `n < 1000000` en puisant un à un les facteurs premiers dans la liste `LP`.

N.B. Il existe aussi une fonction `(print-to-file f x mode)` écrivant la valeur de `x` dans le fichier `f`. Si le fichier existe déjà, il est détruit (mode = `'replace`) ou augmenté en queue (mode = `'append`).

Il existe aussi une fonction plus compliquée `(read-from-url url)` qui permet de récupérer automatiquement le contenu d'une page Web, donc de la liste 1-1000000. Mieux vaut jeter un oeil au code source du teachpack `valrose.rkt`...

Exercice 7.5 ASTUCIEUX. En utilisant la primitive `(sort L rel)`, programmez en une ligne une fonction `(shuffle L)` retournant une permutation aléatoire des éléments de `L`.

N.B. "random shuffle" signifie "mélange aléatoire" en anglais. Trouvez la bonne relation d'ordre `rel` *!*

Exercice 7.6 BIZARRE. Dans le cours 7 page 8, la fonction `(scission L)` avançait de deux éléments dans la liste `L` à chaque étape. Trouvez une solution dans laquelle on n'avance comme d'habitude que d'un seul élément !