8 Exercices sur les listes

Question 1

Définir une fonction `L->2L` qui construit une liste dupliquant tous les éléments d'une liste passée en paramètre.
Appeler `(L->2L '(1 2 3 4 5))` retourne la liste `(1 1 2 2 3 3 4 4 5 5)`.
(*fonctions utiles* : `empty?`, `cons`, `first`, `rest`)

Question 2

Définir une fonction `L->3L` qui triple une liste passée en paramètre.
Appeler `(L->3L '(1 2 3 4 5))` retourne la liste `(1 1 1 2 2 2 3 3 3 4 4 4 5 5 5)`.
(*fonctions utiles* : `empty?`, `cons`, `first`, `rest`)

Question 3

Définir une fonction `L->NL` qui démultiplie N fois une liste passée en paramètre.
Appeler `(L->NL '(1 2 3 4 5) 4)` retourne la liste `(1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4 5 5 5 5)`.
(*fonctions utiles* : `empty?`, `cons`, `first`, `rest`)

Question 4

Définir une fonction `nieme` retournant le N-ième élément d'une liste d'entiers positifs ; si `N` est négatif, alors la fonction retourne `-1` ; si `N` va au delà de la liste, alors la fonction retourne `-1` ; ne pas utiliser `list-ref`.
Appeler `(nieme '(1 2 3 4 5) -1)` retourne `-1`.
Appeler `(nieme '(1 2 3 4 5) 0)` retourne `1`.
Appeler `(nieme '(1 2 3 4 5) 2)` retourne `3`.
Appeler `(nieme '(1 2 3 4 5) 10)` retourne `-1`.
(*fonctions utiles* : `length`, `first`, `rest`)

Question 5

Définir une fonction `position` retournant la première position à laquelle apparait un élément dans une liste ; si l'élément n'est pas dans la liste, alors la fonction retourne `-1`.
Appeler `(position '(5 4 3 2 1) 0)` retourne `-1`.
Appeler `(position '(5 4 3 2 1) 5)` retourne `0`.
Appeler `(position '(5 4 3 2 1) 1)` retourne `4`.
(*fonctions utiles* : `empty?`, `first`, `rest`)

Question 6

Définir une fonction `plat` retournant une liste applatie.
Appeler `(plat '(3 (2 (1 ())))` retourne la liste `(3 2 1)`.
(*fonctions utiles* : `empty?`, `list?`, `append`, `first`, `rest`)

Question 7

Définir un prédicat `isflat?` permettant de savoir si une liste est plate.
Appeler `(isflat? '(1 2 3 4 5))` retourne vrai.
Appeler `(isflat? '(1 (2) 3 4 5))` retourne faux.
(*fonctions utiles* : `empty?`, `list?`, `first`, `rest`)

Question 8

Définir une fonction `fusion` retournant une liste fusionnant deux listes passées en paramètre ; le comportement de `fusion` est similaire à la fonction `append` sans obligatoirement respecter l'ordre des éléments.
(*fonctions utiles* : `empty?`, `cons`, `first`, `rest`)

Question 9

Définir une fonction `sublist` retournant une demi-liste d'une liste ; la liste est passé en argument ; un deuxième argument définit si la liste retournée est la première demi-liste (celle des valeurs d'indices pairs) ou la deuxième demi-liste (celle des valeurs d'indices impairs).
Appeler `(sublist '(1 2 3 4 5 6) 0)` retourne la liste `(1 3 5)`.
Appeler `(sublist '(1 2 3 4 5 6) 1)` retourne la liste `(2 4 6)`.
Appeler `(sublist '(1 2 3 4 5) 1)` retourne la liste `(2 4)`.
(*fonctions utiles* : `empty?`, `cons`, `first`, `rest`, `reverse`)

Question 10

Définir une fonction `rev-dup` inversant une liste en dupliquant ses éléments ; ne pas utiliser `reverse`.
Appeler `(rev-dup '(1 2 3 4))` retourne la liste `(4 4 3 3 2 2 1 1)`.
(*fonctions utiles* : `empty?`, `append`, `first`, `rest`, `list`)

Question 11

Définir un prédicat `isin?` permettant de savoir si un élément appartient à une liste.
Appeler `(isin? '(1 2 3 4 5) 6)` retourne faux.
Appeler `(isin? '(1 2 3 4 5) 3)` retourne vrai.
(*fonctions utiles* : `empty?`, `first`, `rest`)

Question 12

Définir une fonction `remove` permettant de supprimer un élément d'une liste.
Appeler `(remove '(1 2 3 4 5) 6)` retourne la liste `(1 2 3 4 5)`.
Appeler `(remove '(1 2 3 4 5) 3)` retourne la liste `(1 2 4 5)`.
(*fonctions utiles* : `empty?`, `cons`, `first`, `rest`)

Question 13

Définir une fonction `nfirst` retournant la liste des N premiers éléments d'une liste.
Appeler `(nfirst '(1 2 3 4 5) 6)` retourne la liste `(1 2 3 4 5)`.
Appeler `(nfirst '(1 2 3 4 5) 3)` retourne la liste `(1 2 3)`.
(*fonctions utiles* : `empty?`, `append`, `first`, `rest`, `list`)

Question 14

Définir une fonction `nlast` retournant la liste des N derniers éléments d'une liste.
Appeler `(nlast '(1 2 3 4 5) 6)` retourne la liste `(1 2 3 4 5)`.
Appeler `(nlast '(1 2 3 4 5) 3)` retourne la liste `(3 4 5)`.
(*fonctions utiles* : `length`, `rest`)

Question 15

Définir une fonction `last` retournant le dernier élément d'une liste.
Appeler `(last '(1 2 3 4 5))` retourne `5`.
Appeler `(last '(5 4 3 2 1))` retourne `1`.
(*fonctions utiles* : `length`, `rest`, `first`)

Question 16

Définir une fonction `firsts` retournant les éléments d'une liste sans le dernier.
Appeler `(firsts '(1 2 3 4 5))` retourne `'(1 2 3 4)`.
Appeler `(firsts '(5 4 3 2 1))` retourne `'(5 4 3 2)`.
(*fonctions utiles* : `length`, `rest`, `first`)

Question 17

Définir une fonction `maxlist` retournant la valeur max d'une liste.
Appeler `(maxlist '(1 2 3 4 5))` retourne `5`.
Appeler `(maxlist '(1 3 7 5 2))` retourne `7`.
(*fonctions utiles* : `empty?`, `max`, `rest`, `first`)

Question 18

Définir un prédicat `chk?` retournant vrai si `v ope e` est vrai pour tous les éléments `e` d'une liste.
Appeler `(chk? 0 < '(1 2 3 4 5))` retourne `#t`.
Appeler `(chk? 6 >= '(1 3 7 5 2))` retourne `#f`.
(*fonctions utiles* : `empty?`, `rest`, `first`)

Question 19

Définir une fonction `set-at` permettant de redéfinir une valeur dans une liste ; quand la liste considérée n'a pas de valeur à cet indice, la fonction `set-at` retourne une liste non-modifiée ; réaliser cette fonction sans utiliser `list-set`.
Appeler `(set-at '(1 2 3 4 5) 0 22)` retourne la liste `(22 2 3 4 5)`.
Appeler `(set-at '(1 2 3 4 5) -1 22)` retourne la liste `(1 2 3 4 5)`.
Appeler `(set-at '(1 2 3 4 5) 22 22)` retourne la liste `(1 2 3 4 5)`.
(*fonctions utiles* : `empty?`, `length`, `first`, `rest`)

Question 20

Définir les fonctions `diviseurs` et `pgcd` :

- La fonction `diviseurs` retourne la liste des diviseurs d'une valeur ; cette liste est établie par la division successive de la valeur passée en paramètre par des valeurs de plus en plus grandes ; les valeurs pour lesquelles le reste de la division est nul sont ajoutées à la liste ; quand la valeur de division est supérieure à la valeur à diviser, la liste construite est la liste des diviseurs.
- La fonction `pgcd` retourne le plus grand commun diviseur par comparaison des listes de diviseurs.

Appeler `(diviseurs 20)` retourne la liste `(20 10 5 4 2 1)`.
Appeler `(pgcd 52 20)` retourne la valeur 4.
(*fonctions utiles* : `remainder`, `cons`, `first`, `rest`)

Question 21

Définir une fonction `mk-cmd` retournant la fonction à exécuter pour acceder à un élément dans une liste plate ; le programme écrit donc un programme dont l'exécution retourne l'élément recherché ; si l'élément n'est pas dans la liste, alors la fonction retourne un programme qui ne fait rien (*i.e.* retourne la string `(void)` donc retourne `"(void)"`).
Appeler `(mk-cmd '(1 3 5 7 11) 1)` retourne `"(first '( 1 3 5 7 11))"`.
Appeler `(mk-cmd '(1 3 5 7 11) 11)` retourne `"(first (rest (rest (rest (rest '( 1 3 5 7 11))))))"`.
Appeler `(mk-cmd '(1 3 5 7 11) 111)` retourne `"(void)"`.
(*fonctions utiles* : `empty?`, `list->str`, `first`, `rest`)

Question 22

Définir une fonction `conway` retournant le N-ième terme de la suite audioactive de Conway ; le premier terme de la suite de Conway est la liste `'(1)` ; les termes suivants sont obtenus en indiquant le nombre de répétitions de chaque chiffre dans le terme précédent ; si on note X_n le N-ième terme de la suite de Conway, on a :

- X_0 = `'(1)`
- X_1 = `'(11)`
- X_2 = `'(21)`
- X_3 = `'(1211)`
- X_4 = `'(111221)`

Appeler `(conway 4)` retourne `'(111221)"`.
Appeler `(conway 6)` retourne `'(13112221)"`.
(*fonctions utiles* : `empty?`, `first`, `rest`, `let`, `cons`)