TD2

Exercice 1.
Le but de cet exercice est d'implémenter les fonctions de base pour les listes doublement chaînées.

1. Écrire une fonction `Ajouter(x,lst)` qui ajoute l'élement `x` en tête de la liste doublement chaînée `lst`.
2. Écrire une fonction `Retirer(c)` qui retire une case `c` contenue dans une liste doublement chaînée. Quelle est sa complexité ?
3. Écrire une fonction `Insérer(x,c)` qui insère l'élement `x` après la case `c` au sein d'une liste doublement chaînée. Quelle est sa complexité ?

Exercice 2.
On représente un compteur dans une liste `cpt` contenant chacun de ses bits dans sa représentation binaire, de la droite vers la gauche. Par exemple, quand le compteur a atteint la valeur 37, il est représenté par la liste `cpt=[1,0,1,0,0,1]`.

1. Écrire un algorithme `incr` qui incrémente la valeur de `cpt`. Quelle est sa complexité dans le pire cas ? Dans le meilleur cas ?
2. On initialise `cpt` à la liste vide (qui représente 0). Montrer que la complexité de n appels à `incr` est $O(n)$.
   On dit que la complexité amortie de la fonction ***`incr`*** *est* $O(1)$.
3. Écrire une fonction `reduce` qui renvoie la valeur numérique de `cpt`.

Exercice 3.

1. Écrire un algorithme `BienParenthese` qui vérifie si une expression donnée en paramètre sous la forme d'une chaîne de caractères `s` est bien parenthésée.

```
s0 = "(x+y)(z*(x/(z+y)+t)*(z+a))"
s1 = "(x+y*(z-t)"
s2 = "(x+y)+z)*(a*(b+c)
BienParenthese(s0)
>> Vrai
BienParenthese(s1)
>> Faux
BienParenthese(s2)
>> Faux
```

   Cela signifie que dans tout préfixe de l'expression, on a au moins autant de parenthèses ouvrantes "(" que de parenthèses fermantes ")", et un même nombre dans l'expression totale.

2. Même question, mais cette fois-ci différent types de parenthèses peuvent apparaître dans l'expression : `( ) [ ] { }`.

   Cette fois, il faut également vérifier qu'une parenthèse ouvrante n'est pas suivie d'une parenthèse fermante d'un autre type.

Exercice 4.
Voici la description Wikipédia du crible d'Eratosthène.

> L'algorithme procède par élimination : il s'agit de supprimer d'une table des entiers de 2 à N tous les multiples d'un entier (autres que lui-même).
>
> En supprimant tous ces multiples, à la fin il ne restera que les entiers qui ne sont multiples d'aucun entier à part 1 et eux-mêmes, et qui sont donc les nombres premiers.
>
> On commence par rayer les multiples de 2, puis les multiples de 3 restants, puis les multiples de 5 restants, et ainsi de suite en rayant à chaque fois tous les multiples du plus petit entier restant.
>
> On peut s'arrêter lorsque le carré du plus petit entier restant est supérieur au plus grand entier restant, car dans ce cas, tous les non-premiers ont déjà été rayés précédemment.
>
> À la fin du processus, tous les entiers qui n'ont pas été rayés sont les nombres premiers inférieurs à N.

1. En utilisant le crible d'Eratosthène, écrire un algorithme prenant en paramètre un entier n, et qui renvoie la liste de tous les entiers premiers inférieurs à n.
2. Exprimer, à l'aide d'une somme faisant intervenir l'ensemble $\mathbb{P}_n$ des nombres premiers inférieurs à n, le nombre de fois que l'algorithme effectue l'opération *rayer un nombre*. On verra en cours que cette somme vaut approximativement $n \ln \ln n$.
3. Comment pourrait-on faire pour éviter à l'algorithme de rayer plusieurs fois le même nombre ? Proposer une implémentation efficace de cette solution en s'inspirant de la structure des listes doublement chaînées.