B25T

CONCOURS ENSAM - ESTP - ECRIN - ARCHIMEDE

Epreuve d'Informatique MP

durée 3 heures

Indiquez en tête de copie ou de chaque exercice le langage utilisé.

1. Exponentiation rapide modulo m

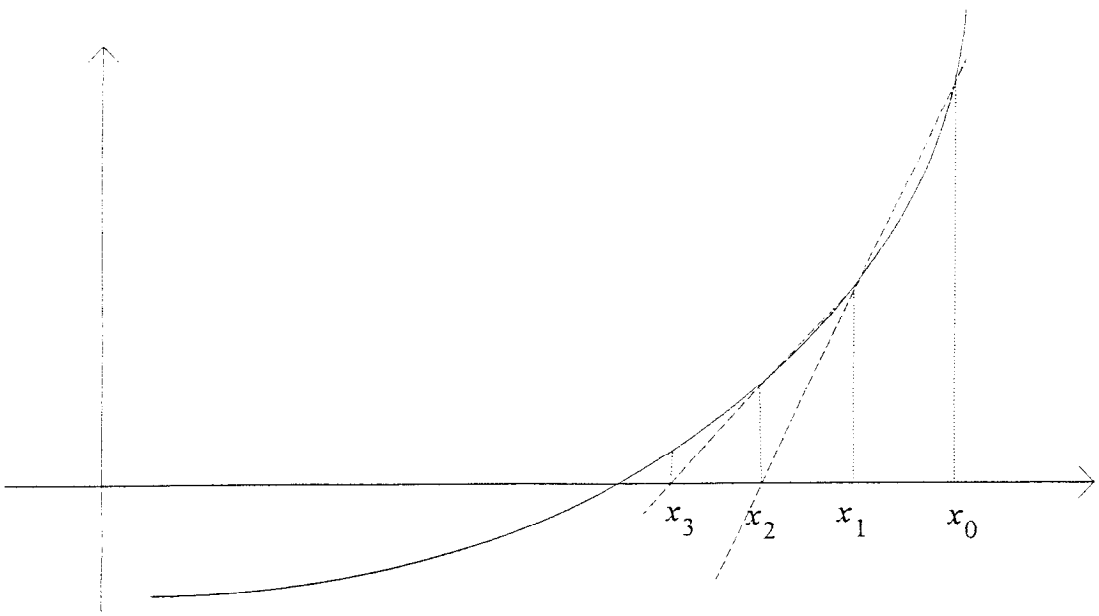
1.1. Écrire la fonction **récursive** qui calcule $x^n[m]$ (x^n modulo m) en utilisant le principe suivant :

$$x^0[m] = 1$$
$$x^{2n}[m] = (x^n[m])^2[m]$$
$$x^{2n+1}[m] = (x \cdot x^{2n}[m])[m]$$

1.2. Donner le nombre de multiplications et de calculs de modulo en fonction de n.

2. Résolution de l'équation $f(x)=0$ par la méthode des sécantes

La méthode de la sécante pour résoudre une équation $f(x)=0$ consiste à partir de deux approximations différentes x_0 et x_1 de la solution et à construire la suite x_i , $i >=2$ telle que x_{i+1} soit l'intersection de l'axe des x avec la droite passant par les points $[x_{i-1}, f(x_{i-1})]$ et $[x_i, f(x_i)]$, et ce jusqu'à ce que l'écart relatif entre x_i et x_{i+1} soit inférieur à une précision donnée (la dérivée seconde de f doit garder son signe pour que la méthode fonctionne)



2.1. Écrire la fonction

```
intersection        données x1,y1,x2,y2 : réels
                    résultat x : réel
```

qui calcule l'abscisse de l'intersection de la droite passant les points$[x_1 , y_1]$et $[x_2 , y_2]$avec l'axe des x.

2.2. Écrire la fonction

```
zero_secante        données f : fonction; x0, x1 réel
                    résultat x : réel
```

qui retourne une valeur approchée à 10^{-5} d'une solution de $f(x)=0$ avec x_0 et x_1 deux valeurs approchées de la solution

3. Carré magique

Un carré magique est une matrice $N*N$ contenant tous les nombres de **1** à N^2 et telle que les sommes des nombres de chaque ligne, chaque colonne et chaque diagonale soient chacune égales à une constante.

Par exemple,

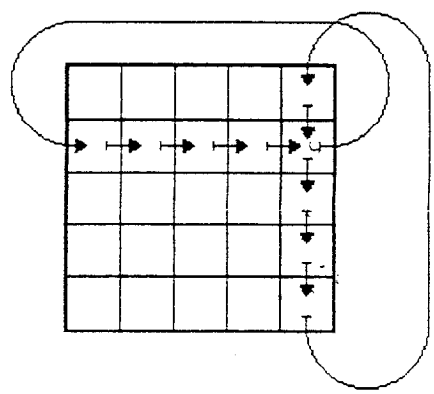
4	9	2
3	5	7
8	1	6

est un carré magique d'ordre **3**

3.1. Quelle est la constante d'un carré magique d'ordre N ?

Dans la suite, on utilisera le type `Matrice` supposé prédéfini qui représentera un tableau à deux dimensions dont les indices commencent à `1`.

3.2. Écrire une fonction prenant en argument une matrice `m` d'ordre `n` et retournant `vrai` ou `faux` si la matrice est un carré magique.



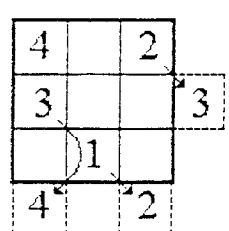
Une des méthodes permettant de construire un carré magique d'ordre impair est la suivante : on considère le tableau comme un tore, à savoir la ligne ***n*** est suivie par la ligne **1** et la colonne ***n*** est suivie par la colonne **1**.

Mettre tous les nombres de 1 à n^2 dans les cases calculées comme suit :

- pour 1, le mettre dans la case située sous la case centrale du carré
- chaque nombre sera mis dans la case située ligne suivante, colonne suivante de celle où on a mis le nombre précédent. Si cette case est déjà remplie, on avance encore d'une ligne et on recule d'une colonne.

Le tableau ainsi constitué est un carré magique.

Voici le début de la construction du carré magique d'ordre 3 :



3.3. Représenter le carré magique d'ordre **5** construit de cette façon.

3.4. Écrire le programme qui remplit selon cette méthode une matrice `m` d'ordre `n` de manière à la rendre magique.

4. Grands nombres

La plupart des langages informatiques représentent le type entier par un sous-ensemble fini de IN (typiquement, $0..2^{p-1}$, où p est la taille en bits du codage d'un entier). Cette représentation conduit à des dépassements de capacité, et, si un utilisateur de ces langages veut travailler sur des grands nombres, il est conduit à écrire des méthodes spécifiques pour les calculs.

Supposons que le langage que vous utilisez n'utilise que le type `Booléen`, le type `Chiffre` (de `0` à `9`) et le type `Liste`. Dans les exemples suivants, une liste sera écrite sous la forme d'une suite de chiffres entre crochets.

Dans la suite, on supposera définies les constantes et fonctions suivantes :

- les booléens : `vrai` et `faux`
- les chiffres : `0 1 2 3 4 5 6 7 8 9`
- la liste vide : `[]`
- `estVide (lst)` : retourne `vrai` si la liste est égale à `[]`, `faux` sinon
  Ex. : `estVide([1,2,3]) --> faux`
- `cons(c,lst)` : retourne la liste composée du chiffre `c` suivie de la liste `lst`
  Ex. : `cons(3,[4,2,1])-->[3,4,2,1]`
- `tete(lst)` : retourne le premier chiffre de la liste `lst`. Met fin au programme si `lst` est vide
  Ex. : `tete([4,2,1])-->4`
- `queue(lst)` : retourne la liste `lst` privée de son premier élément. Met fin au programme si `lst` est vide
  Ex. : `queue([4,2,1])-->[2,1]`
- `resultatPlus(c1,c2)` : retourne le résultat sans retenue de l'addition des chiffres `c1` et `c2`
  Ex. : `resultatPlus(8,9)-->7`
- `retenuePlus(c1,c2)` : retourne la retenue de l'addition des chiffres `c1` et `c2`
  Ex. : `retenuePlus(8,9)-->1`
- `resultatFois(c1,c2)` : retourne le résultat sans retenue de la multiplication des chiffres `c1` et `c2`
  Ex. : `resultatFois(8,9)-->2`
- `retenueFois(c1,c2)` : retourne la retenue de la multiplication des chiffres `c1` et `c2`
  Ex. : `retenueFois(8,9)-->7`

Le but de ce problème est de construire les opérations permettant de manipuler les entiers positifs ou nuls représentés par des listes de chiffres. La représentation choisie, pour faciliter les opérations, placera le chiffre des unités toujours à la même position : en tête de liste. Par exemple, `[2,8,9,1]` représentera le nombre entier `1982`

À l'aide des constantes et fonctions précédentes, écrire les fonctions suivantes. On s'interdira d'utiliser les entiers (`123`, `256`, `...`) autres que des chiffres, et les opérations sur les entiers (`+`, `*`, `...`)

4.1. `estNul(lst)` : retourne vrai si la liste `lst` représente `0`, faux sinon
Ex. : `estNul([]) --> vrai`
`estNul([0,0,0]) --> vrai`
`estNul([0,0,1,0,0]) --> faux`

4.2. `estEgal(lst1,lst2)` : retourne `vrai` si les listes `lst1` et `lst2` représentent le même nombre, `faux` sinon
Ex. : `estEgal([7,1,0,0],[7,1]) --> vrai`

4.3. `resultatAjoute(lst1,lst2)` : retourne la liste des résultats des additions des chiffres de `lst1` et `lst2`
Ex. : `resultatAjoute([5,4,9,4],[3,2,8,7,4]) --> [8,6,7,1,4]`

4.4. `retenueAjoute(lst1,lst2)` : retourne la liste des retenues des additions des chiffres de `lst1` et `lst2`
Ex. : `retenueAjoute([5,4,9,4],[3,2,8,7,4]) --> [0,0,1,1,0]`

4.5. `nombreAjoute(lst1,lst2)` : retourne la liste représentant l'addition des entiers représentés par `lst1` et `lst2`
Ex. : `nombreAjoute([5,4,9,4],[3,2,8,7,4]) --> [8,6,7,2,5]`

4.6. `resultatMultiplie(lst,c)` retourne la liste des résultats des multiplications des chiffres de `lst` par le chiffre `c`
Ex. : `resultatMultiplie([1,7,3,2],6) --> [6,2,8,2]`

4.7. `retenueMultiplie(lst,c)` retourne la liste des retenues des multiplications des chiffres de `lst` par le chiffre `c`
Ex. : `retenueMultiplie([1,7,3,2],6) --> [0,4,1,1]`

4.8. `multiplie(lst,c)` retourne la liste représentant le produit de l'entier représenté par `lst` et du chiffre `c`
Ex. : `multiplie([1,7,3,2],6) --> [6,2,2,4,1]`

4.9. `nombreMultiplie(lst1, lst2)` retourne la liste représentant le produit des entiers représentés par `lst1` et `lst2`
Ex. : `nombreMultiplie([8,2,1],[2,1,5])--> [6,3,5,5,6]`

4.10. `factorielle(lst)` retourne la liste représentant la factorielle de l'entier représenté par `lst`. Comme `12!=479001600`,
Ex : `factorielle([2,1])-->[0, 0, 6, 1, 0, 0, 9, 7, 4]`

4.11. `enleverZero(lst)` retourne la liste représentant l'entier représenté par `lst`, privée de ses 0 non signifiants
Ex. : `enleverZero([9,8,7,1,0,0,0]) --> [9,8,7,1]`

4.12. `nombreChiffres(lst)` : retourne la liste représentant le nombre de chiffres de l'entier représenté par `lst`
Ex. : `nombreChiffres(factorielle([2,1]))--> [9]`