# E3A Informatique MP 2007 — Corrigé

Ce corrigé est proposé par Vincent Puyhaubert (Professeur en CPGE) ; il a été relu par Guillaume Batog (ENS Cachan) et Benjamin Monmege (ENS Cachan).

---

Ce sujet comporte six exercices complètement indépendants qui permettent de réviser la quasi-totalité du cours d'informatique. La difficulté n'est pas homogène : certains exercices ne posent guère de difficultés tandis que d'autres nécessitent des rédactions délicates.

- Le premier exercice concerne la programmation sur des listes et des tableaux. Les codes demandés ne sont pas compliqués. Il faut en revanche réfléchir un peu si l'on ne veut pas se retrouver à écrire des fonctions d'une page de long.
- Le deuxième exercice propose de reconnaître des morceaux de codes. Les deux premiers se comprennent très facilement. En revanche, le dernier est plus délicat à identifier et l'explication de son rôle n'est de surcroît pas simple à rédiger.
- Les exercices 3 et 4 demandent d'écrire des fonctions d'arithmétique. Les codes sont simples et très courts.
- Le cinquième est un problème classique d'automates. C'est la partie la plus difficile à rédiger même si l'intuition du résultat vient rapidement.
- Pour finir, le sixième exercice porte sur la logique. Il est court et sans prétention.

Tout ceci constitue finalement une épreuve variée et intéressante bien qu'elle ne propose pas l'étude approfondie d'une problématique précise. C'est un très bon sujet pour vérifier que l'on maîtrise l'ensemble des connaissances du programme.

## Indications

1.a Utiliser des boucles pour parcourir le tableau tout en comparant les données d'indices $(i, j)$ et $(j, i)$.

1.b Écrire une fonction récursive qui compare dans un premier temps les têtes de listes puis effectue un appel récursif sur les queues.

1.c Utiliser la variable $x$ comme une variable indiquant l'état *possible* du tableau et initialisée à 0 (le tableau est supposé constant au départ). Comparer `T[i]` et `T[i+1]` pour $i$ allant de 0 à $\ell - 1$ et mettre à jour la variable $x$ en fonction des résultats des tests.

Pour le programme de fusion, commencer par déterminer l'ordre des tableaux puis écrire une version modifiée de la fonction de fusion vue en cours sur les tris.

2.a Justifier que le code retourne la plus petite valeur dont le carré est supérieur à 2007.

2.b Justifier que la première fonction extrait une partie d'un tableau `T`. Démontrer ensuite que la seconde fonction renvoie la $\ell$-ième plus petite valeur pour l'ordre naturel sur les entiers parmi les éléments du tableau `T`.

3 Écrire dans un premier temps un programme qui calcule la somme des diviseurs d'un entier $n$ (qui fonctionne à l'aide d'une boucle).

4 Écrire une fonction récursive qui calcule le produit des chiffres d'un entier $n$ en utilisant la division euclidienne de $n$ par 10.

Écrire ensuite un code récursif qui calcule la persistance d'un entier $n$. Remarquer que sauf cas particulier, cette dernière est égale à celle du produit des chiffres de $n$ augmentée de 1.

5.1 Prouver que si $m > n$, le mot $a^m$ est rejeté dans l'automate $\mathcal{A}_n$ à la lecture du $(n+1)$-ième $a$.

5.2 Justifier que $\mathcal{L}_1 = \{a\overline{a}\}^* \{\varepsilon, a\}$.

5.3 Décomposer un mot $u$ de $\mathcal{L}_2$ suivant le parcours effectué dans $\mathcal{A}_2$ à la lecture de ce mot.

5.4.a Utiliser une technique similaire à la question précédente, sachant que l'on doit revenir à 0.

5.4.b Même remarque qu'à la question précédente, sachant que l'on doit cette fois aboutir en $j$.

5.4.c Démontrer par récurrence sur $n$ que l'on peut calculer une expression de $\mathcal{L}_n^{0,j}$ pour tout entier $n$ et tout $j \in [\![ 0\,;\, n ]\!]$.

Remarquer ensuite que $\mathcal{L}_n = \bigcup_{0 \leqslant j \leqslant n} \mathcal{L}_n^{0,j}$.

6.1 Utiliser le connecteur logique $\Longrightarrow$ pour représenter les formules.

6.2 On rappelle que $x \Longrightarrow y$ est équivalent à $(\neg x) \vee y$.

6.3 Démontrer l'impossibilité de contredire la formule à l'aide de la table de vérité.

Pour une épreuve touchant à des domaines aussi variés du programme, il est difficile de dresser une synthèse globale des difficultés rencontrées par les élèves. Toutefois, la lecture du rapport révèle deux principaux écueils :

- les programmes obscurs et/ou non commentés ; si le jury n'accorde pas une importance démesurée à la justesse syntaxique, il est toutefois sensible au fait que chaque fonction soit dissociée en plusieurs sous-programmes ou que chaque partie du code comporte des explications.
- les questions théoriques mal justifiées ; il est fréquent que les résultats de certaines questions soient rapidement devinés, mais les élèves sont parfois incapables de les justifier rigoureusement. Le rapport du jury rappelle donc à juste titre que les épreuves d'informatique nécessitent autant de rigueur que celles de mathématiques.

## Exercice 1

**1.a** L'énoncé n'est pas clair dans cette question. La variable `resultat` suggère de renvoyer un booléen. Pourtant, 0 ou 1 sont du type entier. Faisons donc le choix de respecter la signature de la fonction, et donc de renvoyer des booléens.

Pour déterminer si une matrice A est symétrique, il suffit de vérifier la définition c'est-à-dire vérifier que $A_{i,j} = A_{j,i}$ pour tout $(i,j) \in [\![1\,;n]\!]^2$. Il suffit pour cela d'utiliser deux boucles imbriquées qui parcourent tous les couples d'indices $(i,j) \in [\![1\,;n]\!]^2$. On peut toutefois remarquer que l'on peut se contenter des couples satisfaisant $i < j$ pour raccourcir les boucles. Par défaut, la matrice est supposée symétrique. La variable `x` est ainsi initialisée à `true`. Lorsqu'on trouve un couple $(i,j)$ qui ne vérifie pas la condition $A_{i,j} = A_{j,i}$, cette variable passe à `false` et l'exécution peut s'arrêter. À la fin de l'exécution, il n'y a plus qu'à renvoyer le contenu de `x`.

On obtient alors le code suivant. Le type `'a vect vect` est ici utilisé pour `M` mais d'autres possibilités sont envisageables (comme le type `'a list list`).

```
let Sym M n =
  let x = ref true and i = ref 0 and j = ref 1 in
  while !x = true && !i < n do
    while !x = true && !j < n do
      if M.(!i).(!j) <> M.(!j).(!i) then x := false;
      incr j;
    done;
    incr i;
    j:=i+1;
  done;
!x;;
```

Comme dans la plupart des codes testant si un certain nombre de propriétés sont satisfaites par une donnée, on gagne du temps en stoppant l'exécution quand une condition n'est pas satisfaite. Il faut bien noter toutefois que cela ne change jamais la complexité dans le pire des cas, et que l'on améliore rarement l'ordre de grandeur de la complexité moyenne (qui ici reste en $O\left(n^2\right)$ opérations élémentaires).

**1.b** Pour tester si deux listes sont égales, il suffit de vérifier si leurs têtes et leurs queues sont égales. Cela donne la version récursive suivante de la fonction `Egalite_listes`. La récursivité s'arrête lorsque l'on atteint la fin d'une liste (à savoir une valeur $-1$), auquel cas il faut simplement vérifier que l'on est également à la fin de l'autre liste.

```
let rec Egalite_listes L L' =
  match (L,L') with
    |([-1],_) -> (hd L') = [-1]
    |(_,[-1]) -> (hd L) = [-1]
    |(a::q,b::r) -> if a=b then Egalite_listes q r else false
    |(_,_) -> failwith "Liste non conforme" ;;
```

> Le symbole `"-"` de tiret est interprété comme une soustraction par Caml et ne peut être utilisé au sein d'un nom de variable. C'est pourquoi il a été remplacé par le symbole `"_"` (tiret bas, ou plus communément underscore) dans le nom de la fonction `Egalite_listes`. Suivre scrupuleusement le nom choisi par l'énoncé conduit à une erreur de compilation.
>
> Le programme donné ci-dessus ne fonctionne que pour les listes respectant les conventions de l'énoncé. Notamment, il ne s'applique que pour des listes d'entiers, et génère une erreur s'il n'y a pas de $-1$ dans l'une des listes. Il est très facile de modifier le programme précédent pour qu'il fonctionne quel que soit le type de liste :
>
> ```
> let rec Egalite_listes L L' =
>   match (L,L') with
>     |([],_) -> L' = []
>     |(_,[]) -> L = []
>     |(a::q,b::r) -> (a=b)&&(Egalite_listes q r) ;;
> ```
>
> Au passage, on profite de l'évaluation paresseuse des booléens pour raccourcir le code (l'appel récursif n'est effectué que si le premier test a réussi).

**1.c** Pour déterminer la nature d'un tableau, initialisons la variable `x` à 0, ce qui signifie que le tableau est supposé constant par défaut. Parcourons alors le tableau de gauche à droite en comparant `T[i]` et `T[i+1]` pour $i$ allant de 0 à $n-1$, en mettant à jour la variable `x`. Tant que la variable est à 0, plusieurs cas sont envisageables :

- `T[i] < T[i+1]` : le tableau ne peut plus être constant. En revanche, il peut toujours être rangé par ordre croissant. On passe donc la variable `x` à 1.
- `T[i] > T[i+1]` : à nouveau, le tableau ne peut plus être constant. Cependant, il peut être rangé par ordre décroissant. On passe donc la variable `x` à $-1$.
- `T[i] = T[i+1]` : il n'y a rien à faire.

Lorsque l'on est passé dans l'état 1, il n'y a plus que deux cas de figures pour la mise à jour de `x` :

- `T[i] <= T[i+1]` : il n'y a rien à faire.
- `T[i] > T[i+1]` : le tableau ne peut plus être trié ni par ordre croissant, ni par ordre décroissant. On passe donc la variable `x` à 2.