# TP 6 : Invariants, tableaux multidimensionnels

Informatique Fondamentale (IF121)

1er–12 décembre 2003

## 1 Invariants et boucles while

**Exercice 1 — *La constante de Néper*** La constante de Néper, notée $e$, est définie par :

$$e = \lim_{n \to \infty} \sum_{k=0}^{n} \frac{1}{k!}$$

La suite $u_n = \sum_{k=0}^{n} \frac{1}{k!}$ est une suite croissante qui tend vers $e$. En calculant la suite $u_n$ pour un entier $n$ assez grand, on obtient une bonne valeur approchée de $e$. La propriété suivante permet de contrôler l'erreur commise : pour tout réel $\epsilon > 0$ on a $u_{n+1} - u_n < \frac{\epsilon}{3} \Rightarrow e - u_{n+1} < \epsilon$

Écrire une méthode « `neper` » qui calcule une valeur approchée de $e$, elle prend en argument un double $\epsilon$ qui correspond à la précision désirée sur le résultat.

La méthode comporte une boucle pour laquelle vous préciserez un invariant, afin de prouver la correction du programme.

**Exercice 2 — *Recherche dichotomique*** Le but est d'écrire une fonction de recherche d'une valeur dans un tableau d'entiers.

1. Écrire une méthode « `recherche` » qui prend un tableau d'entiers « `t` » et une valeur entière « `v` » et qui retourne un indice « `i` » tel que `t[i]=v` (si la valeur n'est pas présente dans le tableau, la méthode retourne $-1$).
2. Dans le cas où le tableau est trié par ordre croissant, il est possible de retrouver plus rapidement la valeur recherchée « `v` », par *dichotomie*. Le principe est de comparer à `v` la valeur stockée au milieu du tableau : si elle est plus grande que `v`, il faut chercher dans la moitié inférieure du tableau, et sinon dans la moitié supérieure. On recommence alors sur la portion du tableau de taille moitié et ainsi de suite jusqu'à trouver la valeur. Écrire la méthode « `rechercheDichotomie` » basée sur cette approche. Identifier un invariant et démontrer la correction du programme.

**Exercice 3 — *Deviner un nombre*** Un jeu très amusant consiste à choisir un nombre au hasard et le faire deviner à l'adversaire en lui disant « plus grand » ou « plus petit ». Écrire un programme qui devine un nombre entre 1 et 100 choisi par l'utilisateur. Exemple de dialogue :

```
> java Devin
Est-ce 50 ? plus
Est-ce 75 ? plus
Est-ce 87 ? moins
Est-ce 81 ? plus
Est-ce 84 ? moins
Est-ce 82 ? oui
```

Le programme pourra procéder par dichotomie de manière analogue à l'exercice précédent. Identifier un invariant et démontrer la correction du programme.

**Exercice 4 — *Tri par sélection*** Le but de cet exercice est d'écrire une fonction qui trie un tableau de nombres entiers, en prouvant la validité du programme.

1. Écrire une fonction qui prend en argument un tableau de nombres entiers et renvoie l'indice du plus petit élément. L'expression « `minimum(tableau)` » doit renvoyer un entier « `i` » tel que « `tableau[i]` » soit le plus petit élément du tableau.
2. La fonction précédente utilise forcément une boucle : identifier un invariant pour démontrer que la fonction est correcte.

3. Modifier la fonction « `minimum` » pour qu'elle prenne en argument l'indice où il faut commencer la recherche. Ainsi « `minimum(tableau, début)` » doit renvoyer un entier « `i` » tel que « `tableau[i]` » est le plus petit élément du tableau d'indice supérieur ou égal à « `début` ». Modifier l'invariant pour vérifier la correction de cette nouvelle fonction.

Le principe du tri par sélection est le suivant : on commence par rechercher le plus petit élément du tableau, puis on l'échange avec le premier élément du tableau. On cherche ensuite le plus petit élément à partir du deuxième indice, puis on l'échange avec le deuxième élément du tableau, et ainsi de suite jusqu'à avoir tout trié.

4. À l'aide des fonctions précédentes, écrire une fonction qui effectue un tri par sélection sur un tableau d'entiers. Identifier un invariant pour démontrer la correction de cette fonction.

**Exercice 5 — *Algorithme d'Euclide*** Le but de l'exercice est d'écrire un programme qui calcule le pgcd (plus grand commun diviseur) de deux nombres entiers positifs (non tous nuls), à l'aide de l'algorithme d'Euclide. Celui-ci est basé sur la propriété suivante : si $a$ et $b \neq 0$ sont deux entiers positifs, et si l'on note $r_1$ le reste de la division euclidienne de $a$ par $b$, alors $pgcd(a, b) = pgcd(b, r_1)$. L'idée de l'algorithme est de recommencer la même opération sur le couple $(b, r_1)$, ce qui donne un nouveau reste $r_2$, on poursuit avec le couple $(r_1, r_2)$ et ainsi de suite, jusqu'à tomber sur un reste $r_n$ nul. On a alors $pgcd(a, b) = pgcd(b, r_1) = pgcd(r_1, r_2) = \ldots = pgcd(r_n, 0)$ ; or on voit immédiatement que $pgcd(r_n, 0) = r_n$.

1. Démontrer la terminaison de l'algorithme.
2. Écrire la méthode « `pgcd` » basée sur cet algorithme (elle est construite autour d'une boucle `while`).
3. Exhiber un invariant pour la boucle `while` permettant de justifier la correction du programme.

**Exercice 6 — *Indentation*** On veut écrire un programme qui lit des lignes de programme en Java et qui les indente correctement. Il s'agit de réécrire chaque ligne en s'assurant qu'elles commencent par le bon nombre de caractères "espace", ce nombre étant lié au nombre d'accolades ouvertes. La bonne mesure au niveau de chaque ligne est le nombre d'accolades ouvertes depuis le début du programme et qui n'ont pas encore été fermées.

On aura recours aux fonctions de la classe Deug qui permettent de manipuler les chaînes de caractères (comme « `Deug.length` », « `Deug.charAt` » et « `Deug.subString` », dont on peut trouver la documentation à l'adresse `http://www.pps.jussieu.fr/~beffara/enseignement/deug.html`).

1. Écrire une fonction « `accolades` » qui prend en argument une chaîne de caractères et renvoie le nombre d'accolades ouvertes par cette chaîne, c'est-à-dire le nombre d'occurences de « `{` » moins le nombre d'occurences de « `}` ».
2. Écrire une fonction qui prend en argument une chaîne de caractères et renvoie la même chaîne sans espaces au début.
3. Par convention, si $n$ accolades sont ouvertes (et pas refermées) avant une ligne donnée, on veut qu'il y ait $4n$ espaces au début de cette ligne. Écrire une fonction qui prend en argument un entier $n$ et une chaîne de caractères et revoie la même chaîne avec exactement $4n$ espaces au début.
4. Écrire un programme qui lit les lignes d'un programme avec la fonction « `readLine` » et écrit les mêmes lignes avec le bon nombre d'espaces au début.

Note : si on lance ce programme avec la commande « `java Indentation` », le programme va attendre que l'utilisateur tape les lignes au clavier. Si on le lance avec la commande « `java Indentation < Toto.java` », il va lire les lignes dans le fichier « `Toto.java` » et afficher le résultat.

## 2 Matrices

Les matrices sont des tableaux à deux dimensions de nombres réels, on va donc les représenter par des variables de type « `double[][]` ». Rappelons la syntaxe des manipulations de base sur ces tableaux :

| | |
|---|---|
| `double[][] a;` | déclaration d'une matrice |
| `a = new double[`$m$`][`$n$`];` | création d'une matrice à $m$ lignes et $n$ colonnes |
| `a[`$i$`][`$j$`]` | accès à l'élément $j$ de la ligne $i$ |
| `a.length` | nombre de lignes d'une matrice |
| `a[0].length` | nombre de colonnes d'une matrice |

Les lignes d'une matrice $m \times n$ sont numérotées de 0 à $m-1$ et ses colonnes sont numérotées de 0 à $n-1$.

**Exercice 7 — *Entrées et sorties***

1. Écrire une fonction « `litMatrice` » qui demande à l'utilisateur les dimensions d'une matrice et ses coefficients et renvoie le tableau correspondant.
2. Écrire une fonction « `afficheMatrice` » qui affiche une matrice ligne par ligne. Par exemple, la matrice

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad \text{s'affiche}$$

```
1.0 2.0
3.0 4.0
```

**Exercice 8 — *Addition*** Si $A$ et $B$ dont deux matrices à $m$ lignes et $n$ colonnes, dont les éléments sont $a_{ij}$ et $b_{ij}$, la matrice $A+B$ est la matrice de même taille dont les éléments sont les $a_{ij}+b_{ij}$. Écrire une fonction qui prend en arguments deux matrices (qu'on supposera de même taille) et renvoie leur somme.

**Exercice 9 — *Produit par un scalaire*** Si $A = (a_{ij})$ est une matrice $m \times n$ et $x$ est un nombre réel, la matrice $x\,A$ est la matrice $m \times n$ dont les éléments sont les $x\,a_{ij}$. Écrire une fonction qui prend en arguments un réel « `x` » et une matrice « `a` » et qui renvoie la matrice produit de « `a` » par « `x` ».

**Exercice 10 — *Transposée*** Si $A = (a_{ij})$ est une matrice à $m$ lignes et $n$ colonnes, sa transposée ${}^tA$ est la matrice à $n$ lignes et $m$ colonnes dont les éléments sont les $a_{ji}$. Écrire une fonction qui calcule la transposée d'une matrice.

**Exercice 11 — *Produit*** Si $A = (a_{ij})$ est une matrice à $m$ lignes et $n$ colonnes et $B = (b_{jk})$ est une matrice à $n$ lignes et $p$ colonnes, le produit $AB$ est la matrice à $m$ lignes et $p$ colonnes définie par

$$AB = \begin{pmatrix} c_{0,0} & \cdots & c_{0,p-1} \\ \vdots & \ddots & \vdots \\ c_{m-1,0} & \cdots & c_{m-1,p-1} \end{pmatrix} \qquad \text{avec} \qquad c_{ik} = \sum_{j=0}^{n-1} a_{ij} b_{jk}$$

Écrire une fonction qui prend en arguments deux matrices $A$ et $B$ de tailles supposées compatibles (telles que le nombre de colonnes de $A$ est égal au nombre de lignes de $B$) et renvoie leur produit.

## 3 Polynômes

La série d'exercices qui suit a pour but final d'adapter l'algorithme d'Euclide de l'exercice 5 afin de calculer le pgcd de deux polynômes. Au cours de ces exercices, on représentera les polynômes à coefficients réels à l'aide de tableaux de doubles. Ainsi, $P = \sum_{i=0}^{n} a_i X^i$ sera représenté par un tableau « `t` » tel que `t[i]` $= a_i$ pour tout $i$ entre 0 et $n$. Remarquons qu'il y a plusieurs représentations possibles pour un même polynôme : on peut toujours élargir une représentation sous forme de tableau par un autre tableau plus long et complété par des zéros. Par exemple, le tableau

| | | |
|---|---|---|
| -3 | 0 | 1 |

et le tableau

| | | | | |
|---|---|---|---|---|
| -3 | 0 | 1 | 0 | 0 |

représentent tous les deux le polynôme $X^2 - 3$.
Il en résulte en particulier qu'on ne peut pas déduire directement le degré d'un polynôme à partir de la taille d'un tableau qui le représente.

**Exercice 12 — *Fonctions utilitaires***

1. Rappelons que sur les nombres flottants, il n'y a pas d'égalité fiable. Commencer par écrire une méthode « `egalite` » prenant deux arguments de type double et qui retourne « `true` » lorsqu'ils sont proches à une constante $\epsilon$ près (avec par exemple $\epsilon = 0.000001$).
2. Écrire une méthode « `litPolynome` » qui demande à l'utilisateur de taper les coefficients d'un polynôme et qui retourne sa représentation sous forme d'un tableau.
3. Écrire une méthode « `degrePolynome` » qui retourne le degré d'un polynôme. Par convention, le méthode retournera $-1$ dans le cas du polynôme nul.
4. Écrire une méthode « `creeMonome` » qui fabrique la représentation d'un monôme $aX^n$.
5. Écrire une méthode « `coefficient` » qui prend en paramètres un polynôme $P$ et un entier positif $n$, et qui retourne le coefficient de $X^n$ dans $P$. Pour un entier excédant la taille du tableau représentant le polynôme, la méthode doit retourner zéro.
6. Écrire une méthode d'affichage « `affichePolynome` ». Par exemple, elle affichera le polynôme $2X^2 - 3{,}5$ de la façon suivante : « `2.0*X^2-3.5` ».

**Exercice 13 — *Fonctions arithmétiques simples***

1. Écrire les méthodes de somme et de soustraction de polynômes.
2. Écrire la méthode de multiplication de deux polynômes. On rappelle que si $P = \sum_{i=0}^{n} a_i X^i$ et $Q = \sum_{j=0}^{m} b_j X^j$, alors $P.Q = \sum_{i=0}^{n+m} (\sum_{k=0}^{i} a_k b_{i-k}).X^i$. La méthode pourra utiliser deux boucles « `for` » imbriquées.

**Exercice 14 — *Division euclidienne*** Les polynômes à coefficients réels sont munis d'une opération de division euclidienne analogue à celle des entiers. On a la propriété suivante : si $A$ et $B$ sont deux polynômes (avec $B \neq 0$), il existe un unique couple de polynômes $Q$ (le quotient) et $R$ (le reste) tels que :

- le degré de $R$ est strictement inférieur à celui de $B$,
- $A = BQ + R$.

Le quotient et le reste d'une division euclidienne peuvent se calculer à la main en « posant » la division à la manière d'une division entière. Voici un exemple d'un tel calcul, la division de $2X^4 + 2X^2 + 1$ par $2X^2 + X$ :

$$
\begin{array}{llr|l}
2X^4 & +2X^2 & +1 & 2X^2+X \\ \cline{4-4}
2X^4+X^3 & & & X^2-(1/2)X+(5/4) \\ \cline{1-3}
\quad -X^3 & +2X^2 & +1 & \\
\quad -X^3-(1/2)X^2 & & & \\ \cline{1-3}
 & (5/2)X^2 & +1 & \\
 & (5/2)X^2+(5/4)X & & \\ \cline{1-3}
 & & -(5/4)X+1 &
\end{array}
$$

Programmer la méthode de division euclidienne. Elle repose sur une boucle `while` pour laquelle on exhibera un invariant. Justifier la correction de la méthode.

**Indication :** on pourra utiliser une variable contenant successivement chaque « reste intermédiaire » de l'opération de division (dans l'exemple ci-dessus : $2X^4 + 2X^2 + 1$, $-X^3 + 2X^2 + 1$, $(5/2)X^2 + 1$ et enfin le reste résultat $-(5/4)X + 1$).

**Remarque d'ordre pratique :** comme dans la suite nous serons uniquement intéressés par le reste de la division, la méthode se contentera de retourner le reste.

**Exercice 15 — *Algorithme d'Euclide*** On s'intéresse au pgcd d'un couple de polynômes $\neq (0,0)$. Attention, dans le cas des polynômes, il n'y a pas unicité du pgcd. Il y a une infinité de pgcd possibles pour un couple de polynômes, qui diffèrent par un coefficient de proportionnalité réel (non nul). On peut donc déduire l'ensemble des pgcd de deux polynômes à partir d'un pgcd quelconque.

Le but de l'exercice est de programmer l'algorithme d'Euclide (comme celui de l'exercice 5) pour les polynômes. La méthode est essentiellement la même que pour les entiers.

1. Quel argument permet de démontrer la terminaison de l'algorithme dans le cas des polynômes ?
2. Écrire la méthode « `pgcd` » basée sur cet algorithme (elle est construite autour d'une boucle `while`).
3. Exhiber un invariant pour la boucle `while` permettant de justifier la correction du programme.
4. Rédiger un programme complet qui calcule un pgcd de deux polynômes. On choisit d'afficher le pgcd dont le coefficient dominant est égal à 1.