MMFAI
Cours de programmation

TP numéro 2
semaine du 12 octobre 1998

Matrices, Arithmétique

1 Manipulation de matrices

Nous souhaitons nous constituer ici une librairie de fonctions utiles pour travailler sur des matrices de taille arbitraire.

Notons que en C on à le droit d'écrire:

```
double *v;

v=(double *)malloc(sizeof(double)*100);

v[23] = 0.0;
```

- on déclare la variable v comme pointeur sur données de type double
- on alloue de la mémoire pour contenir 100 éléments de type double (comme cette taille dépend de la machine, on a recours à la fonction sizeof() qui donne l'espace nécessaire pour mémoriser un type donné)
- on demande au système de se débrouiller avec l'arithmétique des pointeurs pour trouver l'adresse du 23ème élément du vecteur (qui est `v+23*sizeof(double)`) en écrivant `v[23]`.

Si on sait que la taille du vecteur v ne changera pas, on peut aussi écrire, pour cela, le programme suivant, qui est plus clair et pour lequel le système fait tout seul l'allocation de mémoire.

```
double v[100];

v[23] = 0.0;
```

Mais pour les matrices cela se passe différemment: on peut écrire

```
double v[10][10];

v[2][3] = 0.0;
```

- on déclare la variable v comme une matrice 10 pour 10
- on demande au système de calculer le pointeur au 3ème élément de la 2ème rangée (qui est `v+(2*10+3)*sizeof(double)`).

Mais pour cela, le système a besoin de connaître la dimension horizontale de la matrice (10), qui est déclarée explicitement. On ne peut pas donc s'attendre à des miracle quand on ne connaît pas à priori la taille de la matrice: en écrivant qque chose comme ce qui suit, on se fait justement jeter par le compilo

```
double *v;

v=(double *)malloc(sizeof(double)*votretaillepreferee);

v[2][3] = 0.0;
```

On doit alors s'organiser pour écrire des fonctions adaptés à ce cas.

Exercice 1.1 (get et set) *Écrivez des fonctions* `get(double *m,int i,int j, int n)` *et* `set(double *m,int i,int j, int n, double v)` *pour accéder et modifier une matrice de largeur n.*

Exercice 1.2 (manipulation de matrices) *Écrivez les fonctions suivantes*

- `double *id(int n)` qui renvoie la matrice identité de dimension n
- `double *madd(double *m1, double *m2, int m1rows, int m1cols)` qui additionne deux matrices de dimensions compatibles
- `double *mmult(double *m1, double *m2, int m1rows, int m1cols, int m2cols)` qui multiplie deux matrices de dimensions compatibles
- `void mprint(double *m, int mrows, int mcols)` qui imprime joliment une matrice de taille $mrows * mcols$ ligne par ligne
- `double *minput(int mrows, int mcols)` qui lit sur l'entrée standard une matrice de taille $mrows * mcols$ ligne par ligne

Réfléchissez à la façon de laquelle vous gérez la mémoire dans vos fonctions: peut-on libérer de la mémoire dans ces fonctions?

2 Fibonacci et théorème de Jones

On va essayer ici, avec le peu d'outils que l'on a en ce moments, d'écrire un premier programme qui vérifie le théorème de Jones vu en TD la semaine passée.
Pour cela, il nous faut d'abord le générateur rapide de nombres de Fibonacci.

Exercice 2.1 (Fibonacci: F_n en $log(n)$) *Écrivez une fonction* `int fibolog(int n)` *qui calcule F_n en temps logn en utilisant la multiplication de matrices vu au cours du TD précédant et les fonctions que vous venez d'écrire.*

Exercice 2.2 (Tester si un entier est un nombre de Fibonacci) *Écrivez un programme bête pour tester si un entier m est un nombre de Fibonacci F_n (retournez aussi n). Pouvez vous proposer un algorithme pas bête?*

Exercice 2.3 (Théorème de Jones) *Écrivez un programme qui affiche tous les couples (x, y) pour lesquels le polynôme*

$$2xy^4 + x^2y^3 - 2x^3y^2 - y^5 - x^4y + 2y$$

prend des valeurs positives, la valeur et le n tel que F_n est cette valeur positive.

Exercice 2.4 (Exponentiel modulo, pas fait la dernière fois) *Écrivez un programme qui lit trois entiers a, b, n et imprime $(a^b) mod\ n$. Pour cela, il peut être utile de savoir que C dispose d'opérateurs pour travailler sur les bits, comme* `<<` *et* `>>` *qui font du roulement a gauche ou à droite de la representation binaire d'un entier (par ex.* `1<<4` *= 16).*

N'oubliez pas de vous déconnecter avant d'abandonner le clavier (voir notes du stage Unix).