

## Atelier 3

### *Objectifs de formation*

- Représenter une matrice sous la forme d'un tableau et gérer les éléments de la matrice via les pointeurs.
- Maîtriser l'allocation dynamique de mémoire en C.
- Maîtriser la mesure du temps de calcul en C.
- Optimiser un code d'algèbre linéaire en remplaçant les indices de type entiers des boucles par des pointeurs.

### 1. : 30 mn

Soit un jeu de plateau où l'objectif est de conquérir la terre des 4 royaumes. Un joueur peut créer des armées composées de trolls, d'orcs, de gremlins et de loups-garous. Un joueur crée donc 4 armées pour chacun des royaumes définies dans le tableau ci-dessous :

	Trolls	Orcs	Gremlins	Loups-garous
armée 1	15	34	27	2
armée 2	8	23	10	1
armée 3	11	17	9	4
armée 4	8	65	45	7

Les créatures se nourrissent de viandes, de céréales, de légumes et de fruits. La ratio journalière de nourriture pour chaque créatures est donnée en kilos dans le tableau ci-dessous :

	Viande	Céréales	Légumes	Fruits
Trolls	1	2	2,5	2
Orcs	0,7	1	1,5	1
Gremlins	0,6	1,1	1,2	0,8
Dragons	10	15	7	5

Le joueur veut savoir, pour chaque aliment, combien de quantité il doit prévoir par jour pour chacune de ses armées. Le résultat est donné dans le tableau ci-dessous :

	Viande	Céréales	Légumes	Fruits
Armée 1	75	123,7	134,9	95,6
Armée 2	40,1	65	73,5	52
Armée 3	68,3	108,9	91,8	66,2
Armée 4	150,5	235,5	220,5	152

- Montrer que le problème conduit au calcul d'un produit matriciel.
- Écrire un programme permettant de résoudre le problème précédent à l'aide de la fonction `void prodmatmat(double *a, double *b, double *p, int n)` qui calcule le produit de deux matrices  $a$  et  $b$  d'ordre  $n$  stockées comme des vecteurs et retourne ce produit dans la matrice  $p$  stockée comme un vecteur.  
On utilisera une formulation classique de type `*(a + i*n + j)` pour désigner l'élément  $a_{ij}$  de la matrice  $a$ .  
Le tester sur le cas précédent.

## 2. : 60 mn - activité donnant lieu à un rendu

- Optimiser la fonction précédente sans effectuer de calcul d'adresse en utilisant les pointeurs.
- Écrire un programme qui :
  - (a) demande à l'utilisateur de rentrer une dimension au clavier sous la forme d'un entier  $n$ .
  - (b) crée dynamiquement trois matrices flottantes  $a$ ,  $b$  et  $p$  de taille  $n$  en double précision stockées comme des vecteurs.
  - (c) initialise les matrices  $a$  et  $b$  avec des valeurs aléatoires dans  $[-1, 1]$ . Pour cela on utilisera la fonction `int rand(void)` définie dans `stdlib.h` qui renvoie aléatoirement un entier compris entre 0 et `RAND_MAX`.
  - (d) calcule et compare les temps de calcul pour  $n = 10, 100$  et  $500$  entre la formulation classique et la formulation optimisée. Pour cela, on utilisera le type `clock_t` et la fonction `clock_t clock(void)` définis dans `time.h` qui renvoie le temps "absolu" en cycles (de type `clock_t`) du processeur.

## 3. : 90 mn

- Aller chercher sur internet l'algorithme de Strassen qui calcule de manière récursive le produit de deux matrices quand la dimension est une puissance de 2.
- Écrire une fonction non récursive  
`void strassen_2(float *a, float *b, float *res)`  
qui calcule le produit de deux matrices  $a$  et  $b$  de dimension 2 dans la matrice  $res$  par l'algorithme de strassen. Tester la fonction sur le produit

$$\begin{pmatrix} 2 & -3 \\ -8 & 4 \end{pmatrix} \times \begin{pmatrix} 5 & 2 \\ 11 & 5 \end{pmatrix} = \begin{pmatrix} -23 & -11 \\ 4 & 4 \end{pmatrix}.$$

- Programmer une fonction récursive  
`void strassen(float *a, float *b, float *res, int k)`

qui calcule le produit de deux matrices  $a$  et  $b$  de dimension  $2^k$  dans la matrice  $res$  par l'algorithme de Strassen. Tester la fonction sur le produit matriciel de l'activité 1.

*Conseils : vous aurez besoin*

- (a) d'une fonction qui fait la somme de deux matrices,*
- (b) d'une fonction qui fait la différence de deux matrices,*
- (c) d'une fonction qui fait le produit élémentaire de deux matrices de dimension 2.*

*La fonction strassen se décompose en 4 parties*

- (a) une allocation dynamique de mémoire de 21 matrices de dimension  $2^{k-1}$*
- (b) la construction des 8 matrices de dimension  $2^{k-1}$  à partir des matrices  $a$  et  $b$ ,*
- (c) la construction de 4 matrices de dimension  $2^{k-1}$ ,*
- (d) la reconstruction de la matrice produit à partir des 4 matrices précédentes.*

*Il est impératif de vous répartir le travail.*