AP1 TP

16

Semaine 16 : Révisions du DS2

16.1 Récursivité

On se propose d'utiliser la récursivité pour calculer la fonction puissance, notée $a^b = pow(a, b)$, de deux entiers a et b. Pour cela, deux formules de récurrence sont possibles :

$$\text{(A) } pow(a,b) = \begin{cases} 1 \text{ si } b = 0 \\ pow(a, b-1) \times a \text{ sinon} \end{cases}$$

$$\text{(B) } pow(a,b) = \begin{cases} 1 \text{ si } b = 0 \\ pow(a \times a, b/2) \text{ si } b > 0 \text{ pair} \\ pow(a \times a, (b-1)/2) \times a \text{ sinon} \end{cases}$$

Question 1
Ecrivez la fonction exponentielle correspondant à la formule de récurrence (A).

Question 2
Ecrivez la fonction exponentielle correspondant à la formule de récurrence (B).

16.2 Manipulation de tableaux

On se propose de faire un programme qui sauvegarde les notes d'étudiants sous forme d'un tableau à deux dimensions. Ce tableau comportera une ligne par étudiant (au maximum `MAX_ETUDIANTS`) et autant de colonnes que de notes à enregistrer (au maximum `MAX_NOTES`).

On vous donne une partie de ce code source.

```
#include <iostream>
#include <cstdlib>
using namespace std;
#define MAX_ETUDIANTS 30
#define MAX_NOTES 6
#define NOTE_MAXIMALE 20

void saisir_notes(double notes[][MAX_NOTES], int &nb_etu, int &nb_notes){
}

double moyenne_note(double notes[][MAX_NOTES], int numero_note, int nb_etu){
}

double moyenne_etudiant(double notes[][MAX_NOTES], int numero_etudiant, int
    nb_notes){
}

int main(){
  double notes_etu [MAX_ETUDIANTS][MAX_NOTES];
  int nb_etu, nb_notes;
  saisir_notes(notes_etu, nb_etu, nb_notes);
  int numero_etu = 1;
  cout << "Moyenne de l'etudiant "<< numero_etu << " : "
     <<moyenne_etudiant(notes_etu, numero_etu, nb_notes) <<endl;
  int numero_note =2;
  cout << "Moyenne des etudiants sur la note "<< numero_note << " : "
     <<moyenne_note(notes_etu, numero_note, nb_etu) <<endl;
}
```

Question 3
Fournir le code de la procédure `void saisir_notes(double notes[][MAX_NOTES], int &nb_etu, int &nb_notes)` qui va demander à l'utilisateur de saisir le nombre d'étudiants ainsi que le nombre de notes (ce nombre de notes sera le même pour chaque étudiant).
Ensuite, pour chaque étudiant, l'utilisateur saisit les notes une à une sur l'entrée standard (le clavier).
Bien sûr, chaque valeur fournie par l'utilisateur fera l'objet d'une vérification.

Question 4
Fournir le code de la fonction `double moyenne_note(double notes[][MAX_NOTES], int numero_note, int nb_etu)` qui retourne la moyenne constatée pour l'exercice `numero_note`.

Question 5
Fournir le code de la fonction `double moyenne_etudiant(double notes[][MAX_NOTES], int numero_etudiant, int nb_notes)` qui retourne la moyenne de l'étudiant d'indice `numero_etudiant` sur la totalité de ses notes.

16.3 Le jeu de la vie

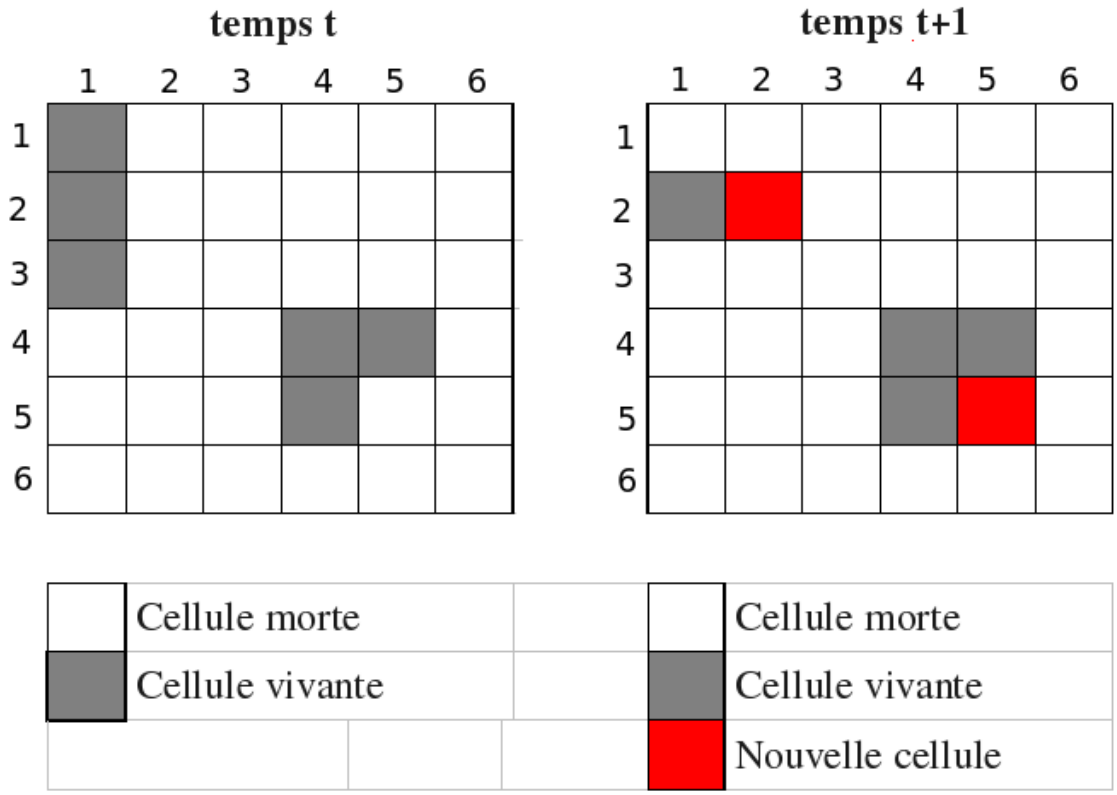
Le jeu de la vie (The Game of Life), inventé en 1970 par John Conway n'est pas réellement un jeu. Il est généralement représenté par un quadrillage du plan (damier) dont les cases (cellules) sont soit blanches (mortes) soit grises

(vivantes).

L'évolution des cellules suit des règles simples décrites dans le tableau ci-dessous. Une cellule a au maximum huit voisines. L'état d'une cellule à l'instant $t+1$ dépend de son état et de celui de ses voisines à l'instant t.

cellule à l'instant t	**cellule à l'instant** $t+1$
Si la cellule est morte et qu'elle possède trois voisines vivantes	vivante
Si la cellule est vivante et qu'elle possède deux ou trois voisines vivantes	vivante
Autres cas	morte

L'image suivante illustre une itération du jeu de la vie. Vous observerez qu'entre deux itérations, 2 cellules sont mortes et deux cellules sont apparues. En effet, en $t+1$ la cellule $(2,2)$ est apparue car en t elle possède trois voisines vivantes (les cellules $(1,1)$, $(2,1)$ et $(3,1)$). La cellule $(3,1)$ est vivante en t mais meurt en $t+1$ car en t elle ne possède qu'une cellule voisine vivante (idem pour la cellule $(1,1)$). En t, la cellule $(5,5)$ est morte et a 3 voisines vivantes, elle sera donc elle même vivante en $t+1$
Entre deux itérations, chaque cellule est donc testée. L'ordre dans lequel sont effectués les tests n'est pas important.



Afin de représenter cette grille, nous définissons le type `damier` comme étant une matrice [6][6] de booléens. Si la case à la position (i,j) contient la valeur `vrai` alors la cellule correspondante est vivante (si elle contient la valeur `faux` alors la cellule est morte).

Question 6
Définir le type `damier`.

Question 7
On souhaite générer un damier contenant un certain nombre de cellules vivantes. Pour cela, on va comparer, pour chaque cellule, un nombre tiré aléatoirement (entre 0 et 1) à une probabilité p définie par l'utilisateur (aussi entre 0 et 1). Si le nombre tiré aléatoirement est plus petit que p, alors la cellule est vivante, sinon la cellule est morte.
Ecrire une fonction `générerDamier` prenant un entrée un `damier` permettant de remplir aléatoirement ce damier. Cette action doit tout d'abord demander à l'utilisateur la probabilité p qu'une cellule soit vivante. Puis `générerDamier` compare pour chaque cellule la probabilité p à un nombre tiré aléatoirement pour déterminer si la cellule est vivante.

Question 8
Ecrire une fonction `nombreVoisin` prenant en entrée un `damier` et deux entiers `i` et `j` et retournant le nombre de cellules vivantes voisines de la cellule à la position (i, j). On considère que le damier passé en entrée est déjà initialisé.

Question 9
Ecrire une fonction `nouvelleGénération` prenant en paramètre un damier `damier_t` et un damier `damier_t+1`, permettant de générer le damier au temps $t + 1$ (`damier_t+1`) à partir du damier au temps t (`damier_t`). On considère que le damier passé en entrée est déjà initialisé et que votre fonction `nombreVoisin` fonctionne.