

Semaine 16 : Révisions du devoir final

16.1 Etude d'un code C++ (4 points)

Soit le programme suivant :

```

1  #include<iostream>
2  #include<cstdlib>
3  using namespace std;
4  const int taille=5;
5  void TinkyWinky(int *n, int * &ptr)
6  {
7      ptr=new int[ taille ];
8      int cpt=0;
9      do
10     {
11         *(ptr+cpt)=n[cpt];
12         cpt++;
13     }
14     while (cpt < taille);
15 }
16 void Dipsy(int *n)
17 {
18     for (int i=0; i< taille;i++){
19         n[i]=i;
20     }
21 }
22 void LaaLaa(int *n)
23 {
24     for (int i=0; i< taille;i++)
25         cout << *(n+i);
26     cout << endl;
27 }
28 void Po(int *n, int*b)
29 {
30     int *c=n;
31     for (int i=0; i< taille-1;i++)
32         n++;
33     int i=0;
34     while (c!=n){
35         b[i]=*n;
36         n--;
37         i++;
38     }
39 }
40 int main()
41 {
42     int *p;
43     int *tabA=new int[ taille ];
44     int *tabB;
45     Dipsy(tabA);
46     LaaLaa(tabA);
47     TinkyWinky(tabA,tabB);
48     LaaLaa(tabB);
49     Po(tabA,tabB);
50     LaaLaa(tabB);
51     return EXIT_SUCCESS;
52 }
```

Question 1

Indiquez ce qu'affiche l'exécution du programme ci-dessus.

Question 2

Le code donné ci-dessus compile parfaitement et ne génère pas d'erreur d'exécution. Cependant, il faudrait rajouter des lignes de code pour que le programme soit réellement correct. Indiquez quelles sont ces lignes de code. Précisez également les numéros de lignes où ce code doit être inséré.

16.2 Récursivité (Exalgo - 4 points)

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

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

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

Question 3

Donnez **en ExAlgo** la fonction exponentielle correspondant à la formule de récurrence (A).

Question 4

Donnez **en ExAlgo** la fonction exponentielle correspondant à la formule de récurrence (B).

Question 5

Quelle est, selon vous, la récurrence la plus efficace ? Justifiez avec l'exemple $2^7 = \text{pow}(2, 7)$.

16.3 Manipulation de tableaux (C++ - 6 points)

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.

```
1  #include <iostream>
2  #include <cstdlib>
3  using namespace std;
4  #define MAX_ETUDIANTS 30
5  #define MAX_NOTES 6
6  #define NOTE_MAXIMALE 20
7
8  void saisir_notes(double notes[][MAX_NOTES], int &nb_etu, int &nb_notes){
9  }
10
11 double moyenne_note(double notes[][MAX_NOTES], int numero_note, int nb_etu){
12 }
13
14 double moyenne_etudiant(double notes[][MAX_NOTES], int numero_etudiant, int
    nb_notes){
15 }
16
17 int main(){
18     double notes_etu [MAX_ETUDIANTS][MAX_NOTES];
19     int nb_etu, nb_notes;
```

```

20  saisir_notes(notes_etu, nb_etu, nb_notes);
21  int numero_etu = 1;
22  cout << "Moyenne de l'etudiant " << numero_etu << " : "
    << moyenne_etudiant(notes_etu, numero_etu, nb_notes) << endl;
23  int numero_note = 2;
24  cout << "Moyenne des etudiants sur la note " << numero_note << " : "
    << moyenne_note(notes_etu, numero_note, nb_etu) << endl;
25  }

```

Question 6

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 7

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 8

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.

16.4 Le jeu de la vie EXALGO - (7 points)

Remarque 1 : Toutes les questions sont indépendantes, il n'est pas NECESSAIRE d'avoir fait une question pour pouvoir utiliser la fonction/action correspondante

Remarque 2 : Ne pas oublier de préciser si les paramètres sont passés en Entrée, Sortie ou Entrée/Sortie.

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.

temps t							temps t+1						
	1	2	3	4	5	6		1	2	3	4	5	6
1													
2													
3													
4													
5													
6													

	Cellule morte			Cellule morte
	Cellule vivante			Cellule vivante
				Nouvelle cellule

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 9

Définir le type `damier`.

Question 10

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 action `générerDamier` prenant en 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 (utiliser la fonction `aléatoire()` : réel retournant un nombre aléatoire entre 0 et 1) pour déterminer si la cellule est vivante.

Question 11

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 12

Ecrire une action `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.