UNIVERSITÉ PARIS-SUD 2018-2019 première session d'examen

Licence 3 et Magistère 1ère année de Physique Fondamentale

Examen d'informatique

mardi 7 mai 2019 8h30 à 10h30 (2 heures)

— *Aucun document n'est autorisé.*

— *Les programmes doivent être écrits en C/C++.*

— *Respecter les notations de l'énoncé : par exemple la variable notée n_t dans l'énoncé devra être écrite* `nt` *dans un programme, n'' devra être écrite* `ns`.

— *L'utilisation de fonctions est laissée à l'appréciation de l'étudiant, sauf dans les cas où elle est imposée.*

— *On suppose que tous les* `#include<iostream> ... #include<math.h> using namespace std;` *nécessaires sont sous-entendus, il n'y a pas à les écrire.*

— *Ne pas faire lire les données au clavier ou dans un fichier par un* `cin >>`*, ou l'équivalent pour un fichier, sauf si cela est explicitement demandé. Par défaut, les données seront donc fournies dans le programme lui-même, par des instructions du type :*

```
dx=0.01; a=1.7; b=1.1;
```

— *Tous les tableaux dont il s'agit dans l'énoncé sont des tableaux dynamiques, à déclarer avec un ou plusieurs* `malloc`.

— *Dans chaque exercice on suppose que le programme principal (s'il y en a un) et les fonctions (s'il y en a) sont écrits dans un unique fichier.*

— *Les exercices sont indépendants les uns des autres.*

— ***Le nombre de points (sur un total de 46) attribué à chaque question est indiqué entre parenthèses (c'est un barème indicatif). Deux points sont réservés à la qualité de la présentation, pour un total de 48 points. La note finale sera le nombre de points total divisé par deux (avec a priori 24 comme note maximale, qui sera arrondie à 20).***

— *Le corrigé pourra être consulté sur le site du cours ultérieurement.*

1. Heures de passage du RER B

Nous avons un fichier texte, appelé *RERB.dat*, qui contient les vraies heures de passage du RER B à la gare d'Orsay-Ville le vendredi 19 avril. Ce fichier a le format suivant : sur la première ligne il y a le nombre d'heures de passage dans le fichier, puis sur les lignes suivantes il y a les heures de passage, une par ligne, triées par ordre croissant, représentées comme des doubles. La partie du double avant le point décimal indique l'heure (entre 0 et 23) tandis que la partie après le point décimal indique les minutes (entre 0 et 59). Le début du fichier est donc par exemple :

```
98
5.12
5.22
5.38
5.54
6.08
⋮
```

Le but de l'exercice est de calculer plusieurs statistiques concernant ces heures de passage : les temps minimal et maximal entre deux trains, le temps moyen entre deux trains, etc.

Question 1 : (2 points)

Écrire une fonction, dont le prototype est `int convheure(double h)`, qui convertit une heure de passage `h`, représentée comme un double comme expliqué ci-dessus, en nombre de minutes depuis 0h00.

Réponse à la question 1 :

```
int convheure(double h){
  return 60*(int)h + (h-(int)h)*100;  // A noter : 60*h + ... n'est pas correct
}
```

On suppose que `t` est un tableau à un indice de `n` entiers qui représentent les heures de passage des trains (en ordre croissant), converties en minutes avec la fonction précédente.

Question 2 : (2 points)

Écrire une fonction, dont le prototype est `int maxtemps(int* t, int n)`, qui calcule le temps maximal (en minutes) entre deux trains consécutifs.
Écrire également une fonction, appelée `mintemps`, qui calcule le temps minimal (en minutes) entre deux trains consécutifs.

Réponse à la question 2 :

```
int maxtemps(int* t, int n){
  int i, max = -1;
  for(i = 1; i < n; i++)
    if(t[i]-t[i-1] > max)
      max = t[i] - t[i-1];
  return max;
}

int mintemps(int* t, int n){
  int i, min = 2000;
  for(i = 1; i < n; i++)
    if(t[i]-t[i-1] < min)
      min = t[i] - t[i-1];
  return min;
}
```

Question 3 : (1 point)

Écrire une fonction, appelée `moytemps`, qui calcule le temps moyen (en minutes) entre deux trains consécutifs. Le résultat de cette fonction sera un double, parce que la valeur moyenne peut bien sûr être une valeur non entière.

Réponse à la question 3 :

```
double moytemps(int* t, int n){
  return (double)(t[n-1]-t[0])/(n-1);
}
```

Question 4 : (2 points)

Écrire une fonction, appelée `ecatemps`, qui calcule l'écart-type (en minutes) des temps entre deux trains consécutifs. Pour rappel, l'écart-type d'une quantité aléatoire x est donné par $\sqrt{\langle x^2\rangle - \langle x\rangle^2}$.

Réponse à la question 4 :

```
double ecatemps(int* t, int n){
  int i;
  double moy, eca = 0;
  for(i = 1; i < n; i++)
    eca += (t[i] - t[i-1])*(t[i] - t[i-1]);
  moy = moytemps(t,n);
  return sqrt(eca/(n-1) - moy*moy);
}
```

À noter : on ne peut pas utiliser la fonction `moytemps` avec un tableau qui contient les `t[i]*t[i]` pour calculer la moyenne des carrés, parce que cela reviendrait à calculer la moyenne de `t[i]*t[i] - t[i-1]*t[i-1]` au lieu de `(t[i]-t[i-1])*(t[i]-t[i-1])`.

Question 5 : (3 points)

Écrire un programme principal (fonction `main`) qui lit le contenu du fichier *RERB.dat*, convertit les valeurs en minutes en utilisant la fonction `convheure` de la question 1, et met les valeurs résultantes dans le tableau `t` (qu'il faut bien sûr d'abord déclarer avec `malloc`). Puis il calcule les temps maximal, minimal et moyen et l'écart-type en faisant appel aux fonctions des questions précédentes, et il affiche les résultats à l'écran.

Réponse à la question 5 :

```
int main(){
  fstream fich;
  int i, n, *t;
  double h;

  fich.open("RERB.dat", ios::in);
  fich >> n;                          // Ici le changement de la question 6
  t = (int*)malloc(n * sizeof(int));
  for(i = 0; i < n; i++){
    fich >> h;
    t[i] = convheure(h);
  }
  fich.close();
  cout << "Maximal : " << maxtemps(t,n) << endl;
  cout << "Minimal : " << mintemps(t,n) << endl;
  cout << "Moyen : " << moytemps(t,n) << endl;
  cout << "Ecart-type : " << ecatemps(t,n) << endl;
  free(t);
  return 0;
}
```

Question 6 : (1 point)

Que faudrait-il changer/ajouter au programme précédent si le fichier ne contenait pas le nombre total d'heures de passage sur la première ligne, et le nombre d'heures de passage dans le fichier était donc inconnu ?

Réponse à la question 6 :

Si l'on n'a aucune idée du nombre de lignes dans le fichier, on est obligé de lire le fichier deux fois : une première fois pour compter le nombre de lignes, dont on a besoin pour `malloc`, puis une deuxième fois pour remplir le tableau.

Au lieu de la ligne `fich >> n;` on met les cinq lignes suivantes :

```
n = 0;
while(fich >> h)
  n++;
fich.close();
fich.open("RERB.dat", ios::in);
```

Le reste du programme ne change pas.

2. Pointeurs

Soit le programme suivant, où il manque les opérateurs `*` et `&` :

```
 1. #include<iostream>
 2. #include<stdlib.h>
 3. using namespace std;
 4. int X = 1;
 5. void echange(int a, int b) {
 6.   int c;
 7.   c = a;
 8.   a = b;
 9.   b = c;
10. }
11. void calcul(int t, int a) {
12.   X *= 2;
13.   a = t[0] + t[1] + X;
14. }
15. int declar1() {
16.   int t = (int)malloc(2 * sizeof(int));
17.   t[0] = 2;
18.   t[1] = 4;
19.   X += 2;
20.   return t;
21. }
22. void declar2(int t) {
23.   t = (int)malloc(2 * sizeof(int));
24.   t[0] = 3;
25.   t[1] = 6;
26. }
27. int main() {
28.   int a = 1, b = 2;
29.   echange(a,b);
30.   cout << a << " " << b << endl;
31.   int t;
32.   t = declar1();
33.   calcul(t,a);
34.   cout << a << endl;
35.   declar2(t);
36.   t += 1;
37.   calcul(t,b);
38.   cout << b << endl;
39.   free(t);
40.   return 0;
41. }
```

Question 1 : (5 points)

Ajouter des opérateurs `*` et `&` aux bons endroits et en bon nombre pour créer un programme qui tourne sans messages d'erreur. Vous devez aussi ajouter des parenthèses autour des pointeurs si elles sont indispensables. Il est interdit de faire d'autres changements au programme. Si vous avez des fonctions qui ne font rien, c'est faux ! Pour faire cela, vous copiez sur votre feuille seulement les lignes qui sont à changer, avec leur numéro. Marquer clairement ce que vous avez ajouté dans ces lignes.

Réponse à la question 1 :

```
 5. void echange(int* a, int* b) {
 7.   c = *a;
 8.   *a = *b;
 9.   *b = c;
11. void calcul(int* t, int* a) {
13.   *a = t[0] + t[1] + X;
15. int* declar1() {
16.   int* t = (int*)malloc(2 * sizeof(int));
```

```
22. void declar2(int** t) {
23.   *t = (int*)malloc(2 * sizeof(int));
24.   (*t)[0] = 3;
25.   (*t)[1] = 6;
29.   echange(&a,&b);
31.   int* t;
33.   calcul(t,&a);
35.   declar2(&t);
36.   *t += 1;        // *t est identique a t[0]
37.   calcul(t,&b);
```

À noter : il faut un pointeur sur un pointeur dans la ligne 22. Sinon `t` serait juste une variable locale dans la fonction `declar2` et l'adresse qu'on y mettrait à la ligne 23 serait perdue à la fin de la fonction et on n'aurait alors pas accès à ce tableau dans le reste du programme.

Question 2 : (3 points)

Indiquer ce qui sera affiché à l'écran par les 3 commandes `cout` sur les lignes 30, 34 et 38.
Enfin, quelle commande faudrait-il ajouter au programme, et à quel endroit, pour écrire un programme plus propre ?

Réponse à la question 2 :

Ligne 30 : `2 1`
Ligne 34 : `12`
Ligne 38 : `22`

Il faut aujouter la commande `free(t);` juste avant la ligne `35. declar2(&t);`. Sinon la mémoire réservée par la fonction `declar1` restera bloquée et inaccessible pour toujours (jusqu'à la fin du programme).

3. Particule au voisinage d'un trou noir

Le 10 avril 2019, la première photo d'un trou noir a été publiée par l'*Event Horizon Telescope* (EHT), validant par la même occasion une nouvelle fois la théorie de la relativité générale d'Einstein. Afin de caractériser de tels objets, cette théorie décrit la gravitation comme étant une force apparente, due à la déformation de la géométrie de l'espace-temps.

Nous nous intéressons au mouvement d'une particule de masse $m = 1$ au voisinage d'un trou noir.[1] Nous plaçons l'origine de notre repère au centre du trou noir. Comme le problème est à symétrie sphérique, il peut être ramené à un problème plan. Dans ce cas, les coordonnées radiale et angulaire de la particule sont respectivement notées r et ϕ, et son temps propre (tel que mesuré par une horloge qui serait accrochée à la particule) noté t. Les équations du mouvement en fonction de t telles que décrites par la relativité générale deviennent alors

$$\frac{dr}{dt} = \left(1 - \frac{R_s}{r}\right) p_r, \tag{1a}$$

$$\frac{d\phi}{dt} = \frac{p_\phi}{r^2}, \tag{1b}$$

$$\frac{dp_r}{dt} = \frac{p_\phi^2}{r^3} - \frac{R_s}{2r^2}\left[\left(1 - \frac{R_s}{r}\right)^{-2} E^2 + p_r^2\right]. \tag{1c}$$

Les grandeurs E, p_r, et p_ϕ sont respectivement l'énergie, la quantité de mouvement radiale, et la quantité de mouvement angulaire de la particule. On définit aussi la constante R_s, le rayon de l'horizon du trou noir (ou rayon de Schwarzschild). Les quantités E et p_ϕ sont conservées au cours du mouvement.

Afin que le problème reste physique, le mouvement de la particule doit respecter la condition de conservation d'énergie-impulsion, qui peut se réécrire ici comme

$$p_r = -\left(1 - \frac{R_s}{r}\right)^{-1} \sqrt{-1 + \left(1 - \frac{R_s}{r}\right)^{-1} E^2 - \frac{p_\phi^2}{r^2}}. \tag{2}$$

La méthode d'Euler explicite $\vec{q}(t+dt) = \vec{q}(t) + \dot{\vec{q}}(t)\, dt$ est dite "à pas simple", dans ce sens où elle n'utilise que l'information du point actuel t pour calculer la fonction $\vec{q}$ au point suivant $t + dt$. Nous allons implémenter une méthode dite "à pas lié", qui tire profit de l'information des valeurs calculées précédemment afin de calculer la valeur suivante. Pour cela, on fait appel à la méthode d'Adams-Bashforth d'ordre 2 (AB2), définie comme suit :

$$\vec{q}(t + dt) = \vec{q}(t) + \left[3\dot{\vec{q}}(t) - \dot{\vec{q}}(t - dt)\right] \frac{dt}{2}. \tag{3}$$

1. On travaille en unités naturelles pour lesquelles la vitesse de la lumière et la constante de Newton valent $c = 1$ et $G = 1$.

Puisque cette méthode requiert le calcul de l'itération précédente, on réalisera la première itération à l'aide de la méthode d'Euler. Les itérations suivantes sont ensuite calculées à l'aide de AB2.

Question 1 : (1 point)

Écrire une fonction avec le prototype `double getpr(double r)` qui renvoie la valeur de p_r en fonction de la coordonnée radiale r, basée sur l'équation (2). On déclarera en global les variables R_s, E, et p_ϕ.

Réponse à la question 1 :

```
double Rs=2., E=0.97, pf=4.1;     // Valeurs donnees en question 4

double getpr(double r){
  return -1./(1.-Rs/r)*sqrt(-1.+E*E/(1.-Rs/r)-pf*pf/r/r);
}
```

Question 2 : (2 points)

Écrire une fonction avec le prototype `void system(double* q, double* qp, int n)` qui prend en argument r, ϕ et p_r combinés dans un pointeur `q` de taille $n = 3$, et qui renvoie leurs dérivées dans un pointeur `qp` sur base du système d'équations différentielles défini par les équations (1).

Réponse à la question 2 :

```
void system(double* q, double* qp, int n){
  double h = 1.-Rs/q[0];
  qp[0] = h*q[2];
  qp[1] = pf/q[0]/q[0];
  qp[2] = pf*pf/q[0]/q[0]/q[0] - Rs/2./q[0]/q[0]*(E*E/h/h+q[2]*q[2]);
}
```

Question 3 : (3 points)

Écrire une fonction avec le prototype `void AB2(void (*syst)(double*,double*,int),double* q,double* qp,double dt,int n)` qui réalise une itération de la méthode Adams-Bashforth 2. Le pointeur `double* qp` sert à la fois à récupérer le résultat précédent $\dot{\vec{q}}(t - dt)$, et à renvoyer celui du pas courant, $\dot{\vec{q}}(t)$.

Réponse à la question 3 :

```
void AB2(void (*syst)(double*,double*,int), double* q, double* qp, double dt, int n){
  int i;
  double* qpt = (double*)malloc(n * sizeof(double));
  for(i = 0; i < n; i++)
    qpt[i] = qp[i];
  syst(q,qp,n);
  for(i = 0; i < n; i++)
    q[i] = q[i] + (3*qp[i] - qpt[i])*dt/2.;
  free(qpt);
}
```

Question 4 : (4 points)

Écrire un programme principal qui résout les équations du mouvement en 10 000 points de temps de $t = 0$ jusqu'à $t_{\text{fin}} = 100$ inclus, en utilisant la méthode AB2. Ce programme doit inclure la première itération calculée par la méthode d'Euler. On prendra $R_s = 2$, $p_\phi = 4.1$ et $E = 0.97$ et comme conditions initiales $\phi_0 = 0$ et $r_0 = 15$.

Le programme doit écrire les valeurs de t, r et ϕ de chaque itération dans un fichier nommé *trounoir.res*.

Réponse à la question 4 :

```
int main(){
  int i, n=3, N=10000;
  double t=0., tfin=100., dt=(tfin-t)/(N-1);
  double* q = (double*)malloc(n * sizeof(double));
  double* qp = (double*)malloc(n * sizeof(double));
  fstream fich("trounoir.res", ios::out);

  q[0] = 15.; q[1] = 0.;
  q[2] = getpr(q[0]);
  fich << t << " " << q[0] << " " << q[1] << endl;
  system(q,qp,n);
  for(i = 0; i < n; i++)
```

```
    q[i] = q[i] + qp[i]*dt;
  t += dt;
  for(i = 1; i < N; i++){
    fich << t << " " << q[0] << " " << q[1] << endl;
    AB2(system,q,qp,dt,n);
    t += dt;
  }                 // On ajoute les lignes de la question 5 avant cette ligne-ci.
  fich.close();
  free(q); free(qp);
  return 0;
}
```

Question 5 : (1 point)

Ajouter une condition au programme précédent afin que le calcul s'arrête et un message soit affiché à l'écran lorsque la particule est absorbée par le trou noir ($r < R_s$).
Ajouter également un test qui affiche un avertissement et le numéro de l'itération à l'écran (mais n'arrête pas le calcul) si la différence entre le $|p_r|$ calculé directement par AB2 et le $|p_r|$ calculé avec la fonction de la question 1 devient plus grande que 0.05 (il faut comparer les valeurs absolues, parce que l'équation (2) donne seulement la solution négative).

Réponse à la question 5 :

On ajoute à la fin de la deuxième boucle `for` :

```
if(q[0] < Rs){
  cout << "Particule tombee dans le trou noir !" << endl;
  break;
}
if(fabs(fabs(q[2])-fabs(getpr(q[0]))) > 0.05)
  cout << i << " : probleme avec p_r !" << endl;
```

4. Formation de cristaux dans un baromètre de FitzRoy

Mise en contexte (lecture facultative)

Les verres-de-tempête, aussi appelés baromètres de FitzRoy, sont des bouteilles de verre contenant un mélange de camphre, d'eau, et d'éthanol. Ce mélange est en proportions telles que le camphre soit très proche de sa température de solubilisation quand la température ambiante avoisine les 20°C. De cette manière, quand la température diminue ou augmente, on observe la formation ou la disparition de cristaux. L'aspect des cristaux contenus dans ces bouteilles a été utilisé par les marins pour prédire la météo au cours du XIX^e siècle. On sait maintenant que ces cristaux ne renseignent pas sur la météo future, mais sur la météo passée. En effet, la forme des cristaux dépend de la concentration en camphre et de la vitesse de diffusion des molécules en solution pendant leur formation.

Nous allons ici nous intéresser à la formation des cristaux dans ce type d'objet. Le point où le cristal commence à se former s'appelle site de nucléation. Dans notre cas, les sites de nucléation seront des ions que l'on ajoute au mélange. Ensuite, la formation d'un cristal suit trois étapes :

— diffusion des atomes ou molécules qui composent le cristal dans le milieu environnant jusqu'à la surface d'un cristal déjà formé,
— diffusion des atomes sur la surface du cristal jusqu'à un site d'accroche,
— cristallisation et évacuation de la chaleur générée lors de l'accroche.

FIGURE 1 – Formation de cristaux dans un baromètre de FitzRoy.

Modélisation

Dans cet exercice, nous allons modéliser la formation d'un cristal de camphre autour d'un site de nucléation en deux dimensions. Notre système est un échantillon carré de côté N_{tab}, qui contient N_{cam} molécules de camphre. Les molécules peuvent seulement se trouver aux noeuds de la grille carrée.

Question 1 : (2 points)

Déclarer deux variables globales : `N_tab = 39` et `N_cam`. Puis écrire le début du programme principal, où vous commencez par demander à l'utilisateur d'entrer la concentration `conc` de camphre (valeur réelle entre 0 et 1 ; utiliser `cin`) et par calculer la valeur de `N_cam`, donnée par $(N_{\text{tab}} - 2)^2 \times$ `conc` (on va garder une bordure vide

autour de notre système, ce qui explique le -2).

Enfin, écrire les commandes pour déclarer et réserver la mémoire pour deux tableaux d'entiers à deux dimensions :

— l'un, appelé `tab`, représente notre système, l'échantillon carré de côté N_{tab} ;

— le second, appelé `cam`, de N_{cam} lignes et 2 colonnes, contiendra les coordonnées des molécules de camphre dans l'échantillon.

Réponse à la question 1 :

```
int N_tab = 39, N_cam;

int main(){
  int i, **tab, **cam;
  double conc;

  cout << "Concentration de camphre ? ";
  cin >> conc;
  N_cam = (N_tab-2) * (N_tab-2) * conc;
  tab = (int**)malloc(N_tab * sizeof(int*));
  for(i = 0; i < N_tab; i++)
    tab[i] = (int*)malloc(N_tab * sizeof(int));
  cam = (int**)malloc(N_cam * sizeof(int*));
  for(i = 0; i < N_cam; i++)
    cam[i] = (int*)malloc(2 * sizeof(int));
```

On choisira dans la suite, d'inscrire un 1 dans le tableau `tab` quand on voudra indiquer la présence d'une molécule de camphre libre de diffuser, un 2 quand on voudra représenter une molécule qui fait déjà partie du cristal, et un 0 sinon.
Astuce générale : À part la mise à zéro initiale et l'affichage, il ne sera jamais nécessaire de parcourir tout le tableau `tab`. Pour effectuer des opérations sur les molécules, il est beaucoup plus pratique de parcourir le tableau `cam`.

Question 2 : (3 points)

Écrire une fonction de type `void`, appelée `initialisation`, qui prend en entrée les deux tableaux définis dans la question 1. Cette fonction doit

— initialiser le tableau `tab` à zéro ;

— placer un 2 au centre du tableau `tab` pour représenter le site de nucléation, et mettre ses coordonnées dans la première ligne du tableau `cam` ;

— tirer aléatoirement les coordonnées initiales des $N_{\text{cam}} - 1$ molécules de camphre restantes et les écrire dans le tableau `cam`, où l'on veillera à ne placer une molécule que sur un site libre et à exclure la bordure de `tab`, donc les emplacements avec une coordonnée égale à 0 ou à $N_{\text{tab}} - 1$;

— inscrire des 1 à l'emplacement des molécules de camphre dans le tableau `tab` (hors celle au centre).

Réponse à la question 2 :

```
void initialisation(int** tab, int** cam){
  int i, j, k;
  for(i = 0; i < N_tab; i++)
    for(j = 0; j < N_tab; j++)
      tab[i][j] = 0;
  tab[N_tab/2][N_tab/2] = 2;
  cam[0][0] = N_tab/2;
  cam[0][1] = N_tab/2;
  for(k = 1; k < N_cam; k++){
    do {
      i = (N_tab-2) * drand48() + 1;
      j = (N_tab-2) * drand48() + 1;
    } while(tab[i][j] > 0);
    cam[k][0] = i;
    cam[k][1] = j;
    tab[i][j] = 1;
  }
}
```

Question 3 : (1 point)

Écrire une fonction appelée `conc_cam_crist` qui calcule quelle fraction des molécules de camphre est cristallisée (est un 2).

Réponse à la question 3 :

```
double conc_cam_crist(int** tab, int** cam){
  int i, N=0;
  for(i = 0; i < N_cam; i++)
    if(tab[cam[i][0]][cam[i][1]] == 2)
      N++;
  return (double)N / N_cam;
}
```

Dans la suite de notre modélisation, nous allons faire une hypothèse très forte. Nous allons supposer que dès qu'une molécule de camphre arrive à proximité d'une molécule appartenant déjà au cristal, alors elle est immédiatement piégée et appartient elle-même au cristal.

Question 4 : (4 points)

Écrire une fonction `void`, appelée `cristallisation`, qui prend en entrée les deux tableaux définis dans la question 1 et qui cristallise toutes les molécules libres (les 1 dans le tableau `tab`) au contact du cristal (les 2). On définit qu'il y a contact si au moins l'un des 8 sites voisins les plus proches du 1 contient un 2. La cristallisation ne progresse qu'une couche par itération : une molécule qui vient d'être piégée par le cristal ne peut cristalliser ses voisines qu'à l'itération suivante. [2]

Réponse à la question 4 :

```
void cristallisation(int** tab, int** cam){
  int k, i, j, ii, jj;
  for(k = 0; k < N_cam; k++){
    ii = cam[k][0];
    jj = cam[k][1];
    i = ii - 1;
    while(tab[ii][jj] == 1 && i <= ii+1){
      j = jj - 1;
      while(tab[ii][jj] == 1 && j <= jj+1){
        if(tab[i][j] == 2)
          tab[ii][jj] = 2;
        j++;
      }
      i++;
    }
  }
}
```

Les molécules qui ne sont pas cristallisées (les 1) sont soumises au mouvement brownien. Nous allons ici le modéliser sommairement.

Question 5 : (4 points)

Écrire une fonction `void`, appelée `mvt_brownien`, qui tire au hasard pour chaque molécule libre (les 1) une nouvelle position, qui est soit l'une des 8 positions voisines, soit la même position (la molécule ne bouge alors pas), toutes à probabilité égale. Si la nouvelle position est dans la bordure de l'échantillon ou déjà occupée par une autre molécule, alors la molécule ne bouge pas. Sinon il faut déplacer la molécule dans le tableau `tab` et conserver ses nouvelles coordonnées dans le tableau `cam`.

Réponse à la question 5 :

```
void mvt_brownien(int** tab, int** cam){
  int k, i, j, ii, jj;
  for(k = 0; k < N_cam; k++){
    ii = cam[k][0];
    jj = cam[k][1];
    if(tab[ii][jj] == 1){
      i = ii + 3 * drand48() - 1;
      j = jj + 3 * drand48() - 1;
      if(i > 0 && i < N_tab-1 && j > 0 && j < N_tab-1 && tab[i][j] == 0){
        tab[i][j] = 1;
```

2. Pour faire cela proprement il faudrait écrire dans une copie du tableau `tab`, mais pour faire plus court cela ne vous est pas demandé.

```
          tab[ii][jj] = 0;
          cam[k][0] = i;
          cam[k][1] = j;
        }
      }
    }
  }
```

Question 6 : (2 points)

On souhaite arrêter le programme quand 50 % des molécules de camphre initialement présentes sont intégrées au cristal. En faisant appel à toutes les fonctions précédentes, compléter la fonction `main` commencée en question 1 pour créer un programme qui simule la formation de cristaux de camphre dans une grille en deux dimensions. Le programme doit afficher à l'écran le tableau `tab` initial et le tableau `tab` final.

Réponse à la question 6 :

On écrit d'abord une fonction pour l'affichage du tableau (l'utilisation d'une fonction n'est pas obligatoire) :

```
void affiche_tableau(int** tab){
  int i, j;
  for(i = 0; i < N_tab; i++){
    for(j = 0; j < N_tab; j++)
      cout << tab[i][j] << " ";
    cout << endl;
  }
  cout << endl;     // Ligne blanche apres le tableau
}
```

Puis la fin de la fonction `main` dont le début se trouve en question 1 :

```
  srand48(time(NULL));
  initialisation(tab, cam);
  affiche_tableau(tab);
  while(conc_cam_crist(tab,cam) < 0.5){
    cristallisation(tab, cam);
    mvt_brownien(tab, cam);
  }
  affiche_tableau(tab);
  for(i = 0; i < N_tab; i++)
    free(tab[i]);
  free(tab);
  for(i = 0; i < N_cam; i++)
    free(cam[i]);
  free(cam);
  return 0;
}
```

Ce modèle de cristallisation est très simplifié, et a été inspiré par les travaux de Yasuko Tanaka, dans lesquels il observe la formation de structures dendritiques pour une solution de camphre et d'éthanol soumise à un gradient de température (voir figure, extraite de `https://doi.org/10.1016/j.jcrysgro.2008.01.037`). N'hésitez pas à lire son article si le sujet vous a intéressés !