DS1 MP2I : langage C

Démonstration. □

1. On fait de la Science : toute affirmation doit être justifiée.
2. Ecrire les questions dans l'ordre quitte à laisser des blancs.
3. Les réponses sont rédigées uniquement sur les feuilles quadrillées distribuées en début de devoir. Elles doivent comporter vos noms prénoms classes et ne pas être agrafées ni cornées. le crayon à papier est interdit.
4. Au début de chaque réponse, on indique le numéro de la question qu'elle résout.
5. Écrire « Oui » ou « Non », n'est pas une réponse acceptable en CPGE. La phrase réponse doit contenir les éléments essentiels de la question. Par exemple, si la question est « Le capitaine est-il en mesure de naviguer ? » la réponse peut être « Oui, le capitaine semble bien en mesure de naviguer compte tenu de . . . ».
6. Si une réponse apparaît peu lisible... et bien, elle ne sera pas lue !

Codes donnés

Dans tous les exercices ci-dessous, on suppose que les fichiers d'en-tête nécessaires ont bien été inclus sauf mention du contraire.

Q.1 Y a-t-il un problème pour ce code ? Si oui, expliquer.

```
int main(){
  int *p = calloc(4 , sizeof(int));
  for (int i=0; i<4;i++)
     printf("%d,",p[i]);
  return 0;
}


```

Solution. Oublie de la libération. □

Q.2 La fonction `nextafterf` est déclarée dans **math.h**. On considère le **main** du fichier **next.c** :

```
int main(){
  float from1 = 1, to1 = nextafterf(from1, 2);
  printf("The next representable float after %.2f is %.20g\n",
   from1, to1);
  return 0;
}

```

(a) Écrire les inclusions nécessaires dans **next.c**.

(b) Écrire une commande de compilation du fichier **next.c** pour en faire l'exécutable **toto**.

Solution. `gcc -Wall next.c -o toto -lm`

Inclure **stdio.h** et **math.h**.

```
    #include <stdio.h>
    #include <math.h>

```

□

Q.3 Y a-t-il un problème pour ce code ? Si oui, expliquer.

```c
int main(){
  int i = 3;
  int *p = &i;
  *p=4;
  free(p); return 0;
}

```

Solution. Libération d'un pointeur alloué sur la pile et non par un `malloc`.

FFF j'avais oublié le `return 0` mais ce n'est pas le problème principal. en C99+ labsence de return 0 ; dans main est acceptable ; en C90, non garanti. □

Q.4 Le code suivant compile-t-il ? Si non, indiquer ce qui vous semble un problème.

```c
int main(){
  int t[3] ;
  int tab[3]={1,2,3};
  t=tab; return 0;
}


```

Solution. Ne compile pas car un tableau n'est pas une lvalue. □

Q.5 Le code suivant compile-t-il ? Si non, indiquer ce qui vous semble un problème.

```c
void init(int n, int t[n]){
  for(int i=0;i<n;i++)
    t[i]=1;
}

int main(){
  int n=3;
  int *p = malloc(n*sizeof(int));
  init(n,p);
  return 0;
}
```

Solution. Le code compile. Mais la libération a été oubliée. □

Q.6 On suppose que les portions de code non indiquées sont écrites correctement :

```c
void display(int n, int t[n][n]){
  //code correct pour afficher une matrice carrée n * n
}

int main(){
  int n=3;
  int **p = malloc(...);
  for (int i=0;i<3;i++){
    //code correct pour que p représente une matrice 3 * 3  pleine de 0
  }
  display(n,p);
  return 0;
}
```

Y a-t-il un problème pour ce code ? Si oui, expliquer.

Solution. Passage d'un `int**` là ou on attend un `int[][]`
Les lignes du premier cas ne sont pas contiguës mais dans le second cas, elles le sont.
Le code peut compiler mais on aura un seg fault à l'exécution.
Enfin, on a oublié de libérer.

□

Q.7 On donne :

```
int i = 2, j=6;
int const *p1 = &i;
int * const p2 = &i;
```

Pour chaque ligne ci-dessous, indiquer si elle est correcte ou non :

```
*p1=3;// est-ce correct ?
 p1=&j;// est-ce correct ?
 *p2=12;// est-ce correct ?
 p2=&j;// est-ce correct ?

```

Solution. `p1` est un pointeur sur un `int` constant. On ne peut pas changer la valeur de `i` par `*p1=3`. L'autre affectation est valable.
`p2` est un pointeur constant sur un `int`. On ne peut pas changer la valeur de `p2` par `p2=&i`. L'autre affectation est valable. □

Q.8 Que produit la commande d'affichage ? Expliquer.

```
void swap(int x, int y){
  int buff = x;
  x=y;
  y=buff;
}

int main(){
  int i = 2, j=6; swap(i,j);
  printf("%d,%d",i,j);
  return 0;
}

```

Solution. On affiche 2 puis 6. On ne peut pas faire d'effet de bord sans passer de pointeur en argument. □

Q.9 On donne le code suivant :

```
void mystere (int t[], int n){
  for(int i=0;i<=n/2;i++){
    int b = t[i];
    t[i] = t[n-1-i]; t[n-1-i] = b;
  }
}

int main(){
  int t[4]={1,2,3,4};
  mystere(t,4);
  return 0;
}

```

Que contient `t` après l'action de `mystere` ?

Solution. `{4,2,3,1}` □

Q.10 Le code suivant est-il correct ? Sinon, expliquer.

```
int *p = {1,2,3};
```

Solution. Un pointeur n'est pas un tableau. Un pointeur simple ne peut recevoir qu'une seule valeur, une adresse, ce que n'est pas $\{1, 2, 3\}$.

□

Q.11 Le code suivant compile-t-il ? Sinon, expliquer.

```
int t[3];
t= {1,2,3};

```

Solution. Sinon, un tableau est un pointeur constant sur son premier élément ; ce n'est pas une lvalue. Le code ne compile pas.

□

Q.12 En utilisant les conventions de tailles usuelles en classes préparatoires, donner les valeurs qui sont affichées au lignes 7 et 8 :

```
1  long unsigned int display(int t[]){
2    return sizeof(t) / (sizeof t[0]);
3  }
4
5  int main(void){
6    int t[5];
7    printf("%ld\n",sizeof(t) / (sizeof t[0]));// valeur affichée ?
8    printf("%ld\n",display(t));// valeur affichée ?
9    return 0;
10 }
11
```

Solution. 5 et 2. □

Q.13 Dans le code suivant, on suppose que toutes les inclusions nécessaires ont été réalisée.

```
//inclusions nécessaires réalisées
int* f(void){
  int t[3]={1,2,3};
  return t;
}
int main(void){
  int *p = f();
  printf("%d",p[0]);
  return 0;
}

```

On le compile et on l'exécute. Qu'obtient-on ?

Solution. Une erreur de segmentation puisqu'on renvoie une adresse sur la pile.
FFFF il y avait une coquille : j'ai oublié une parenthèse int main(void() au lieu de int main(void) □

Problème : Modélisation de la propagation d'une épidémie

L'étude de la propagation des épidémies joue un rôle important dans les politiques de santé publique. Les modèles mathématiques ont permis de comprendre pourquoi il a été possible d'éradiquer la variole à la fin des années 1970 et pourquoi il est plus difficile d'éradiquer l'apparition d'épidémies de grippe tous les hivers. Aujourd'hui, des modèles de plus en plus complexes et puissants sont développés pour prédire la propagation d'épidémies à l'échelle planétaire telles que le SRAS, le virus H5N1 ou le virus Ebola. Ces prédictions sont utilisées par les organisations internationales pour établir des stratégies de prévention et d'intervention.

On s'intéresse plus précisément ici à une méthode de simulation numérique (dite *par automates cellulaires*). On la programme en langage C. Toutes les importations de bibliothèques nécessaires sont supposées avoir été réalisées.

Dans tout le problème, on peut utiliser une fonction demandée précédemment même si la question correspondante n'a pas été traitée.

Pour mieux prendre en compte la dépendance spatiale de la contagion, il est possible de simuler la propagation d'une épidémie à l'aide d'une grille. Chaque case de la grille peut être dans un des quatre états suivants : saine, infectée, rétablie, décédée. On choisit de représenter ces quatre états par les entiers :

0 (Sain), 1 (Infecté), 2 (Rétabli) et 3 (Décédé)

L'état des cases d'une grille évolue au cours du temps selon des règles simples. On considère un modèle où l'état d'une case à l'instant $t+1$ ne dépend que de son état à l'instant t et de l'état de ses huit cases voisines à l'instant t (une case du bord n'a que 5 cases voisines et trois pour une case d'un coin). Les *règles de transition* sont les suivantes :

— une case décédée reste décédée ;

— une case infectée devient décédée avec une probabilité p_1 ou rétablie avec une probabilité $(1-p_1)$;

— une case rétablie reste rétablie ;

— une case saine devient infectée avec une probabilité p_2 si elle a au moins une case voisine infectée et reste saine sinon.

On initialise toutes les cases dans l'état sain, sauf une case choisie au hasard dans l'état infecté.

Dans ce qui suit, on représente une grille de taille $n \times n$ (pour $n \in \mathbb{N}^*$) par un pointeur `int ** g` dont chaque pointeur interne (les « lignes ») pointe sur une zone de n entiers.

Q.14 Ecrire la fonction `int ** grille(int n)` qui prend en paramètre un entier n et renvoie une grille de $n \times n$ cases toutes nulles.

Solution. Voici

```
int ** grille (int n){
  int ** p = malloc(n * sizeof(int*));
  for (int i=0; i<n ;i++)
    p[i] = calloc(n,sizeof(int));
  return p;
}
```

□

Le hasard en C est modélisé dans ce devoir par la fonction `int myrand()`. Cette fonction renvoie un entier entre 0 et une constante `RAND_MAX`, un nombre entier qui dépend des implémentations mais est toujours supérieur à 32767. Chaque nombre a la même probabilité d'être choisi dans l'intervalle $[\![0, \texttt{RAND\_MAX}]\!]$.

Q.15 Les valeurs obtenues avec `myrand()` sont comprises entre 0 et `RAND_MAX`. Il est intéressant de limiter l'intervalle de valeurs de 0 à $N-1$ pour un certain entier N. Pour commencer, une méthode simple consiste à utiliser l'opérateur modulo :

```
int choose(int N){
  return myrand() % N;
    }
```

Toutefois, une telle méthode ne renvoie pas toutes les valeurs de $[\![0, N-1]\!]$ avec la même probabilité. En supposant `RAND_MAX` $= 25$ et $N = 5$, établir que certaines valeurs ont plus de chance d'être obtenues que les autres.

Solution. Restes nuls : 0,5,10,15,20,25 Reste 1 : 1,6,11,16,21

0 a plus de chance d'être obtenu. □

Pour renvoyer un nombre entier situé dans un intervalle $[\![0, N-1]\!]$ avec équiprobabilité, nous utilisons la méthode de « mise à l'échelle ». Pour simuler le hasard, nous employons désormais uniquement la fonction suivante :

```
int choose(int N){
  return (int)(myrand() / (double)RAND_MAX * (N - 1));
  }
```

Q.16 Écrire une fonction `int ** init(int n)` qui construit une grille G de taille $n \times n$ ne contenant que des cases saines, choisit aléatoirement une des cases et la transforme en case infectée, et enfin renvoie G.

Q.17 Écrire la fonction `bool est_grille_initialisation(int ** g, int n)` qui revoie `true` uniquement si la grille $n \times n$ passée en argument a toutes ses cases à 0 et une à 1.

Solution.
```
int ** init(int n){
  int ** g = grille(n);
  int i=choose(n), j=choose(n);
  g[i][j]=1;
  return g;
}
bool est_grille_initialisation (int **g, int n) {
    int nb_ones = 0;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            int v = g[i][j];
            if (v == 1) {
                nb_ones++;
                if (nb_ones > 1) return false;   // plus d'un '1'  faux
            } else if (v != 0) {
                return false; // valeur différente de 0/1  faux
            }
        }
    }
    return nb_ones == 1; // exactement un '1' et le reste à 0
}

```

□

Q.18 Écrire une fonction `int * compte(int n, int ** G)` qui a pour argument une grille G et sa dimension et renvoie un tableau dynamique `{n0,n1,n2,n3}` formé des nombres de cases dans chacun des quatre états.

Solution. Voici

```
int * compte(int n, int ** g){
  int * r = calloc(4,sizeof(int));
  for (int i=0; i<n; i++)
    for (int j=0; j<n; j++)
      r[g[i][j]] ++;
  return r;
}

```

□

D'après les règles de transition, pour savoir si une case saine peut devenir infectée à l'instant suivant, il faut déterminer si elle est exposée à la maladie, c'est-à-dire si elle possède au moins une case infectée dans son voisinage. Pour cela, on écrit la fonction `est_exposee(G,i,j)` suivante :

```
??? est_exposee(??? n,??? G, ??? i, ???j){
    if ((i == 0) && (j == 0))
    return (G[0][1]-1)*(G[1][1]-1)*(G[1][0]-1) == 0;
  else if ((i == 0) && ( j == n-1))
    return (G[0][n-2]-1)*(G[1][n-2]-1)*(G[1][n-1]-1) == 0;
  else if ((i == n-1) && (j == 0))
    return (G[n-1][1]-1)*(G[n-2][1]-1)*(G[n-2][0]-1) == 0;
  else if ((i == n-1 ) && ( j == n-1))
              return (G[n-1][n-2]-1)*(G[n-2][n-2]-1)*(G[n-2][n-1]-1) == 0;
  else if  (i== 0)  ???
  else if (i == n-1)
```

```
12      return (G[n-1][j-1]-1)*(G[n-2][j-1]-1)*(G[n-2][j]-1)*(G[n-2][j+1]-1)*(G[n-1][j+1]-1) == 0;
13    else if (j == 0)
14      return (G[i-1][0]-1)*(G[i-1][1]-1)*(G[i][1]-1)*(G[i+1][1]-1)*(G[i+1][0]-1) == 0;
15    else if (j == n-1)
16          return (G[i-1][n-1]-1)*(G[i-1][n-2]-1)*
17          (G[i][n-2]-1)*(G[i+1][n-2]-1)*(G[i+1][n-1]-1) == 0;
18    else   ???
19  }// est_exposee
20
```

Il s'agit bien sûr de compléter les portions indiquées par des `???`.

Q.19 (a) Écrire proprement le prototype de la fonction `est_exposee(G,i,j)` donné en ligne 1.

(b) Écrire la branche positive du test `else if (i==0)`.

(c) Écrire ce qui vient après `else`.

Solution. Le prototype :

```
1  bool est_exposee(int n,int ** G, int i, int j)
2
```

Le test `i==0` :

```
1  else if  (i== 0)
2     return ( G [0][j-1] -1)*
3       ( G [0][ j+1] -1)*      \
4             ( G [1][j-1] -1)*
5       ( G [1][j] -1)*
6       ( G [1][ j+1]-1)== 0;
7             //5 termes
```

La branche négative `else` :

```
1   else
2     return  ( G [i-1][j-1] -1)*( G [i-1][j] -1)*( G [i-1][ j+1] -1)*
3                 ( G [i ][j-1] -1)*( G [i][ j+1] -1)*
4                 ( G [i+1][j-1] -1)*( G [i+1][j] -1)*( G [i+1][ j+1] -1)== 0;
5                 // 8 termes
```

□

Une variable aléatoire suit une loi de *Bernoulli* de paramètre $p \in [0,1]$ lorsque son résultat vaut 1 avec une probabilité p et 0 avec la probabilité $1-p$.

Q.20 Ecrire la fonction `bool bernoulli(float p)` qui renvoie `true` avec une probabilité p et `false` avec la probabilité $1-p$.

Q.21 Écrire la fonction `void copy(int n, int ** G1, int ** G2)` qui copie `G1` dans `G2`.

Q.22 Écrire une fonction `bool suivant(int n,int ** G,float p1,float p2)` qui prend en paramètres une grille `G` de taille $n \times n$ et deux probabilités `p1,p2`. La grille `G` représente l'état de la population à un instant t. Les arguments `p1` et `p2` sont les probabilités qui interviennent dans les règles de transition pour les cases infectées et les cases saines.
La fonction fait évoluer toutes les cases de la grille `G` à l'aide des règles de transition. La grille est modifiée pour représenter, après action de la fonction, l'état de la population à l'instant suivant de la simulation.
La fonction renvoie `true` si une case a été modifiée et `false` sinon.

Solution. Voici

```
1  bool bernoulli (float p){
2    float x  = (float) choose(RAND_MAX+1) / (float) (RAND_MAX+1);
3    return x < p;
4  }// comme cela ça marche même si p=1  : on renvoie toujours vrai
5
```

Et

```
void copy(int n, int ** G1, int ** G2){
  for (int i=0; i<n; i++)
    for (int j=0; j<n; j++)
      G2[i][j]=G1[i][j];
}

bool suivant(int n,int ** G,float p1,float p2){
  int c =0; //compteur de modifications
  int ** g = grille(n);//nvle grille vierge
  for (int i=0; i<n; i++)
    for (int j=0; j<n; j++){
      if ((G[i][j]==0) && (est_exposee(n,G,i,j))
    && (bernoulli(p2))){
          //case saine infectable devient infectée
      g[i][j]=1;
            c+=1;}// fin case saine infectable
  else if (G[i][j]==1){//case infectée
      if (bernoulli(p1))
              g[i][j]=3;// décès
      else
        g[i][j]=2;// retablissement
      c++;
  }// fin case infectée
  else g[i][j]=G[i][j];
    }// for j
  copy(n,g,G);
  //libérer g
  for (int i=0; i<n;i++)
    free(g[i]);
  free(g);
  return c>0;
}
```

□

Avec les règles de transition du modèle utilisé, l'état de la grille évolue entre les instants t et $t+1$ tant qu'il existe au moins une case infectée.

Q.23 Écrire une fonction `float * simulation(int n,float p1, float p2)` qui réalise une simulation complète avec une grille de taille $n \times n$ pour les probabilités `p1` et `p2` et renvoie le tableau `{x0,x1,x2,x3}` formée des fréquences de cases dans chacun des quatre états à la fin de la simulation (une simulation s'arrête lorsque la grille n'évolue plus).

Solution. Voici

```
float * simulation(int n,float p1,float p2){
  int ** g =init(n);
  bool modif = true;
  while (modif){
    modif = suivant(n,g,p1,p2);
  }
  int * c = compte(n,g);
  float * r = calloc(4,sizeof(float));
  for (int i=0; i<4;i++)
     r[i] = c[i]/ (float) (n*n);
  free(c);
  for (int i=0; i<n;i++)
    free(g[i]);
  free(g);
  return r;
}

```

□

Q.24 Quelle est la valeur de la proportion des cases infectées `x1` à la fin d'une simulation ?

Quelle relation vérifient `x0,x1,x2` et `x3` ?

Comment obtenir à l'aide des valeurs de `x0,x1,x2` et `x3` la valeur `x_atteinte` de la proportion des cases qui ont été atteintes par la maladie pendant une simulation ?

Solution. Il n'y a plus de case infectée, donc $x_1 = 0$
On a $x_0 + x_2 + x_3 = 1$
Les personnes qui ont été atteintes sont rétablies ou mortes. La proportion des cases qui ont été atteintes par la maladie pendant une simulation est $x_2 + x_3$ ou $1 - x_0$ ce qui revient au même □

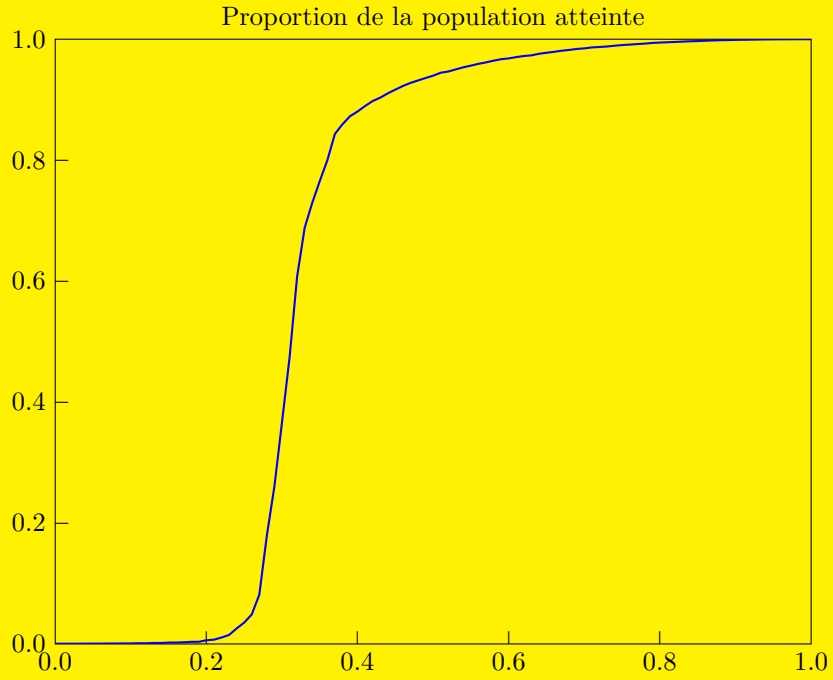

FIGURE 1 – Représentation de la proportion de la population qui a été atteinte par la maladie pendant la simulation en fonction de la probabilité $p2$

Q.25 On appelle *seuil critique de pandémie* la valeur de `p2` à partir de laquelle plus de la moitié de la population a été atteinte par la maladie à la fin de la simulation. On suppose que les valeurs de `p2` et `x_atteinte` utilisées pour tracer la courbe de la figure 1 on été stockées dans deux tableaux de même longueur k : `Lp2` (valeurs en abscisses dans le graphique 1) et `Lxa` (valeurs en ordonnées dans le graphique 1). Ces deux tableaux sont triés par ordre croissant et `Lxa[i]` désigne le nombre de personnes atteintes calculées par la simulation pour la probabilité `Lxa[i]`.

Écrire une fonction `float * seuil(int k, float *Lp2,float *Lxa)` qui détermine un encadrement $[m, M]$ du seuil critique de pandémie avec la plus grande précision possible ($|M - m|$ doit être minimum). Le retour de la fonction est donc un tableau dynamique qui contient 2 valeurs.
On supposera que la liste `Lp2` croît de 0 à 1 et que la liste `Lxa` des valeurs correspondantes est croissante.

Une complexité logarithmique en k est attendue.

Solution. Voici

```c
float * seuil(int k, float *Lp2,float *Lxa){
  int g =0 , d = k-1;
  while ((d - g) >1){
    int m = g + (d-g)/2;
    if (Lxa[m] < 0.5)
      g = m;
    else d = m;
  }
  float * r = malloc(2*sizeof(float));
  r[0] = Lp2[g]; r[1]=Lp2[d];
  return r;
}
```

□

Pour étudier l'effet d'une campagne de vaccination, on immunise au hasard à l'instant initial une fraction `q` de la population. On a écrit la fonction `int ** init_vac(int n,float q)` :

```
1  int ** init_vac(int n,float q){
2    int ** G = init(n);
3    int nvac = (int) (q*n*n);
4    int k = 0;
5    while (k < nvac){
6      int i = choose(n);
7      int j = choose(n);
8      if (G[i][j] == 0){
9        G[i][j] = 2;
10       k = k+1  ;
11       }//if
12   }//while
13  return G;
14 }
15
```

Q.26 Peut-on supprimer le test en ligne 8 ?

Q.27 Que renvoie l'appel `init_vac(5,0.2)` ?

Solution.

1. Non, il y a une personne infectée à l'instant initial, et le vaccin est sans effet sur elle. De plus il ne faudrait pas vacciner deux fois la même personne.
2. Une grille 5×5 avec $25\frac{2}{10} = 5$, donc 5 personnes notées comme rétablies (vaccinées) et une malade.

□