I Exercice : *Algorithmique*

Pour une formation, nous disposons de tableaux de notes (de type *float*) contenant les moyennes des étudiants (numérotés à partir de 0) et d'un tableau contenant le nom complet de chaque étudiant Par exemple, si le nom complet d'un étudiant est `Bernard Pivot`, qu'il a eu `12.5` en mathématiques, et que son numéro est 5 :
- la cinquième case du tableau de noms vaut "Bernard Pivot", et
- la cinquième case du tableau de notes correspondant aux mathématiques vaut `12.5`.

> Dans les questions suivantes, pour chaque fonction, et dans cet ordre :
> - Indiquez le profil de la fonction que vous allez écrire ;
> - Décrivez en une phrase ses arguments et la valeur qu'elle retourne ;
> - Écrivez la fonction en C.

(a) (1 pt) Quel est le type d'un tableau de noms ?

(b) (2 pts) Écrire une fonction Affiche qui prend en entrée un tableau de notes, un tableau de noms, et le nombre d'étudiants, et qui affiche les notes (avec l'étudiant correspondant) avec une précision de 3 chiffres après la virgule.

(c) (2 pts) Écrire une fonction Moyenne qui prend en entrée un tableau de float représentant les notes d'une matière, le nombre d'étudiants, et qui retourne la moyenne de la classe pour la matière représentée par le tableau.

(d) (2 pts) Écrire une fonction Major qui, a partir d'un tableau de notes, retourne le numéro de l'étudiant ayant obtenu la meilleure note

(e) (2 pts) Faire une fonction Passe qui prend en entrée les un tableau de notes et le nombre d'étudiants, et rend en sortie un tableau t d'entiers avec $t[i] = 1$ si l'élève de numéro i a la moyenne, et $t[i] = 0$ sinon.

(f) (4 pts) Écrire une fonction MiseAJour qui prend un numéro d'étudiant et propose de mettre à jour la moyenne de cet étudiant dans une matière. Pour cela, cette fonction :
- affiche la note courante courante de l'étudiant dans cette matière ;
- demande une nouvelle note, puis :
  - si l'utilisateur rentre une note, la fonction met à jour la note de l'étudiant ;
  - sinon, l'ancienne note est inchangée.

II Exercice : *Bichaînes*

On fixe la structure suivante pour contenir les couples (⟨clef⟩,⟨valeur⟩).

```
struct bichaine_s {
int l1 ;
int l2 ;
char * clef ;
char * valeur ;
};
```

(a) (1 pt) Définir le type `bichaine` des pointeurs vers une structure `bichaine_s`.

(b) (2 pts) Écrire une fonction `nouvelle_bichaine` qui prend en entrée 2 chaînes de caractères et leur longueur, et rend une valeur de type `bichaine`.

(c) (2 pts) Écrire deux fonctions `clef` et `valeur` qui prennent en entrée une bichaîne et rendent respectivement sa clef et sa valeur.

(d) (1 pt) Écrire une fonction qui affiche une bichaîne dans le format ⟨`clef`⟩`:` ⟨`valeur`⟩ suivi dans cet ordre par les deux caractères `\n` et `\f`.

(e) (3 pts) Écrire une fonction qui prend en entrée **les adresses** de 2 valeurs de type `bichaine`, et rend un entier positif, nul, ou négatif si la première est plus grande, égale, ou plus petite que la seconde. Pour comparer 2 bichaines :

— si les deux sont `NULL`, elles sont égales ;
— si une seule des deux est `NULL`, c'est la plus grande ;
— si aucune n'est nulle, alors le résultat est celui de la comparaison de leur **clef**.

En particulier, on note que si deux bichaînes ont la même clef, mais des valeurs différentes, elles seront considérées comme égales.

III Exercice : *Calcul de majorité (14 pts)*

Lors de l'ouverture de fichiers, il est demandé de faire attention aux cas possibles d'erreur.

Un *automate cellulaire unidimensionnel borné* est défini par un tableau fini de *cellules*. Chaque *cellule* est définie par un état. Cet automate cellulaire *évolue* suivant une *fonction de transition* qui indique comment chaque cellule doit évoluer en regardant son état et celui de ses voisines.

Cet exercice porte sur le problème de recherche de majorité avec un automate cellulaire unidimensionnel borné. Les états ne peuvent être que `oui` ou `non`, et on veut faire évoluer un automate pour que toutes ses cellules soient à `oui` s'il y a une majorité de oui-s, ou `non` s'il y a une majorité de non-s.

Une solution (qui ne marche pas toujours, car le problème n'a pas de solutions en général) consiste à utiliser la fonction de transition suivante :

— Si l'état de la cellule i est `oui`, alors le nouvel état est celui de la majorité entre cette cellule, celle à l'adresse $i-1$, et celle à l'indice $i-3$;

— Si l'état de la cellule i est `non`, alors le nouvel état est celui de la majorité entre cette cellule, celle à l'adresse $i+1$, et celle à l'indice $i+3$.

Les indices dans le tableau sont modulo n, le nombre de cellules dans l'automate cellulaire unidimensionnel.

(a) (1 pt) Pour simplifier le calcul de majorité, on veut définir une cellule comme pouvant avoir 2 états, `oui` et `non`, avec `oui` qui vaut 1, et `non` qui vaut -1. Définir le type `cellule` des cellules.

(b) (1 pt) On définit maintenant les automates comme des couples ayant :

1. un tableau de cellules ;
2. un entier n indiquant le nombre de cellules dans l'automate.

. Définir la structure `automate_s` de ces couples, ainsi que le type `automate` des pointeurs vers ces structures.

(c) (2 pts) Écrire une fonction nouvel_automate qui prend en entrée un entier n, et qui rend un automate pouvant contenir n cellules. *Il n'est pas demandé d'initialiser la valeur de ces cellules.*

Pour simplifier le traitement des erreurs, dans la suite, vous pouvez utiliser la fonction *erreur*, qui s'utilise exactement comme la fonction *printf*, mais termine le programme immédiatement quand elle est appelée. Par exemple :

```
if ( ( f = fopen( fichier, "r") ) == NULL )
    erreur ( "Impossible d'ouvrir le fichier %s en lecture, fin." , fichier ) ;
```

(d) (2 pts) Écrire une fonction lire_etat_initial qui prend en entrée le *nom* d'un fichier, et qui :

— ouvre ce fichier en lecture ;

— lit un entier n, qui sera la taille de l'automate, dans ce fichier ;

— construit un automate ayant n cellules ;

— lit les n valeurs de cellules stockées dans le fichier (le caractère `'.'` pour `non`, et le caractère `'*'` pour `oui`, aucun autre caractère n'est autorisé).

— ferme le fichier.

Exemple. On donne ci-dessous un fichier bien formatté qui doit être lu sans erreurs :

```
30
...**.*.***..**..**...***..***
```

(e) (2 pts) Écrire une fonction ecrire_automate qui prend en entrée un fichier ouvert en écriture (de type `FILE *`) et un automate, et écrit les cellules de l'automate sur une ligne de ce fichier.

(f) (2 pts) Écrire une fonction vote qui prend en entrée un automate a et le numéro i d'une cellule de cet automate, et renvoie le nouvel état pour la cellule i de a. *Rappel : l'opérateur C % calcule le "modulo" :* $7\%5 == 2$ *mais* $-3\%5 == -3$.

(g) (1 pt) Écrire une fonction suivant qui prend en entrée un automate et renvoie un nouvel automate contenant l'évolution de toutes les cellules de l'automate en argument de la fonction.

(h) (2 pts) Écrire une fonction evolution qui prend en entrée le nom d'un fichier et :

1. lit l'automate a qui est définit au début de ce fichier ;
2. ré-ouvre ce fichier en *écriture en ajout* ;
3. calcule les 100 évolutions de l'automate a, et écrit dans le fichier tous les automates calculés ;
4. referme le fichier.

Comme précédemment, il est demandé de faire très attention à tous les cas possibles d'erreur.

(i) (1 pt) Écrire la fonction main d'un programme qui prend en entrée exactement 1 argument, qui est le nom d'un fichier. Le programme doit compléter le fichier en écrivant les 100 automates qui suivent celui qui est déjà dans le fichier.

Exemple. À la fin, le fichier a la forme suivante (les dernières lignes sont enlevées car toutes identiques) :

```
30
...**.*.***..**..**...***..***
....****.**..**..**..*.**.****
...*.***.**..**..**.**..******
..*.*.**.**..**..**.**.*******
.***...*.**..**..**.**********
****..*...*..**..**************
****..*......**.**************
****..*.......****************
****..*......*.****************
****..*.....*.*.***************
****..*....*.*.*.**************
****..*...*.*.*.*.*************
****..*..*.*.*.*.*.************
****..*.***.*.*.*.*.**********
****.***.*****.*.*.*.*********
*****************.*.*.********
*********************.*.*******
**********************.******
```

****
****

IV Exercice : *Calendrier (9pts)*

Dans cette partie, on modélise une date par un triplet jour-mois-année.

(a) (1pt) On veut modéliser les mois en utilisant dans le programme les noms des mois (janvier, février, ...) plutôt que des entiers. Donnez la déclaration C à écrire.

(b) (0.5pt) Donnez la définition de structure permettant de modéliser une date.

(c) (1pt) Écrire une fonction date qui prend en entrée un jour, un mois, et une année, et qui rend une date.

(d) (1pt) Écrire une fonction affichant une date.

(e) (0.5pt) Écrire une fonction bissextile qui rend vrai si l'année donnée en argument est bissextile, et faux sinon.

Rappel : une année n est bissextile si, et seulement si, n est divisible par 400 ou si n est divisible par 4 mais pas par 100.

(f) (1pt) Écrire une fonction nb_jours qui prend en entrée une date et rend le nombre de jours du mois de cette date.

Exemples: #1
#1

(g) (3pts) Écrire une fonction jour_suivant qui change la date en argument en la date du jour suivant.

Indication : une possibilité est d'écrire 3 fonctions changeant la date en, respectivement, le premier janvier de l'année suivant, le premier jour du mois suivant, et finalement le jour suivant.

Exemples: #1
#1

(h) (1pt) Écrire une fonction avance qui change une date en celle de n jours plus tard.

Exemple: #1

V Exercice : *Calendrier (bis) (5pts)*

Cette partie est plus complexe que les précédentes ; il est déconseillé de l'aborder avant d'avoir terminé ce qui précède. On cherche à nouveau à modéliser des dates, mais cette fois dans le format jour-semaine-an :
- le dimanche de la semaine 0 de 2012 est le 01/01/2012 ;
- le mercredi de la semaine 2 de 2012 est le 10/01/2012 ;
- pour plus d'exemples, voir les deux calendriers en annexe.

Chaque année est divisée en 53 ou 54 semaines, dont une ou deux peuvent être incomplètes. Une semaine commence soit un lundi, soit le premier janvier (semaine 0), et se termine soit le dimanche, soit le 31 décembre (semaine 52). Une année normale dure $52 \times 7 + 1$ jours, et une année bissextile dure $52 \times 7 + 2$ jours. Dans les questions suivantes, il est conseillé de définir des fonctions auxiliaires pour faire les calculs. Il est possible de réutiliser les fonctions de l'exercice II.

(a) (1pt) Donnez les types de données que vous allez utiliser.

(b) (2pts) Écrire une fonction rendant la date du jour de l'an d'une année n.

(c) (2pts) Écrire deux fonctions transformant une date jour-mois-an en une date jour-semaine-an et réciproquement.

VI Exercice : *Championnat*

Le but de cet exercice est de modéliser des championnats, c'est-à-dire des matchs aller-retour entre des équipes de telle sorte que chaque équipe rencontre deux fois chaque autre équipe, une fois à domicile, une fois à l'extérieur.

(a) (1pt) Écrire une fonction qui demande un nom d'équipe (une chaîne de caractères) à l'utilisateur et renvoie cette chaîne de caractères. La chaîne de caractères dans laquelle la fonction doit stocker le nom de l'équipe est passée en argument.

(b) (2pts) On veut avoir une structure championnat qui contient le nombre (un entier) et les noms des équipes (des chaînes de caractères), et la longueur maximale (un entier) du nom d'une équipe. Donnez la déclaration de la structure championnat.

Une *rencontre* est modélisée par 4 entiers `equipe1`, `equipe2`, `score1`, `score2` qui représentent :

— l'indice de l'équipe 1 dans le tableau du championnat ;
— l'indice de l'équipe 2 dans le tableau du championnat ;
— le score de l'équipe 1 dans le match ;
— le score de l'équipe 2 dans le match ;

Les scores sont à -1 tant que le match n'est pas joué.

(c) (1pt) Donnez la déclaration de la structure rencontre en fonction de ces données.

(d) (3pts) Écrire une fonction qui demande à l'utilisateur :

— un nombre d'équipes ;
— la taille maximale du nom d'une équipe ;
— puis les noms de toutes les équipes du championnant un par un ;

et qui renvoie un championnat représentant ces informations.

(e) (2pts) Écrire une fonction qui prend en entrée une variable de type championnat et une variable de type rencontre et qui affiche le résultat de la rencontre dans le format :

```
<nom Équipe1> <score1> - <score2> <nom Équipe2>
```

Par exemple :

```
Stade Toulousain 18 - 15 RC Toulon
```

(f) (3pts) Écrire une fonction qui prend en entrée un championnat, et rend un tableau de rencontres entre les équipes de ce championnat dans lequel deux équipes différentes se rencontrent exactement 2 fois. Les scores des rencontres seront initialisés à -1.

VII Exercice : *Correction d'un programme C*

(a) (5 pts) Corriger le programme suivant pour qu'il affiche le maximum de 3 entiers (répondre sur le sujet)

```
#include ⟨stdio.h⟩
int main (int argc, char *argv[]){
      int nb1,nb2,nb3,n1,n2,* n3,
      int max;
      /* Demande des nombres */
      printf("Donner votre premier nombre : ");
      do { nb1 = scanf("%f",&n1); while (getchar() != '\n' ); } while ( n1 == 0 );
      printf("Donner votre deuxieme nombre : ");
      do { nb2 = scanf("%d",&n2); while (getchar() != '\n' ); } while ( nb2 = 0 );
      printf("Donner votre troisieme nombre : ");
      do { nb3 = scanf("%s",n3); while (getchar() != '\n' ); } while ( nb3 = 0 );
      if (n1>n2);
      {
           max = n1;
           if (n3 > max )
                max = *n3;
      }
      else
      {
           max = n2;
           if (n3 > max )
           {
                max = *n3
           }
      }
      printf ("Le maximum de %d %d %d est %d \n", &n1,&n2,n3,&max);
      return 0;
}
```

VIII Exercice : *Algorithmique*

On organise des élections pour lesquelles il y a C candidats et S États. Les résultats des candidats dans chaque État sont stockés dans un tableau. S'il y a $C = 3$ candidats et $S = 2$ états, ce tableau contient par exemple :

état\candidat	0	1	2
0	150	200	10
1	33	120	3

(a) (2 pts) Écrire une fonction initialisation qui prend en entrée le nombre C de candidats et le nombre N d'états, et rend l'adresse d'un tableau pouvant stocker les résultats du vote pour une élection à C candidats et S états.

(b) (1 pt) Écrire une fonction affichageCandidatEtat qui prend en entrée le tableau stockant les résultats du vote, un numéro de candidat c et un numéro d'état s et qui affiche le nombre de voix obtenu par le candidat c dans l'état s.

(c) (1 pt) Écrire une fonction entreeCandidatEtat qui prend en entrée le tableau stockant les résultats du vote, un numéro de candidat c et un numéro d'état s, qui demande à l'utilisateur le nombre de voix obtenu par le candidat c dans l'état s, et qui met à jour le tableau de résultats.

(d) (2 pts) Écrire une fonction totalVoixCandidat qui rend le nombre total des voix obtenues par ce candidat.

(e) (2 pts) Écrire une fonction totalVoix qui rend le nombre total de votes.

(f) (2 pts) Écrire une fonction president qui rend le candidat ayant remporté le plus d'états.

IX Exercice : *Jeu de la vie*

Le *jeu de la vie* est un jeu informatique qui se joue dans un tableau à deux dimensions $n \times m$. Chaque case de ce tableau, appelée *cellule*, est soit vivante, soit morte. Chaque cellule a 8 *voisines* (dans les 8 directions nord, nord-ouest, ouest, sud-ouest, sud, sud-est, est, nord-est). Les cellules au-delà des bords sont toutes considérées comme mortes. Un *état* est un tableau dans lequel chaque cellule est soit vivante, soit morte.

On passe de l'état courant à l'état suivant en examinant, pour chaque cellule en dehors des bords, le nombre k de voisines en vie :

— s'il y en a 3, et que la cellule est morte, elle devient vivante ;
— s'il y en a 2 ou 3, et que la cellule est vivante, elle le reste ;
— Dans tous les autres cas, la cellule meurt.

(a) (1 pt) Déclarer le type pour représentant l'état des cellules.

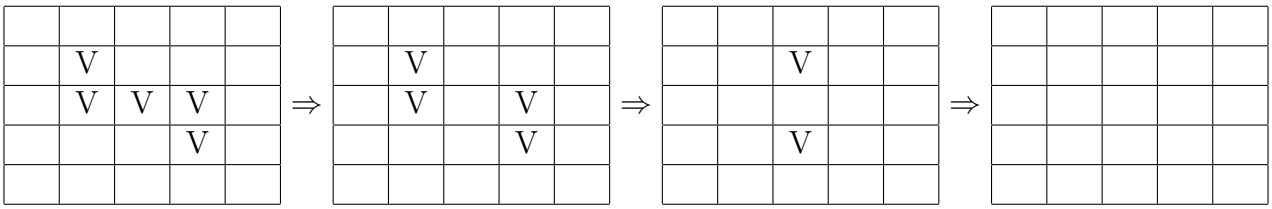
(b) (1 pt) Déclarer un type pour représenter les tableaux dans lesquels se jouent le jeu de la vie.

(c) (1 pt) Écrire un fonction prenant en entrée deux entiers n et m, et rendant un tableau pour le jeu de la vie dans lequel toutes les cellules sont mortes.

(d) (1 pt) Écrire une fonction donnant le nombre de voisines vivantes d'une cellule dans un tableau.

(e) (3 pts) Écrire une fonction prenant en entrée un tableau (ainsi qu'éventuellement d'autres arguments) représentant un état courant, et rendant un tableau contenant l'état suivant.

Exemple. Dans la suite d'états suivante, une cellule morte est vide, une vivante est représentée par un V.



X Exercice : *Matrices (7 pts)*

Le but de cet exercice est d'évaluer les contributions individuelles au projet.

(a) (1 pt) On définit une *matrice entière* par :
— un nombre de lignes et de colonnes ;
— un tableau de lignes, chaque ligne étant un tableau d'entiers.
Définir le type `matrice` des pointeurs vers des structures C représentant des matrices entières.

(b) (1 pt) Écrire une fonction **matrice_nulle** qui prend en entrée un nombre de lignes et de colonnes, et renvoie une matrice dont tous les coefficients sont à 0.

(c) (2 pts) Écrire une fonction qui renvoie le produit de 2 matrices, ou `NULL` si le produit n'est pas défini.

(d) (1 pt) Écrire une fonction **taille_int** qui calcule le nombre minimal de symboles à utiliser (y compris le `'-'` pour les nombres négatifs) pour écrire un entier en base 10.

(e) (1 pt) Écrire une fonction **taille_mat** qui renvoie le maximum des tailles des entiers des coefficients d'une matrice passée en argument.

(f) (1 pt) Le *mineur* $M_{i,j}$ d'une matrice carrée M de taille $n \times n$ est la matrice carrée de taille $(n-1) \times (n-1)$ construite à partir de M en enlevant la $i$ème ligne et la $j$ème colonne. En numérotant les lignes et les colonnes à partir de 0, le *déterminant* de M est noté $|M|$ et vaut :

$$|M| = \Sigma_{j=0}^{n-1}(-1)^j |M_{0,j}|$$

Donnez un ordre de grandeur, en fonction de n, pour le nombre de calculs à effectuer dans une fonction calculant un déterminant en utilisant cette formule. *Il n'y a pas de code C à écrire pour cette question !*

XI Exercice : *Matrices (7pts)*

On revoit dans cette partie certaines fonctions écrites dans le projet. On représente les matrices avec des structures contenant :
- un nombre `nb_lig` de lignes ;
- un nombre `nb_col` de colonnes ;
- un tableau `coefficients` de lignes, chaque ligne étant un tableau de valeurs de type double .

(a) (1pt) Donnez la déclaration d'une structure matrice correspondant à cette définition.

(b) (2pts) On veut n'avoir qu'une fonction, appelée matrice, qui crée des matrices en appelant la fonction malloc. Écrire cette fonction.

(c) (2pts) Écrire une fonction qui rend le résultat de l'addition de deux matrices. Vérifiez que les matrices en entrée sont bien définies et que l'addition des deux matrices en entrée est bien définie.

(d) (2pts) Écrire une fonction qui rend le résultat du *produit* de deux matrices. Vérifiez que les matrices en entrée sont bien définies et que la multiplication des deux matrices en entrée est bien définie.

XII Exercice : *Matrices*

Le but de cet exercice est d'implémenter des fonctions opérant sur des matrices. Une *matrice* est un tableau de float à deux dimensions.

(a) (1 pt) Déclarez une structure Matrice_s qui contient :
— le nombre de lignes et de colonnes ;
— un tableau de lignes, chaque ligne étant un tableau de float ;
ainsi que le type Matrice des pointeurs vers ces structures.

Dans la suite, une matrice ayant l lignes et c colonnes est dite $l \times c$. Une matrice $l \times c$ peut être écrite $(a_{i,j})_{\substack{1 \leq i \leq l \\ 1 \leq j \leq c}}$; dans ce cas on dit que $a_{i,j}$ est son coefficient à la ligne i et à la colonne j. Enfin, on dit qu'une matrice est *carrée* si $l = c$.

(b) (2 pts) Écrivez une fonction matrice créant une nouvelle matrice ayant l lignes et c colonnes et dont tous les coefficients sont à 0.

(c) (1 pt) L'addition de deux matrices $A = (a_{i,j})_{\substack{1 \leq i \leq l_a \\ 1 \leq j \leq c_a}}$ et $B = (a_{i,j})_{\substack{1 \leq i \leq l_b \\ 1 \leq j \leq c_b}}$ est définie si, et seulement si, $l_a = l_b$ et $c_a = c_b$. Écrire une fonction indiquant si deux matrices peuvent être additionnées.

(d) (2 pts) Si deux matrices $(a_{i,j})_{\substack{1 \leq i \leq l \\ 1 \leq j \leq c}}$ et $(b_{i,j})_{\substack{1 \leq i \leq l \\ 1 \leq j \leq c}}$ sont additionnables, le résultat de leur addition est la matrice $(r_{i,j})_{\substack{1 \leq i \leq l \\ 1 \leq j \leq c}}$ avec :

$$r_{i,j} = a_{i,j} + b_{i,j}$$

Écrire une fonction rendant le résultat de l'addition de deux matrices.

(e) (2 pts) Le produit de deux matrices $(a_{i,j})_{\substack{1 \leq i \leq l \\ 1 \leq j \leq c}}$ et $(b_{i,j})_{\substack{1 \leq i \leq l' \\ 1 \leq j \leq c'}}$ est défini lorsque $c = l'$, et dans ce cas, le résultat du produit est la matrice $(r_{i,j})_{\substack{1 \leq i \leq l \\ 1 \leq j \leq c'}}$ définie par :

$$r_{i,j} = \Sigma_{k=1}^{c} a_{i,k} \cdot b_{k,j}$$

Écrire une fonction rendant le produit de deux matrices.

(f) (2 pts) On donne ci-dessous un schéma pour effectuer un calcul de puissance rapide d'une matrice carrée $n \times n$ A. Dans ce schéma, I_n désigne la matrice identité $n \times n$ (ses coefficients $r_{i,i}$ valent 1, tous les autres valent 0).

```
// Calcul de A^k pour une matrice n x n A
R:= I_n
x := A ;
Tant que k > 0 faire
   Si k mod 2 == 0 Alors
       k := k / 2 ; x := x * x
   Sinon
       k := k - 1 ; R := R * x ;
```

```
   Fin Si
Fin tant que
retourner R.
```

Écrire une fonction implémentant ce calcul rapide de puissance d'une matrice carrée.

(g) (3 pts) Soient $A = (a_{i,j})_{\substack{1 \leq i \leq l \\ 1 \leq j \leq c}}$ et $B = (b_{i,j})_{\substack{1 \leq i \leq c \\ 1 \leq j \leq c'}}$ deux matrices telles que le produit $A \times B$ soit défini. On peut alors découper A et B en blocs tels que toutes les opérations ci-dessous (additions et produits de matrices) soient définies :

$$\left(\begin{array}{c|c} A_1 & A_2 \\ \hline A_3 & A_4 \end{array}\right) \times \left(\begin{array}{c|c} B_1 & B_2 \\ \hline B_3 & B_4 \end{array}\right) = \left(\begin{array}{c|c} A_1 \times B_1 + A_2 \times B_3 & A_1 \times B_2 + A_2 \times B_4 \\ \hline A_3 \times B_1 + A_4 \times B_3 & A_3 \times B_2 + A_4 \times B_4 \end{array}\right)$$

Écrire une fonction effectuant le produit de deux matrices en commençant par les séparer en blocs de taille adéquate, puis en regroupant les différents résultats. *Il est rappelé que la compréhension de l'énoncé fait partie de la note, et que les dernières questions peuvent être longues.*

XIII Exercice : *Permutations*

Le but de cet exercice est d'écrire des fonctions qui réalisent des permutations d'un tableau d'entiers. On veut faire une permutation aléatoire en combinant les deux opérations de base suivantes :

— Une transposition entre le premier élément et le second élément ;
— Un cycle dans lequel tous les éléments du tableau sauf le dernier sont déplacés d'une case vers la droite, et le dernier élément est mis en premier.

On donne un exemple de combinaison et de cycles dans la Fig. 1.

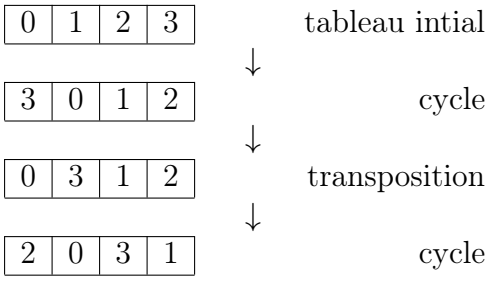


FIGURE 1 – Exemple de combinaison de permutations

(a) (1pt) Donnez la déclaration d'une structure perm qui contient :
— un pointeur `val` vers un tableau d'entiers ;
— un entier `taille` qui est la taille de ce tableau ;

(b) (2pts) Écrivez une fonction nouveau qui prend en entrée un entier N et rend une permutation dans laquelle `val` pointe vers un tableau d'entiers de taille N.

(c) (2pts) Écrivez une fonction initialise qui prend en entrée une permutation et qui rend une permutation dans lequel la case i du tableau vaut i.

(d) (2pts) Écrivez une fonction transpose qui échange les deux premiers éléments de la permutation donnée en entrée.

(e) (2pts) Écrivez une fonction cycle qui effectue une permutation circulaire sur les éléments de la permutation donnée en
entrée.

(f) (3pts) Écrivez une fonction aleatoire qui effectue une permutation circulaire sur les éléments de la permutation donnée en entrée. Pour cela, elle prend en entrée une permutation de taille N, et effectue N tirages aléatoires d'entier. À chaque tirage, si l'entier tiré est pair, la fonction fait une transposition, et s'il est impair, elle applique une permutation cyclique.

XIV Exercice : *Points dans le plan*

Le but de cet exercice est d'implémenter des fonctions permettant de travailler avec des points dans un plan (un espace de dimension 2). Cette première série de questions porte sur les opérations de base sur les points.

(a) (1 pt) Déclarez une structure Point_s, ainsi que le type Point des pointeurs vers ces structures.

(b) (1 pt) Écrivez une fonction point créant une nouvelle valeur de type Point à partir de ses arguments.

(c) (0,5 pt) Écrivez une fonction distance calculant la distance habituelle entre deux points. Précisez la bibliothèque à inclure, et les options de compilation nécessaires.

(d) (1 pt) Écrivez deux fonction ord_compare et abs_compare comparant deux points suivant respectivement leur ordonnée et leur abscisse. Ces fonctions rendent un entier négatif, nul, ou positif si le premier point a une ordonnée (resp. une abscisse) inférieure, égale, ou supérieure à celle du second.

On va travailler maintenant sur les tableaux de points.

(e) (0,5 pt) Déclarer un type Points pour les tableaux de points.

(f) (3 pts) Écrire deux fonctions lire_points et ecrire_points permettant respectivement de lire et d'écrire un tableau de points dans un fichier (représenté par une valeur de type FILE * passée en argument). Il est demandé que la fonction de lecture puisse lire un fichier écrit par la fonction d'écriture.

(g) (4 pts) Écrire deux fonctions permettant de trier un tableau de points en fonction respectivement de leur abscisse et de leur ordonnée. Il est conseillé, mais non requis, d'écrire des fonctions intermédiaires.

XV Exercice : *Points*

On donne une structure `Point_s` qui contient des points de IR^3 :

```
struct Point_s {
    float x;
    float y;
    float z;
};
```

(a) (1 pt) Définir le type Point des pointeurs vers une structure struct Point_s.

(b) (1 pt) Écrire une fonction qui prend en entrée un fichier f (de type FILE *) et un point P, et qui écrit sur une ligne les trois coordonnées de ce point avec 3 chiffres après la virgule. Par exemple, comme résultat de l'écriture de 2 points, on peut avoir le fichier :

```
1.000   2.000   3.000
2.000   4.000   6.000
```

On définit le type des tableaux de points de la manière suivante :

```
typedef struct tab_s{
    int taille;
    Point * t;
}* tab;
```

(c) (3 pts) Écrire une fonction qui prend en entrée un nom de fichier (de type *char**) et un tableau de points (de type tab) et qui écrit les points du tableau dans le fichier (à ouvrir puis à refermer).

(d) (2 pts) De la même manière, écrire une fonction qui va lire un fichier dont le nom est passé en argument et rendre le tableau de points (de type tab) contenu dans ce fichier.

XVI Exercice : *Calcul d'une racine carrée (4,5 pts)*

On peut calculer la racine carrée d'un nombre positif d en calculant la limite d'une suite récursive :

$$\begin{cases} u_0 &= 1 \\ u_{n+1} &= \frac{1}{2} \cdot (u_n + \frac{d}{u_n}) \end{cases}$$

Pour la suite des questions, il peut être utile de définir d'autres fonctions que celles demandées.

(a) (2,5 pts) Écrire une fonction racine17 qui retourne la racine carrée de 17 à 10^{-3} près. On admettra que cette précision est atteinte lorsque deux éléments consécutifs de la suite récursive sont distants de moins de 10^{-3}.

(b) (1 pt) Écrire une fonction racine qui calcule la racine carrée d'un nombre de type double à ε près.

(c) (1 pt) Écrire une fonction qui demande un nombre de type double à l'utilisateur et affiche la racine carrée de ce nombre (on supposera que les entrées de l'utilisateur forment du premier coup un nombre de type double strictement positif.)

XVII Exercice : *Tableaux de bichaînes*

Pour l'envoi de messages HTTP, on a juste besoin d'ajouter des bichaînes dans un tableau initialement vide. Pour cela, on va créer des tableaux d'une taille fixée (TAILLE_INITIALE_TABLEAU_BC), et s'il y a besoin, lors de l'ajout d'une bichaîne, on augmentera cette taille.

(a) (1 pt) Définir une structure `tab_bc_s` qui contient l'adresse d'un tableau de bichaînes et le nombre d'éléments dans ce tableau.

(b) (1 pt) Définir le type `tab_bc` des pointeurs vers une structure `tab_bc_s`.

(c) (1 pt) Définir la constante TAILLE_INITIALE_TABLEAU_BC comme valant 10.
Dans le tableau de bichaînes d'une structure `tab_bc_s`, un élément peut être soit `NULL` (pour indiquer une case vide), soit une valeur de type `bichaine`.

(d) (2 pts) Écrire une fonction `nouveau_tab_bc` qui renvoie une valeur de type `tab_bc` dont le tableau contient TAILLE_INITIALE_TABLEAU_BC cases **vide**.

(e) (3 pts) Écrire une fonction `ajoute_tab_bc` qui insère une bichaîne au premier emplacement `NULL` d'un tableau de bichaînes. S'il n'y a aucun emplacement disponible, cette fonction va créer un nouveau tableau dont la taille est 2 fois l'ancienne taille, correctement initialisé, puis recopier l'ancien dans le nouveau, et enfin insérer la nouvelle bichaîne dans ce tableau (toujours au premier emplacement non-nul).

Lors de la lecture d'un message, la norme RFC 2616 indique que si deux "bichaînes" avec la même clef apparaissent, il faut fusionner les valeurs en les séparant par des points-virgule. Par exemple, il faut fusionner les bichaînes `("codage","8859-1")` et `("codage","UTF-8")` en une seule bichaîne `("codage","8859-1;UTF-8")`. Pour effectuer cette fusion, on va :

1. Trier le tableau de bichaînes en utilisant la fonction `qsort` définie dans `stdlib.h` :

   $$\texttt{qsort} : \texttt{void}* \times \texttt{int} \times \texttt{int} \times (\texttt{int}()(\texttt{void}* \times \texttt{void}*)) \to \texttt{void}$$

   avec par exemple `qsort ( t , 10 , sizeof ( int ) , intcmp )` qui trie un tableau t de 10 éléments (des `int`) dont chacun est de la taille `sizeof ( int )` en comparant les éléments avec une fonction `intcmp` qui prend en entrée les adresses de deux entiers et renvoie leur différence.
2. Pour chaque élément non-nul du tableau trié :
   — Pour chaque élément suivant qui a la même clef :
      — copier la valeur de l'élément suivant à la suite de celle de l'élément courant (en réservant de la mémoire supplémentaire si nécessaire) ;
      — mettre l'élément suivant à `NULL`
      — passer à l'élément d'après
   — passer au premier élément suivant non-nul.
3. Re-trier le tableau pour mettre les éléments nuls en fin de tableau.

Exemple d'exécution.

encodage	8859-1
referer	"http ://www.irit.fr/"
encodage	UTF-8

↓ Premier tri

encodage	8859-1
encodage	UTF-8
referer	"http ://www.irit.fr/"

↓ Fusion des éléments de même clef

encodage	8859-1 ;UTF-8
NULL	
referer	"http ://www.irit.fr/"

↓ Second tri

encodage	8859-1 ;UTF-8
referer	"http ://www.irit.fr/"
NULL	

(f) (3 pts) Écrire une fonction `normalisation` qui prend en entrée une valeur de type `tab_bc`, et implémente l'algorithme ci-dessus.

XVIII Exercice : *Tableaux dynamiques*

On veut créer une structure permettant d'adapter la taille d'un tableau au nombre d'éléments qu'il contient. En théorie, une manière efficace d'adapter la taille est, pour un tableau de n éléments pouvant en contenir N :

- Si $n = N$, *i.e.*, le tableau est plein, changer le tableau en un tableau de taille $2 \times N$;
- Si $n \leq N$, *i.e.*, le tableau est aux $3/4$ vide et si $N > 10$, changer le tableau en un tableau de taille $N/2$.

(a) (1 pt) Écrire une fonction qui prend en entrée un entier positif n et qui rend la plus petite puissance de 2 qui est plus grande que n : $f(n) = \min\{N \mid N = 2^k \text{ et } N \geq n\}$

(b) (1 pt) Déclarer le type td des pointeurs vers une structure td_s qui contient :

- un nombre maximal d'éléments ;
- le nombre courant d'éléments ;
- un tableau `val` d'éléments.

(c) (2 pts) Écrire une fonction rendant un tableau dynamique pouvant contenir N éléments mais n'en contenant aucun. Le nombre N est passé en argument.

(d) (2 pts) Écrire une fonction qui prend en entrée deux tableaux dynamiques tels que le premier peut contenir plus d'éléments que n'en possède le second, et qui recopie tous les éléments du second tableau dans le premier. Cette fonction rend le nombre d'éléments copiés.

(e) (2 pts) Écrire deux fonctions :

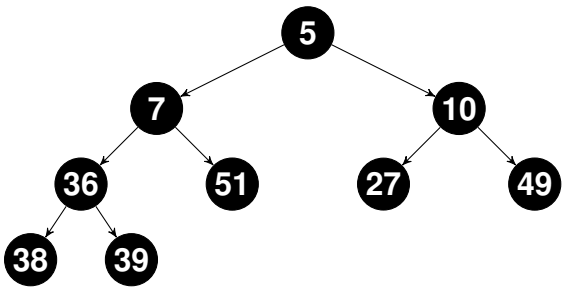
- une qui insère un élément à la fin d'un tableau dynamique passé en argument ;
- une qui supprime le dernier élément d'un tableau dynamique passé en argument.

Si N désigne le nombre d'éléments que peut contenir le tableau dynamique, et n le nombre d'éléments contenus après l'insertion ou la suppression, on recopiera le tableau dans un tableau pouvant contenir $N/2$ éléments si $n \leq N/4$, ou $2 \times N$ si $n = N$

(f) (1 pt) Donnez une manière de représenter un tableau dynamique dans un fichier. *Cette question est une préparation aux deux questions suivantes.*

(g) (1.5 pts) Écrire une fonction prenant un tableau dynamique et un argument de type FILE * correspondant à un fichier ouvert en écriture, et stockant ce tableau dynamique suivant le format défini à la question précédente.

(h) (1.5 pts) Écrire une fonction prenant un argument de type FILE * correspondant à un fichier ouvert en lecture, et lisant un tableau dynamique qui y est stocké suivant le format défini à la question précédente. Cette fonction renvoie ce tableau dynamique.

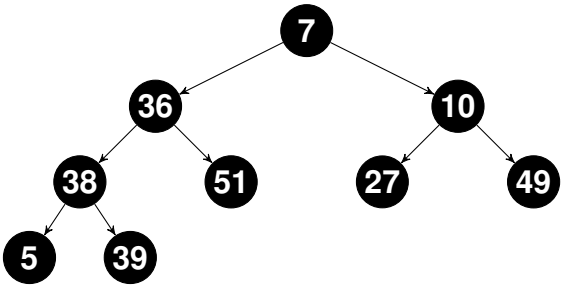


(a) Un tas, dont la racine a pour valeur 5, son fils gauche a une racine de valeur 7 et son fils droit a pour racine un nœud dont la valeur est 10. Les arbres vides ne sont pas représentés.

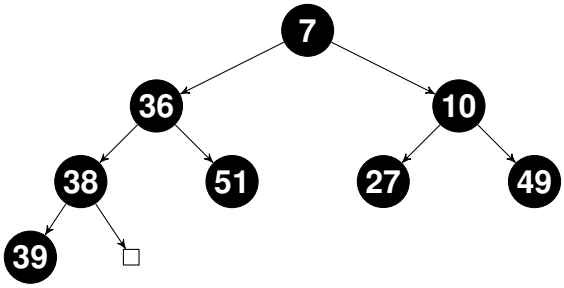
5	7	10	36	51	27	49	38	39			
0		2			5						

(b) Le tableau correspondant à ce tas : la valeur de la racine est dans la case 0, celle de son fils droit dans la case $2 = 2 \times 0 + 2$, et celle du fils gauche de ce fils droit est dans la case $5 = 2 \times 2 + 1$.

FIGURE 2 – Tas et leur représentation par un tableau



(a) Le tas de la Figure 2a après l'étape 2 de la suppression.



(b) Le tas de la Figure 2a après l'étape 5 de la suppression. Le nœud vide représente la décrémentation du nombre d'éléments. 39 est plus grand que 38, donc l'étape 6 ne fait rien

FIGURE 3 – Tas aux différentes étapes de la suppression.

XIX Exercice : *Tas*

Préliminaires. Un *arbre* est soit un arbre vide, soit est un nœud (la *racine* de cet arbre) qui est le père de deux *fils*, son fils droit et son fils gauche, qui sont eux-mêmes des arbres.

Les *tas* sont des tableaux qui codent certains arbres de la manière suivante :

— la valeur de la racine de l'arbre est dans la case 0 ;
— si la valeur d'un nœud est stockée dans la case i, alors la valeur de la racine de son fils gauche est stockée dans la case $2 \times i + 1$, et la valeur de la racine de son fils droit est stockée dans la case $2 \times i + 2$;
— la valeur d'un nœud est toujours plus petite que la valeur de ses fils.

Pour savoir quand s'arrêtent les valeurs du tas (les cases blanches de la Figure 2), on a besoin, en plus du tableau, du nombre d'éléments dans le tas. Pour décrire des tas en C, on utilise les déclarations suivantes :

```
struct tas_s {
    int nb_elts; /* nombre d'éléments dans le tas */
    int * val; /* tableau des valeurs */
};
typedef struct tas_s * tas;
```

(a) (0.5 + 0.5 + 1 = 2 pts) Écrire trois fonctions fils_gauche, fils_droit, et pere qui, en fonction d'un indice i, calculent l'indice correspondant respectivement au fils gauche, au fils

1. On note i l'indice courant. Au départ, l'indice courant est l'indice de la première case vide dans le tableau (dans la Figure 2b, i vaut au départ 9) ;
2. mettre l'élément à ajouter dans la case d'indice i ;
3. Tant que :
   — l'indice courant est strictement positif ;
   — et la valeur du père de l'indice courant est plus grande que la valeur de l'indice courant

   échanger ces deux valeurs, et positionner l'indice courant sur son père ;
4. rendre le tas ;

(a) Algorithme d'insertion d'un élément.

1. On note i l'indice courant. Au départ, l'indice courant vaut 0 ;
2. Tant que le fils droit de l'indice courant est dans le tableau :
   — échanger la valeur du nœud courant avec celle du plus petit de ses fils ;
   — l'indice courant devient celui du fils avec lequel l'échange a été fait.
3. Si l'élément courant a un fils gauche, on échange ces deux éléments, et l'élément courant devient l'indice du fils gauche ;
4. on échange ensuite la valeur de l'élément courant et celle du dernier élément du tas ;
5. on décrémente de 1 le nombre d'éléments dans le tas ;
6. on remonte, comme pour l'insertion, l'élément courant tant qu'il est plus petit que son père.

(b) Algorithme de suppression du plus petit élément.

FIGURE 4 – Algorithmes sur les tas

droit, et au père du nœud dont la valeur est stockée dans la case i. *Par exemple, pere(5)=2, et fils_gauche(2)=5.*

(b) (1 pt) Écrire une fonction qui prend en entrée un nombre maximal d'éléments pour un tas, et qui rend un tas contenant 0 éléments mais *pouvant* contenir ce nombre maximal d'éléments.

L'algorithme permettant d'insérer un élément dans un tas est donné informellement dans la Figure 4a.

(c) (1 pt) Justifiez (informellement) que l'insertion d'un élément dans un tas permet d'obtenir un tas, *i.e.*, que dans le tas obtenu, tout élément est plus petit que ses fils.

(d) (2 pts) Écrire la fonction d'insertion d'un élément en C. On suppose qu'il y a strictement moins d'éléments dans le tas que le nombre maximal qu'il ne peut en contenir.

(e) (2 pts) Écrire la fonction de suppression du plus petit élément en C en se basant sur l'algorithme de la figure 4b. On suppose qu'il y a au moins un élément dans le tas.

XX Exercice : *Algorithmique (7 pts)*

La bibliothèque universitaire a actuellement des difficultés à gérer son stock car elle est très fréquentée en ce moment et ne dispose pas d'un nombre suffisant d'employés. Vous décidez de l'aider en écrivant un programme informatique permettant de soulager le travail des employés.

Ce que doit faire votre programme : La bibliothèque possède *nbLivres* livres indexés de 0 à $nbLivres - 1$. Chaque jour, un certain nombre de clients demandent à emprunter des livres pour une certaine durée. Si le livre est disponible, la requête du client est satisfaite, sinon le client repart sans livre.

Votre programme doit d'abord lire sur une première ligne deux entiers : $nbLivres \leq 10000$ et *nbJours*. Pour chacun des jours, votre programme lira un entier *nbClients* sur une ligne puis *nbClients* lignes de deux entiers. Le premier entier correspond à l'indice du livre et le second la durée correspondante. (voir l'exemple d'entrée). Il affichera ensuite, sur des lignes séparées, pour chaque client un 1 si le livre peut être prêté et 0 dans le cas contraire.

On remarquera que si un client emprunte un livre le jour *iJour* pendant une durée *duree* alors celui-ci ne sera de nouveau disponible qu'au jour $iJour + duree$. De plus, si plusieurs personnes demandent le même livre pendant une journée, seule la première a une chance d'être satisfaite.

Exemple d'entrée. Les lignes commençant par `//` sont des commentaires décrivant le contenu de la ligne suivante.

```
// nombre de livres
10
// nombre de jours à traiter
2
// nombre de clients le premier jour
1
// emprunt du livre 3 pour 2 jours
3 2
// nombre de clients le second jour
2
// emprunt du livre 3 pour 1 journée
3 1
// emprunt du livre 0 pour 2 jours
0 2
```

Le programme doit afficher :

```
// premier emprunt accepté
1
// deuxième emprunt refusé car le livre a été prêté la veille
0
// troisième emprunt accepté
1
```

(a) (1 pt) Écrire une fonction qui lit le nombre de livres et alloue un tableau contenant, pour chaque livre, le jour (un entier) où il sera livre. Initialement, aucun livre n'est emprunté.

(b) (2 pts) Écrire une fonction qui prend entrée le tableau des disponibilités, lit une ligne correspondant à un emprunt (2 entiers), et :

— affiche 0 si l'emprunt est refusé, et 1 s'il est accepté ;

— met à jour le tableau si l'emprunt est accepté.

(c) (2 pts) Écrire une fonction qui prend entrée le tableau des disponibilités, lit le nombre d'emprunts sur une journée, et met le tableau à jour pour cette journée.

(d) (2 pts) Écrire le programme résolvant le problème.

XXI Exercice : *Algorithmique (4 pts)*

Ecouter de la musique peut être très agréable mais lorsqu'un morceau est vraiment très répétitif, il arrive parfois qu'on s'ennuie un peu. Aussi le professeur de composition musicale du conservatoire a décidé d'imposer une règle très stricte : quand il relit les morceaux composés par ses élèves, dès qu'il voit deux notes identiques côte à côte, il les efface toutes les deux ! Il continue ainsi d'effacer tant qu'il existe deux notes égales consécutives.

Ce travail étant long et fastidieux, il se demande s'il n'est pas possible de l'automatiser.

(a) (4 pts) Écrire une fonction C prenant en argument une chaîne de caractères représentant le morceau de musique (on supposera qu'elle est correcte) et qui affiche version du morceau "corrigée" où tous les doublons sont supprimés tant qu'il en existe. Les notes de musiques sont représentées par les lettres 'a', 'b', 'c', 'd', 'e', 'f' et 'g'.