

NOM :

Prénom :

- tous les documents (poly, slides, TDs, livres, brouillon du voisin...) sont **interdits**.
- les réponses doivent toutes tenir dans les cases prévues dans l'énoncé. Si cela ne suffit pas, utilisez la dernière page blanche (pas de feuilles supplémentaires).
- Dans l'exercice 2, le code des fonctions doit être donné en langage C. Dans le reste de l'examen, le code des algorithmes/fonctions pourra être donné soit en C soit en pseudo-code : la syntaxe n'a pas besoin d'être exacte, mais le code doit être présenté clairement (accolades, indentation, écriture lisible...). En pseudo-code, aucune syntaxe n'est imposée, mais tout devra être suffisamment détaillé pour ne pas laisser de place aux ambiguïtés (écrire "parcourir les sommets du graphe G" n'est pas assez précis, mais "parcourir les voisins du sommet S" est correct).
- le barème actuel de ce contrôle aboutit à une note sur 100 points. La note finale sur 20 sera dérivée de la note sur 100 grâce à une fonction croissante qui n'est pas encore définie (cela ne sert à rien de la demander).
- n'oubliez pas de remplir votre nom et votre prénom juste au dessus de ce cadre.

**Exercice 1 : QCM**

(20 points)

Chaque bonne réponse rapporte 1 point. Chaque mauvaise réponse enlève 1 point. Si vous n'êtes pas certains de votre réponse, ne répondez pas au hasard, la note totale peut être négative !

```
_____ code1 _____  
1 #include <stdio.h>  
2 int main(int argc, char* argv[]) {  
3     int i, sum=0, n=atoi(argv[1]);  
4     printf("0");  
5     for (i==1; i<n; i++) {  
6         printf(" + %d", i*i);  
7         sum += i*i;  
8     }  
9     printf(" = %d.\n", sum);  
10    return 0;  
11 }
```

**1.a]** Le code ci-dessus contient une erreur qui fait qu'il risque de ne pas s'exécuter correctement. À quelle ligne cette erreur se trouve-t-elle ?

- 2,                       3,                       5,                       9.

**1.b]** Qu'affiche le code ci-dessus si on le compile (après avoir corrigé l'erreur) en un programme appelé `code1` et qu'on l'exécute la commande `./code1 5` ?

- $0 + 1*1 + 2*2 + 3*3 + 4*4 + 5*5 = 55.$   
  $0 + 1 + 4 + 9 + 16 = 30.$   
  $+ 1*1 + 4*4 + 9*9 + 16*16 = 30.$   
  $0 + 1 + 4 + 9 + 16 + 25 = 15.$

**1.c]** Laquelle des propositions suivantes peut être programmée à l'aide d'une boucle `while` mais pas d'une boucle `for` ?

- une boucle infinie,
- une boucle qui s'exécute 0 fois,
- un calcul de pgcd,
- rien, tout ce qui se fait avec un `while` peut se faire avec un `for`.

**1.d]** Laquelle des fonctions récursives suivantes retourne la valeur de factoriel `n` ?

---

```
1 int fact(int n) {
2   if (n!=0) {
3     return n*fact(n-1);
4   }
5   return 0;
6 }
```

---

---

```
1 int fact(int n) {
2   if (n==1) {
3     return n*fact(n-1);
4   }
5   return n;
6 }
```

---

---

```
1 int fact(int n) {
2   if (n==0) {
3     return 1;
4   }
5   return fact(n)*(n-1);
6 }
```

---

---

```
1 int fact(int n) {
2   if (n>1) {
3     return n*fact(n-1);
4   }
5   return 1;
6 }
```

---

**1.e]** Que manque-t-il à la définition de structure suivante pour qu'elle soit syntaxiquement correcte ?

---

```
1 struct node{
2   int val;
3   node *left, *right;
4 };
```

---

- un `typedef`
- un `;`
- un `struct`
- rien, elle est correcte

**1.f]** Combien d'`int` peut-on stocker au maximum dans un tableau `tab` alloué avec la commande : `int* tab = (int*) malloc(1000);` ?

- 1000,
- `1000/sizeof(int)`
- `1000×sizeof(int)`
- on ne peut pas savoir.

**1.g]** La complexité d'un algorithme permet de mesurer son temps d'exécution en fonction de :

- la taille de son entrée,
- son nombre de lignes de code,
- son nombre de paramètres,
- son implémentation.

**1.h]** Il faut faire au moins 7 mouvements de disque pour résoudre le problème des tours de Hanoï à 3 disques. Combien faut-il faire de mouvements (au minimum) pour résoudre le problème à 4 disques ?

- 8,
- 14,
- 15,
- 49.

**1.i]** Quand une fonction récursive atteint sa "condition d'arrêt" elle...

- ... ne peut plus exécuter d'instructions autres que `return`,
- ... ne fera plus d'appels à elle même,
- ... renvoie une erreur,
- ... arrête l'exécution du programme.

**1.j]** Lequel des algorithmes de tri suivants a la meilleure complexité *dans le pire cas* ?

- le tri par insertion,
- le tri à bulles,
- le tri fusion,
- le tri rapide.

**1.k]** On effectue un tri fusion (programmé comme celui vu en cours) sur le tableau  $\{4, 7, 3, 1, 2, 8\}$ . Comment les éléments du tableau sont-ils rangés *juste avant* la dernière étape de fusion ?

- $\{1, 2, 3, 4, 7, 8\}$
- $\{3, 4, 7, 1, 2, 8\}$
- $\{1, 2, 3, 8, 7, 4\}$
- $\{1, 4, 2, 7, 3, 8\}$

**1.l]** Laquelle des structures de données suivantes ne peut pas être implémentée efficacement à l'aide d'un tableau ?

- une pile,
- un tas,
- une file,
- une liste chaînée.

**1.m]** Pour implémenter une *file* à l'aide d'une liste chaînée, que faut-il de plus que pour une liste chaînée standard ?

- un pointeur sur le dernier élément,
- un compteur du nombre d'éléments dans la file,
- le dernier élément doit pointer sur le premier (liste cyclique),
- un tableau pour stocker les éléments.

**1.n]** On implémente un dictionnaire de  $n$  mots à l'aide d'une table triée permettant une recherche dichotomique. Quelles sont les complexités respectives des opérations de recherche et d'insertion dans ce dictionnaire ?

- $\Theta(\log n)$  et  $\Theta(\log n)$ ,
- $\Theta(n \log n)$  et  $\Theta(n)$ ,
- $\Theta(\log n)$  et  $\Theta(n)$ ,
- $\Theta(n)$  et  $\Theta(\log n)$ .

**1.o]** Lequel des parcours suivants affiche les nœuds d'un arbre binaire de recherche dans l'ordre croissant ?

- en largeur,
- préfixe,
- infixé,
- postfixé.

**1.p]** Un tas (implémenté comme vu en cours) contient les éléments 18, 13, 9, 5 et 4. On y insère les éléments 17 et 11 puis on en extrait deux fois à la suite sa racine. Quelle sera alors la nouvelle racine ?

- 17,
- 13,
- 11,
- 5.

**1.q]** Que peut-on calculer grâce à un parcours en profondeur d'un graphe ?

- un plus court chemin,
- un tri topologique,
- une fermeture transitive,
- le nombre de chemins entre 2 sommets.

**1.r]** L'algorithme de Dijkstra permet de calculer des plus courts chemins dans un graphe pondéré. Laquelle des conditions suivantes doit être vérifiée pour qu'il fonctionne ?

- le graphe doit être sans cycles,
- le graphe doit être orienté,
- tous les poids doivent être positifs,
- tous les poids doivent être différents.

**1.s]** On cherche à construire un graphe orienté à 6 sommets dont la fermeture transitive réflexive contiendrait tous les arcs possibles. Combien d'arcs au minimum doit-il avoir ?

- 5,
- 6,
- 15,
- 36.

**1.t]** Lequel des langages suivants (que pourraient reconnaître un automate) est différent des 3 autres ?

- $(a^*b)^+$
- $a^*(ba^*)^*b$
- $(a^*ba^*)^*ab$
- $(a^*(a|b))^*b$


## Exercice 2 : Programmation en C

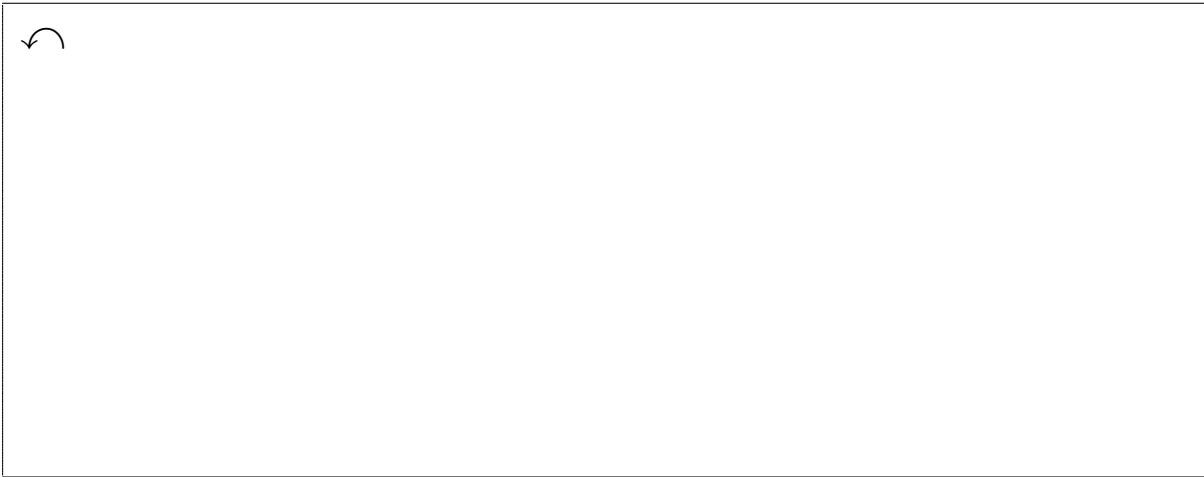
(25 points)

Dans cet exercice il est demandé de répondre aux questions avec du vrai code C. La syntaxe doit être correcte, vous devez déclarer toutes vos variables et penser à **libérer** la mémoire dont vous n'avez plus besoin... Si vous voulez être certains d'obtenir tous les points, votre code doit pouvoir être compilé et exécuté sans erreurs.

- (3 pts) **2.a]** Écrivez une fonction `int suite(int n)` qui calcule (et renvoie) le  $n$ -ième terme de la suite :  $u_n = u_{n-1} + 2u_{n-2}$  avec  $u_0 = 1$  et  $u_1 = 1$ . Votre fonction ne doit faire que des calculs entiers et doit avoir une complexité d'au plus  $\Theta(n)$  (vous pouvez par exemple utiliser de la programmation dynamique).

- (4 pts) **2.b]** Écrivez une fonction `void plus_frequent(char* str)` qui prend en argument une chaîne de caractères et recherche la caractère qui apparaît le plus de fois dans cette chaîne de caractères. Cette fonction doit ensuite afficher ce caractère et le nombre de fois qu'il apparaît. Si plusieurs caractères apparaissent le même nombre de fois, il suffit d'en afficher un.

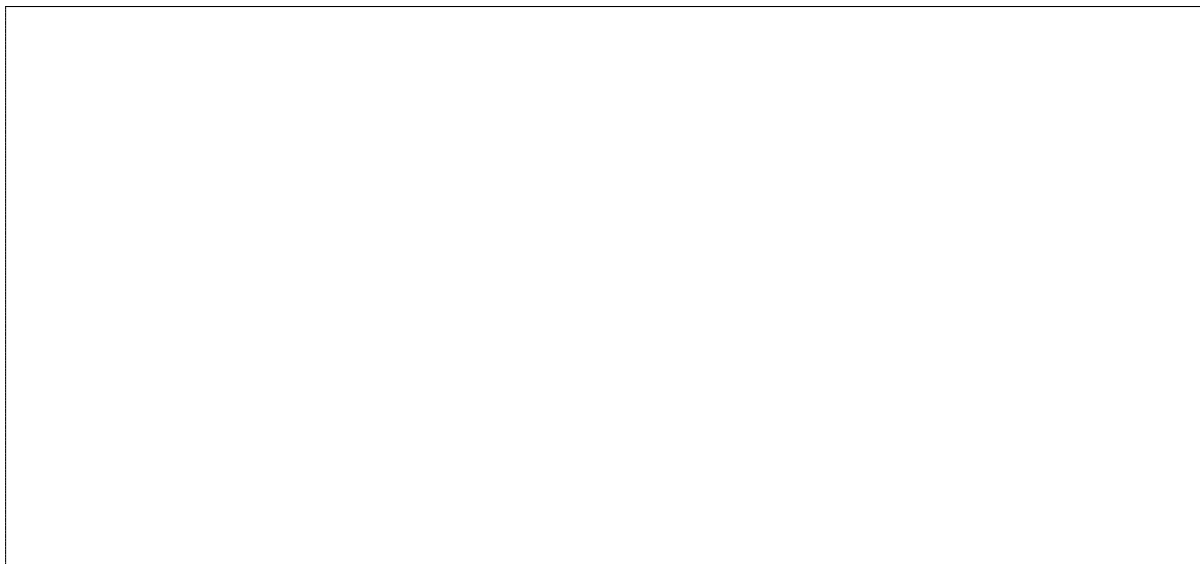




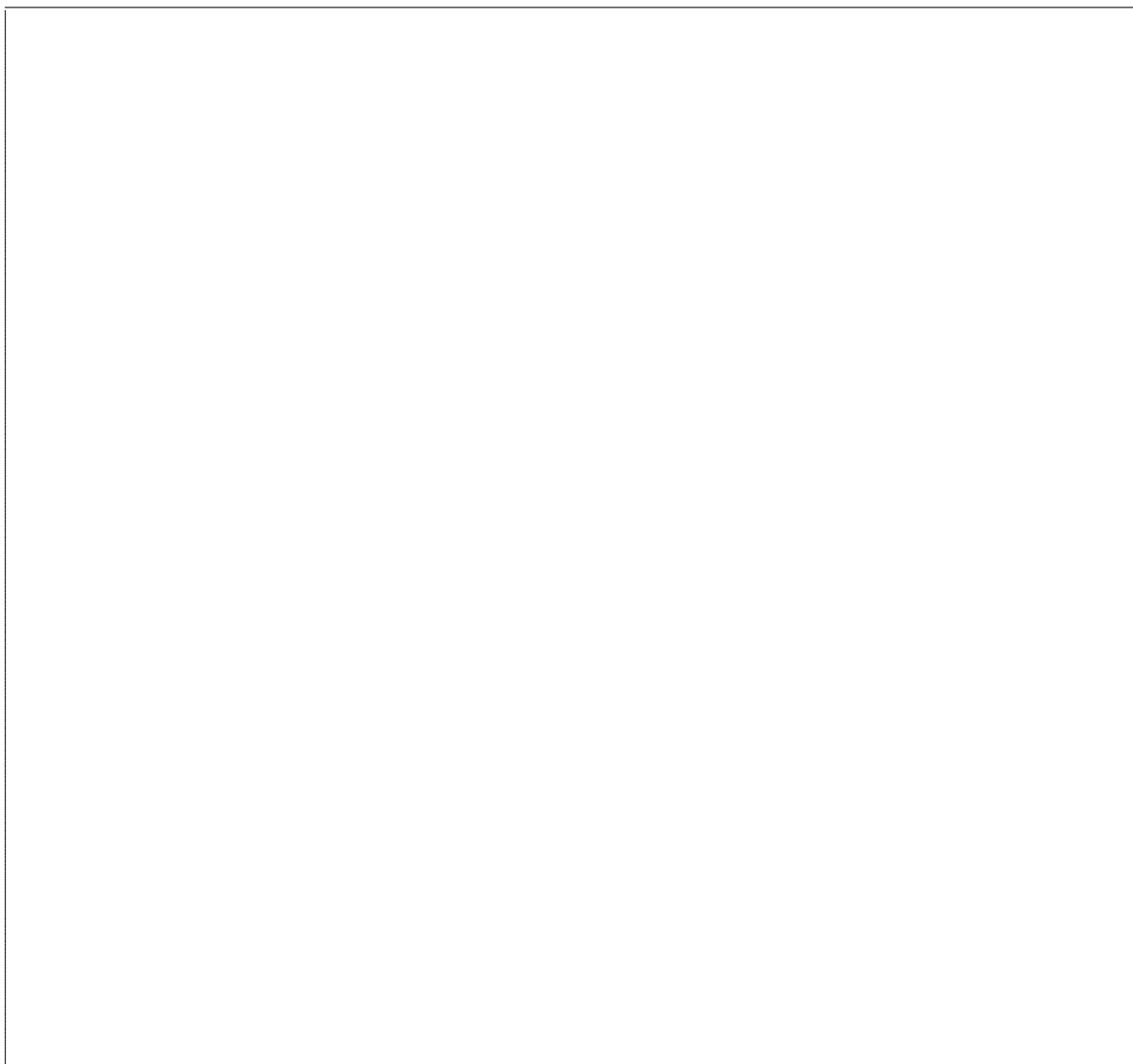
- (5 pts) **2.c]** L'algorithme quicksort permet de trier un tableau récursivement en choisissant un élément pivot et en mettant tous les éléments plus petits au début du tableau et les éléments plus grands à la fin du tableau. Écrivez une fonction `int partition(int* tab, int p, int r)` qui va faire cette opération sur le tableau `tab` entre les positions `p` (incluse) et `r` (exclue). L'élément `tab[p]` sera utilisé comme pivot et la fonction doit retourner la position `q` à laquelle se retrouve le pivot à la fin : les éléments plus petits que `tab[p]` seront donc entre les indices `p` et `q-1` et les éléments plus grands entre `q+1` et `r-1`. On suppose que tous les éléments de `tab` sont différents.



- (2 pts) **2.d]** Écrivez les structures C nécessaires pour représenter un graphe sous forme de listes de successeurs.



- (4 pts) **2.e]** Écrivez une fonction `float average(char* filename)` qui ouvre (et referme à la fin) le fichier dont le nom est passé en argument en lecture et calcule la moyenne des nombres écrits dans ce fichier. On suppose que le fichier ne contient que des nombres entiers, séparés par des espaces.



- (2 pts) **2.f]** Définissez une structure de liste chaînée (une liste simple, comme vue en cours, contenant des entiers) et écrivez une fonction récursive permettant d'afficher le contenu d'une telle liste.



- (5 pts) **2.g]** Définissez une structure de file (implémentée avec un tableau) et programmez les fonctions `push` et `pop` qui permettent d'ajouter ou d'extraire un élément de cette structure.



**Exercice 3 :** Plus court chemin dans un graphe

(10 points)

- (2 pts) **3.a]** On veut écrire un algorithme qui calcule efficacement le plus court chemin entre deux sommets d'un graphe orienté. Quelle représentation utiliseriez vous pour votre graphe, et pourquoi ?

- (8 pts) **3.b]** Écrivez une fonction prenant en argument un graphe et les indices de 2 sommets de ce graphe et qui effectue un parcours en largeur du graphe en partant du premier de ces sommets pour trouver le plus court chemin vers le deuxième. Votre fonction doit construire le tableau des pères qui permettra de retrouver le chemin le plus court et maintenir à jour un tableau des distances ainsi qu'un coloriage des nœuds du graphe. On suppose qu'une structure de file est déjà implémentée et vous pouvez directement utiliser les fonctions `push` et `pop`. Votre algorithme doit s'arrêter dès que le plus court chemin est trouvé et ensuite afficher la longueur de ce chemin ainsi que tous les nœuds par lesquels ce chemin passe (*vous pouvez écrire en pseudo-code*).



**Exercice 4 : Arbres binaires de recherche**

(20 points)

- (2 pts) **4.a]** Donnez des définitions courtes (mais exactes) de ce que sont un arbre binaire et un arbre binaire de recherche.

- (2 pts) **4.b]** On insère les nœuds : 12, 5, 11, 10, 3, 17, 18, 6, 1, 2 et 15 (dans cet ordre) dans un arbre binaire de recherche standard. Dessinez l'arbre que l'on obtient à la fin.

- (1 pts) **4.c]** Les arbres AVL sont une sorte d'arbres binaires de recherche équilibrés. Expliquez ce qu'ils ont de plus que les arbres binaires de recherche standard et qui permet de garantir leur équilibre.

- (2 pts) **4.d]** On insère les nœuds : 12, 5, 11, 10, 3, 17, 18, 6, 1, 2 et 15 (dans cet ordre) dans un arbre AVL. Dessinez l'arbre que l'on obtient à la fin, et donnez le nombre de rotations et de doubles rotations que l'on doit effectuer au cours de ces insertions.

Pour les questions qui suivent, on supposera qu'une structure d'arbre AVL est définie ainsi :

---

```
1 typedef struct node_st {
2   int val; // valeur
3   int h;   // hauteur
4   int eq;  // équilibrage
5   struct node_st* fg; // fils gauche
6   struct node_st* fd; // fils droit
7 } node;
```

---

- (5 pts) **4.e]** Écrivez une fonction qui prend en argument un arbre AVL **A** et deux valeurs entières **p** et **q** et affiche (en ordre croissant) tous les nœuds de **A** qui ont des valeurs comprises entre **p** et **q** (ou égales à **p** ou **q**). Si l'arbre contient  $n$  nœuds et que  $k$  nœuds doivent être affichés, la complexité de votre fonction ne doit pas dépasser  $\Theta(k + \log n)$ .

(8 pts) **4.f]** Plutôt que d'utiliser une implémentation normale d'arbre AVL, quelqu'un a eu la mauvaise idée d'utiliser la structure `node` précédente dans une implémentation d'arbre binaire de recherche standard. Après insertion de plusieurs nœuds, l'arbre peut donc être déséquilibré. Écrivez une fonction qui permet de rééquilibrer complètement un tel arbre (pour lui donner une vraie structure d'arbre AVL) en recalculant les hauteurs et équilibrages de tous les nœuds de l'arbre et en effectuant toutes les rotations nécessaires.

*On suppose que des fonction `rotg` et `rotd` effectuant des rotations à gauche et à droite sont déjà implémentées. Aussi, au lancement de l'algorithme, `h` vaut -1 dans tous les nœuds de l'arbre (permet de savoir si un nœud a déjà été traité).*

⚠ Votre algorithme peut avoir à effectuer plusieurs rotations pour rééquilibrer un même nœud.

**Exercice 5 : Implémentation d'un automate non-déterministe**

(15 points)

Dans cet exercice on cherche à implémenter un automate non-déterministe et son parcours. Habituellement, pour utiliser un automate non-déterministe on commence par le déterminer, mais ici le but est de le parcourir sans le déterminer.

On considère donc un automate ayant  $m$  états numérotés de 0 à  $m - 1$  et un alphabet de taille  $s$  avec des valeurs de 0 à  $s - 1$ . L'état initial est l'état  $\{0\}$  et à chaque étape du parcours l'automate est donc dans un ensemble d'états courant (inclus dans  $[0, m - 1]$ ). La fonction de transition fait correspondre à un état de départ et un symbole de l'alphabet un ensemble d'états d'arrivée. En lisant un symbole, on part donc d'un ensemble d'états de départ pour arriver dans l'union des ensembles d'arrivée correspondants. L'automate comporte aussi une liste d'états finaux : si l'ensemble d'états courant contient l'un de ces états finaux, alors l'automate est dans un état final.

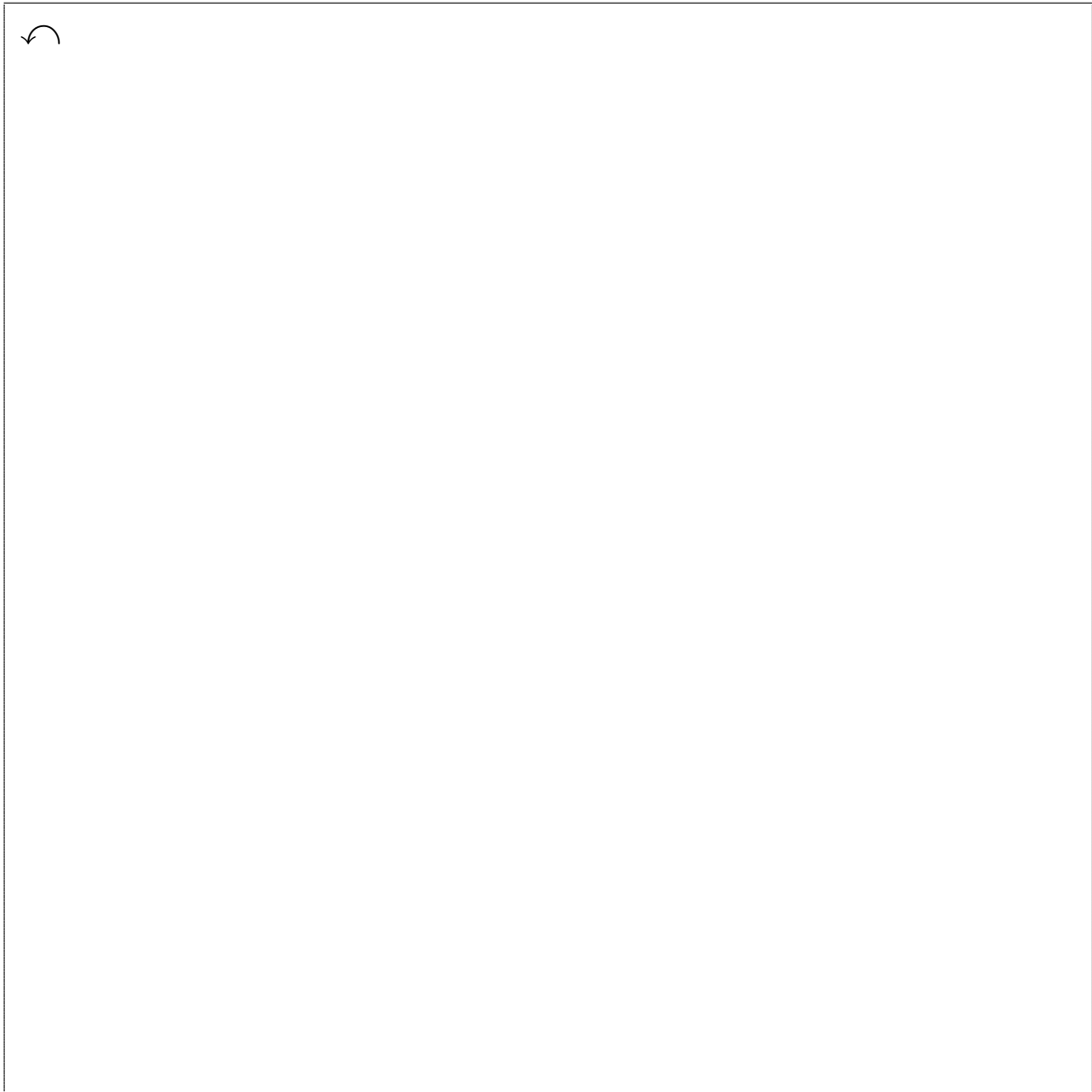
(3 pts) **5.a]** Quelles structures (soit des tableaux, soit des listes chaînées, soit des combinaisons de cela) utiliseriez vous pour représenter :

- la fonction de transition,
- les états finaux,
- l'ensemble d'états courant.

Justifiez vos choix en fonctions des contraintes suivantes : on veut une utilisation mémoire minimale, mais il faut quand même pouvoir, le plus efficacement possible, parcourir les états d'arrivée d'une transition, faire l'union de plusieurs ensembles d'arrivée (tester si un état d'arrivée fait déjà partie de l'ensemble d'états courant), savoir si l'état courant est un état final.

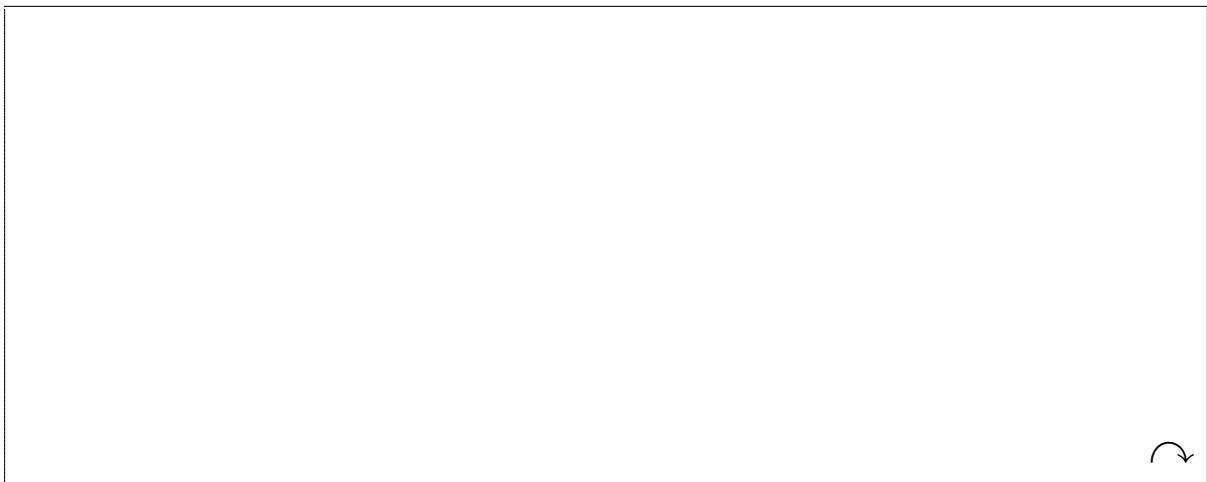
(7 pts) **5.b]** Écrivez une fonction qui prend en argument un automate non-déterministe (sa fonction de transition et ses états finaux), un tableau d'éléments de l'alphabet (des entiers entre 0 et  $s - 1$ ) et la longueur de ce tableau et fait lire le tableau par l'automate. Votre fonction doit renvoyer 1 ou 0 selon que l'automate arrive dans un état final ou pas.





(5 pts) **5.c]** Écrivez maintenant une fonction, similaire à la précédente, qui va déterminer un automate non-déterministe : elle devra construire la fonction de transition du nouvel automate déterministe ainsi que la liste de ses états finaux.

*On suppose que l'automate déterministe a  $2^m$  états (même si tous ne sont pas utilisés) et qu'une fonction `index` prenant en argument un ensemble d'états et retournant un indice entre 0 et  $2^m - 1$  est disponible.*



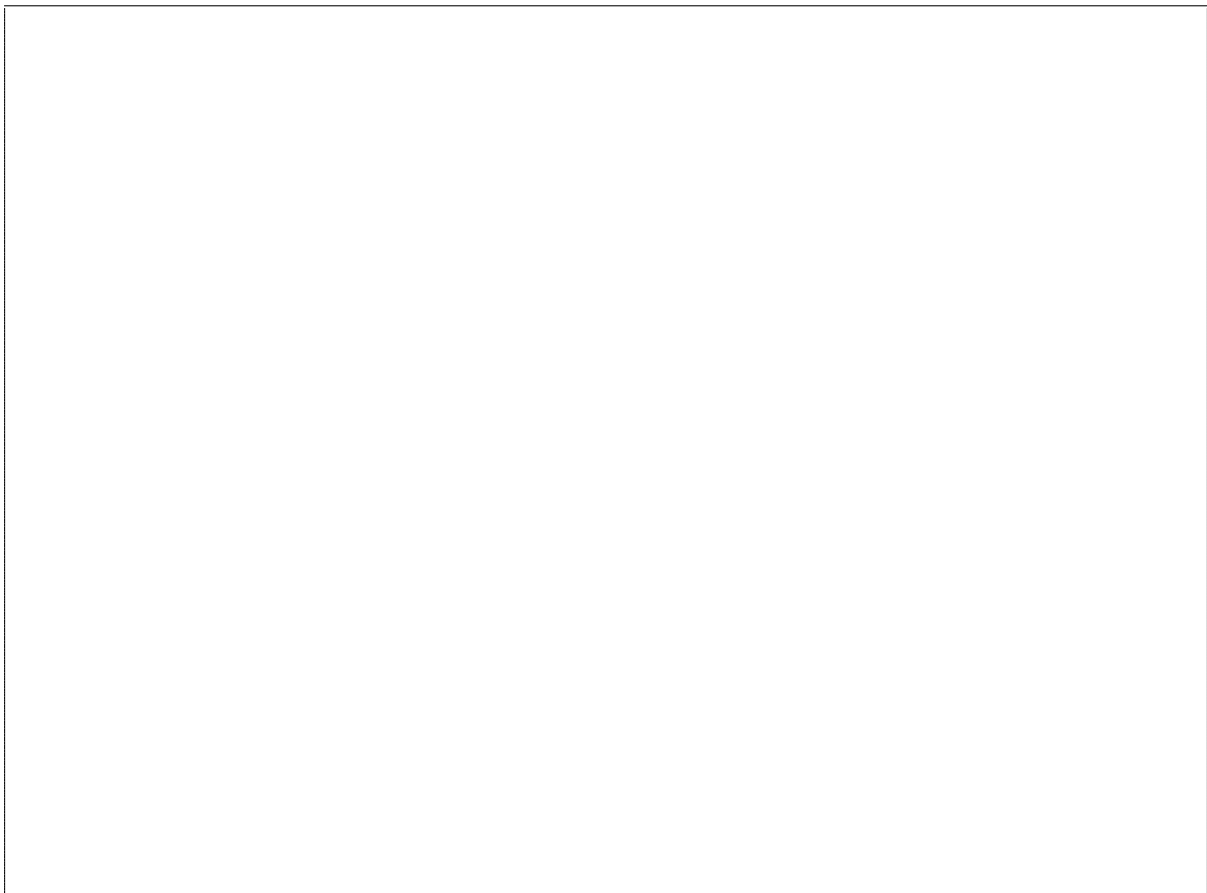


**Exercice 6 :** Dessine-moi un automate

(10 points)

(3 pts)

**6.a]** Dessinez un automate déterministe pour la recherche du motif *abcacaba*. Votre automate doit rester dans l'état final une fois le motif trouvé.



- (3 pts) **6.b]** Dessinez deux automates déterministes qui permettent de tester si un entier est un multiple de 5 pour l'un et un multiple de 6 pour l'autre. L'alphabet est composé des chiffres de 0 à 9 et l'entier est lu chiffre par chiffre en commençant par la gauche : 1, puis 2, puis 4 pour 124 par exemple.



- (2 pts) **6.c]** Dessinez un automate déterministe reconnaissant tous les mots (sur l'alphabet  $\{a, b\}$ ) contenant au moins 4 caractères  $a$ .



- (2 pts) **6.d]** Dessinez un automate déterministe reconnaissant tous les mots (sur l'alphabet  $\{a, b\}$ ) ne contenant aucune séquence de 3  $a$  ou 3  $b$  d'affilée.

