be-OI 2012 **Demi-finale** 1 Février 2012	**Remplissez ce cadre en MAJUSCULES et LISIBLEMENT, svp** PRÉNOM : ..... NOM : ..... ÉCOLE :	**Réservé**

Olympiades belges d'Informatique (durée : 3h maximum)

Ce document est le questionnaire de la demi-finale des Olympiades belges d'Informatique 2012. Il comporte huit questions qui doivent être résolues en **3h au maximum**. Chaque question est accompagnée d'un temps de résolution, donné à titre purement indicatif.

Notes générales (à lire attentivement avant de répondre aux questions)

1. N'indiquez votre nom, prénom et école **que sur la première page**. Sur toutes les autres pages, vous ne pouvez écrire que dans les **cadres prévus** pour votre réponse. Si, suite à une rature, vous êtes amené à écrire hors d'un cadre, répondez obligatoirement sur la même feuille, sans quoi votre réponse ne pourra être corrigée.

2. Vous ne pouvez avoir que de quoi écrire avec vous ; les calculatrices, GSM, ... sont **interdits**.

3. Vos réponses doivent être écrites **au stylo ou au bic** bleu ou noir. Pas de réponses laissées au crayon. Si vous désirez des feuilles de brouillon, demandez-en auprès d'un surveillant.

4. Vous **devez** répondre aux questions ouvertes en **pseudo-code** ou dans l'un des **langages de programmation autorisés** (Java, C, C++, Pascal, Python et PHP). Les erreurs de syntaxe ne sont pas prises en compte pour l'évaluation. Sauf mention contraire, vous ne pouvez utiliser aucune fonction prédéfinie à l'exception de `max(a,b)`, `min(a,b)` et `pow(a,b)` qui calcule a^b.

5. Dans l'ensemble de cet énoncé, si a et b sont des entiers, `a / b` et `a % b` représentent respectivement la division entière et le reste de la division entière de a par b. Par exemple, si a et b valent respectivement 14 et 3, `a / b` vaut 4 et `a % b` vaut 2.

6. Les tableaux sont indicés de 0 à $n-1$, où n correspond à leurs tailles. La notation **`for`** $(i \leftarrow a$ **`to`** b **`step`** $k)$ indique une boucle qui se répète tant que $i \leq b$ pour i initialisé à a et incrémenté de k à la fin de chaque itération.

7. Vous **ne pouvez** à aucun moment **communiquer** avec qui que ce soit, excepté avec les surveillants. Toute question logistique peut leur être posée. A des fins d'égalité entre les participants des différents centres, vous ne **pouvez pas** poser de question **de compréhension** aux surveillants. Toute erreur dans l'énoncé doit être considéré comme faisant partie de l'épreuve.

8. Les participants **ne peuvent en aucun cas quitter leur place** pendant l'épreuve, par exemple pour aller aux toilettes ou pour fumer une cigarette. Selon le lieu où vous présentez l'épreuve, il peut vous être interdit de manger ou boire durant cette dernière.

9. Vous avez **exactement trois heures** pour répondre à ce questionnaire. Un **aide-mémoire** sur le pseudo-code est fourni en annexe à la dernière page.

Bonne chance !

Questionnaire demi-finale

Question 1 – Échauffement (5 min)

Considérons le code suivant :

```
i ← 1
// A
while(i < 5)
{
  // B
  Affiche(i)
  // C
}
```

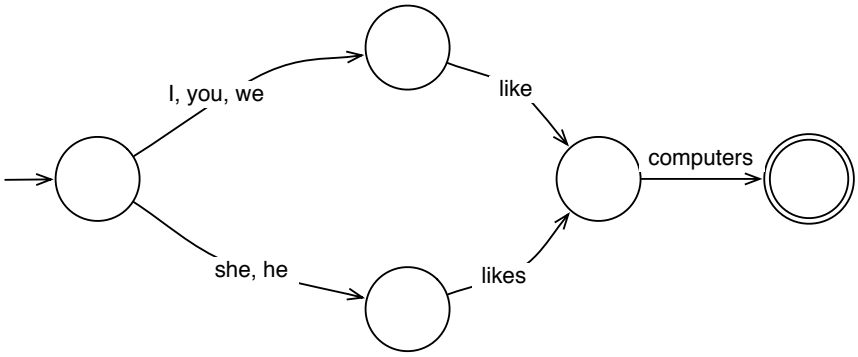
Où faut-il augmenter la variable `i` de 1 pour que cette boucle affiche `1 2 3 4`, sachant que la fonction `Affiche(i)` affiche la valeur de `i` à l'écran ?

Cochez la position à laquelle il faut placer l'instruction $i \leftarrow i + 1$.

Q1	☐ A	☐ B	☐ C

Question 2 – *Let's learn English !* (15 min)

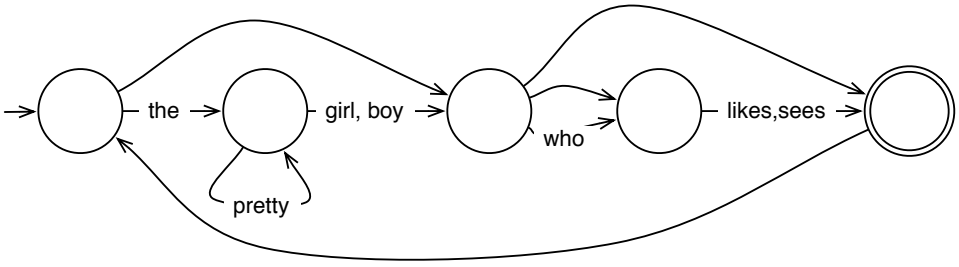
On propose la méthode suivante pour expliquer certaines règles grammaticales de la langue anglaise. Il s'agit d'une méthode basée sur l'utilisation de schémas composés de cercles, appelés *états*, et de flèches, appelées *transitions*, reliant les états entre eux et comportant potentiellement des *étiquettes* (séparées par des virgules s'il y en a plusieurs). En voici un exemple :



On remarque deux états particuliers sur cette figure : l'état le plus à gauche est marqué d'une courte flèche dont l'origine n'est pas un état. Cet état est l'*état initial*. Par ailleurs, l'état le plus à droite est doublé : c'est l'*état final*.

Ce type de diagramme sert à représenter de façon compacte une ou plusieurs phrases. Il se lit comme suit : on commence dans l'état initial, et on parcourt un chemin d'état en état, dans le diagramme, en suivant les transitions, jusqu'à atteindre l'état final. À chaque fois que l'on traverse une transition, on choisit une de ses étiquettes (si elle en comporte). La séquence des étiquettes choisies forme une des phrases représentées. Remarquons qu'il est permis de traverser plusieurs fois l'état final, à condition que le chemin s'y termine. Par exemple, le diagramme ci-dessus représente, entre autres, la phrase « *he like**s** computers* », mais pas « *I like**s** computers* ».

Voici maintenant un autre exemple, plus complexe, qui représente tant des phrases correctes qu'incorrectes en anglais :



Indiquez, pour chacune des phrases ci-dessous, si elle peut être représentée (oui ou non) par ce diagramme. *(1 point par case cochée correcte, -1 point par case cochée incorrecte, pas de score négatif à cette question)*

Oui	Non	
☐	☐	« *the pretty pretty girl sees the boy who likes the pretty girl* »
☐	☐	« *the girl who likes pretty pretty boy* »
☐	☐	« »
☐	☐	« *who sees the pretty girl* »

Question 3 – Un peu de logique (15 min)

Q3(a) – Informaticiens namurois

Cochez la négation de la phrase suivante : "Tous les habitants de Namur qui ont les yeux bruns deviendront informaticiens et prendront leur retraite avant 50 ans."

☐	Il existe au moins un informaticien aux yeux bruns qui n'est pas namurois et qui prendra sa retraite avant 50 ans.
☐	Aucun informaticien namurois aux yeux bruns prendra sa retraite avant 50 ans.
☐	Il existe un habitant de Namur qui a les yeux bruns, qui ne deviendra pas informaticien ou qui prendra sa retraite après 50 ans.
☐	Il n'existe pas de namurois aux yeux bruns devenant informaticien qui prendra sa retraite après 50 ans.

Q3(b) – La chasse à la *galinette cendrée*

Des chasseurs ont organisé une chasse à la *galinette cendrée* dans les bois du *bouchonnois*. N'en ayant attrapé aucune jusqu'au dernier jour de la traque, ils organisent un lâcher de 171 volatiles. Les chasseurs finissent par les abattre toutes sauf 67. **Combien** en reste-t-il ?

Q3(b)	**un nombre entier**
. . . . .	

Q3(c) – La maison hantée

Dans une maison hantée, les esprits se manifestent sous deux formes différentes, un chant obscène et un rire sardonique. On peut cependant influencer leurs comportements en jouant de l'orgue ou en brûlant de l'encens.

Après quelques investigations sur place, on a découvert que :

– le chant ne se fait pas entendre, à moins que l'on ne joue de l'orgue sans que le rire ne se fasse entendre.
– si on brûle de l'encens, le rire se fait entendre uniquement lorsque le chant se fait entendre.

En ce moment, le chant se fait entendre et le rire est silencieux. **Peut-on en conclure** que pour l'instant, on joue de l'orgue et on ne brûle pas d'encens ?

Q3(c)	☐ Oui	☐ Non	☐ On ne sait pas le dire avec certitude

Question 4 – Maximum absolu (25 min)

L'algorithme décrit ci-dessous renvoie la valeur d'un tableau la plus éloignée de 0. Par exemple, pour le tableau tab_1, l'algorithme devrait renvoyer -9.

tab_1

1	7	5	−9	0	2

```
Input  : tab, un tableau d'entiers
         n, la taille de tab. n > 0
Output : Renvoie la valeur de tab la plus éloignée de zéro. Si plusieurs solutions
         sont équivalentes, renvoie la première rencontrée (de gauche à droite).

num ← 0

for (i ← 0 to n − 1 step 1)
{
    if ( [...] )
    {
        num ← tab[i]
    }
}

return num
```

Pour chacune des expressions suivantes, **cochez *oui*** si celles-ci compléteraient cet algorithme de manière à ce qu'il renvoie la valeur attendue quel que soit le tableau donné en entrée, **cochez *non*** sinon. *(1 point par case cochée correcte, -1 point par case cochée incorrecte, pas de score négatif à cette question)*

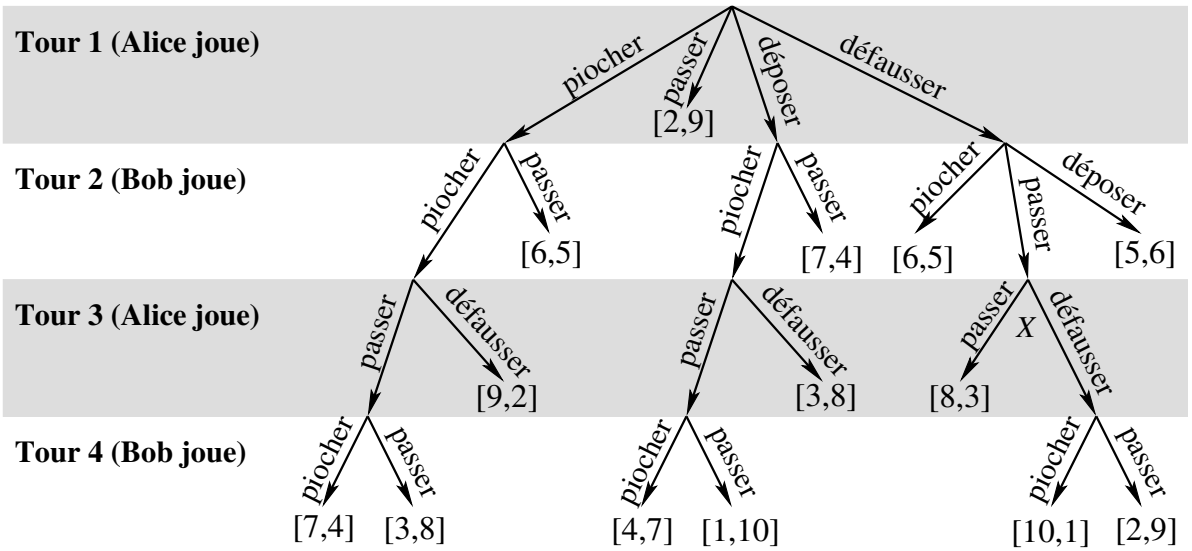
Oui	Non	
☐	☐	$(tab[i] < 0$ **and** $-tab[i] > num)$ **or** $tab[i] > num$
☐	☐	$(num < 0$ **and** $(tab[i] < num$ **or** $tab[i] > -num))$ **or** $(num > 0$ **and** $(tab[i] > num$ **or** $tab[i] < -num))$
☐	☐	$tab[i] > 0$ **and** $tab[i] > num$
☐	☐	$(num \geq 0$ **and** $(tab[i] > num$ **or** $tab[i] < -num))$ **or** $(num \leq 0$ **and** $(tab[i] < num$ **or** $tab[i] > -num))$
☐	☐	$(tab[i] < 0$ **and** $-tab[i] > num)$ **or** $(tab[i] > 0$ **and** $tab[i] > num)$
☐	☐	$-tab[i] > num$ **or** $tab[i] > num$

Question 5 – Jeu de stratégie (15 min)

Alice et Bob jouent à un jeu de stratégie complexe dans lequel ils doivent construire une civilisation plus puissante que celle de leur adversaire. Une civilisation est déterminée par un ensemble de cartes posées par un joueur.

A la fin d'une partie, le score de chacun est déterminé en partageant 11 points de victoire entre les deux joueurs selon différents critères (2 points pour la puissance militaire, 3 points pour l'avancement technologique, etc.). Le joueur avec le score le plus élevé est déclaré vainqueur. Par exemple, une partie endiablée peut se terminer par la victoire sur le fil d'Alice sur un score de 6 points contre 5 points pour Bob (par convention, un tel résultat de fin de partie est noté $[6, 5]$).

Le tirage au sort désigne Alice pour commencer la partie. Elle décide de ne rien laisser au hasard et dessine toutes les situations possibles du jeu (voir ci-dessous). Pour son premier tour de jeu, Alice dispose de 4 coups possibles (*piocher*, *passer*, *déposer* et *défausser*). Ensuite, c'est au tour de Bob de jouer et ainsi de suite jusqu'à ce que le jeu soit terminé, ce qui est représenté par le score de fin de partie. Le nombre de coups possibles dépend de la situation du jeu. Vous pouvez ainsi remarquer que certaines parties peuvent se terminer assez rapidement, par exemple par une défaite d'Alice si celle-ci passe au premier tour.



Q5(a) Si la stratégie de Bob consiste à toujours *passer* invariablement et qu'Alice le sait, **par quel coup** Alice doit-elle commencer pour maximiser son score **et quel sera celui-ci** ?

Q5(a) **(un coup et un nombre)**

. . . . . .

Q5(b) Lors d'une autre partie, Alice et Bob utilisent la même stratégie optimale et ils le savent tous les deux. Les deux joueurs jouent donc du mieux possible, chacun tentant de maximiser son propre score. Par exemple, dans la situation du Tour 3 notée X ci-dessus, Alice choisit de *passer* (pour un score de 8) car si elle choisit de *défausser* (en espérant obtenir 10), Bob décidera de *passer* (et Alice n'obtiendra qu'un score de 2). Le hasard désigne à nouveau Alice pour débuter la partie. **Par quel coup** Alice doit-elle commencer pour maximiser son score **et quel sera celui-ci** ?

Q5(b) **(un coup et un nombre)**

. . . . . .

Question 6 – Croissance de population (20 min)

Bart est un statisticien spécialiste dans l'étude des croissances de population. Dans le cadre d'une comparaison entre différentes cultures, il mesure la taille d'une population donnée à des instants régulièrement espacés dans le temps. Bart aimerait alors, sur base de ces données, prédire la croissance future de la population en question.

Bart fait appel à son amie Lisa, une informaticienne, afin de mettre au point un algorithme qui calcule la taille d'une population à un instant donné, et ce, sur base de la taille de cette population telle qu'observée à un ou plusieurs instants dans le passé. Bart fournit à Lisa une série d'observations qui comprennent la taille initiale de la population. À l'aide de ces données, Lisa calcule la relation qui lie les mesures successives de la population et écrit l'algorithme demandé.

Les algorithmes de Lisa viennent d'arriver. Mais avant qu'il ne les utilise pour des prédictions, Bart aimerait être certain que ces algorithmes fournissent des résultats corrects pour les observations qu'il avait fournies à Lisa.

Aidez Bart à calculer, pour chacun des algorithmes suivants, la taille de la population aux instants $t = 0, 1, 2, 3, 4$ et 5. Le paramètre $t \geq 0$ indique l'instant auquel la taille de la population a été mesurée. L'instant $t = 0$ correspond à la taille initiale de la population.

```
function population1(t)
{
    if (t = 0)
    {
        return 0
    }
    else
    {
        return population1(t-1) + 3
    }
}
```

Q6(a) Calculez la valeur t_i renvoyée par l'appel `population1(i)` :

Q6(a)	$t_0 =$.....	$t_1 =$.....	$t_2 =$.....	$t_3 =$.....	$t_4 =$.....	$t_5 =$.....

```
function population2(t)
{
    if (t = 0)
    {
        return 1
    }
    else
    {
        return population2(t − 1) × 3
    }
}
```

Q6(b) Calculez la valeur t_i renvoyée par l'appel `population2(i)` :

Q6(b)	$t_0 =$.....	$t_1 =$.....	$t_2 =$.....	$t_3 =$.....	$t_4 =$.....	$t_5 =$.....

Question 6 (Suite)

Lisa vient de recevoir une série de nouvelles mesures de la part de Bart et doit maintenant écrire un algorithme qui concorde avec ces données. Vous trouverez ci-dessous une série de mesures faites par Bart et un algorithme incomplet proposé par Lisa. Pour cet algorithme, **indiquez la série d'expressions** qui garantit que le résultat concorde avec les mesures de Bart.

$t_0 = 0$	$t_1 = 1$	$t_2 = 4$	$t_3 = 9$	$t_4 = 16$	$t_5 = 25$

```
function population3(t)
{
    if ([...])                  // A
    {
        return [...]            // B
    }
    else
    {
        return [...]            // C
    }
}
```

Q6(c)	A	B	C
☐	`t = 0`	`0`	`population3(t-1) + t × t`
☐	`t ≤ 1`	`t`	`population3(t-1) × 2`
☐	`t = 0`	`0`	`population3(t-1) + ( 2 × t ) - 1`
☐	`t < 1`	`t`	`population3(t-1) × population3(t-1)`

Question 6 (Suite)

Vous trouverez ci-dessous de nouveau un algorithme incomplet proposé par Lisa. **Remplissez les parties manquantes** de manière à ce que le résultat de l'algorithme concorde avec les mesures de Bart.

$t_0 = 0$	$t_1 = 1$	$t_2 = 1$	$t_3 = 2$	$t_4 = 3$	$t_5 = 5$	$t_6 = 8$

```
function population4(t)
{
  if (t < 2)
  {
    return [...]            // Q6(d)
  }
  else
  {
    return [...]            // Q6(e)
  }
}
```

Q6(d) **(une expression)**

. . . . . .

Q6(e) **(une expression)**

. . . . . .

Question 7 – La gare de triage (25 min)

Dans la composition d'un train de marchandises, l'ordre des wagons a son importance : les wagons doivent être placés dans l'ordre dans lequel leur contenu doit être livré. De cette façon, il suffit au conducteur de train de détacher, à chaque gare, le dernier wagon du train (contenant le chargement destiné à cette gare). Si l'ordre est incorrect, il faut détacher des wagons au milieu du train, les déplacer vers une autre voie, *etc*, ce qui coûte beaucoup de temps et d'efforts. Le métier de « trieur de wagons » a été découvert par hasard, par un cheminot qui s'occupait d'un ponton de chemin de fer tournant. Le ponton de chemin de fer tourne autour d'un axe, placé au centre de la rivière qu'il enjambe. En faisant tourner le pont de 90 degrés, on permet aux bateaux de passer tant à gauche qu'à droite du pont. Le premier *trieur de wagons* découvrit également que le pont pouvait fonctionner si on y plaçait deux wagons. En faisant alors tourner le pont de 180 degrés, on échange les positions des wagons dans la composition du train. L'ordre correct pour les wagons peut alors être obtenu par une série d'échanges successifs.

Aujourd'hui, presque tous les *trieurs de wagons* sont pensionnés et la SNCB désire automatiser leur travail. Une part du programme informatique nécessaire à cette automatisation consiste en une fonction qui, pour une composition de train donnée, calcule le nombre minimum d'échanges de wagons voisins qui sont nécessaires pour obtenir un train ordonné. On vous assigne la tâche d'écrire cette fonction. Votre fonction reçoit en entrée un nombre entier N correspondant à la longueur du train et un tableau `wagons` de longueur N qui donne l'ordre initial des wagons, sous forme d'une permutation des nombres de 1 à N. Par exemple, si la valeur de `wagons` est $[3, 2, 1]$, les wagons doivent être ré-ordonnés par une série d'échanges de telle manière que le wagon 1 arrive en première position (c'est-à-dire *wagons*$[0] = 1$), le wagon 2 arrive en deuxième position (c'est-à-dire *wagons*$[1] = 2$), jusqu'au wagon N en dernière position. Donc, si nous commençons avec *wagons* valant $[3, 2, 1]$, la fonction doit retourner le tableau $[1, 2, 3]$. Cela est possible en échangeant d'abord *wagons*$[0]$ et *wagons*$[1]$ (on obtient alors le tableau $[2, 3, 1]$), puis *wagons*$[1]$ et *wagons*$[2]$ (on obtient $[2, 1, 3]$), et finalement à nouveau *wagons*$[0]$ et *wagons*$[1]$ (et on obtient bien $[1, 2, 3]$)). Dans cet exemple, trois échanges ont été nécessaires pour réordonner les wagons. Avec trois wagons, au plus trois échanges seront nécessaires pour réordonner les wagons (vous pouvez facilement le vérifier vous-même). L'exemple $[3, 2, 1]$ est donc un *pire cas* pour réordonner trois wagons.

Considérons, pour commencer, les instances suivantes pour N et *wagons*. Pour chaque instance, **indiquez le nombre minimum d'échanges nécessaires** pour obtenir un train ordonné.

Q7(a)	$N = 4$	`wagons`$= [2, 4, 3, 1]$	.....
Q7(b)	$N = 4$	`wagons`$= [4, 3, 2, 1]$	.....
Q7(c)	$N = 5$	`wagons`$= [5, 4, 3, 2, 1]$	.....
Q7(d)	$N = 5$	`wagons`$= [4, 5, 1, 3, 2]$	.....

Donnez, sur base de ces exemples, **le nombre minimal d'échanges nécessaires** *au pire cas* pour un train de longueur N.

Q7(e)	☐ N	☐ $2N$	☐ $\sqrt{N}$	☐ $\frac{N \times (N-1)}{2}$	☐ N^3

Question 7 (suite)

Une fonction qui, quel que soit l'entrée, calcule le nombre minimal d'échanges nécessaires à la forme suivante. **Complétez** cette fonction.

```
Input:  wagons, un tableau d'entiers positifs compris entre 0 et N − 1 (inclus),
        tous différents.
        N, la taille du tableau wagons.
Output: le nombre minimal de permutations nécessaires afin que wagons soit trié.

function compte_echanges(wagons, N)
{
    nbrEchanges ← 0
    for (k ← 0 to (N − 2) step 1)
    {
      for (i ← 0 to [...] step [...])      // Q7(f), Q7(g)
      {
        if (wagons[i] > wagons[i + 1])
        {
          swap(i, i + 1)                   // échange les valeurs de wagons[i] et wagons[i + 1]
          [...]                            // Q7(h)
        }
      }
    }
    return nbrEchanges
}
```

Q7(f) **(une expression)**

. . . . . .

Q7(g) **(une expression)**

. . . . . .

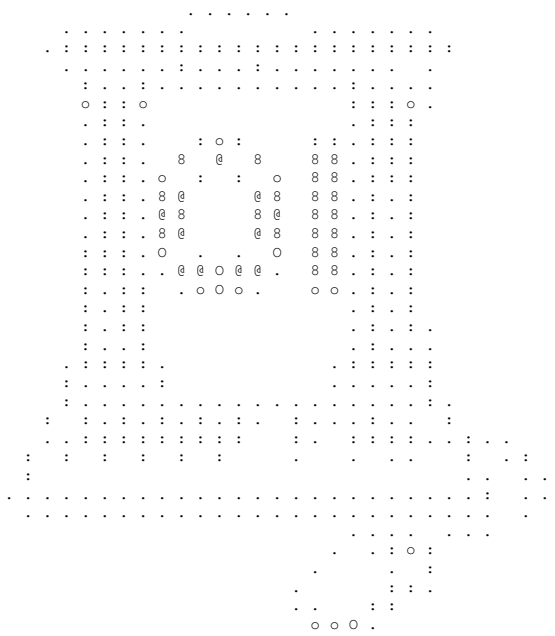
Q7(h) **(une instruction)**

. . . . . .

Question 8 – ASCII art (25 min)

L'*ASCII-Art* est une représentation graphique d'une image dans laquelle des points ou groupes de points sont remplacés par des caractères. La difficulté de l'*ASCII-Art* réside dans le choix du caractère le plus représentatif pour un groupe de points donnés. Il existe des techniques très élaborées pour effectuer automatiquement ce choix.

La technique mise en oeuvre dans cette question est simple : elle repose sur l'utilisation de différents caractères pour figurer différentes intensités de gris. A titre d'illustration, la figure ci-dessous montre le logo de l'olympiade informatique (initialement une image de 120×160 points) convertie en une matrice de 30×40 caractères. Pour effectuer cette conversion, les 7 caractères donnés dans le tableau à droite de l'image ont été utilisés. Dans cet exemple, chaque caractère correspond à l'intensité moyenne d'un groupe de 16 points (4×4), chaque point de l'image d'origine ayant une intensité de gris mesurée entre 0 (blanc) et 255 (noir).

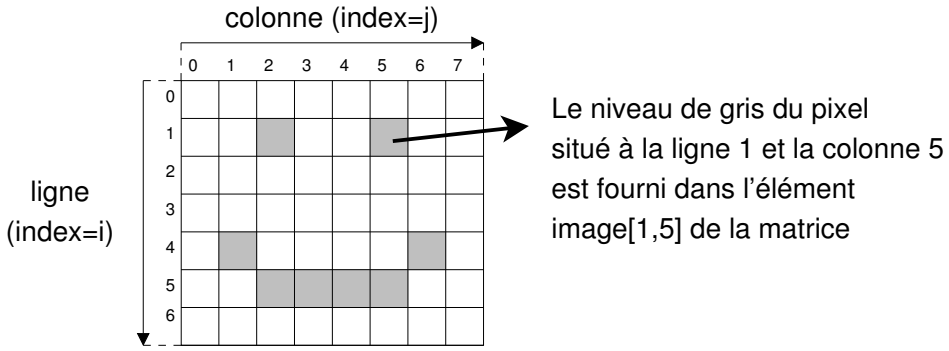


Niveau de gris de l'image initiale	Caractère ASCII résultant
$0 - 36$	(espace)
$37 - 73$	.
$74 - 109$	:
$110 - 146$	o
$147 - 182$	O
$183 - 219$	8
$220 - 255$	@

L'algorithme permettant de convertir en *ASCII-Art* une image exprimée en niveaux de gris vous est fourni ci-après. Cependant, certaines parties de l'algorithme ont été omises. A vous de les **compléter** !

L'algorithme prend en entrée l'image d'origine fournie sous la forme d'une matrice d'entiers, le tableau de caractères utilisés pour la conversion, ainsi qu'une matrice destinée à stocker le résultat, c'est-à-dire les caractères de la représentation *ASCII-Art*.

L'image d'origine est fournie sous la forme d'une matrice `image` de taille $m \times n$ où m est le nombre de lignes et n est le nombre de colonnes. Chaque élément de la matrice `image[i,j]` est un entier compris dans l'intervalle $[0, 255]$ qui représente l'intensité de gris du point (*pixel*) situé à la ligne i et à la colonne j. Dans la notation `image[i,j]`, i est l'index de la ligne ($0 \leq i < m$) et j est l'index de la colonne ($0 \leq j < n$). L'exemple fourni ci-dessous illustre la représentation par une matrice 7×8 d'une image de 8 pixels de large sur 7 pixels de haut.



Les caractères à employer pour représenter les niveaux de gris sont fournis dans le tableau `caracteres` de longueur k. Les caractères du tableau sont ordonnés du plus petit niveau de gris (blanc) au plus grand niveau de gris (noir). Il est possible d'invoquer l'algorithme de conversion avec des ensembles de caractères de longueurs et de contenus variables. L'implémentation de l'algorithme doit s'adapter automatiquement au tableau de caractères fourni. Par exemple, si le tableau contient $k = 4$ caractères, le caractère d'index 0 sera utilisé pour représenter les niveaux de gris de 0 à 63, celui d'index 1 pour les niveaux de 64 à 127 et ainsi de suite.

Le résultat de l'algorithme est placé dans la matrice `ascii` de taille $p \times q$ telle que m (respectivement n) est un multiple entier de p (respectivement q) de sorte que chaque élément `ascii[k,l]` de la matrice soit un caractère qui représente l'intensité de gris d'un rectangle de points de l'image d'origine. Les conventions pour les indices des colonnes et des lignes sont identiques à celles de la matrice `image`.

```
Input  : image, une matrice non-vide de dimension m × n d'entiers compris entre [0, 255].
         caracteres, un tableau de caractères de taille k > 0.
         ascii, une matrice non-vide de caractères de dimension p × q telle que
         m et n sont respectivement multiples entiers de p et q.
Output : Les éléments de la matrice ascii ont reçu une valeur.
         L'algorithme ne renvoie rien.

function converti(image, m, n, caracteres, k, ascii, p, q)
{
  for (i ← 0 to p − 1 step 1)
  {
    for (j ← 0 to q − 1 step 1)
    {
      s ← 0
      for (a ← 0 to [...] step 1)                          // Q8(a)
      {
        for (b ← 0 to [...] step 1)                        // Q8(b)
        {
          s ← s + image[ [...] , [...] ]                   // Q8(c), Q8(d)
        }
      }
      ascii[i, j] ← caracteres[ [...] ]                    // Q8(e)
    }
  }
}
```

Q8(a) **(une expression)**

. . . . . .

Q8(b) **(une expression)**

. . . . . .

Q8(c) **(une expression)**

. . . . . .

Q8(d) **(une expression)**

. . . . . .

Q8(e) **(une expression)**

. . . . . .

Aide-mémoire pseudo-code

Les données sont stockées dans des variables et il est possible d'effectuer des opérations arithmétiques avec ces dernières. En plus des quatre opérateurs classiques ($+$, $-$, $\times$ et $/$), vous pouvez également utiliser $\%$ (voir définitions en première page). La valeur d'une variable est modifiée avec $\leftarrow$. Il y a des variables permettant de stocker des entiers, des nombres réels, des tableaux et des booléens : vrai (**`true`**) ou faux (**`false`**).

```
annee_naissance ← 1992
age ← 2012 − annee_naissance
```

On peut n'exécuter du code que si une certaine condition est vérifiée en utilisant une instruction **`if`** et éventuellement un autre code dans le cas contraire en ajoutant une instruction **`else`**. L'exemple suivant vérifie si une personne est majeure ou non et stocke la réponse dans la variable booléenne (vrai ou faux) $majeur$.

```
if (age ≥ 18)
{
    majeur ← true
}
else
{
    majeur ← false
}
```

Pour manipuler plusieurs éléments d'un coup, on utilise un tableau. Il s'agit d'une liste de valeurs qu'on va pouvoir accéder en utilisant les indices des éléments. Le premier élément de la liste est l'élément d'indice 0. Le second est celui d'indice 1 … et le dernier est celui d'indice $N-1$ si N indique le nombre total d'éléments de la liste (sa taille). Par exemple, pour le tableau tab qui vaut $[5, 9, 12]$, $tab[0]$ indique le premier élément (à savoir 5) et $tab[2]$ indique le dernier (à savoir 12).

On peut exécuter une boucle en utilisant **`for`** (définition à la première page). L'exemple suivant calcule la somme des éléments du tableau tab en supposant que sa taille vaut N. La somme se trouve dans la variable sum à la fin de l'exécution de l'algorithme.

```
sum ← 0
for (i ← 0 to N − 1 step 1)
{
    sum ← sum + tab[i]
}
```

On peut également exécuter une boucle avec **`while`** qui repète du code tant que sa condition est vraie. Dans l'exemple suivant, on va diviser un nombre entier positif N par 2, puis par 3, ensuite par 4 … jusqu'à ce qu'il ne soit plus composé que d'un seul chiffre (c'est-à-dire qu'il est < 10).

```
d ← 2
while (N > 10)
{
    N ← N/d
    d ← d + 1
}
```