be-OI 2014	Remplissez ce cadre en MAJUSCULES et LISIBLEMENT, svp	O
Finale samedi 8 février 2014	PRÉNOM : ..... NOM : ..... ÉCOLE :	**Réservé**

Olympiade belge d'Informatique (durée : 1h15 maximum)

Ce document est le questionnaire de la finale de l'Olympiade belge d'Informatique 2014. Il comporte quatre questions qui doivent être résolues en **1h15 au maximum**. Chaque question est accompagnée d'un temps de résolution indicatif, ainsi que de son score maximal possible.

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

1. N'indiquez votre nom, prénom et école **que sur la première page**. Indiquez vos réponses sur les pages prévues à cet effet, à la fin du formulaire. 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. Quand vous avez terminé, remettez au surveillant la première page (avec votre nom) et les pages avec les réponses. Vous pouvez conserver les autres.

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

4. 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.

5. Tous les extraits de code de l'énoncé sont en **pseudo-code**. Vous trouverez, sur la page suivante, une **description** du pseudo-code que nous utilisons.

   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.

6. Pour toutes les questions à choix multiples (cases à cocher), une mauvaise réponse vous fait *perdre* le même nombre de points qu'elle peut vous en rapporter. Ne pas répondre ne fait ni perdre ni gagner de points. Si vous obtenez un score négatif à une question, votre score pour cette question sera ramené à zéro. Pour répondre à une question à choix multiples, cochez la case (☒) correspondant à votre réponse. Pour annuler une réponse, noircissez entièrement la case (■).

7. Vous **ne pouvez** à aucun moment **communiquer** avec qui que ce soit, excepté avec les surveillants. Vous pouvez par exemple leur demander du papier de brouillon, mais vous ne pouvez pas poser de question au sujet de l'épreuve. Toute erreur dans l'énoncé doit être considérée comme faisant partie de l'épreuve.

8. Vous **ne pouvez pas quitter votre place** pendant l'épreuve. Si vous devez vous rendre aux toilettes, faites-en la demande à un surveillant. Ce dernier pourra décider d'accepter ou de refuser votre requête selon qu'il soit possible, ou pas, de vous accompagner tout en assurant la surveillance de l'épreuve. Selon le lieu où vous présentez l'épreuve, il peut vous être interdit de manger ou boire durant celle-ci.

Solutions

Cette oeuvre est mise à disposition selon les termes de
la Licence Creative Commons Attribution 2.0 Belgique.

Aide-mémoire pseudo-code

Les données sont stockées dans des variables. On change la valeur d'une variable à l'aide de $\leftarrow$. Dans une variable, nous pouvons stocker des nombres entiers, des nombres réels, ou des tableaux (voir plus loin), ainsi que des valeurs booléennes (logiques): vrai/juste (**`true`**) ou faux/erroné (**`false`**). Il est possible d'effectuer des opérations arithmétiques sur des variables. En plus des quatre opérateurs classiques ($+$, $-$, $\times$ et $/$), vous pouvez également utiliser $\%$: si a et b sont des nombres entiers, alors a/b et $a\%b$ désignent respectivement le quotient et le reste de la division entière. Par exemple, si $a = 14$ et $b = 3$, alors: $a/b = 4$ et $a\%b = 2$. Dans le code suivant, la variable age reçoit 20.

```
annee_naissance ← 1993
age ← 2013 − annee_naissance
```

Pour exécuter du code uniquement si une certaine condition est vraie, on utilise l'instruction **`if`** et éventuellement l'instruction **`else`** pour exécuter un autre code si la condition est fausse. L'exemple suivant vérifie si une personne est majeure et stocke le prix de son ticket de cinéma dans la variable $prix$. Notez les commentaires dans le code.

```
if (age ≥ 18)
{
    prix ← 8 // Ceci est un commentaire.
}
else
{
    prix ← 6 // Prix réduit
}
```

Pour manipuler plusieurs éléments avec une seule variable, on utilise un tableau. Les éléments individuels d'un tableau sont indiqués par un index (que l'on écrit entre crochets après le nom du tableau). Le premier élément d'un tableau tab est d'indice 0 et est noté $tab[0]$. Le second est celui d'indice 1 et le dernier est celui d'indice $N - 1$ si le tableau contient N éléments. Par exemple, si le tableau tab contient les 3 nombres 5, 9 et 12 (dans cet ordre), alors $tab[0] = 5$, $tab[1] = 9$, $tab[2] = 12$. La longueur est 3, mais l'index le plus élevé est 2.

Pour répéter du code, par exemple pour parcourir les éléments d'un tableau, on peut utiliser une boucle **`for`**. La notation **`for`** $(i \leftarrow a$ **`to`** b **`step`** $k)$ représente une boucle qui sera répétée tant que $i \leq b$, dans laquelle i commence à la valeur a, et est augmentée de k à la fin de chaque étape. 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 écrire une boucle à l'aide de l'instruction **`while`** qui répète 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
}
```

Solutions

Question 1 – Échauffement (15 min – 16 pts)

Dans cette question, on vous demande de prévoir l'effet de plusieurs extraits de code, qui dépendent tous de la valeur d'une variable N.

Considérons le programme suivant:

```
Input : une valeur entière strictement positive N

V ← 1

for (i ← 1 to N step 1)
{
   for (j ← 1 to N step 1)
   {
      Afficher(V)
      V ← V + 1
   }
}
```

Q1(a) [1 pt]	**Si $N = 3$ quelle est la séquence de valeurs affichées par ce programme ?**
Solution: 1, 2, 3, 4, 5, 6, 7, 8, 9	

Q1(b) [1 pt]	**Si $N = 16$ quelle est la plus grande valeur affichée par le programme ?**
Solution: 256	

Q1(c) [2 pts]	**Donnez, en fonction de N, la plus grande valeur affichée par le programme**
Solution: N^2	

Considérons ce nouveau programme. La différence par rapport au précédent a été encadrée:

```
Input : une valeur entière strictement positive N

V ← 1

for (i ← 1 to N step 1)
{
   for (j ← 1 to [i] step 1)
   {
      Afficher(V)
      V ← V + 1
   }
}
```

Q1(d) [2 pts]	**Si $N = 3$ quelle est la séquence de valeurs affichées par ce programme ?**
Solution: 1, 2, 3, 4, 5, 6	

Solutions

Q1(e) [3 pts]	**Si $N = 16$ quelle est la plus grande valeur affichée par le programme ?**
Solution: 136	

Finalement, considérons ce dernier programme avec trois boucles imbriquées:

```
Input : une valeur entière strictement positive N

V ← 1

for (i ← 1 to N step 1)
{
  for (j ← 1 to i step 1)
  {
    for (k ← 1 to j step 1)
    {
      Afficher(V)
      V ← V+1
    }
  }
}
```

Q1(f) [3 pts]	**Si $N = 3$ quelle est la séquence de valeurs affichées par ce programme ?**
Solution: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10	

Q1(g) [4 pts]	**Si $N = 5$ quelle est la plus grande valeur affichée par le programme ?**
Solution: $1 + (1 + 2) + (1 + 2 + 3) + (1 + 2 + 3 + 4) + (1 + 2 + 3 + 4 + 5) = 35$	

Solutions

Question 2 – Sudoku (15 min – 16 pts)

Un sudoku se compose d'un carré de 9 lignes et 9 colonnes, et de 9 carrés de 3 par 3, délimités par des lignes noires. Sur chaque ligne, sur chaque colonne, et dans chaque carré, tous les nombres de 1 à 9 ne doivent apparaître qu'une et une seule fois.

La figure de gauche, ci-dessous, est un exemple de sudoku correctement rempli. Celle de droite n'est pas correctement remplie. En effet, le nombre 6 apparaît deux fois dans le carré du haut, à gauche, par exemple.

2	9	5	7	4	3	8	6	1
4	3	1	8	6	5	9	2	7
8	7	6	1	9	2	5	4	3
3	8	7	4	5	9	2	1	6
6	1	2	3	8	7	4	9	5
5	4	9	2	1	6	7	3	8
7	6	3	5	2	4	1	8	9
9	2	8	6	7	1	3	5	4
1	5	4	9	3	8	6	7	2

2	9	5	7	4	3	8	6	1
4	**6**	1	8	6	5	9	2	7
8	7	**6**	1	9	2	5	4	3
3	8	7	4	5	9	2	1	6
6	1	2	3	8	7	4	9	5
5	4	9	2	1	6	7	3	8
7	6	3	5	2	4	1	8	9
9	2	8	6	7	1	3	5	4
1	5	4	9	3	8	6	7	2

Le programme suivant vérifie qu'un sudoku est correctement rempli. Le seul argument de la fonction `sudoku` est un carré de 9 par 9 qui est complètement rempli. Tous les nombres du sudoku sont compris entre 1 et 9, inclus. Certaines lignes du code doivent être complétées.

Q2(a) [4 pts]	**Donnez l'expression** `(a)`
Solution: $b[s[j][i] - 1] + 1$	

Q2(b) [3 pts]	**Donnez la valeur** `(b)`
Solution: 1	

Q2(c) [5 pts]	**Donnez l'expression** `(c)`
Solution: $3 \times m + i$	

Q2(d) [4 pts]	**Donnez l'expression** `(d)`
Solution: $a[s[x][y] - 1] + 1$	

Solutions

```
Input : s, un tableau de 9 par 9, contenant des nombres entiers de 1 à 9.
Output : OK contient true si et seulement si s contient un sudoku correct.

Le code fait usage de deux tableaux supplémentaires a et b (contenant 9 entiers).

OK ← true

for(i ← 0 to 8 step 1)
{
  for(j ← 0 to 8 step 1)
  {
    a[j] ← 0
    b[j] ← 0
  }

  for(j ← 0 to 8 step 1)
  {
    a[s[i][j] − 1] ← [ ... ] // pas demandé
    b[s[j][i] − 1] ← [ ... ] // (a)
  }

  for (j ← 0 to 8 step 1)
  {
    if(a[j] > [...] or b[j] > [...]) // (b) (ne donnez que la 1ère expression)
    {
      OK ← false
    }
  }
}

for(m ← 0 to 2 step 1)
{
  for(n ← 0 to 2 step 1)
  {
    for (i ← 0 to 8 step 1)
    {
      a[i] ← 0
    }

    for(i ← 0 to 2 step 1)
    {
      for(j ← 0 to 2 step 1)
      {
        x ← [...] // (c)
        y ← [...] // pas demandé

        a[s[x][y] − 1] ← [...] // (d)
      }
    }

    for (j ← 0 to 8 step 1)
    {
      if(a[j] > [...]) // pas demandé
      {
        OK ← false
      }
    }
  }
}
```

Solutions

Question 3 – Recherche dans un tableau ordonné (25 min – 24 pts)

On s'intéresse à la recherche d'un élément t dans un tableau tab à deux dimensions. On appelle $tab[i][j]$ l'élément à la i^e ligne et j^e colonne (où, comme d'habitude, les lignes et les colonnes sont numérotées à partir de 0). On suppose que le tableau possède N lignes et M colonnes et qu'il est ordonné, c'est-à-dire que les valeurs dans chaque ligne et chaque colonne sont en ordre croissant. Voici un exemple d'un tableau ordonné avec $N = 2$ et $M = 3$:

1	2	2
1	3	7

Supposons que $N = M = 5$. Pour chacune des quatre conditions suivantes, **barrez** les cases du tableau qui suit où on est **sûr** que t ne peut **pas** se trouver.

Q3(a) [1 pt] **Dans quelles cases t ne peut-il pas se trouver si $t < tab[2][2]$?**

Solution:

	0	1	2	3	4
0					
1					
2			×	×	×
3			×	×	×
4			×	×	×

Q3(b) [1 pt] **Dans quelles cases t ne peut-il pas se trouver si $t > tab[2][4]$?**

Solution:

	0	1	2	3	4
0	×	×	×	×	×
1	×	×	×	×	×
2	×	×	×	×	×
3					
4					

Q3(c) [2 pts] **Dans quelles cases t ne peut-il pas se trouver si $tab[0][2] < t < tab[0][3]$ *et* $tab[2][0] < t < tab[3][0]$?**

Solution:

	0	1	2	3	4
0	×	×	×	×	×
1	×			×	×
2	×			×	×
3	×	×	×	×	×
4	×	×	×	×	×

Solutions

Q3(d) [3 pts]	**Dans quelles cases t ne peut-il pas se trouver si $tab[0][2] < t < tab[0][4]$ *et* $tab[1][0] < t < tab[3][0]$?**

Solution:

	0	1	2	3	4
0	×	×	×		×
1	×				×
2					×
3	×	×	×	×	×
4	×	×	×	×	×

Si l'on souhaite rechercher un élément t dans un tel tableau, une simple technique pourrait consister à parcourir tout le tableau et vérifier pour chaque case si $tab[i][j] = t$. On souhaiterait néanmoins trouver un algorithme plus efficace, qui exploite le caractère trié du tableau pour ne pas devoir le parcourir entièrement. Le professeur Yunoac vous propose l'algorithme suivant:

On commence par définir la ligne du milieu comme la ligne numéro $m = \frac{N-1}{2}$. On compare t au dernier élément de la ligne du milieu, c'est-à-dire $tab[m][M-1]$. Si t est supérieur ou égal à cet élément, on rappelle l'algorithme sur le tableau résultant de l'élimination des lignes 0 à m inclus. Sinon, on rappelle l'algorithme sur le tableau résultant de l'élimination des lignes $m+1$ à $N-1$. Quand le tableau ne possède plus qu'une ligne on regarde si l'élément se trouve dans cette ligne en le comparant avec tous les éléments de la ligne.

Le professeur Yunoac était manifestement mal réveillé, le jour où il a proposé cet algorithme, car une de ses étudiantes se rend immédiatement compte qu'il est faux ! Elle prétend avoir trouvé un tableau avec seulement 3 lignes et 2 colonnes, et une valeur t pour lesquels l'algorithme retourne une réponse fausse. Pouvez vous faire de même ?

Q3(e) [3 pts]	**Donnez un tableau ordonné avec $N = 3$ et $M = 2$, et une valeur t qui prouvent que l'algorithme du professeur Yunoac est faux.**

Solution:

	0	1
0	1	3
1	2	4
2	3	5

$t = 2$

L'étudiante du professeur Yunoac propose alors un algorithme tout a fait différent.

On commence à la case en bas à gauche du tableau, c'est-à-dire, la position $tab[N-1][0]$. Avant chaque pas, on compare t avec la position courante. Si t est trouvé à la position courante, on s'arrête (en affichant que t est trouvé). Si t est plus grand que la position courante, alors on avance d'une position vers la droite (on va sur la colonne suivante). Sinon on avance d'une position vers le haut (on va sur la ligne précédente). Si on sort du tableau, alors on s'arrête et on affiche que t n'est pas dans tab.

Solutions

Complétez le pseudo-code suivant de façon à implémenter cet algorithme. A la fin de cet algorithme la variable $trouve$ doit contenir $true$ (vrai) ou $false$ (faux) selon que la valeur t se trouve dans le tableau ou non.

```
Input:
- tab: un tableau ordonné
- N: le nombre de lignes du tableau
- M: le nombre de colonnes du tableau

i ← N-1
j ← 0
trouve ← false

while( !trouve and i >= 0 and [...])          // (f)
{
  if( [...] )                                // (g)
  {
    trouve ← true
  }
  else if(t > tab[i][j])
  {
    [...]                                    // (h)
  }
  else
  {
    [...]                                    // (i)
  }
}
```

Q3(f) [4 pts]	**Complétez la condition** `(f)`
Solution: $j < M$	

Q3(g) [2 pts]	**Complétez la condition** `(g)`
Solution: $t = tab[i][j]$	

Q3(h) [2 pts]	**Complétez l'instruction** `(h)`
Solution: $j \leftarrow j + 1$	

Q3(i) [2 pts]	**Complétez l'instruction** `(i)`
Solution: $i \leftarrow i - 1$	

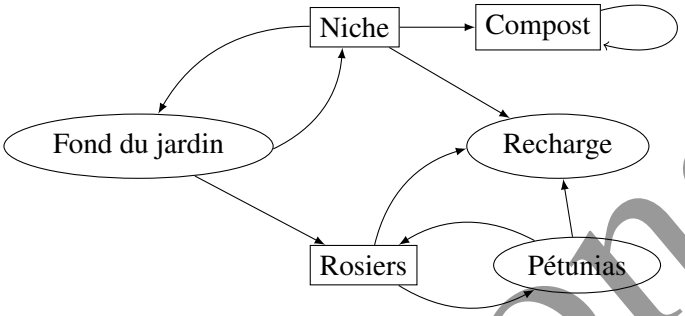
Q3(j) [4 pts]	**Quel est, en fonction de N et de M, le nombre maximum de fois que la condition** `(g)` **sera testée ?**
Solution: $N + M - 1$	

Solutions

Question 4 – Le robot jardinier (20 min – 19 pts)

Le SupraMow 3000 ® est un nouveau robot jardinier qui se charge de la tonte du gazon et du ramassage des feuilles mortes. Les concepteurs du robot font appel à vous pour les aider à réaliser le logiciel de contrôle qui lui permettra de se déplacer sans encombre dans le jardin.

Le SupraMow 3000 ® dispose d'une carte du jardin, dont on voit un exemple ci-dessous. Chaque ellipse et chaque rectangle représente un *lieu*, indiqué par son nom. Les flèches représentent les déplacements possibles. Dans l'exemple ci-dessous, le robot commence son parcours dans le lieu *fond du jardin* et souhaite atteindre la *station de recharge* pour charger ses batteries:



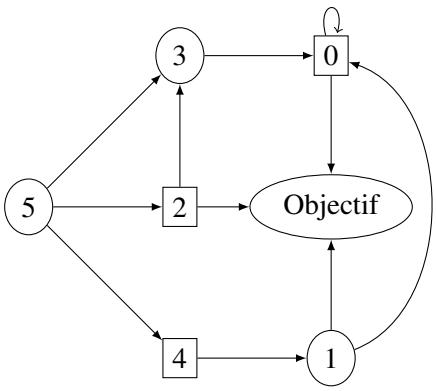
Malheureusement, le SupraMow 3000 ® ne contrôle pas tout ce qui se passe quand il se déplace, ce qui est reflété sur l'image du dessus par les deux formes différentes (rectangle et ellipse) pour représenter les différents lieux. Les lieux de forme elliptiques sont appelés *contrôlables*: le SupraMow 3000 ® peut décider où aller. Par exemple, du fond du jardin, le robot peut décider d'aller vers la niche du chien ou vers les rosiers. Les lieux représentés par des rectangles sont *incontrôlables*: le SupraMow 3000 ® ne contrôle pas où il va depuis ces endroits. Par exemple, la niche du chien: si le chien sort de sa niche et donne un coup de patte dans le robot, celui-ci peut être renvoyé dans le fond du jardin, ou tomber sur le tas de compost. Du côté des rosiers, le terrain est en pente, et le robot pourrait glisser vers les pétunias (d'où il peut décider de rejoindre la station de recharge ou de retourner vers les rosiers).

On voit donc que, depuis le *fond du jardin*, choisir d'aller vers la *niche* n'est pas une bonne *stratégie*, car le chien pourrait, de façon répétée, renvoyer le robot dans le *fond*. Pire, le chien pourrait envoyer le SupraMow 3000 ® sur le tas de *compost*, d'où il ne peut plus s'échapper. Par contre, aller vers les *rosiers* est un bon choix: qu'il glisse vers les *pétunias* ou pas, le robot pourra toujours atteindre la station de *recharge*. On voit aussi que certains lieux sont *sûrs*: ce sont ceux à partir desquels le robot possède une stratégie qui lui permet de garantir d'atteindre la station de recharge en un nombre fini de mouvements. Par exemple le *fond du jardin* est *sûr*, mais pas le *compost*. En particulier, la station de *recharge* est évidemment *sûre*.

Q4	**Les autres lieux sont-ils sûrs ?**		
	Sûr	Pas sûr	Lieu
Q4(a) [1 pt]	☒	☐	Rosiers
Q4(b) [1 pt]	☒	☐	Pétunias
Q4(c) [1 pt]	☐	☒	Niche

Solutions

Sur la carte suivante, quels sont les lieux *sûrs*, à partir desquels le robot peut garantir d'atteindre le lieu *objectif* ?



Q4(d) [4 pts]	**Donnez la liste des lieux sûrs.**
Solution: Objectif, 1, 4, 5	

Finalement, on propose l'algorithme ci-dessous pour déterminer, étant donné une telle carte, si un lieu est *sûr*. On vous demande de le compléter. L'algorithme suppose que les différents lieux sont numérotés de 0 à $N-1$, et que le lieu qu'on souhaite atteindre est le lieu $N-1$. Un lieu i doit donc être déclaré *sûr* si le robot peut garantir de visiter $N-1$ à partir de i. Un tableau `acces` de taille $N \times N$ contient des valeurs Booléennes pour indiquer quels sont les déplacements possibles du robot: chaque case `acces[i][j]` contient **`true`** si et seulement si le robot peut se déplacer du lieu i au lieu j (il y a un flèche allant de i à j sur le diagramme). Enfin, un tableau `controle` de taille N contient également des valeurs Booléennes pour indiquer quels lieux sont contrôlables: le lieu i est contrôlable si et seulement si `controle[i]` contient **`true`**. Voici, par exemple, les contenus des tableaux `acces` et `controle` pour la carte donnée à l'exercice précédent (en supposant que l'objectif a le numéro 6, et en représentant **`true`** par **T** et **`false`** par F).

`acces`:

	0	1	2	3	4	5	6
0	**T**	F	F	F	F	F	**T**
1	**T**	F	F	F	F	F	**T**
2	F	F	F	**T**	F	F	**T**
3	**T**	F	F	F	F	F	F
4	F	**T**	F	F	F	F	F
5	F	F	**T**	**T**	**T**	F	F
6	F	F	F	F	F	F	F

`controle`:

0	1	2	3	4	5	6
F	**T**	F	**T**	F	**T**	**T**

Q4(e) [2 pts]	**Complétez l'expression** `(e)`
Solution: $N-1$	

Q4(f) [4 pts]	**Complétez l'expression** `(f)`
Solution: `sur[j] = false and acces[i][j] = true` ou `not sur[j] and acces[i][j]`	

Solutions

Q4(g) [3 pts]	**Complétez l'instruction** (g)
Solution: `flag`←**`false`**	

Q4(h) [3 pts]	**Complétez l'expression** (h)
Solution: `sur[`i`]` ou `sur[`i`] = true`	

Solutions

```
Input : Les tableaux acces et controle comme décrits ci-dessus
Output : un tableau sur de taille N tel que sur[i] contient true
         si le lieu i est sûr, et false autrement.

Initialement, le tableau sur est initialisé à false dans toutes les cases.

sur[...] ← true // (e)
fini ← false

while (not fini) // fini est égal à false
{
  fini ← true
  for (i ← 0 to N − 1 step 1)
  {
    if (not sur[i])
    {
      if (controle[i]) // i est un lieu contrôlable.
      {
        for (j ← 0 to N − 1 step 1)
        {
          if (sur[j] and acces[i][j])
          {
            sur[i] ← true
          }
        }
      }
      else // i n'est pas un lieu contrôlable.
      {
        flag ← true

        for (j ← 0 to N − 1 step 1)
        {
          if ([...]) // (f)
          {
            [...] // (g)
          }
        }
        sur[i] ← flag
      }

      if ([...]) // (h)
      {
        fini ← false
      }

    }
  }
}
```

Solutions

Donnez vos réponses ici !

Q1(a)	**Une séquence de valeurs**	/1
Q1(b)	**Une valeur**	/1
Q1(c)	**Une expression de N**	/2
Q1(d)	**Une séquence de valeurs**	/2
Q1(e)	**Une valeur**	/3
Q1(f)	**Une séquence de valeurs**	/3
Q1(g)	**Une valeur**	/4
Q2(a)	**Une expression**	/4
Q2(b)	**Une valeur**	/3
Q2(c)	**Une expression**	/5
Q2(d)	**Une expression**	/4

Q3(a) **Barrez les cases où $t < tab[2][2]$ ne peut pas se trouver.** / 1

	0	1	2	3	4
0					
1					
2					
3					
4					

Solutions

Q3(b) **Barrez les cases où $t > tab[2][4]$ ne peut pas se trouver.** / 1

	0	1	2	3	4
0					
1					
2					
3					
4					

Q3(c) **Barrez les cases où t ne peut pas se trouver si** $tab[0][2] < t < tab[0][3]$ ***et*** $tab[2][0] < t < tab[3][0]$**.** / 2

	0	1	2	3	4
0					
1					
2					
3					
4					

Q3(d) **Barrez les cases où t ne peut pas se trouver si** $tab[0][2] < t < tab[0][4]$ ***et*** $tab[1][0] < t < tab[3][0]$**.** / 3

	0	1	2	3	4
0					
1					
2					
3					
4					

Q3(e) **Un tableau (3 lignes et 2 colonnes) et une valeur de t.** / 3

	0	1
0		
1		
2		

$t =$ .....

Q3(f) **Une expression** /4

.....

Solutions

Q3(g)	**Une expression**	/2
Q3(h)	**Une instruction**	/2
Q3(i)	**Une instruction**	/2
Q3(j)	**Une expression**	/4

	Sûr	Pas sûr	Lieu	
Q4(a) /1	☐	☐	Rosiers	/3
Q4(b) /1	☐	☐	Pétunias	
Q4(c) /1	☐	☐	Niche	

Q4(d)	**Une liste de lieux**	/4
Q4(e)	**Une expression**	/2
Q4(f)	**Une expression**	/4
Q4(g)	**Une instruction**	/3
Q4(h)	**Une expression**	/3

Solutions