be-OI 2014 **Demi-Finale** mercredi 27 novembre 2013	**Remplissez ce cadre en MAJUSCULES et LISIBLEMENT, svp** PRÉNOM : ..... NOM : ..... ÉCOLE :	100 **Réservé**

Olympiade belge d'Informatique (durée : 3h maximum)

Ce document est le questionnaire de la demi-finale de l'Olympiade belge d'Informatique 2014. Il comporte huit questions qui doivent être résolues en **3h 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. 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. 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.

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 suivante, 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.

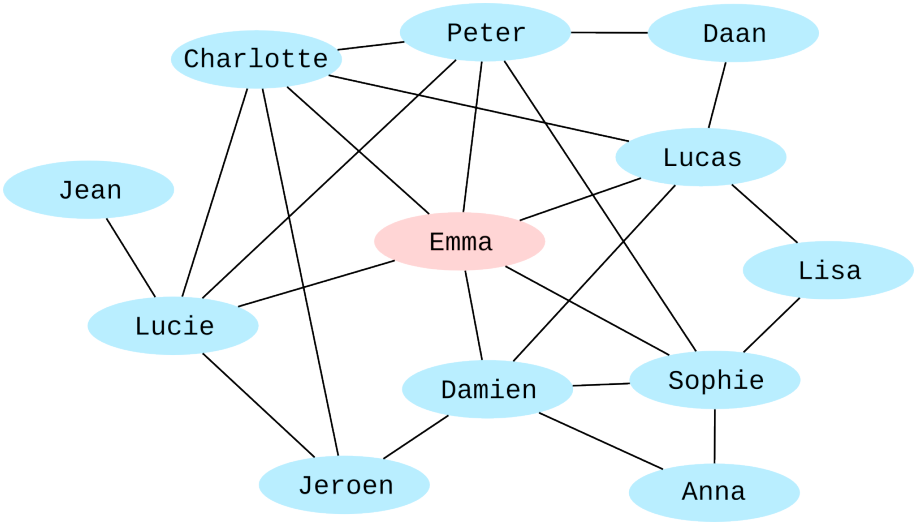
```
sum ← 0
for (i ← 0 to N − 1 step 1)
{
    sum ← sum + tab[i]
}
```

On peut également écrire une boucle à l'aide de l'instruction **`while`** qui 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
}
```

Question 1 – Les réseaux sociaux (5 min – 6 pts)

Emma est inscrite sur un réseau social où elle peut être *amie* avec d'autres personnes inscrites sur le réseau. Le diagramme ci-dessous montre les *amis* d'Emma ainsi que les *amis* des *amis* d'Emma.



Un trait entre deux personnes signifie qu'elles sont *amies* sur le réseau social. Par exemple, Sophie est *amie* avec Lisa et Lisa est *amie* avec Lucas, mais Sophie n'est pas *amie* avec Lucas.

Chaque personne peut partager ses photos uniquement avec ses amis. Lorsqu'une personne donne accès à une photo à un *ami*, celui-ci peut laisser un commentaire sur la photo. Lorsque une personne commente une photo, ses *amis* peuvent voir la photo et le commentaire associé, mais ne peuvent pas la commenter à leur tour, sauf s'ils étaient autorisés à la commenter au départ.

Emma a posté une photo de son animal de compagnie, un chihuahua prénommé Paris, mais elle ne veut pas que Damien puisse la voir. Elle est sûre que Damien ne deviendra pas *ami* avec d'autres de ses *amis* (c'est-à-dire que le diagramme ci-dessus ne changera pas).

Parmi les personnes suivantes, à qui Emma a-t-elle le droit de donner l'accès à la photo, tout en garantissant que Damien ne pourra jamais la voir ?

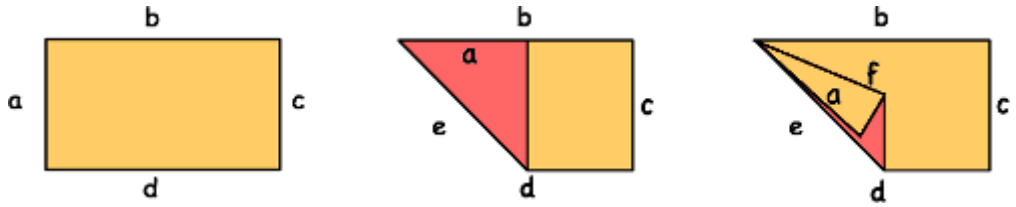
		Personne
Q1(a) [1 pt]	OUI / NON	Charlotte
Q1(b) [1 pt]	OUI / NON	Lucas
Q1(c) [1 pt]	OUI / NON	Damien
Q1(d) [1 pt]	OUI / NON	Lucie
Q1(e) [1 pt]	OUI / NON	Daan
Q1(f) [1 pt]	OUI / NON	Peter

Question 2 – Pliage (5 min – 6 pts)

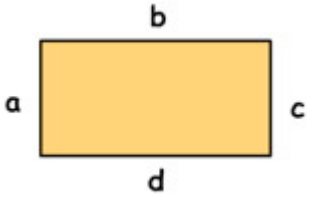
Votre camarade de classe Sakura est très douée pour construire des cocottes en papier. Vous trouvez ça très amusant et intéressant, alors vous désirez, vous aussi, vous mettre à cette discipline. Cependant, étant japonaise et fraîchement arrivée en Belgique, elle ne parle pas très bien français. Vous mettez donc au point un nouveau langage pour qu'elle puisse vous apprendre simplement comment copier ses oeuvres.

Ce nouveau langage consiste en une instruction $fold$ qui permet de plier un bord d'un morceau de papier vers un autre. L'instruction $z = fold(x, y)$ signifie que l'on doit plier le bord nommé x vers le bord nommé y. Le nouveau côté est nommé z.

Par exemple, voici ce qu'il se passe lorsque l'on effectue les instructions $e = fold(a, b)$ et $f = fold(a, e)$ successivement.

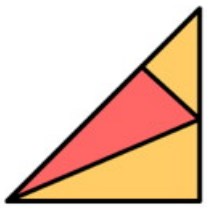


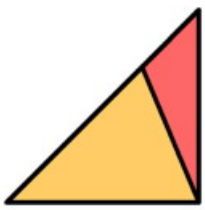
Si nous prenons la feuille de papier suivante avec le côté b deux fois plus long que le côté a.



Si nous appliquons les instructions: $e = fold(c, a)$, $f = fold(c, d)$ et $g = fold(a, f)$

Q2(a) [6 pts]	**Parmi les quatre figures ci-dessous, laquelle représente l'état final du papier ?**

1. 

2. 

3. 

4. 

Question 3 – Vive le général Alcazar, c'est un malabar ! (15 min – 12 pts)

La révolution faire rage au San Theodoros. L'infâme général Tapioca vient de renverser le bon général Alcazar, qui a heureusement pu se réfugier dans la jungle grâce à l'aide de ses fidèles Picaros.

Afin de donner plus de grandeur à son triomphe, le général Tapioca a décidé de faire disparaître toute mention du nom de son prédécesseur dans les textes de loi et autres archives du San Theodoros. Il désire même que chaque occurrence des mots « général Alcazar » soit remplacée par « grand et génial général Tapioca », le fourbe !

Les archives du San Theodoros ont été informatisées. Chaque texte est stocké dans un tableau, caractère par caractère (le premier caractère dans la première case du tableau, le second dans le seconde case, etc). Par exemple:

L	e		g	é	n	é	r	a	l		A	l	c	a	z	a	r		d	é	c	r	è	t	e		...

Avant de commencer à écrire le programme qui réalise les changements, les informaticiens du San Theodoros se posent une première question, qui va les aider à calculer les moyens de stockage nécessaires une fois les substitutions effectuées. Supposons que le texte original contient N caractères; que la chaîne de caractères à remplacer (dans ce cas « général Alcazar ») est de longueur L_{anc}; que la chaîne à lui substituer (« grand et génial général Tapioca ») est de longueur L_{nouv}; et que la chaîne à remplacer apparaît k fois dans le texte d'origine.

Q3(a) [3 pts]	**En général, quelle sera la taille (en nombre de caractères) du texte obtenu après remplacement ? Donnez une expression en fonction de N, k, L_{nouv} et L_{anc}.**

Afin de réaliser les changements nécessaires, les informaticiens san théodoriens proposent l'algorithme ci-dessous. Comme l'indiquent les commentaires, celui-ci consiste à recopier, caractère par caractère, le contenu du texte de loi originel (chaîne $texte$) dans la chaîne $destination$, tout en comparant chaque caractère avec ceux de la chaîne $ancien$. Chaque fois qu'une occurrence complète de la chaîne $ancien$ est détectée, celle-ci est remplacée par $nouveau$ dans la chaîne $destination$. Cet algorithme fait l'hypothèse que $destination$ est suffisamment grand pour stocker le résultat de la substitution.

Malheureusement, certaines expressions ont été effacées, suite à une cyber-attaque révolutionnaire des Picaros. Nous vous demandons de retrouver ces expressions.

Q3(b) [3 pts]	**Quelle est l'expression** `(b)` **?**

Q3(c) [2 pts]	**Quelle est l'expression** `(c)` **?**

Q3(d) [2 pts]	**Quelle est l'expression** `(d)` **?**

Q3(e) [2 pts]	**Quelle est l'expression** `(e)` **?**

```
Input : une chaîne de caractères texte de longueur N
        une chaîne de caractères ancien de longueur L_anc
        une chaîne de caractères nouveau de longueur L_nouv
Output : une chaîne de caractères destination, obtenue en remplaçant chaque occurrence
         de la chaîne ancien par la chaîne nouveau dans la chaîne texte

it ← 0      // Pour parcourir texte
id ← 0      // Pour parcourir destination
j ← 0       // Pour parcourir ancien

while (it < N)
{
  destination[id] ← texte[it]

  // A-t-on trouvé un nouveau caractère d'ancien dans texte ?
  if (...)          // (b)
  {
    j ← j + 1
  }
  else
  {
    j ← 0
    if (destination[id] = ancien[0])
    {
      j ← 1
    }
  }

  it ← it + 1
  id ← id + 1

  // On vérifie s'il est nécessaire de remplacer les L_anc derniers
  // caractères insérés dans destination par la chaîne nouveau.
  if (...)             // (c)
  {
    for (i ← 0 to L_nouv − 1 step 1)
    {
      destination[...] ← nouveau[i] // (d)
    }
    id ← ...     // (e)
    j ← 0
  }
}
```

Question 4 – Min-Max (30 min – 20 pts)

Les fonctions min et max sont deux fonctions couramment utilisées en programmation. Ces deux fonctions prennent deux arguments. Si a et b sont deux nombres entiers, alors $min(a, b)$ renvoie le plus petit nombre et $max(a, b)$ renvoie le plus grand.

On peut combiner les fonctions min et max, ainsi que les opérateurs $(+,-,\times,/)$ pour écrire de longues *expressions*. L'intérêt de min et max est alors qu'elles peuvent être utilisées pour opérer une sélection: elles sélectionnent le plus petit ou le plus grand nombre, qui peut être utilisé dans la suite, ce qui permet parfois de remplacer une instruction *if*.

Voici un exemple d'expression où min, max et les opérateurs $+$ et $-$ apparaissent (cette expression ne calcule rien de particulièrement intéressant):

```
max(a-c, min(b+a, c+a))
```

Nous vous demandons maintenant de faire la démarche inverse: nous souhaitons effectuer un certain traitement sur un ou plusieurs nombres, et nous vous demandons de composer une expression qui effectue ce traitement. Votre expression ne peut utiliser que les opérateurs de base $(+,-,\times,/)$ et les fonctions min et max.

a) Le plus grand nombre

```
Input :  trois nombres a, b en c
Output : le plus grand des trois nombres a, b et c

return [...] // (a)
```

Q4(a) [3 pts]	**Donnez une expression équivalente, la plus courte possible**

b) Valeur absolue

La valeur absolue d'un nombre est ce nombre sans son signe (la valeur absolue de 3 est 3, la valeur absolue de -5 est 5).

```
Input : un nombre a
Output : la valeur absolue de a

return [...] // (b)
```

Q4(b) [3 pts]	**Donnez une expression équivalente, la plus courte possible.**

c) Le nombre le plus proche

```
Input :  trois nombres a, b et c
Output : la distance  entre a et le nombre parmi b et c qui est le plus proche de a

return [...] // (c)
```

Q4(c) [5 pts]	**Donnez une expression équivalente, la plus courte possible.**

d) Tout ou rien

On peut introduire cet exercice comme suit: on souhaite déplacer un objet de poids b, mais on peut porter au maximum un poids a. Il est impossible de n'emporter qu'une partie de l'objet (il faut donc tout prendre, ou rien). Quel poids va-t-on devoir porter ?

Nous sommes ici à la recherche de la réponse correcte pour tous nombres *réels* a et b (en d'autres termes, a et b peuvent être des nombres à virgule)

```
Input : deux nombres réels positifs a ≠ b
Output : 0 si a < b, sinon b

return [...] // (d)
```

Q4(d) [5 pts]	**Donnez une expression équivalente, la plus courte possible**

e) Tout ou rien, partie 2

Si nous nous restreignons à des nombres a et b qui sont *entiers*, il existe une expression encore plus courte. N'oubliez pas que *division entière* fonctionne comme décrit dans les pages d'introduction au pseudo-code, au début de ce document.

Pourrez-vous trouver la plus courte expression qui est valable pour les nombres entiers ?

```
Input : deux nombres entiers positifs  a ≠ b
Output : 0 si a < b, sinon b

return [...] // (e)
```

Q4(e) [4 pts]	**Donnez une expression équivalente, la plus courte possible.**

Question 5 Guirlande électrique (15 min – 14 pts)

Pour rehausser la célébration du nouvel an chinois, les autorités taïwanaises ont décidé d'installer des guirlandes électriques un peu partout dans la ville. Ces guirlandes électriques sont *programmables*: on peut en allumer et éteindre les différentes ampoules selon un programme donné. Afin de pouvoir prévoir l'effet des différentes guirlandes, on vous demande d'en réaliser un simulateur. Celui-ci maintiendra un tableau G de taille N (le nombre d'ampoules), tel que $G[i] = 1$ si la i^{e} ampoule est allumée, et $G[i] = 0$ si elle est éteinte.

Le programme de la guirlande est stocké dans un tableau Act. Chaque case du tableau Act peut contenir une des instructions suivantes:

1. SL: effectue une *rotation circulaire vers la gauche*. Après l'exécution de cette instruction, le contenu de chaque case i (pour $1 \leq i \leq N-1$) est recopié dans la case $i-1$ (la case à gauche). Le contenu de la case 0 est recopié dans la case $N-1$.
2. SR: opération symétrique de SL qui effectue une *rotation circulaire vers la droite*.
3. ON: allume toutes les ampoules.
4. OFF: éteint toutes les ampoules.
5. INV: inverse toutes les ampoules (éteint les ampoules allumées, et allume les ampoules éteintes).

Par exemple, si nous avons les tableaux G et Act:

$G =$ | 0 | 0 | 1 | 0 | 1 | $Act =$ | SL | SR | SL | INV |

Le tableau G contiendra successivement, après l'exécution de chaque action de Act, les valeurs suivantes:

Après l'exécution de $Act[0]$: $G =$ | 0 | 1 | 0 | 1 | 0 |

Après l'exécution de $Act[1]$: $G =$ | 0 | 0 | 1 | 0 | 1 |

Après l'exécution de $Act[2]$: $G =$ | 0 | 1 | 0 | 1 | 0 |

Après l'exécution de $Act[3]$: $G =$ | 1 | 0 | 1 | 0 | 1 |

On vous demande de prévoir l'effet d'une série de programmes sur le tableau G suivant:

$G =$ | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |

Q5(a) [1 pt]	**Que contiendra G après l'exécution de $Act =$**	SL	INV	SR	**?**
Q5(b) [1 pt]	**Que contiendra G après l'exécution de $Act =$**	SL	SL	SL	**?**
Q5(c) [1 pt]	**Que contiendra G après l'exécution de $Act =$**	SL	INV	SR	SR	INV	SR	INV	ON	**?**
Q5(d) [1 pt]	**Que contiendra G après l'exécution de $Act =$**	INV	SR	SR	SR	INV	SR	OFF	INV	**?**

Afin d'avoir un simulateur plus réaliste, on ajoute la possibilité d'utiliser l'instruction Loop(T), où T est un entier strictement positif. Cette instruction doit nécessairement se trouver en fin du tableau Act, et T donne le nombre de répétitions des instructions qu'il faut effectuer. Ainsi, le programma SL SR Loop(3) produit le même effet que SL SR SL SR SL SR.

À nouveau, on vous demande de prévoir l'effet d'une série de programmes sur le tableau G donné ci-dessus.

Q5(e) [1 pt]	**Que contiendra G après l'exécution de** $Act =$ | SL | SR | Loop(30) | **?**

Q5(f) [1 pt]	**Que contiendra G après l'exécution de** $Act =$ | SL | INV | SR | INV | Loop(6205) | **?**

Q5(g) [1 pt]	**Que contiendra G après l'exécution de** $Act =$ | SL | SL | SL | Loop(36) | **?**

Q5(h) [1 pt]	**Que contiendra G après l'exécution de** $Act =$ | SL | INV | SR | Loop(245) | **?**

Nous souhaitons maintenant écrire le programme du simulateur. Nous vous demandons le programme qui simule l'effet de certaines actions sur le tableau G, à l'aide de la boucle **`for`** suivante, potentiellement encadrée de deux instructions:

```
[...] // (1)
for (i ← 0 to [...] step 1) // (2)
{
  G[i] ← [...] // (3)
}
[...] // (4)
```

Par exemple, pour simuler l'opération OFF, il faut remplacer (2) par $N - 1$, (3) par 0, mais laisser vides (1) et (4).

Q5(i) [3 pts]	**Par quoi faut-il remplacer les expressions `(1)`, `(2)`, `(3)` et `(4)` pour simuler l'opération INV (indiquer « rien » si nécessaire. Des variables supplémentaires peuvent être utilisées) ?**

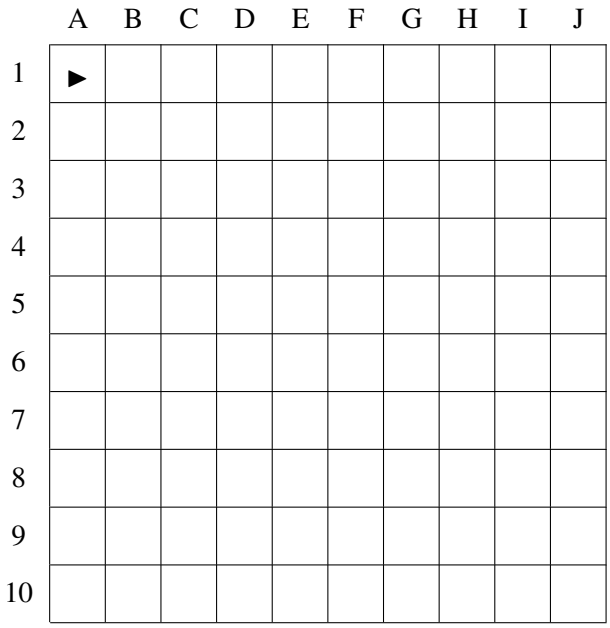
Q5(j) [3 pts]	**Par quoi faut-il remplacer les expressions `(1)`, `(2)`, `(3)` et `(4)` pour simuler l'opération SL (indiquer « rien » si nécessaire. Des variables supplémentaires peuvent être utilisées) ?**

Question 6 – Suivez le robot (10 min – 8 pts)

Vous trouverez ci-dessous une série d'espaces en deux dimensions dans lesquels un robot est représenté. Le robot a une position bien déterminée dans chacun des espaces, ainsi qu'une orientation (nord = ▲, est = ►, sud = ▼, ouest = ◄).

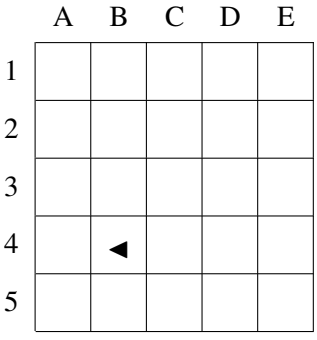
Le robot est dirigé au moyen d'un programme informatique. Chaque fois que le programme fait appel à la procédure `Avance`, le robot se déplace d'une case en suivant sa direction courante. Chaque fois que le programme fait appel à la procédure `Tourne`, le robot tourne de 90° dans le sens des aiguilles d'une montre, tout en restant sur la même case. Le robot ne peut pas se déplacer en dehors de l'espace: chaque fois qu'il entre en collision avec un des bords, il reste tout simplement sur la même case (l'opération `Avance` n'a donc pas d'effet).

Considérons les trois espaces ci-dessous dans lesquels la position et l'orientation de départ du robot sont à chaque fois indiquées. À côté de chacun d'eux se trouve un programme. On demande de prédire quelle sera la **position** du robot **à l'issue** de l'exécution du programme. Les positions sont indiquées au moyen d'une lettre et d'un chiffre (par exemple, la position de départ du robot dans le premier des trois espaces est **A1**).



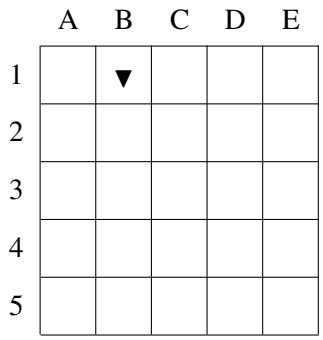

```
x ← 1
while (x < 9)
{
  Avance()
  Tourne()
  Avance()
  for (y ← 0 to 2 step 1)
  {
    Tourne()
  }
  x ← x + 1
}
```

Q6(a) [2 pts]	**Quelle est la position du robot après l'éxécution du programme ci-dessus ?**




```
x ← 0
while (x < 66)
{
  Tourne()
  Avance()
  Avance()
  x ← x + 1
}
```

Q6(b) [3 pts]	**Quelle est la position du robot après l'éxécution du programme ci-dessus ?**




```
x ← 1
y ← 1
z ← 0

while (x + y < 10)
{
  x ← x + 1
  y ← y + x
  Tourne()
  z ← y
  while (z > 0)
  {
    z ← z − 1
    Avance()
  }
}
```

Q6(c) [3 pts]	**Quelle est la position du robot après l'exécution du programme ci-dessus ?**

Question 7 – L'intrus (20 min – 20 pts)

A l'annonce de l'organisation de l'Olympiade Internationale d'Informatique à Taiwan, la ville de Taipei a décidé d'ouvrir une nouvelle ligne de production de nouilles à l'anguille, plat traditionnel, pour nourrir les nombreux participants. Cette ligne de production doit produire des nouilles qui ont toutes une taille différente les unes des autres. Les longueurs sont entières et mesurées en mm, en commençant à 0 (une nouille de la taille de quelques microns). Malheureusement, suite à une panne, deux nouilles de même taille sont sorties de la ligne. Il vous est demandé de retrouver au plus vite la longueur de nouille en défaut.

On vous demande pour cela d'écrire un algorithme. Celui-ci prendra en entrée un tableau tab de taille n contenant un ensemble de nombres de 0 à $n-2$ (compris) dans un ordre quelconque, mais dont un seul se répète. Votre algorithme devra retourner ce nombre. Par exemple, si $tab = [2, 4, 0, 4, 1, 3]$ et $n = 6$, votre algorithme doit retourner 4.

Les responsables de la ville, affolés, ont déjà fait l'ébauche d'un premier algorithme qui consiste à parcourir chaque case du tableau, et à vérifier si le nombre contenu dans la case se répète dans une case qui suit. Aidez les à le terminer.

```
Input: un tableau tab de taille n contenant des entiers de 0 à n−2 (compris) dont
       un seul se répète

Output: à la fin de l'exécution, sol contient la valeur qui apparaît deux fois
        dans tab.

  sol ← −1

  for (i ← 0 to n − 1 step 1)
  {
    for (j ←[...] to n − 1 step 1)    // (a)
    {
      if (tab[i] = tab[j])
      {
        sol ← tab[i]
      }
    }
  }
```

Q7(a) [4 pts]	**Quel doit être l'expression** `(a)` **?**

Très rapidement, vous remarquez que, vu qu'il y a plus de 100 000 tailles de nouilles différentes, cet algorithme évaluera plus de 10 milliards de fois le "`if`" et ne se terminera donc probablement pas à temps pour le début de l'olympiade. L'un des membres de l'équipe propose alors d'optimiser l'algorithme précédent en utilisant un tableau intermédiaire. Complétez l'algorithme suivant.

```
Input: un tableau tab de taille n contenant des entiers de 0 à n − 2 (compris) dont
       un seul se répète.

Output: à la fin de l'exécution, sol contient la valeur qui apparaît deux fois
        dans tab.

tab2 ← un tableau de taille n initialisé avec la valeur 0 à chaque indice.
sol ← −1

for (i ← 0 to n − 1 step 1)
{
  if ([...])                    // (b)
  {
    [...]                       // (c)
  }
  else
  {
    sol ← tab[i]
  }
}
```

Q7(b) [4 pts]	**Quelle doit être l'expression** `(b)` **?**

Q7(c) [4 pts]	**Quelle doit être l'instruction** `(c)` **– il s'agit d'une affectation ?**

Malheureusement, un responsable de l'usine vous fait remarquer que cet algorithme devra tourner sur une machine de l'usine qui ne possède que très peu de mémoire (RAM) et qu'il serait donc bon de pouvoir se passer de ce tableau intermédiaire de taille n. Il existe une solution n'utilisant qu'un nombre (de taille arbitraire) comme variable tout en évitant les deux boucles "`for`" imbriquées, saurez-vous la trouver ?

```
Input: un tableau tab de taille n contenant des entiers de 0 à n − 2 (compris) dont
       un seul se répète

Output: à la fin de l'exécution, sol contient la valeur qui apparaît deux fois
        dans tab.

sol ← −1
x ← 0

for (i ← 0 to n − 1 step 1)
{
  x ← [...]                     // (d)
}
sol ← [...]                     // (e)
```

Q7(d) [4 pts]	**Quelle doit être l'expression** `(d)` **?**

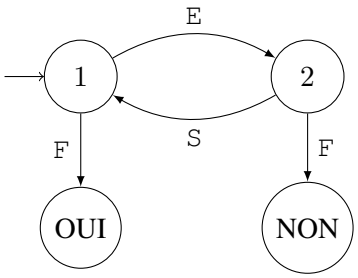
Q7(e) [4 pts]	**Quelle doit être l'expression** `(e)` **?**

Question 8 – La machine d'Alan (25 min – 22 pts)

Pour éviter la fraude aux examens, beaucoup d'universités imposent qu'aucun étudiant ne peut entrer dans la salle si un autre étudiant en est déjà sorti. Votre ami Alan vous propose de mettre au point un système permettant de détecter ce type de fraude, de manière infaillible. Le plan élaboré par Alan est le suivant: vous vous placez à l'entrée de la salle d'examen, et vous notez sur un ruban la séquence des entrées et sorties: un `E` quand un étudiant entre, un `S` quand un étudiant sort et un `F` unique à la fin de vos observations, tout cela dans l'ordre observé. Par exemple, si un étudiant entre, puis un étudiant sort, puis un autre étudiant entre, vous noterez sur le début du ruban:

E	S	E	F	...

Ensuite, Alan introduit le ruban dans une machine de son invention. Celle-ci va lire le ruban, et va indiquer, si oui ou non la séquence d'entrée est valide. Pour répondre à cette question, Alan vous explique que l'on peut programmer sa machine, et que les programmes peuvent être représentés par des diagrammes comme ceci:



Sur ce diagramme, chaque cercle est appelé un *état*. On distingue trois états particuliers: les états OUI et NON qui permettent à la machine d'indiquer si oui ou non la séquence sur le ruban est valide, et l'état initial qui est l'état dans lequel la machine commence son exécution. Sur cet exemple, il s'agit de l'état 1: il est indiqué par la petite flèche à gauche, qui ne provient d'aucun autre état. Les flèches sont appelées *transitions*, et permettent à la machine de lire un caractère du ruban et de changer d'état. La machine ne peut se trouver, à tout moment, que dans un seul état.

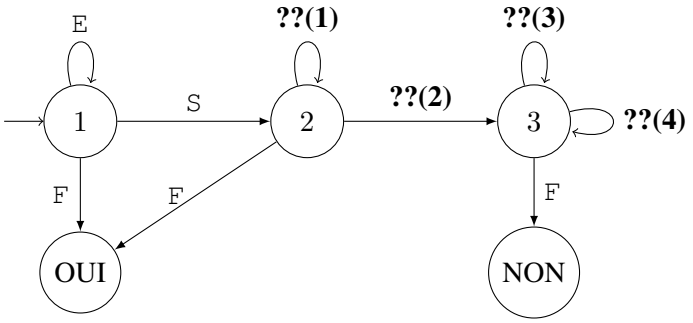
Pour lire les caractères, la machine possède une tête de lecture qui se trouve sur la première case du ruban au début de l'exécution (la plus à gauche). Ensuite, quand la machine est dans un état donné, elle peut exécuter une transition qui part de cet état, à condition que la lettre indiquée par la transition soit celle qui se trouve sous la tête de lecture. L'effet est alors de déplacer la tête de lecture d'une position vers la droite (c'est-à-dire de passer au caractère suivant), et de changer l'état courant, qui devient celui à l'extrémité de la flèche.

Par exemple, sur le contenu de ruban donné ci-dessus, la machine est initialement dans l'état 1, avec sa tête de lecture sur le premier `E`. Elle exécute alors la transition de l'état 1 vers l'état 2, et la tête de lecture se trouve sur le premier `S`. Puis, la machine revient dans l'état 1 avec la tête sur le second `E`, et ainsi de suite. L'exécution se termine dans l'état NON.

Dans quel état la machine termine-t-elle son exécution quand on lui fournit les rubans suivants ?

		Ruban
Q8(a) [2 pts]	OUI / NON	E S E S F ...
Q8(b) [2 pts]	OUI / NON	F ...

Pour contrôler les entrées et sorties, Alan propose le programme suivant pour sa machine. Malheureusement, certaines lettres sont effacées (remplacées par des points d'interrogation). Alan vous indique qu'avec ce programme, sa machine atteindra l'état NON *uniquement quand un étudiant rentre alors qu'un autre est déjà sorti*.

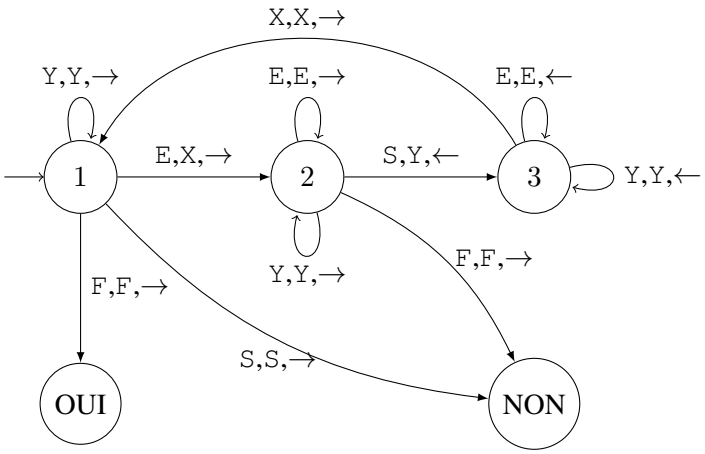


Aidez Alan en donnant les quatre lettres qui doivent remplacer **??(1)**, **??(2)**, **??(3)** et **??(4)** (dans cet ordre)

Q8(c) [4 pts]	**Aidez Alan en donnant les quatre lettres qui doivent remplacer ??(1), ??(2), ??(3) et ??(4) (dans cet ordre).**

Finalement, Alan vous propose un ultime raffinement de sa machine. Celle-ci peut maintenant, à chaque transition, remplacer le caractère lu sur le ruban (la tête permet donc maintenant d'écrire), et décider dans quel sens le ruban se déplace (on peut donc maintenant déplacer le ruban vers la gauche pour revenir sur des parties du ruban visitées précédement). Les étiquettes des transitions sont maintenant de la forme $\texttt{A}, \texttt{B}, d$, indiquant que la machine doit lire `A` à l'emplacement de la tête, remplacer le `A` lu par un `B` (avec, éventuellement, $\texttt{A} = \texttt{B}$), et déplace la tête dans la direction d, qui peut être soit $\leftarrow$ (pour la gauche), soit $\rightarrow$ (pour la droite).

Pour cette nouvelle version de la machine, Alan vous propose le programme suivant:



Considérons le contenu initial du ruban suivant:

E	S	F	...

Q8(d) [2 pts]	**Dans quel état la machine terminera-t-elle son exécution ?**

Q8(e) [2 pts]	**Quel sera le contenu du ruban quand la machine terminera son exécution ?**

Même question pour le contenu initial suivant:

E	E	S	F	...

Q8(f) [2 pts]	**Dans quel état la machine terminera-t-elle son exécution ?**

Q8(g) [2 pts]	**Quel sera le contenu du ruban quand la machine terminera son exécution ?**

Parmi ces conditions, quelles sont celles qui garantissent que la machine acceptera comme valide le contenu initial du ruban, en supposant que celui-ci est composé d'une séquence de `E`, suivie d'une séquence de `S`, suivie d'un unique `F` ?

		Condition
Q8(h) [2 pts]	Garantit / Ne garantit pas	Le nombre de `E` est égal au nombre de `S`
Q8(i) [2 pts]	Garantit / Ne garantit pas	Le nombre de `E` est supérieur ou égal au nombre de `S`
Q8(j) [2 pts]	Garantit / Ne garantit pas	Le nombre de `E` est strictement supérieur au nombre de `S`

Donnez vos réponses ici !

	OUI	NON	Personne	/6
Q1(a) /1	☐	☐	Charlotte	
Q1(b) /1	☐	☐	Lucas	
Q1(c) /1	☐	☐	Damien	
Q1(d) /1	☐	☐	Lucie	
Q1(e) /1	☐	☐	Daan	
Q1(f) /1	☐	☐	Peter	

Q2(a)	**Un numéro de proposition entre 1 et 4**	/6
Q3(a)	**Une expression**	/3
Q3(b)	**Une expression**	/3
Q3(c)	**Une expression**	/2
Q3(d)	**Une expression**	/2
Q3(e)	**Une expression**	/2
Q4(a)	**Une expression**	/3
Q4(b)	**Une expression**	/3
Q4(c)	**Une expression**	/5
Q4(d)	**Une expression**	/5
Q4(e)	**Une expression**	/4

Q5(a) **Le contenu de G**	/1
Q5(b) **Le contenu de G**	/1
Q5(c) **Le contenu de G**	/1
Q5(d) **Le contenu de G**	/1
Q5(e) **Le contenu de G**	/1
Q5(f) **Le contenu de G**	/1
Q5(g) **Le contenu de G**	/1
Q5(h) **Le contenu de G**	/1
Q5(i) **Les expressions (1), (2) et (3) et (4)**	/3
Q5(j) **Les expressions (1), (2) et (3) et (4)**	/3
Q6(a) **Une position**	/2
Q6(b) **Une position**	/3
Q6(c) **Une position**	/3
Q7(a) **Une expression**	/4
Q7(b) **Une expression**	/4

Q7(c) **Une instruction (affectation)**	/4
. . . . .	

Q7(d) **Une expression**	/4
. . . . .	

Q7(e) **Une expression**	/4
. . . . .	

	OUI	NON	Ruban	/4
Q8(a) /2	☐	☐	E S E S F …	
Q8(b) /2	☐	☐	F …	

Q8(c) **Quatre lettres**	/4
. . . . .	

Q8(d) **Un état**	/2
. . . . .	

Q8(e) **Un contenu de ruban**	/2
. . . . .	

Q8(f) **Un état**	/2
. . . . .	

Q8(g) **Un contenu de ruban**	/2
. . . . .	

	Garantit	Ne garantit pas	Condition	/6
Q8(h) /2	☐	☐	Le nombre de E est égal au nombre de S	
Q8(i) /2	☐	☐	Le nombre de E est supérieur ou égal au nombre de S	
Q8(j) /2	☐	☐	Le nombre de E est strictement supérieur au nombre de S	