OI 2010 **Demi-finale** 24 Mars 2010	**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 pour la catégorie secondaire. Il comporte six questions. Les deux premières sont des QCM et les quatre dernières sont des questions ouvertes. 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.

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**. Pas de réponses laissées au crayon. Si vous désirez des feuilles de brouillon, demandez-en auprès d'un surveillant.

4. Pour les questions de type QCM, vous ne devez choisir qu'**une seule réponse**. Une réponse correcte rapporte 1 point, une abstention vaut 0 point et une mauvaise réponse sera sanctionnée par $-0{,}5$ point.

5. Vous pouvez répondre aux questions ouvertes en pseudo-code, à l'aide d'organigramme ou avec un des langages de programmation autorisés : Java, C, C++, C#, Pascal, Python, Ruby, PHP et Visual Basic. Dans ce cas, vous ne pouvez utiliser **aucune librairie**, vous ne pouvez utiliser que les constructions standards du langage. Les erreurs de syntaxe ne sont pas prises en compte pour l'évaluation. En cas de doute, faites appel à un surveillant.

6. Vous pouvez définir des sous-fonctions et utiliser de la récursion, sauf mention explicite contraire.

7. La notation **`for`** $(i \leftarrow a$ **`to`** b **`step`** $k)$ indique une boucle qui se répète tant que $i \leq b$ pour i partant de a par pas de k.

8. Vous avez **exactement trois heures** pour répondre à toutes les questions.

Bonne chance !

Question 1 – Pliage (10 min)

Soit une feuille de papier de x mm d'épaisseur. Si je plie cette feuille en deux, j'obtiens une épaisseur de $2x$ mm. Si j'effectue un nouveau pli, l'épaisseur sera $4x$ mm … et ainsi de suite. L'algorithme suivant prend deux paramètres : la valeur de x et une épaisseur souhaitée g. Il calcule le nombre minimum de plis qu'il faut effectuer pour obtenir au moins g mm d'épaisseur.

```
Input  : x > 0, épaisseur du papier en mm
         g > 0, épaisseur minimale souhaitée en mm
Output : nombre de plis à faire, au minimum, pour avoir au moins g mm d'épaisseur

n ← 0
while (x < g)
{
    [...]
}
return n
```

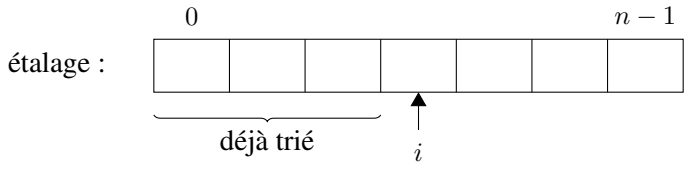
Quelles sont les instructions manquantes à l'algorithme pour que ce dernier calcule ce qu'il faut ?

☐	$x \leftarrow n^2$ $n \leftarrow n + 1$
☐	$x \leftarrow 2 \cdot x$ $n \leftarrow n + 1$
☐	$n \leftarrow n + 1$ $x \leftarrow 2 \cdot n$
☐	$x \leftarrow x + n$ $n \leftarrow n + 1$

Question 2 – Un peu de rangement (15 min)

Vous êtes engagé comme jobiste dans un supermarché et êtes responsable du département qui vend des chaussures. Votre patron vous demande de classer ces chaussures en fonction de leurs pointures, le tout de manière croissante. Vous avez à votre disposition n paires de chaussures, chacune se trouvant déjà sur l'étalage, à une certaine position. On numérote les n positions de l'étalage de 0 à $n-1$. L'algorithme suivant, où $etalage[pos]$ indique la pointure de la paire de chaussure à la position pos de l'étalage, permet de trier les paires de chaussures de manière croissante.

L'algorithme va parcourir l'étalage de gauche à droite. À tout moment, vous prenez la paire de chaussures en position i. La première boucle (**while**) permet de retrouver l'endroit, à votre gauche, où la paire de chaussures que vous avez en mains doit aller (pos). La seconde boucle (**for**) va décaler toutes les chaussures afin de faire de la place en pos pour y mettre la paire de chaussures que vous avez dans vos mains.




```
Input  : etalage, les pointures des paires de chaussures
         n > 0, nombre de paires de chaussures
Output : etalage a été trié de manière croissante

for (i ← 1 to n − 1 step 1)
{
    pos ← 0
    while (etalage[pos] < etalage[i])
    {
        pos ← pos + 1
    }
    en_mains ← etalage[i]

    for ([...])
    {
        etalage[j + 1] = etalage[j]
    }
    etalage[pos] = en_mains
}
```

Quels sont les valeurs de j qui doivent être parcourues lors de la seconde phase ?

☐	$j \leftarrow i$ **to** pos **step** -1
☐	$j \leftarrow i+1$ **to** pos **step** -1
☐	$j \leftarrow i-1$ **to** pos **step** -1
☐	$j \leftarrow i$ **to** 1 **step** -1

Question 3 – Collier de perles (20 min)

Votre sœur joue avec des perles, des blanches et des noires, et elle souhaite constituer des colliers. Dans un souci d'esthétique, elle désire alterner les couleurs, tout en préservant une certaine périodicité. Elle va donc placer x perles blanches, suivies de y perles noires et elle répètera ce motif n fois.

Par exemple, supposons qu'elle possède $x = 12$ perles blanches et $y = 6$ perles noires. Elle a plusieurs possibilités :

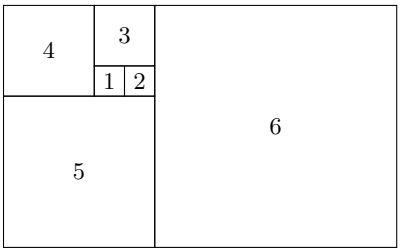
- une répétition (12 noires, 6 blanches) ○○○○○○○○○○○○●●●●●●
- deux répétitions (6 noires, 3 blanches) ○○○○○○●●●○○○○○○●●●
- trois répétitions (4 noires, 2 blanches) ○○○○●●○○○○●●○○○○●●
- six répétitions (2 noires, 1 blanche) ○○●○○●○○●○○●○○●○○●

Écrivez un algorithme qui calcule, étant donné deux entiers strictement positifs x et y, représentant respectivement les nombres de perles blanches et noires, le nombre maximum de répétitions que votre sœur va pouvoir faire avec ces perles.

Q3

Question 4 – Rectangles de carrés (20 min)

Votre petit frère a reçu des pièces carrées en bois, dont les longueurs des cotés vous sont inconnues. Il va construire un motif particulier grâce à ces pièces. Il commence par retrouver la plus petite pièce et en place deux côte-à-côte afin de former un rectangle de largeur 1 et de longueur 2 (carrés 1 et 2). Il va ensuite trouver la pièce carrée dont la longueur du coté est égale à la longueur du rectangle obtenu (carré 3), il place cette pièce le long de cette longueur pour former un nouveau rectangle. Il continue à répéter ce processus plusieurs fois, les longueurs des rectangles obtenus valant respectivement $3, 5, 8, 13, \ldots$



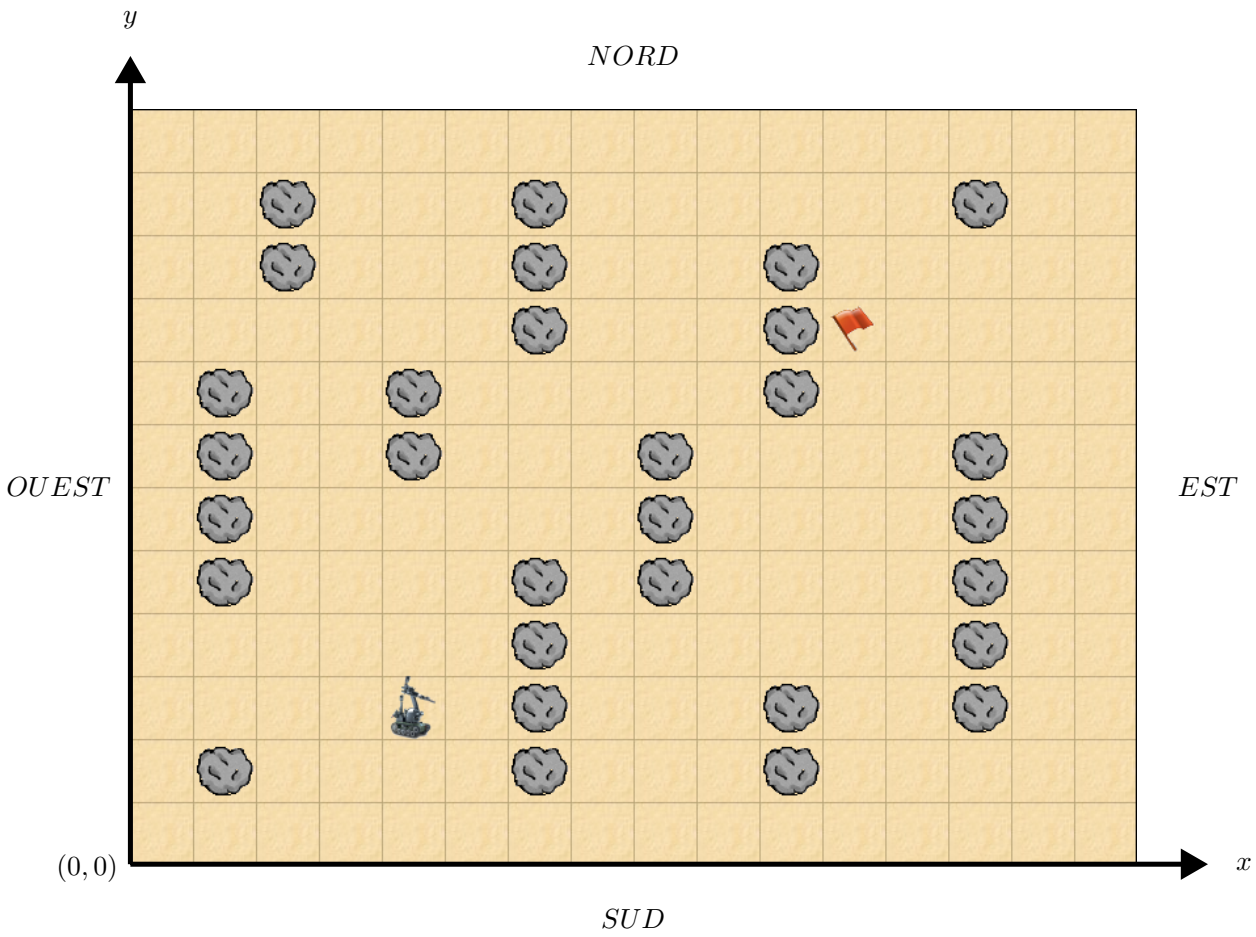
On peut définir une fonction mathématique L qui donne la longueur du rectangle obtenu après avoir placé n pièces selon la procédure utilisée par votre frère. Écrivez un algorithme qui permet d'imprimer les valeurs des longueurs des rectangles pour les entiers $i = 1, 2, 3, \cdots$ jusqu'à un entier n donné, tel que $n \geq 1$.

$$L(n) = \begin{cases} 0 & \text{, si } n = 0 \\ 1 & \text{, si } n = 1 \\ L(n-1) + L(n-2) & \text{, si } n > 1 \end{cases}$$

Q4 **! Vous ne pouvez pas utiliser de sous-fonction, ni de récursion**

Question 5 – Par où dois-je aller ? (50 min)

Vous êtes engagé par l'ESA (Agence Spatiale Européenne) pour programmer le futur robot d'exploration lunaire. Vous devez écrire un algorithme qui va permettre de déplacer le robot afin d'atteindre une certaine cible (drapeau). Le terrain est découpé en cases et le robot est initialement orienté vers l'est. Le terrain est jonché d'obstacles infranchissables, ce sont toujours des lignes verticales de rochers. Il est toujours possible de faire tout le tour de la ligne, en longeant les rochers la constituant. On suppose également que le robot est toujours situé à l'ouest de la cible. Le terrain ci-dessous est un exemple possible, votre robot doit fonctionner dans toutes les situations qui respectent les conditions sus-mentionnées, la taille du terrain peut varier, le nombre et la position des obstacles également.



Votre robot est capable d'effectuer un certain nombre limité d'opérations, et vous ne pouvez utiliser que ces dernières pour votre algorithme. Certaines opérations effectuent une action, et d'autres permettent d'interroger le robot ou d'obtenir des informations sur l'environnement.

`move`	Fait avancer le robot d'une case dans sa direction courante, s'il n'est pas contre un rocher ou contre le bord du terrain. Sinon, ne fait rien.
`turnLeft`	Fait faire une rotation de 90 degrés vers la gauche au robot.
`turnRight`	Fait faire une rotation de 90 degrés vers la droite au robot.
`canMove`	Teste si le robot peut bouger (c'est-à-dire n'est pas contre un rocher ou contre le bord).
`getX, getY`	Obtient la coordonnée x ou y du robot.
`getDirection`	Obtient la direction du robot (`NORD`, `SUD`, `OUEST` ou `EST`).
`targetX, targetY`	Obtient la coordonnée x ou y de la cible.
`isOnTarget`	Teste si le robot se trouve sur la cible.

Voyons un exemple d'algorithme. On veut que, tant que le robot n'est pas sur la cible, il essaye de bouger tant que c'est possible. Et une fois que ce n'est plus possible, il fait une rotation de 90 degrés vers la gauche.

```
while (not isOnTarget)
{
    while (canMove)
    {
        move
    }

    turnLeft
}
```

Proposez un algorithme qui permet au robot d'atteindre la cible, en minimisant les déplacements du robot (faire le moins de `move` possible). Plusieurs solutions sont possibles, proposez celle qui vous semble la meilleure. Vous pouvez faire plusieurs propositions, en donnant d'abord la plus bête qui prend plein de temps et ensuite une plus maligne.

Q5 **! Vous ne pouvez utiliser que les primitives données**

Q5 (suite)

Question 6 – Super Programmer Star (50 min)

Vous avez été engagé par une chaine de télévision privée qui va lancer un nouveau concours fou. Les participants vont devoir vivre sur une ile déserte et programmer, en équipe, un jeu vidéo. Ils vont devoir s'entendre sur les aspects de programmation pure, sur le design, les scénarios et la psychologie des personnages. Toutes les semaines, plusieurs équipes sont éliminées, en fonction de votes exprimés par le public.

Chaque équipe possède un numéro, qui est un naturel $n > 0$. Chaque téléspectateur va pouvoir voter pour plusieurs équipes, et la liste $votes$ contient le nombre de votes enregistrés pour chaque équipe. Toutes les équipes, qui n'ont pas reçu un nombre de votes qui est supérieur ou égal au nombre moyen de votes reçus pour toutes les équipes, sont éliminées.

Vous devez écrire un algorithme qui prend en entrée un nombre d'équipe N et une liste $votes$. Il doit renvoyer une liste de même longueur que $votes$, dont les premières entrées contiennent les numéros des équipes éliminées et toutes les autres entrées contiennent -1. On désigne par $votes[i]$ le nombre de votes obtenus par l'équipe numéro i.

Par exemple, supposons qu'il y a 10 équipes, dont les numéros sont donc $1, 2, \cdots, 10$. La production a reçu 15 votes repris dans la liste $votes = [3, 7, 3, 3, 9, 9, 5, 1, 2, 5, 7, 1, 9, 5, 2]$. On peut calculer le nombre de votes reçus pour chaque équipe :

Équipe	1	2	3	4	5	6	7	8	9	10
Votes	2	2	3	0	3	0	2	0	3	0

Le nombre moyen de votes est obtenu en divisant le nombre total de votes (15) par le nombre d'équipes ayant au moins reçu un vote (6). On ignore donc les équipes n'ayant reçu aucun vote. Cela donne donc :

$$m = \frac{15}{6} = 2{,}5$$

Toutes les équipes n'ayant pas obtenu au minimum $m = 2{,}5$ votes doivent être éliminées. La liste qui doit être calculée par l'algorithme est donc :

$result = [1, 2, 4, 6, 7, 8, 10, -1, -1, -1, -1, -1, -1, -1, -1]$

Q6

Q6 (suite)