OI 2010 **Finale** 12 Mai 2010	**Remplissez ce cadre en MAJUSCULES et LISIBLEMENT, svp** PRÉNOM : ..... NOM : ..... ÉCOLE :	**Réservé**

Olympiades belges d'Informatique (durée : 1h15 maximum)

Ce document est le questionnaire de **la partie papier** de la finale des Olympiades belges d'Informatique pour **la catégorie secondaire**. Il comporte huit questions qui doivent être résolues en **1h15 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.

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. Pour les questions de type QCM, vous ne devez choisir qu'**une seule réponse**. Cochez la case de votre choix. Si vous vous êtes trompé, noircissez la case erronée pour annuler votre choix. Une réponse correcte rapporte 1 point, une abstention vaut 0 point et une mauvaise réponse est sanctionnée par $-0{,}5$ point.

5. Vous **devez** répondre aux questions ouvertes en **pseudo-code**. 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. 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 ou les organisateurs. Toute question portant sur la compréhension de la question ou liée à des problèmes techniques ne peut être posée qu'aux organisateurs. Toute question logistique peut être posée aux surveillants.

8. Il est strictement **interdit de manger ou boire** durant l'épreuve. Les participants **ne peuvent en aucun cas quitter leur place** pendant l'épreuve, par exemple pour aller aux toilettes ou pour fumer une cigarette.

9. Vous avez **exactement une heure et quart** pour répondre à toutes les questions.

Bonne chance !

Questionnaire finale papier secondaire

Question 0 – Amuse-bouche (10 min)

(a) Soit la fonction `notdivisible (x, n)` qui renvoie **`true`** si `x` n'est pas divisible par `n` et **`false`** sinon. Quelle est l'expression qui permet de vérifier que `x` n'est pas divisible par 5, ni par 3, ni par 7 ?

☐	**not** (notdivisible (x, 5) **or** notdivisible (x, 3) **or** notdivisible (x, 7))
☐	notdivisible (x, 5) **or** notdivisible (x, 3) **or** notdivisible (x, 7)
☐	notdivisible (x, 5) **and** notdivisible (x, 3) **and** notdivisible (x, 7)
☐	**not** notdivisible (x, 5) **or not** notdivisible (x, 3) **or** notdivisible (x, 7)

(b) Laquelle des expressions suivantes est équivalente à : **not** $(a > 4)$ **and** $3a = b$

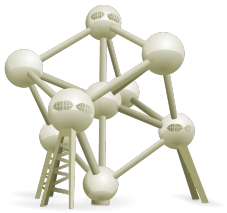
☐	**not** $(a \leq 4$ **or** $3a \neq b)$
☐	**not** $(a > 4$ **or** $3a \neq b)$
☐	**not** $(a \leq 4$ **and** $3a = b)$
☐	**not** $(a > 4$ **or** $3a \neq b)$

(c) Quelle est la valeur de n après exécution de cet algorithme ?

```
n ← 0
a ← 3
while (a ≤ 4)
{
    n ← n + a
}
```

☐	0
☐	7
☐	11
☐	aucune des solutions ci-dessus

Question 1 – Colis souvenirs (10 min)



Afin de promouvoir la Belgique à l'étranger, vous avez à votre disposition deux types d'objets. Vous possédez a miniatures de l'atomium et b aimants bière. Vous désirez faire des colis, chacun de ceux-ci comportant un certain nombre d'atomiums et de bières. Afin de toucher un maximum de personnes, vous avez mis au point un algorithme qui calcule le nombre maximal de colis que vous allez pouvoir faire, sachant que tous les colis doivent comporter le même nombre X d'atomiums et le même nombre Y de bières. De plus, il ne doit pas rester d'objets n'ayant pas été placés dans un colis.

Voyons quelques exemples. Si on a un stock de 13 atomiums et 7 aimants bière, on va pouvoir faire 1 seul colis contenant 13 atomiums et 7 aimants bières. Par contre, avec 12 atomium et 2 aimants bières, on pourra faire 6 colis contenant chacun 6 atomiums et 1 aimants bière.

```
Input  : a et b, deux naturels non-nuls tels que a ≥ b
Output : nombre maximal de colis que vous allez pouvoir faire

n ← a
m ← b
r ← a  mod b

while (r ≠ 0)
{
    n ← m
    m ← r
    [...]
}

return m
```

Quelle instruction manque-t-il à cet algorithme afin qu'il produise le résultat attendu ?

Q1 **(une instruction)**

. . . . . .

Question 2 – Combien de combinaisons possibles ? (5 min)

Si j'ai n éléments et que je souhaite en choisir p parmi ces n, combien de choix différents vais-je pouvoir faire ? On peut par exemple être intéressé par le nombre de mains différentes qu'il est possible de faire au poker. Ce nombre est appelé nombre binomial et est noté $\binom{n}{p}$ ou C_n^p. On peut le calculer grâce au triangle de Pascal. La première colonne de ce triangle, ainsi que son hypothénuse, contiennent des 1. Ensuite, la valeur de chaque case du triangle est égale à la somme de la valeur dans la case se situant juste au-dessus et de celle se situant à gauche de celle se trouvant au-dessus.

Voici les cinq premières lignes du triangle de Pascal :

1				
1	1			
1	2	1		
1	3	3	1	
1	4	6	4	1

L'algorithme suivant permet de calculer les n premières lignes du triangle de Pascal. Le résultat est calculé dans une matrice carrée, et toutes les entrées ne faisant pas partie du triangle valent 0.

```
Input  : n, un entier positif
Output : une matrice de n lignes et n colonnes contenant les n premières lignes du
         triangle de Pascal

c ← matrice d'entiers de n lignes et n colonnes, initialisée avec des 0
c[0][0] ← 1

for (i ← 1 to n − 1 step +1)
{
    c[i][0] ← 1
    for (j ← 1 to i − 1 step +1)
    {
        [...]
    }
    c[i][i] ← 1
}

return c
```

Quelle instruction faut-il ajouter pour que l'algorithme calcule le résultat attendu ?

Q2 **(une instruction)**

. . . . . .

Question 3 – Quand les maths se cachent … (10 min)

En rangeant votre grenier, vous êtes tombé sur un vieux livre de maths. En le feuilletant, vous êtes tombé sur une page avec un algorithme assez mystérieux que voici, où `odd (n)` vaut **`true`** si n est impair et **`false`** sinon.

```
Input  : k, z, deux entiers positifs
Output : ?

y ← 1

while (k ≠ 0)
{
    if (odd (k))
    {
        k ← k − 1
        y ← y · z
    }
    k ← k div 2                          % div représente la division entière
    z ← z · z
}

return y
```

Cet algorithme vous parait assez intéressant. Malheureusement, les pages expliquant ce qu'il calcule sont manquantes. Trouvez quelle est la fonction mathématique calculée.

Q3 **(une expression mathématique)**

. . . . . .

Question 4 – Division entière (5 min)

Soient deux entiers x et y, avec $y \neq 0$. La division entière de x par y produit un quotient q et un reste r, tels que :

$$x = q \cdot y + r \qquad \text{tel que } \mid r \mid < \mid y \mid$$

Voici quelques exemples :

x	y	q	r
12	7	1	5
−12	7	−1	−5
12	−7	−1	5
−12	−7	1	−5

L'algorithme suivant permet de calculer le quotient et le reste de la division entière. La notation $\mid x \mid$ représente la valeur absolue de x.

```
Input  : x et y, deux entiers
Output : (q,r), le quotient et le reste de la division entière de x par y

q ← 0
r ← | x |

while (r ≥ | y |)
{
    q ← q + 1
    r ← r− | y |
}

if (x · y < 0)
{
    q ← −q
}

if ([...])
{
    r ← −r
}

return (q,r)
```

Cet algorithme est incomplet. Il manque en effet une condition pour le **`if`**. Quelle est-elle ?

Q4 **(une condition)**

. . . . . .

Question 5 – Mais où est mon DVD ? (5 min)

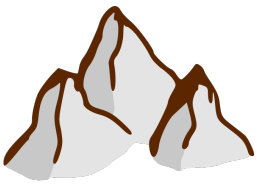
C'est chaque fois la même chose ! Quand vous voulez vous faire un petit film tranquille, pas moyen de retrouver le bon DVD ! Afin de vous aider, votre frère a élaboré un algorithme qui consiste à parcourir tous les DVDs de l'étagère, mais selon un ordre bien particulier et en étant deux à chercher. L'une des personnes regarde si le DVD cherché n'est pas le premier, et la seconde personne examine le dernier DVD. La première personne poursuit en regardant le second DVD, et l'autre personne l'avant-dernier DVD, etc.

```
Input  : dvds, tableau de n DVDs, indicé de 0 à n − 1
         x, DVD recherché
Output : true si le DVD x se trouve dans le tableau dvds et false sinon

found ← false
i ← 0
while (not found and i ≤ n div 2)                    % div représente la division entière
{
    if ([...])
    {
        found ← true
    }
    i ← i + 1
}

return found
```

Quelle condition faut-il pour l'instruction **if** afin que l'algorithme calcule le résultat attendu ?

☐	$dvds[i] \neq x$ **and** $dvds[n-i] \neq x$
☐	$dvds[i] = x$ **or** $dvds[n-i] = x$
☐	$dvds[i] = x$ **or** $dvds[n-1-i] = x$
☐	$dvds[i] \neq x$ **and** $dvds[n-1-i] \neq x$

Question 6 – Le plus long plateau (10 min)



Le mois prochain, vous allez faire une randonnée en montagne avec des amis. Vous avez reçu le plan précis du parcours qui sera effectué, avec l'altitude de chaque tronçon. Une question ne cesse de vous hanter ; vous voulez absolument connaitre la longueur de la plus longue partie du parcours complètement plate, c'est-à-dire le nombre maximal de tronçons consécutifs ayant la même altitude.

Prenons un exemple. Soit le tableau suivant $[5, 2, 2, 2, 4, 4, 5]$. Le plus long plateau est le sous-tableau constitué de 2 et sa longueur vaut 3.

Heureusement, il y a un algorithme pour répondre à votre question :

```
Input  : altitude, tableau d'entiers positifs de taille n, indicé de 0 à n − 1
Output : longueur de la plus longue partie de parcours plat

e ← 0
t ← −1
g ← 0
for (i ← 0 to n − 1 step +1)
{
    if (altitude[i] ≠ t)
    {
        if (e > g)
        {
            g ← e
        }
        e ← 1
        t = altitude[i]
    }
    else
    {
        e ← e + 1
    }
}

return [...]
```

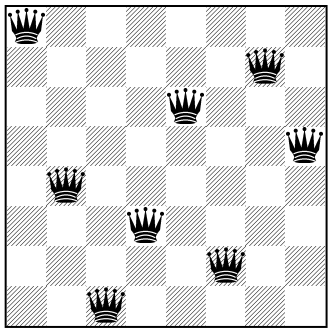
Quelle est la valeur que doit renvoyer l'algorithme pour que son résultat soit celui attendu ?

Q6 **(une expression)**

. . . . . .

Question 7 – Je les mets où mes reines ? (15 min)

Comme vous êtes nul aux échecs et que vous ne savez pas quoi faire avec votre échiquier, vous avez décidé d'inventer un petit jeu. Ce que vous voulez faire, c'est placer huit reines sur l'échiquier, de manière à ce qu'aucune d'entre elles ne puisse en attaquer une autre du plateau. Pour rappel, une reine peut attaquer toute autre reine se trouvant sur la même ligne, colonne ou diagonale. Voici une solution possible :



On se rend facilement compte qu'il y aura forcément une seule reine pour chaque colonne. On peut donc représenter une solution au problème comme un tableau où chaque élément contient le numéro de la ligne où placer la reine. Pour la solution montrée ci-dessus, ce tableau est $[0, 4, 7, 5, 2, 6, 1, 3]$

Étant donné que ce problème devient difficile à résoudre à la main lorsque la taille de l'échiquier augmente, en bon informaticien, vous avez développé un algorithme pour le résoudre. Pour cela, vous vous basez sur un autre algorithme (`unsafe`) qui est décrit ci-dessous. Ce dernier permet de tester si une reine est en danger ou non.

```
Input  : b, tableau de n entiers compris entre 0 et n − 1
         y, entier positif compris entre 0 et n − 1
Output : true, si la reine placée en b[y] se fait attaquer par au moins une des
         reines placées dans une des colonnes k telles que k < y, et false sinon

function unsafe (b, y)
{
    x ← b[y]
    for (i ← 1 to y step +1)
    {
        t ← b[y − i]
        if ([...])
        {
            return true;
        }
    }
    return false;
}
```

Cet algorithme est incomplet, il manque en effet une condition pour le **`if`**. Quelle est-elle ?

Q7a **(une condition)**

. . . . . .

Question 7 (suite)

Voici maintenant l'algorithme qui permet de résoudre votre jeu, pour un plateau de taille n donné.

```
Input  : n, entier strictement positif, taille de l'échiquier
Output : b, tableau de n entiers, où b[i] indique le numéro de la ligne où placer
         la (i+1)e reine, de manière à ce qu'aucune ne puisse attaquer une autre

b ← tableau d'entiers de taille n, indicé de 0 à n-1, initialisé avec des 0
y ← 0

while (y < n)
{
    while (b[y] ≠ n and unsafe (b, y))
    {
        b[y] ← b[y] + 1
    }

    if (b[y] < n)
    {
        if (y < n-1)
        {
            y ← y + 1
            b[y] ← 0
        }
        else
        {
            return b
        }
    }
    else
    {
        [...]
        b[y] ← b[y] + 1
    }
}
```

Il manque également une instruction à cet algorithme. Trouvez-la !

Q7b **(une instruction)**

. . . . . .

Question 8 – Exponentiation (20 min)

Soit un algorithme qui prend en entrée trois entiers positifs x, n et m. Cet algorithme calcule $(x^n) \mod m$, c'est-à-dire le reste de la division entière de x^n par m. Voici l'algorithme en question :

```
Input  : x, n, m, trois entiers positifs
Output : la valeur de (x^n) mod m

result ← 1
while (n ≠ 0)
{
    result ← result * x
    n ← n − 1
}
return result mod m
```

Cet algorithme n'est pas efficace. En effet, le temps d'exécution est proportionnel à n. De plus, si on suppose qu'on utilise des entiers encodés sur 32 bits, les calculs sur des grandes valeurs de n ne seront pas valides. Il est possible d'écrire un algorithme plus efficace, pour lequel le temps d'exécution est proportionnel à $\log_2 n$ et permettant de calculer, par exemple, $17^{1000000} \mod 13$ en n'utilisant que des entiers 32 bits. Cet algorithme fait l'objet de cette question et il vous est demandé de compléter son code ci-dessous. Vous pouvez utiliser les opérateurs div et mod, où div représente la division entière et mod le reste de cette division.

```
result ← 1
while (n ≠ 0)
{
    if ([...])      % (a)
    {
        [...]       % (b)
    }
    [...]           % (c)
}
return result
```

Q8a **(une condition)**

. . . . . .

Q8b **(une instruction)**

. . . . . .

Q8c **(deux instructions)**

. . . . . .

. . . . . .