be-OI 2012 **Finale** 14 mars 2012	**Remplissez ce cadre en MAJUSCULES et LISIBLEMENT, svp** 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 2012. Il comporte six 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. 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. Vous ne pouvez utiliser aucune fonction prédéfinie à l'exception de celles présentées à la page suivante.

5. 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.

6. Les participants **ne peuvent en aucun cas quitter leurs places** 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.

7. Vous avez **exactement une heure et quinze minutes** pour répondre à ce questionnaire. Un **aide-mémoire** sur le pseudo-code et les conventions prises se trouve à la page suivante.

Bonne chance !

001

Questionnaire finale

Aide-mémoire du pseudo-code

Dans ce questionnaire, il y a des questions pour lesquelles vous devez compléter une partie de code. On vous demande de les compléter avec soit des *expressions*, soit des *instructions*. Les expressions ont une valeur, les instructions effectuent une action.

Les expressions possèdent une valeur et sont utilisées dans les calculs, pour modifier les valeurs des variable et dans les conditions. Certaines expressions ont comme valeur un nombre (entier ou réel) et d'autres un booléen (**`true`** ou **`false`**). Les instructions représentent des actions à exécuter. Un algorithme est une suite d'instructions.

Les expressions les plus basiques sont les entiers (comme 3, -12...) et les variables (comme x, sum...). On peut également construire des expressions avec des opérateurs comme le montre le tableau suivant.

opération	**notation**	**exemples**	
égalité	$=$	$3 = 2$ vaut **`false`**	$1 = 1$ vaut **`true`**
différence	$\neq$	$1 \neq 4$ vaut **`true`**	$3 \neq 3$ vaut **`false`**
comparaison	$>, \geq, <, \leq$	$1 > 3$ vaut **`false`**	$5 \leq 5$ vaut **`true`**
addition et soustraction	$+, -$	$3 + 2$ vaut 5	$1 - 9$ vaut -8
multiplication	$\times$	2×7 vaut 14	-2×3 vaut -6
division entière	$/$	$10/3$ vaut 3	$9/3$ vaut 3
reste de la division entière	$\%$	$10\%3$ vaut 1	$9\%3$ vaut 0
« non » logique	**`not`**	**`not`** $3 = 2$ vaut **`true`**	**`not`** $1 = 1$ vaut **`false`**
« et » logique	**`and`**	$3 = 3$ **`and`** $5 \leq 9$ vaut **`true`**	$1 \geq 3$ **`and`** $2 = 2$ vaut **`false`**
« ou » logique	**`or`**	$3 = 3$ **`or`** $12 \leq 9$ vaut **`true`**	$1 \geq 3$ **`or`** $2 \neq 2$ vaut **`false`**
accès à un élément de tableau	`[]`	soit tab valant `[1,2,3]`, alors tab`[1]` vaut 2	
appel à une fonction retournant une valeur		`size (`$list$`)`	`min (12, 5)`

Les fonctions que vous pouvez utiliser sont, en plus de celles définies dans l'énoncé et celles définies sur les tableaux ci-dessous, les fonctions `max (`a`, `b`)`, `min (`a`, `b`)` et `pow (`a`, `b`)` qui permettent respectivement de calculer le maximum entre deux nombres, le minimum entre deux nombres et la puissance (a^b).

Le tableau suivant reprend les instructions de base :

opération	**notation**	**exemple**
affectation	$\leftarrow$	$x \leftarrow 20 + 11 \times 2$
renvoi	**`return`**	**`return`** 42
appel à une fonction (qui ne renvoie rien)	nom de la fonction	`sort (`$list$`)`

En plus, il y a également des *instructions de contrôle* qui sont au nombre de trois : **`if`**–**`else`**, **`while`** et **`for`**. Ces instructions vont faire en sorte d'exécuter un groupe d'instructions ou non, en fonction de conditions. 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.

Dans un algorithme, outre les variables permettant de stocker une valeur, on peut également utiliser des *tableaux* pour stocker plusieurs valeurs. Un tableau tab de taille n est indicé de 0 à $n-1$. La notation tab`[`i`]` permet d'accéder au $(i+1)^{\text{e}}$ élément du tableau. Ainsi, le premier élément du tableau est tab`[0]`. On peut créer un nouveau tableau tab de taille n, dont les éléments sont initialisés à 0, avec la notation $tab \leftarrow$ `newArray (`n`)`. On peut faire une copie d'un tableau tab de taille n vers un nouveau tableau $newTab$ avec la notation $newTab \leftarrow tab$.

À propos du cout des opérations

Les questions 3 et 5 vous demandent d'écrire des algorithmes efficaces afin d'obtenir le score maximum.

Lorsqu'on s'intéresse à la complexité temporelle d'un algorithme (une estimation du temps qu'il va prendre pour s'exécuter), une manière de procéder consiste à compter le nombre d'opérations élémentaires qu'il va effectuer. Toutes les instructions de base coutent une unité de temps. La création d'un tableau de taille n, et également la copie d'un tel tableau, coutent n unités de temps. Pour les instructions **`if`**–**`else`**, **`while`** et **`for`**, il faut compter une unité de temps pour le calcul de la condition. Puis, bien sûr compter les instructions qui seront exécutées.

Question 1 – Echauffement (5 min)

Complétez cet algorithme.

```
Input  : tab, un tableau d'entiers de taille n.
Output : renvoie le nombre de valeurs de tab ne valant ni 5 ni 7.

count ← 0

for (i ← 0 to n − 1 step 1)
{
    if ( [...] )                                          // Q1
    {
        count ← count + 1
    }
}
return count
```

Q1 **(une expression)**

. . . . . .

Question 2 – La machine à réécrire (15 min)

John C. Onway a inventé une nouvelle machine. La machine possède deux rubans : un ruban d'entrée sur lequel la machine peut lire, et un ruban de sortie sur lequel la machine peut écrire. Sur le ruban d'entrée, John écrit une série de symboles `0` ou `1`. La machine fonctionne selon un principe très simple : elle regarde chaque paire consécutive de symboles sur le ruban d'entrée, et écrit, pour chaque paire, un nouveau symbole sur le ruban de sortie, et ce, en suivant les régles que voici :

Entrée			Sortie
0	0	→	0
0	1	→	1
1	0	→	1
1	1	→	0

La première règle signifie que, si la machine lit un `0` suivi d'un `0` sur le ruban d'entrée, elle écrit un unique `0` sur le ruban de sortie.

La machine traite ainsi, à chaque étape, une paire de symboles sur le ruban d'entrée et décale sa position sur les rubans de lecture et d'écriture d'une position vers la droite. Pour le dernier symbole sur le ruban d'entrée, la machine suppose que le symbole suivant est implicitement un `0`. Évidemment, John s'arrange pour que les rubans d'entrée et de sortie aient toujours la même taille.

Q2(a) John a chargé un ruban d'entrée avec les symboles suivants dans la machine : | 1 | 0 | 1 | 0 | 1 |

Quels symboles se trouveront sur le ruban de sortie après que la machine a fini de traiter le ruban d'entrée ?

Q2(a) **(une série de symboles)**

... ...

John charge dans la machine le ruban de sortie obtenu à la question Q2(a), comme ruban ruban d'entrée, et charge un ruban vierge comme ruban de sortie de la machine. **Quels symboles** se trouvent maintenant sur le nouveau ruban de sortie ?

Q2(b) **(une série de symboles)**

... ...

John répète cette procédure 6 nouvelles fois (la sortie de l'étape précédent devient à chaque fois l'entrée de l'étape suivante, en commençant avec la réponse à la questions Q2(b) comme entrée). **Quels symboles** se trouvent maintenant sur le ruban après la 6^{me} iteration ?

Q2(c) **(une série de symboles)**

... ...

John répète maintenant cette procédure 73 fois, en commençant avec la réponse de la question Q2(c). **Quels symboles** se trouvent sur le ruban après la 73^{me} iteration ?

Q2(d) **(une série de symboles)**

... ...

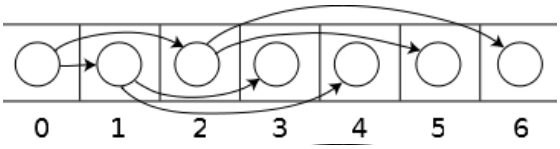
Question 3 – Nos ancêtres communs (15 min)

Un arbre généalogique ascendant est un arbre qui recense les parents, grand-parents, arrière-grand-parents, etc d'une personne. Cette personne est le point de départ de l'arbre (la racine). Juste au dessus de cette personne, l'arbre se divise en 2 branches : l'une représente son père (la branche de gauche) et l'autre représente sa mère la branche de droite). Chacune de ces deux branche se divise à nouveau en deux branches pour représenter les deux parents du père et de la mère (ce sont donc les grands-parents de la personne à la racine). Chacune de ces branches se séparera aussi en deux, et ainsi de suite. On choisit de toujours placer le père dans la branche de gauche et la mère dans la branche de droite.

Ma cousine sous-germaine a étudié de manière approfondie son arbre généalogique ascendant : elle est donc la racine de cet arbre et a retrouvé tous ses ancêtres jusqu'au 16ème siècle. Je lui ai demandé de me fournir la partie de son arbre qui me concerne, à savoir l'arbre généalogique de notre arrière-grand-mère commune.

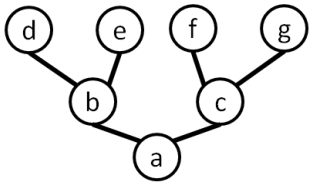
Vu qu'elle a son arbre généalogique ascendant au format électronique, on souhaite avoir une programme qui extrait une partie de cet arbre généalogique à partir d'une personne présente dans cet arbre (dans notre cas, on voudrait extraire la partie de l'arbre commençant à notre arrière-grand-mère commune).

En informatique, on manipule souvent ce genre de structure de données qu'on appelle arbre binaire. Chaque élément de l'arbre est appelé noeud et le noeud initial est appelé racine. A l'instar des arbres binaires, notre arbre généalogique ascendant peut être rangé dans un tableau (illustré ci-dessous) : la personne qui est à la racine de l'arbre se trouve dans la première cellule du tableau (celle d'indice 0). Un individu qui se trouve dans une cellule d'indice i, a ses parents qui se trouvent aux indices $2i + 1$ et $2i + 2$. Dans l'exemple ci-dessous, si je suis à l'indice 0, mon père sera à l'indice 1 et ma mère à l'indice 2. Mon grand-père paternel (le père de mon père) est à l'indice 3 et ma grand-mère paternelle (la mère de mon père) est à l'indice 4.



Nous considèrerons des arbres parfaits, c'est-à-dire que si notre arbre contient h générations, elles sont toutes complètes : pour chaque génération i : $1 \leq i \leq h$, le nombre d'individus est $2^{(i-1)}$. La taille totale du tableau est donc $2^h - 1$. Dans notre exemple ci-dessus, il y a 3 générations et donc $(2^3 - 1)$ individus dans l'arbre, ce qui correspond bien aux 7 cellules de notre tableau.

Exemple : l'arbre ci-dessous sera représenté comme suit $[a, b, c, d, e, f, g]$. Sa racine est a. Si on souhaite extraire la partie de l'arbre à partir de l'individu nommé c, on aura le résultat suivant :$[c, f, g]$.



On cherche un algorithme qui permette d'extraire une partie d'un arbre généalogique, stocké sous forme de tableau, à partir d'un individu donné et qui le stockera dans un autre tableau.

Complétez l'algorithme ci-dessous de façon à ce qu'il permette d'extraire d'un arbre généalogique parfait (représenté par un tableau $tab1$), une partie de l'arbre à partir d'un individu situé dans une cellule d'indice i dans $tab1$ ($0 \leq i < N$) où N est la taille de $tab1$. Ce sous arbre sera placé dans un tableau $tab2$ en respectant la même convention d'arrangement.

```
Input  :
 - tab1, un tableau de taille N contenant un arbre généalogique parfait et
   où il existe un h ≥ 1 tel que N = 2^h − 1.
 - 0 ≤ i < N.
 - tab2 est un tableau dont la taille est celle du sous arbre à extraire de tab1.
Output :
 - tab2 contient la partie extraite de tab1 à partir de l'individu situé à l'indice i.

k ← 0
n ← [...]                                   // Q3(a)
while (i < N)
{
    for(x ← 0 to n − 1 step 1)
    {
        [...]                               // Q3(b)
        k ← k + 1
    }
    i ← [...]                               // Q3(c)
    n ← [...]                               // Q3(d)
}
```

Q3a **(une expression)**

. . . . . .

Q3b **(une affectation)**

. . . . . .

Q3c **(une expression)**

. . . . . .

Q3d **(une expression)**

. . . . . .

Question 4 – L'algorithmique rend riche ! (10 min)

Vous vous lancez dans la collection des cartes *beOI*, des cartes à collectionner des 50 meilleurs candidats aux Olympiades d'informatique. Ces cartes sont vendues par paquet dont la composition est connue avant l'achat et contenant de deux à cinq cartes. Chacune des 50 cartes est au moins présente dans l'un des paquets mais peut, bien entendu, se retrouver dans plusieurs de ceux-ci. Chaque paquet ne comporte aucun doublon.

Vous aimeriez réaliser la collection complète des cartes, tout en dépensant le moins d'argent possible. Tous les paquets sont vendus à un même prix. Etant donné que la liste des paquets et leur contenu est public, vous réalisez qu'il doit être possible de calculer cela sur votre ordinateur.

Vous rendant vite compte que ce problème semble compliqué, vous demandez à un ami de vous trouver la stratégie optimale pour ce problème. Voici sa proposition :

> *Acheter le paquet contenant le plus grand nombre de cartes différentes non encore collectées.*
> *Recommencer cette règle jusqu'à ce que vous ayez chacune des cartes.*

La proposition de votre ami permet-elle bien d'obtenir, dans tous les cas, la solution optimale (la moins chère) ?

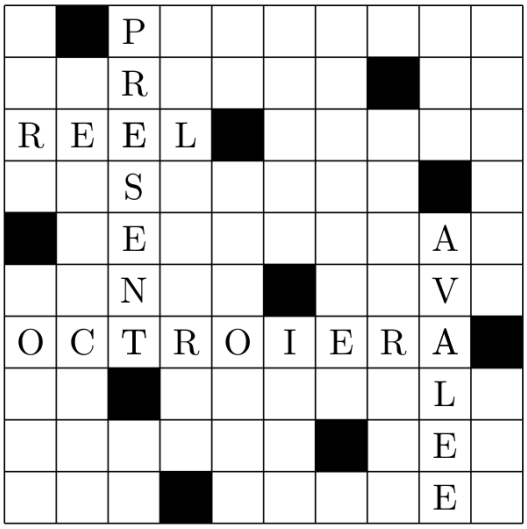
Q4	☐ Oui	☐ Non

Si vous avez répondu *non* à cette question, indiquez un contre-exemple via le tableau ci-dessous. Cochez les cartes présentes pour chacun des paquets. Vous ne devez pas nécessairement utiliser toutes les cartes ni tous les paquets disponibles, mais vous ne pouvez pas rajouter de colonne ni de ligne.

	Carte A	Carte B	Carte C	Carte D	Carte E	Carte F	Carte G
Paquet 1	☐	☐	☐	☐	☐	☐	☐
Paquet 2	☐	☐	☐	☐	☐	☐	☐
Paquet 3	☐	☐	☐	☐	☐	☐	☐
Paquet 4	☐	☐	☐	☐	☐	☐	☐
Paquet 5	☐	☐	☐	☐	☐	☐	☐
Paquet 6	☐	☐	☐	☐	☐	☐	☐

Question 5 – Mot Croisés (15 min)

Cruciverbiste dans l'âme, vous désirez écrire un logiciel de *mots croisés assistés par ordinateur*. Vous décidez de représenter une grille de mots croisés par un tableau à deux dimension pouvant contenir soit des caractères alphabétiques, soit le caractère *vide*, soit la valeur '`0`' représentant une case noire. Voici un exemple de grille et son équivalent sous forme de tableau stocké en mémoire.




```
crosswords[10][10] = {
    {' ','0','P',' ',' ',' ',' ',' ',' ',' '},
    {' ',' ',' ','R',' ',' ',' ',' ',' ','0',' ',' ',' '},
    {'R','E','E','L','0',' ',' ',' ',' ',' ',' '},
    {' ',' ',' ','S',' ',' ',' ',' ',' ',' ','0',' ',' '},
    {'0',' ',' ','E',' ',' ',' ',' ',' ',' ','A',' ',' '},
    {' ',' ',' ','N',' ',' ',' ','0',' ',' ',' ','V',' ',' '},
    {'O','C','T','R','O','I','E','R','A','0'},
    {' ',' ',' ','0',' ',' ',' ',' ',' ',' ','L',' ',' '},
    {' ',' ',' ',' ',' ',' ',' ',' ','0',' ',' ','E',' ',' '},
    {' ',' ',' ',' ','0',' ',' ',' ',' ',' ','E',' ',' '}
}
```

On accède à la case en `(i,j)` du tableau comme suit : `crosswords[i][j]`. Par exemples, la case `crosswords[0][1] = 'O'` et la case `crosswords[2][3] = 'L'`.

L'une des premières fonctions que vous devez écrire est la possibilité de compter tous les mots à remplir dans une grille, sachant qu'un mot doit comporter au moins deux lettres. Dans notre exemple, l'algorithme donnerait pour résultat **32**.

Il vous est demandé de compléter l'algorithme suivant.

```
Input  :
 - nb, un entier strictement positif.
 - crosswords, un tableau de caractères de nb lignes et nb colonnes
   pouvant contenir soit:
    - un caractère vide (' '),
    - un caractère alphabétique,
    - le caractère '0' pour indiquer une case noire.
Output : nbwords le nombre de mots d'au moins deux lettres à remplir dans cette grille
         de mots croisés.

nbwords ← 0
for i ← 0 to nb − 1 step 1
{
    p ← 0
    q ← 0
    for j ← 0 to nb − 1 step 1
    {
        if ( [...] )                                    // Q5(a)
        {
            if (p > 1)
            {
                nbwords ← nbwords + 1
            }
            [...]                                       // Q5(b)
        }
        else
        {
            p ← p + 1
        }
        if ( [...] )                                    // Q5(c)
        {
            if (q > 1)
            {
                nbwords ← nbwords + 1
            }
            [...]                                       // Q5(d)
        }
        else
        {
            q ← q + 1
        }
    }
    if ( [...] )                                        // Q5(e)
    {
        nbwords ← nbwords + 1
    }
    if ( [...] )                                        // Q5(f)
    {
        nbwords ← nbwords + 1
    }
}
return nbwords
```

Q5(a) **(une condition)**

. . . . . .

Q5(b) **(une affectation)**

. . . . . .

Q5(c) **(une condition)**

. . . . . .

Q5(d) **(une affectation)**

. . . . . .

Q5(e) **(une condition)**

. . . . . .

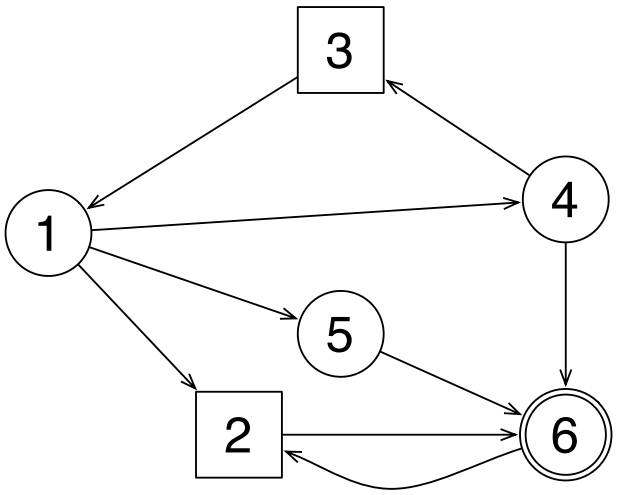
Q5(f) **(une condition)**

. . . . . .

Question 6 – Bob et Bobette (15 min)

Bob et Bobette jouent à un jeu de stratégie qu'ils ont inventé. Le jeu se joue sur une aire de jeu dont les cases sont de type rondes ou carrées. Le jeu se joue en déplaçant un galet sur les cases de l'aire de jeu. Entre les cases, se trouvent des flèches qui indiquent les déplacement autorisés. Le type de case indique le tour : quand le galet est sur une case carrée, c'est à Bobette de choisir où elle déplace le galet, et quand il s'agit d'une case ronde, c'est Bob qui choisit. Tous deux doivent respecter les flèches, mais la partie ne doit pas alterner strictement entre les deux joueurs.

Voici un exemple d'aire de jeu :



Un exemple de partie serait de commencer le jeu en 1, où c'est Bob qui joue. Il décide alors de mettre le galet sur 2, et Bobette peut jouer. Là, elle n'a pas d'autres choix que de se diriger vers 6, où Bob joue. Il place le galet sur 2, etc.

Pour déterminer qui gagne la partie, Bob et Bobette ont marqué une case comme étant gagnante pour Bobette. Sur l'image d'exemple, c'est la case 6. Remarquons que cela peut être une case ronde comme une case carrée. Les deux amis ont décidé que Bobette gagnerait la partie si cette dernière atteint la case désignée. Bob gagne par contre s'il arrive à faire durer la partie infiniment longtemps sans passer par la case gagnante. Dans l'exemple de début de partie ci-dessus, c'est Bobette qui gagne car la partie atteint 6.

Supposons que la partie commence sur la case 1 (initialement, le galet est sur la case 1). **Dans quelle case** Bob doit-il placer le jeton au premier coup, et chaque fois que la partie atteindra 1, pour être certain de gagner ? Si vous pensez qu'il n'y aucune solution, répondez 0.

Q6(a) **(un numéro de case)**

. . . . . .

Si par contre la partie commençait sur une autre case, Bobette aurait peut-être la possibilité d'adopter une stratégie qui lui garantisse de gagner. Pour chacune des cases, peut-on y commencer la partie pour garantir à Bobette de pouvoir gagner si elle utilise la stratégie optimale et quelque soit le jeu de Bob ? (1 point par case cochée correcte, -1 point par case cochée incorrecte, pas de score négatif à cette sous-question)

Q6(b)	**1**	**2**	**3**	**4**	**5**	**6**
Oui	☐	☐	☐	☐	☐	☐
Non	☐	☐	☐	☐	☐	☐