be-OI 2016 **Finale - SENIOR** samedi 19 mars 2016	**Remplissez ce cadre en MAJUSCULES et LISIBLEMENT, svp** PRÉNOM : ..... NOM : ..... ÉCOLE :	100 **Réservé**

Olympiade belge d'Informatique (durée : 2h maximum)

Ce document est le questionnaire de la finale de l'Olympiade belge d'Informatique 2016. Il comporte 10 questions qui doivent être résolues en **2h au maximum**.

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

1. Vérifiez que vous avez bien reçu la bonne série de questions (mentionnée ci-dessus dans l'en-tête):
   - Pour les élèves jusqu'en deuxième année du secondaire: catégorie **cadets**.
   - Pour les élèves en troisième ou quatrième année du secondaire: catégorie **junior**.
   - Pour les élèves de cinquième année du secondaire et plus: catégorie **senior**.
2. N'indiquez votre nom, prénom et école **que sur la première page**.
3. Indiquez vos réponses sur les pages prévues à cet effet, à la fin du formulaire.
4. Si, suite à une rature, vous êtes amené à écrire hors d'un cadre, répondez obligatoirement sur la même feuille (au verso si nécessaire).
5. Écrivez de façon **bien lisible** à l'aide d'un **stylo ou stylo bille** bleu ou noir.
6. Vous ne pouvez avoir que de quoi écrire avec vous; les calculatrices, GSM, ... sont **interdits**.
7. Vous pouvez toujours demander des feuilles de brouillon supplémentaires, au surveillant ou à votre enseignant.
8. Quand vous avez terminé, remettez la première page (avec votre nom) et les pages avec les réponses. Vous pouvez conserver les autres.
9. 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.
10. Si vous devez répondre en code, vous **devez** répondre en **pseudo-code** ou dans un **langages de programmation courant** (Java, C, C++, Pascal, Python, ...). Les erreurs de syntaxe ne sont pas prises en compte pour l'évaluation.
11. Les questions à choix multiples fonctionnent comme suit: une bonne réponse vous fait gagner des points, une mauvaise réponse vous en fait perdre. Si vous ne répondez pas, vous n'obtenez pas de point. Pour répondre à une question à choix multiples, il suffit de faire une croix (☒) dans la case qui correspond à la bonne réponse. Pour annuler une réponse, il faut complètement remplir la case (■).
12. Le surveillant ou l'enseignant ne peut répondre à aucune question. Toute erreur dans l'énoncé doit être considérée comme faisant partie de l'épreuve.

Bonne chance !

L'Olympiade Belge d'Informatique est possible grâce au soutien de nos sponsors et de nos membres:

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 $a\%b = 2$. Par exemple, dans le code suivant, la variable age reçoit 20.

```
annee_naissance ← 1994
age ← 2014 − 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 // moins cher !
}
```

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'indice 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
}
```

Question 1 – Échauffement

Q1(a) Voici un algorithme où n est un nombre entier strictement positif :

```
count ← 0
for (i ← 0 to n − 1 step 1)
{
    for (j ← 0 to n − 1 step 1)
    {
        if (i < j) { count ← count − 1 }
        else { count ← count + 1 }
    }
}
```

Q1(a) [3 pts]	Quelle est la valeur de la variable $count$ à la fin de l'exécution de l'algorithme ?
☐	0
☐	n
☐	$n(n-1)$
☐	n^2

Q1(b) Considérons le code suivant:

```
i ← 0
while (i ≤ 15)
{
    i ← 2 ∗ i + 1
}
```

Q1(b) [3 pts]	Quelle est la valeur de la variable i à la fin de l'exécution de l'algorithme ?

Q1(c) Le code `x%2 = 0` signifie que x est pair.

Q1(c) [3 pts]	Quelle condition est équivalente à « a *et* b *sont impairs* » ?
☐	`not (a%2 = 0 and b%2 = 0)`
☐	`not (a%2 = 0 or b%2 = 0)`
☐	`a%2 = 0 and b%2 = 0`
☐	`a%2 = 0 or b%2 = 0`

Question 2 – Le calcul caché

En rangeant votre grenier, vous tombez sur une ancien livre de mathématiques, dont une des pages présente le mystérieux algorithme ci-dessous:

La fonction `odd(k)` renvoie **`true`** si k est un nombre impair, et **`false`** si k est pair.

```
Input : k et z, deux nombres naturels
Output : ?

y ← 1
while ( k ≠ 0 )
{
  if (odd (k))
  {
    k ← k − 1
    y ← y ∗ z
  }
  k ← k/2
  z ← z ∗ z
}
return y
```

Cet algorithme vous parait assez intéressant. Malheureusement, les pages expliquant ce qu'il calcule sont manquantes.

Q2(a) [8 pts]	**Quelle fonction mathématique cet algorithme calcule-t-il ?**

Question 3 – Les parenthèses

En guise de punition, Bart Simpson doit recopier 100 expressions mathématiques sur l'ordinateur, et il veut vite en être débarrassé ! Son institutrice, Madame Krabappel, le surveille de près. Elle a même conçu un algorithme qui vérifie si les parenthèses sont correctement utilisées.

Une expression est une séquence de caractères. Les parenthèses sont correctes si, à chaque parenthèse ouvrante « (» correspond un parenthèse fermante «) » qui suit. De plus, chaque parenthèse fermante «) » doit correspondre à une parenthèse ouvrante « (» qui précède.

Par exemple, la séquence de caractères $(a + 2 + (c/4) - (1 + e))$ utilise les parenthèses de manière correcte. Les séquences $)a + (2 - x)/(1$ et $a + (2 - 3)/((3 + a) + 3$, par contre, ne les utilisent pas correctement.

L'algorithme de Madame Krabappel est donné ci-dessous, mais il vous faut d'abord le compléter !

```
Input  : s, un tableau de caractères.
         n, un nombre naturel, la longueur du tableau s.
Output : true si l'expression donnée dans le tableau s utilise les
         parenthèses de façon correcte, sinon false.

num ← 0
for (i ← 0 to [...] step 1)                                    // (a)
{
    if ( s[i] = '(' )
    {
        [...]                                                  // (b)
    }
    if ( s[i] = ')' )
    {
        [...]                                                  // (c)
    }
    if ( [...] )                                               // (d)
    {
        return false
    }
}
return (num = 0) // Ceci renvoie true si num vaut 0, et false sinon
```

Q3(a) [2 pts]	**Complétez** (a)

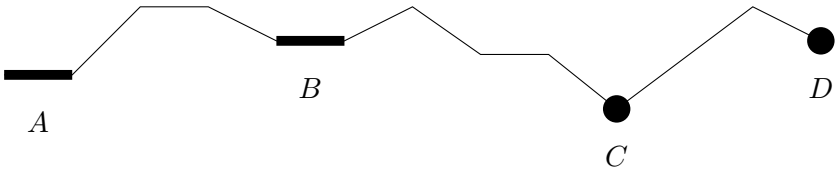
Q3(b) [2 pts]	**Complétez** (b)

Q3(c) [2 pts]	**Complétez** (c)

Q3(d) [2 pts]	**Complétez** (d)

Question 4 – Des montagnes et des vallées

Vous partez en expédition scientifique dans l'Oural, et vous souhaitez installer votre tente au fond d'une vallée, car il s'y trouve certainement un cours d'eau. La figure ci-dessous donne les altitudes successives tout au long de votre chemin:



Le *plateau d'une vallée* est un endroit que l'on peut quitter uniquement en montant, dans toutes les directions possibles. Il peut s'agir aussi bien de points que de segments. Sur le dessin ci-dessus, il y a quatre *plateaux*, indiqués par des lettres A, B, C et D (le petit morceau plat entre B et C n'est pas un plateau, car il est possible d'en descendre pour atteindre C). Un chemin qui serait complètement plat n'a qu'un seul plateau.

Vous pouvez facilement collecter les altitudes tout au long de votre chemin dans un tableau de nombres entiers, et ce, grâce à votre GPS. Vous obtenez alors un tableau comme $[5, 4, 3, 3, 3, 9, 9, 8]$. Celui-ci possède deux plateaux: la partie plate d'altitude 3, et le dernière point d'altitude 8. Le tableau $[4]$ possède un plateau, tout comme le tableau $[3, 1, 8]$. Le tableau $[4, 3, 2, 2, 9, 9, 8]$ en contient 2.

Q4(a) [2 pts]	**Combien de plateaux y a-t-il dans le tableau** $[5, 2, 2, 1, -7]$ **?**

Q4(b) [2 pts]	**Combien de plateaux y a-t-il dans le tableau** $[1, -1, -1, -5, 5, 9, 9, -9, -8, -6, 12, 17, 17, 4, -4, -1, -12]$ **?**

Vous souhaitez obtenir rapidement un aperçu de tous les campements possibles. L'algorithme ci-dessous calcule le nombre de plateaux dans un tableau.

```
Input  : tab, un tableau non-vide de nombres entiers.
         n, un entier strictement positif, la longueur de tab.
Output : le nombre de plateaux dans tab. (tab n'est pas modifié.)

nb ← 0
z ← true
for (i ← 1 to n − 1 step 1)
{
    if ([...])                                          // (a)
    {
        nb ← nb + 1
        z ← false
    }
    z ← [...]                                           // (b)
}
if (z)                                                  // (pour le dernier élément)
{
    nb ← nb + 1
}
return nb
```

Q4(c) [3 pts]	**Que manque-t-il en** `(a)` **?**

Q4(d) [3 pts]	**Que manque-t-il en** `(b)` **?**

Question 5 – Les cartes à collectionner…

Vous êtes un joueur passionné par le jeu de cartes *Hassle: The Dorkening*. Pour pouvoir gagner le plus de parties possibles, vous devez posséder le plus grand nombre possible de cartes uniques. Chaque carte possède un numéro. Votre collection de cartes est triée selon ces numéros, du plus petit au plus grand, et placée en pile. Cette pile est appelée p_1 et la carte avec le plus grand numéro se trouve au-dessus. Vous n'avez aucune carte en double.

Vous avez acheté l'intégralité de la collection d'un autre joueur. Cette collection est triée de la même manière, sous forme d'une autre pile p_2. Il n'y a pas de doublon dans p_2 non plus, mais certaines cartes de p_2 sont peut-être déjà dans votre collection !

Évidemment, vous ne souhaitez conserver que les cartes que vous ne possédiez pas encore. Vous souhaitez revendre les cartes en double (et faire un bénéfice au passage!)

12
10
8
5
3
1

p_1

11
10
8
3

p_2

Vous ne pouvez voir que les cartes qui se trouvent au sommet des piles. Et il n'est naturellement pas possible d'extraire une carte du milieu de la pile sans avoir retiré les cartes qui se trouvent par dessus.

Pour résumer les 4 opérations possibles sur les piles sont les suivantes:

- `top(`p`)` : pour lire le numéro de la carte au sommet de la pile p ;
- `pop(`p`)` : pour retirer la carte au sommet de la pile p ;
- `push(`p`,`e`)` : pour ajouter la carte numéro e au sommet de la pile p ;
- `isEmpty(`p`)` : pour vérifier si la pile p est vide ou pas.

Vous souhaitez maintenant trier toutes les cartes en deux nouvelles piles: *collection* et *doubles*. La pile *collection* doit contenir une collection de cartes aussi complète que possible. Mais elle ne peut contenir qu'un seul exemplaire de chaque carte. La pile *doubles* doit contenir les cartes en double uniquement.

Ces deux nouvelles piles doivent aussi être triée, mais cette fois-ci avec le plus grand numéro au bas de la pile, et le plus petit sur le dessus. Si nous appliquons cela à la figure ci-dessus, nous obtenons le résultat suivant:

1
3
5
8
10
11
12

collection

3
8
10

doubles

L'algorithme ci-dessous effectue cette opération, mais vous devez le compléter ! (Clarification: la fonction **not**(x) renvoie l'inverse de x).

```
Input  : p1, p2 : deux piles, triées par ordre
           croissant, avec le plus grand élément au-dessus,
           et auxquelles on peut appliquer les opérations
           ci-dessus.

Output : La pile collection, qui contient une collection
           aussi complète que possible, triée à
           l'envers.
         La pile doubles, qui contient tous les doubles,
           triée à l'envers.



collection ← une nouvelle pile vide
doubles ← une nouvelle pile vide

while ([...])                                   // (a)
{
    if ([...])                                  // (b)
    {
        m ← pop(p1)
        push(collection, m)
    }
    else if (top(p1) = top(p2))
    {
        m ← pop(p1)
        push([...], m)                          // (c)
    }
    else
    {
        m ← pop(p2)
        push(collection, m)
    }
}

while ( not(isEmpty(p1)) )
{
    m ← pop(p1)
    [...]                                        // (d)
}

while ( not(isEmpty(p2)) )
{
    m ← pop(p2)
    [...]                                        // (d)
}
```

Q5(a) [2 pts]	**Quelle est la condition manquante en** (a) **?**

Q5(b) [2 pts]	**Quelle est la condition manquante en** `(b)` **?**

Q5(c) [2 pts]	**Quelle est la pile manquante en** `(c)` **?**

Q5(d) [2 pts]	**Quelle est l'instruction manquante en** `(d)` **(présente à deux reprises) ?**

Question 6 – L'informaticien lunatique

Un informaticien lunatique a écrit le code suivant, où on suppose que la variable v ne contient que des valeurs entières:

```
Input : v, une valeur entière

if ( (v > 1 and v ≤ 2) or (v ≥ 4 and v ≤ 6) )
{
   if ((v > 2 and v < 4) or (v > 5 and v ≤ 9)
   {
      Afficher('Luke, I am your father!')
   }
   else
   {
       if (v ≥ 3 and v ≤ 4) // condition ⋆
       {
           Afficher('May the force be with you!')
       }
   }
}
else
{
   Afficher ('Hhhrrrrrrrrrrraaaaaaaaaaaaeeeeerrrrrrr')
}
```

Pour les valeurs suivantes de v, le programme va-t-il, oui ou non, afficher *Hhhrrrrrrrrrrraaaaaaaaaaaaeeeeerrrrrrr* (ndlr: au cas où vous ne l'auriez pas reconnu, il s'agit de Chewbacca qui chante sous la douche) ?

Q6(a) [1 pt]	Oui / non	$v = 1$
Q6(b) [1 pt]	Oui / non	$v = 2$
Q6(c) [1 pt]	Oui / non	$v = 3$
Q6(d) [1 pt]	Oui / non	$v = 4$

Q6(e) [1 pt]	**Qu'est-ce que le programme affichera pour $v = 2$?**

Q6(f) [1 pt]	**Qu'est-ce que le programme affichera pour $v = 4$?**

Q6(g) [2 pts]	**Donnez toutes les valeurs de v qui garantissent l'affichage de *Luke, I am your father!***

On vous donne maintenant le droit de simplifier la condition $\star$ du programme de façon à ce que le résultat du code reste le même après la modification.

Q6(h) [3 pts]	**Par quelle condition plus simple remplacez vous la condition $\star$?**

Question 7 – Qui cherche trouve

On reçoit un tableau de nombres entiers, triés du plus petit au plus grand, ainsi qu'un nombre entier N. On vous demande une méthode qui retourne la position de N dans le tableau (on suppose que le nombre s'y trouve).

```
Input : vec, un tableau d'entiers triés par ordre croissant
        lg, un entier strictement positif, la longueur de vec
        N, un entier.

deb ← 0
fin ← lg − 1

while (deb ≤ fin)
{
    mil ← [...]                    // (a)
    if (tab[mil] < N)
    {
        [...]                      // (b)
    }
    if (tab[mil] > N)
    {
        [...]                      // (c)
    }
    if (tab[mil] = N)
    {
        return mil
    }
}
```

Q7(a) [3 pts]	**Que manque-t-il en `(a)` ?**

Q7(b) [3 pts]	**Que manque-t-il en `(b)` ?**

Q7(c) [3 pts]	**Que manque-t-il en `(c)` ?**

Question 8 – Les tableaux farceurs

Habituellement, on parcourt un tableau V en commençant par la première case $V[0]$, puis en visitant $V[1]$, $V[2]$, jusqu'à la dernière case. On considère ici une variante, appelée les « tableaux farceurs ». L'ordre de parcours des éléments est donné par un second tableau $suiv$ et une variable t.

Voici l'explication: le premier élément que l'on visite est celui à l'indice t (par exemple, si $t = 6$, on il s'agit de la septième case, puisqu'on numérote les case à partir de 0). Le deuxième élément est à l'indice $suiv[t]$. Le troisième est à l'indice $suiv[suiv[t]]$, etc. Si un élement d'indice i est le dernier du tableau, on met $suiv[i] = -1$.

Voici un exemple:

$t = 5$

$V =$	89	19	34	57	71	16	43	33	30	88	66	82	93	85	17	34	43	29	37	78
$suiv =$	12	17	18	10	19	14	3	15	7	0	4	13	-1	9	1	2	6	8	16	11

Q8(a) [4 pts]	**Quelle est la séquence représentée par cet exemple ?**

Voici un algorithme pour trier ces tableaux. L'idée générale de cet algorithme est la suivante. On introduit un variable tri telle que. Si on parcourt le tableau V en commençant à la position tri, puis à la position $suiv[tri]$, puis $suiv[suiv[tri]]$, *etc*, et ce, jusqu'à rencontrer la valeur -1, alors on visite une séquence triée par ordre croissant. La variable tri donne donc accès à une portion triée du tableau. es éléments accessibles à partir de t formant la partie du tableau qui n'a pas encore été triée.

Initialement, $tri = -1$, car on n'a encore rien trié. À chaque itération de la boucle **`while`** extérieure, on prélève un élément de la partie non-visitée (accessible à partir de t), et on vient l'insérer en bonne position dans la partie triée.

(Suite au verso)

```
Input : V, un tableau de nombres entiers
        suiv, un tableau de nombres entiers
        N, un entier strictement positif, qui est la taille de V et suiv
        t, un entier compris entre 0 et N − 1 (inclus)

tri ← −1
while (t ≠ −1)                              // Boucle while la plus extérieure
{
   j ← tri
   while (j ≠ −1 et V[t] > V[j])
   {
      jprec ← [...]                         // (1)
      j ← suiv[j]                           // Ligne ⋆
   }

   if ([...])                               // (2)
   {
       tri ← t
   }
   else
   {
       suiv[jprec] ← t
   }
   tmp ← suiv[t]
   suiv[t] ← j
   [...] ← tmp                               // (3)
}
t ← sorted
```

Q8(b) [2 pts]	**Complétez l'expression** (1)

Q8(c) [2 pts]	**Donnez la condition** (2)

Q8(d) [2 pts]	**Donnez la variable manquante** (3)

Q8(e) [2 pts]	***Au pire cas*, combien d'itérations de la boucle `while` la plus extérieure exécutera-t-on (en supposant que le tableau est de taille N ?**

Q8(f) [2 pts]	***Dans le meilleur des cas*, combien d'itérations de la boucle `while` la plus extérieure exécutera-t-on (en supposant que le tableau est de taille N ?**

Fixons maintenant le tableau $suiv$ et la variable t comme suit (pour $N = 5$):

$$t = 0$$

$$suiv = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & -1 \\ \hline \end{array}$$

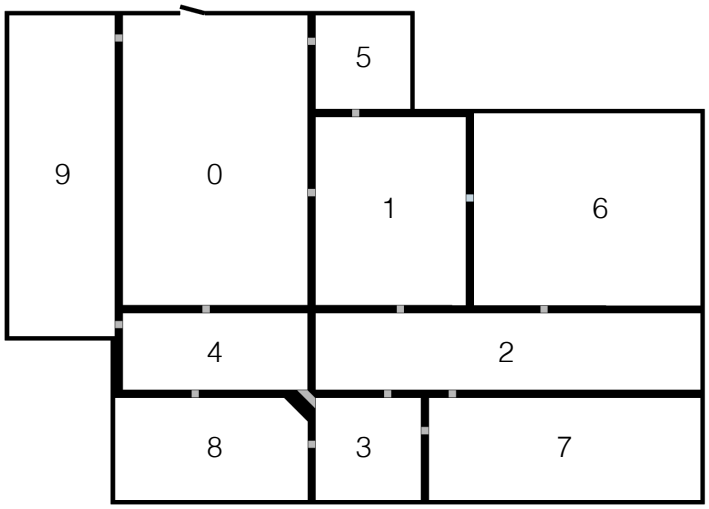
Q8(g) [3 pts]	**Donnez des valeurs pour le tableau V qui garantissent que la ligne $\star$ est exécutée 25 fois.**

Question 9 – Éteindre la lumière

Auteur : *Jorik Jooken, Gilles Geeraerts*

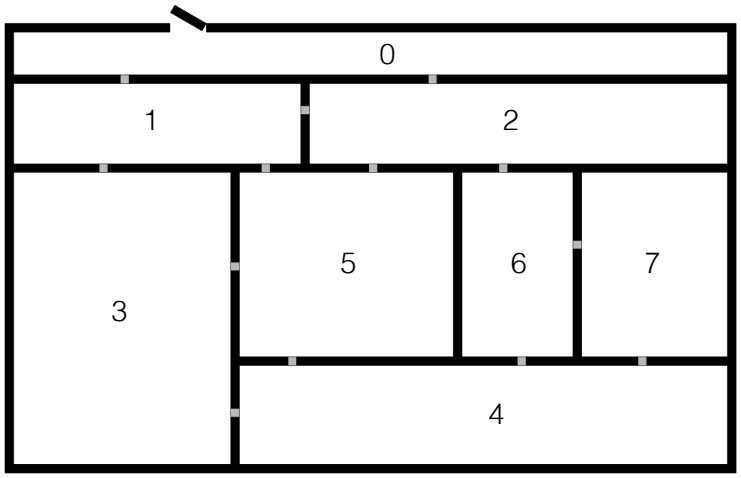
Pierre et ses amis partent en vacance. Juste avant de partir, Pierre se rend compte qu'il a oublié d'éteindre toutes les lumières de la maison. Dans cette maison, il y a une ampoule dans chaque couloir, qui est commandée par un interrupteur à mi-chemin. Une fois que Pierre a traversé un couloir et éteint la lumière, il ne peut donc pas y repasser car il y fait trop noir. Pierre doit donc trouver une manière de visiter toute les pièces et tous les couloirs de se maison *en ne traversant chaque couloir qu'une seule fois*. Il peut traverser les pièces plusieurs fois mais ne peut pas faire demi-tour au milieu d'un couloir. Pierre commence son parcours dans la pièce numéro 0 (où se trouve la porte d'entrée) et doit y revenir à la fin de son parcours pour pouvoir sortir de la maison. Étant donné une carte de la maison où les murs sont représentés en noir, et les couloirs entre deux pièces en gris, on vous demande de déterminer si de tels parcours sont possibles

Voici une autre carte:



Q9(a) [3 pts]	**Donnez, si possible, un chemin qui va de la pièce 0 à la pièce 0, et visite une et une seule fois chaque couloir. Indiquez « impossible » si vous pensez qu'un tel chemin n'existe pas.**

Voici une carte de la maison:



Q9(b) [3 pts]	**Donnez, si possible, un chemin qui va de la pièce 0 à la pièce 0, et visite une et une seule fois chaque couloir. Indiquez « impossible » si vous pensez qu'un tel chemin n'existe pas.**

Q9(c) [6 pts]	**Quelle condition doit on vérifier sur le nombre de couloirs et de pièces pour être certain qu'il existe un chemin allant de la pièce 0 à la pièce 0, et visitant une et une seule fois chaque couloir ? La condition doit garantir qu'une solution existe, et doit également assurer qu'aucune solution n'existe si elle n'est pas vérifiée.**

Question 10 – The Usual Suspects

Auteur : *Niels Joncheere*

Un crime a été commis en ville, mais la police ne sait pas qui sont les coupables. Il y a en ville 5 criminels: Alex, Bernard, Camille, David et Eva. La police les arrête tous les 5 comme suspects. Ils sont interrogés et affirment:

1. Alex dit: « Le crime a été commis par Bernard, avec l'aide de Camille et/ou d'Eva. »
2. Bernard dit: « Si Camille est impliquée dans le crime, alors Alex est complice. »
3. Camille dit: « Soit Bernard n'a rien à voir avec ce crime, ou bien Eva n'a rien à voir avec ce crime. »
4. David dit: « Si Camille n'est pas impliquée dans ce crime, alors Alex est complice. »

S'il est dit qu'une personne *pourrait être impliquée dans ce crime*, et que personne n'affirme le contraire, alors la police peut emprisonner cette personne jusqu'au lendemain. Pour savoir qui peut être emprisonné, les policiers analysent les réponses fournies séparément.

Q10(a) [2 pts]	**Combien de personnes passeront la journée en prison si la police ne prend en compte que la réponse 1 ?**

Q10(b) [2 pts]	**Quels sont les suspects qui sont *certainement* impliqués dans le crime si la police ne prend en compte que la réponse 1 ?**

Q10(c) [4 pts]	**La police pense que les suspects sont de mèche. En d'autres termes: les vrais coupables sont ceux qui, selon les affirmations 1, 2, 3 et 4, ne sont certainement pas liés au crime. Quels suspects sont les vrais coupables ?**

Donnez vos réponses ici !

Q1(a)		**Ne cochez qu'UNE seule réponse !**	/3
	☐	0	
	☐	n	
	☐	$n(n-1)$	
	☐	n^2	

Q1(b) **une valeur**	/3
. . . . .	

Q1(c)		**Ne cochez qu'UNE seule réponse !**	/3
	☐	`not (a%2 = 0 and b%2 = 0)`	
	☐	`not (a%2 = 0 or b%2 = 0)`	
	☐	`a%2 = 0 and b%2 = 0`	
	☐	`a%2 = 0 or b%2 = 0`	

Q2(a) **une fonction mathématique**	/8
. . . . .	

Q3(a) **Une expression**	/2
. . . . .	

Q3(b) **Une instruction**	/2
. . . . .	

Q3(c) **Une instruction**	/2
. . . . .	

Q3(d) **Une expression**	/2
. . . . .	

Q4(a) **un nombre**	/2
. . . . .	

Q4(b) **un nombre**	/2
. . . . .	

Q4(c) **une condition**	/3
. . . . .	

Q4(d) **une expression logique**	/3
Q5(a) **Une condition**	/2
Q5(b) **Une condition**	/2
Q5(c) **Le nom d'une pile**	/2
Q5(d) **Une instruction**	/2

	Oui	non		
Q6(a) /1	☐	☐	$v = 1$	/4
Q6(b) /1	☐	☐	$v = 2$	
Q6(c) /1	☐	☐	$v = 3$	
Q6(d) /1	☐	☐	$v = 4$	

Q6(e) **Une phrase affichée par le programme, ou bien « rien ».**	/1
Q6(f) **Une phrase affichée par le programme, ou bien « rien ».**	/1
Q6(g) **Une ou plusieurs valeurs de v**	/2
Q6(h) **Une condition**	/3
Q7(a) **Une expression**	/3
Q7(b) **Une instruction**	/3
Q7(c) **Une instruction**	/3
Q8(a) **Une séquence de nombres**	/4

Q8(b) **Une expression**	/2
Q8(c) **Une condition**	/2
Q8(d) **Une variable**	/2
Q8(e) **Une expression dépendant de N**	/2
Q8(f) **Une expression dépendant de N**	/2
Q8(g) **Un tableau V de taille $N = 5$**	/3
Q9(a) **Un chemin ou bien « impossible ».**	/3
Q9(b) **Un chemin ou bien « impossible ».**	/3
Q9(c) **Une condition.**	/6
Q10(a) **Un nombre**	/2
Q10(b) **Un ou plusieurs noms**	/2
Q10(c) **Un ou plusieurs noms**	/4