be-OI 2013 **Finale** 9 mars 2013	**Remplissez ce cadre en MAJUSCULES et LISIBLEMENT, svp** PRÉNOM (**MAJUSCULES !**): ..... NOM (**MAJUSCULES !**): ..... ÉCOLE :	**Réservé**

Olympiade belge d'Informatique (durée : 1h15 maximum)

Ce document est le questionnaire de la finale de l'Olympiade belge d'Informatique 2013. Il comporte six questions qui doivent être résolues en **1h15 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**. 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. 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.

5. 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 tandis que ne pas répondre est neutre (aucun point perdu ni gagné). 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 (■).

6. 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 ne peut être posée qu'aux organisateurs. Toute question logistique peut être posée aux surveillants.

7. Vous **ne pouvez pas quitter votre place** pendant l'épreuve. Si vous devez absolument 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.

Cette œuvre est mise à disposition selon les termes de la Licence Creative Commons Attribution 2.0 Belgique.

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 19.

```
annee_naissance ← 1993
age ← 2012 − annee_naissance
```

On peut n'exécuter du code que si une certaine condition est vérifiée en utilisant une instruction **`if`** et éventuellement un autre code dans le cas contraire en ajoutant une instruction **`else`**. L'exemple suivant vérifie si une personne est majeure ou non et stocke la réponse dans la variable booléenne (vrai ou faux) $majeur$.

```
if (age ≥ 18)
{
    majeur ← true
}
else
{
    majeur ← false
}
```

Pour manipuler plusieurs éléments d'un coup, 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 la 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, on peut faire usage d'une boucle **`for`**. La notation **`for`** $(i \leftarrow a$ **`to`** b **`step`** $k)$ représente une boucle qui sera répétée aussi longtemps que $i \leq b$, dans laquelle i commence à la valeur a, et est augementée 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 exécuter une boucle avec **`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 $4 \ldots$ 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 – (5 min – 4 pts)

Considérons le fragment de programme suivant :

```
if ((a < b) = (b < c) ) {
   Afficher (b)
} else if ( (a > c) = (b < c) ) {
   Afficher (c)
} else {
   Afficher (a)
}
```

Indiquez, dans chacun des cas suivants, ce que ce fragment de programme affiche.

(a) Quand $a = 10$, $b = 0$ et $c = -10$.

Q1(a) [1 pt] **un nombre**

. . . . . .

(b) Quand $a = 2013$, $b = 2000$ et $c = 2012$.

Q1(b) [1 pt] **un nombre**

. . . . . .

(c) Quand $a = 400$, $b = 400$ et $c = 400$.

Q1(c) [1 pt] **un nombre**

. . . . . .

(d) Si a est l'âge de John Dundee, b l'âge de Sheila Dundee (la fille de John) et c l'âge de Priscilla Dundee (la mère de John).

Q1(d) [1 pt] **L'âge de quelle personne ?**

. . . . . .

Question 2 – Un programme clair… (10 min – 7 pts)

Considérons le code suivant, qui reçoit en entrée un valeur entière x positive ou nulle ($x \geq 0$) :

```
output ← −1
while (output=−1)
{
  if (x ≤ 3)
  {
    if (x > 1)
    {
      output ← x % 2
    }
    else
    {
      output ← x
    }
  }
  else
  {
    if (x − 3 ≤ 3)
    {
      if (x % 2 = 1)
      {
        output ← x / 5
      }
      else
      {
        output ← (x % 4) / 2
      }
    }
    else
    {
       x ← x % 7
    }
  }
}
Afficher(output)
```

Indiquez, pour chacune des affirmations suivantes, si elle est **correcte** ou **incorrecte**.

	Correcte	Incorrecte	
Q2(a) [1pt]	☐	☐	0 est affiché *uniquement* si x vaut 0, 2 ou 4
Q2(b) [1pt]	☐	☐	La valeur 2 n'est jamais affichée
Q2(c) [1pt]	☐	☐	Si $x = 5$, la valeur 1 est affichée
Q2(d) [1pt]	☐	☐	Il existe une valeur de x qui fait en sorte que la valeur -1 soit affichée
Q2(e) [1pt]	☐	☐	Si $x = 6$, la valeur 0 est affichée
Q2(f) [1pt]	☐	☐	Si la valeur de x est paire, la valeur 0 est affichée
Q2(g) [1pt]	☐	☐	1 est affiché si $x\%7$ vaut 1, 3, 5 ou 6

Question 3 – Ces boucles me rendent malade (15 min – 12 pts)

Considérons le fragment de programme suivant :

```
for (i ← 1 to 100 step 1)
{
  for (j ← 1 to 100 step 1)
  {
    for (k ← 1 to 100 step 1)
    {
      if (i < j and j < k and i + j + k = 100)
      {
        Afficher(i, j, k)
      }
    }
  }
}
```

Ce fragment affiche tous les triplets de nombres dont la somme fait 100 (en ne considérant que des nombres entiers strictement supérieurs à 0).

Le premier triplet affiché par ce fragment est '`1 2 97`'.

Quel est le troisième triplet à être affiché ?

Q3(a) [1 pt] **trois nombres entiers**

. . . . . .

Quel est l'avant-dernier triplet à être affiché ?

Q3(b) [2 pts] **trois nombres entiers**

. . . . . .

Ré-écrivez le programme de façon à éviter les tests `i<j` et `j<k` mais de façon à ce que le programme résultant affiche toujours les mêmes triplets, dans le même ordre.

Remplissez les parties manquantes dans ce fragment **[1 pt par réponse]**

```
for (i ← [...] to 100 step 1)     // (c)
{
  for (j ← [...] to 100 step 1) // (d)
   {
     for (k ← [...] to 100 step 1) // (e)
      {
        if (i + j + k = 100)
         {
           Afficher(i, j, k)
         }
      }
   }
}
```

Q3(c) **une expression**

. . . . . .

Q3(d) **une expression**

. . . . . .

Q3(e) **une expression**

. . . . . .

Réécrivez le programme de façon à éviter une instruction **`if`** et à n'avoir que deux boucles imbriquées au lieu de trois. Les mêmes triplets doivent toujours être affichés, dans le même ordre.

Remplissez les parties manquantes dans ce fragment.

```
for (i ← [...] to [...] step 1) // (f)
{
  for (j ← [...] to [...] step 1) // (g)
   {
     Afficher(i, j, [...])            // (h)
   }
}
```

Q3(f) [2 pts] **deux expressions**

. . . . . .

. . . . . .

Q3(g) [3 pts] **deux expressions**

. . . . . .

. . . . . .

Q3(h) [1 pt] **une expression**

. . . . . .

Question 4 – Traverser une rivière en ménageant les susceptibilités (15 min – 9 pts)

Quatre compères (un loup-garou, une momie, un vampire et le monstre de Frankenstein) veulent traverser une rivière, mais ne disposent que d'une seule barque pour aller de la rive A à la rive B. Malheureusement, pour pouvoir traverser cette rivière, ils vont devoir tenir compte de deux contraintes :

1. la barque n'a que deux places et
2. Tous ces joyeux drilles ne s'entendent pas très bien. La momie supporte le loup-garou, mais pas le vampire, etc... On veut donc éviter que certains de ces personnages se retrouvent *seuls* avec un autre qu'il ne supportent pas (la présence d'un troisième compère garantit que les ennemis ne se battront pas). Les couples possibles sont résumés dans le tableau ci-dessous, où **L** représente le loup-garou, **M**, la momie, **V** le vampire, **F** le monstre de Frankenstein ; ☺ indique un couple qui peut rester seul, et ↯ indique un couple qui ne peut pas rester seul :

	M	**V**	**F**
L	☺	☺	↯
M		↯	☺
V			☺

Par exemple, si tout le monde se trouve sur la rive A, on pourra faire embarquer **L** et **M** ensemble pour aller à la rive B. Ensuite, **L** pourra revenir seul vers la rive A, mais ne pourra pas embarquer **F** pour revenir avec lui vers la rive B, car ils se retrouveraient seuls dans la barque, ce qui est interdit...

Combien de trajets sont nécessaires, au minimum, pour faire traverser la rivière à tout le monde ?

Q4(a) [1 pt] **un nombre**

. . . . . .

Dans le problème que nous avons décrit jusqu'à présent, on cherche de passer d'une *situation* initiale, où tout le monde (ainsi que la barque) se trouve sur la rive A, à une situation *finale*, où tout le monde (ainsi que la barque) se trouve sur la rive B. Pour aller d'une situation à l'autre, il faut clairement passer par des *situations intermédiaires*, où certains membres du groupe sont sur une des rives, les autres sur la rive opposée, et la barque amarrée d'un des deux côtés.

Ces situations intermédiaires doivent respecter le tableau ci-dessus. Les trajets qui permettent de passer d'une situation à l'autre devront aussi respecter cette règle. Quand une situation ou un trajet respecte le tableau, nous dirons que cette situation ou ce trajet est *acceptable*. Clairement, une solution ne peut passer que par des situations acceptables et n'utiliser que des trajets en barque acceptables eux aussi.

On peut représenter l'ensemble des situations intermédiaires acceptables et les trajets possibles entre elles de façon graphique. Pour ce faire, on utilisera les abréviations **L**, **M**, **V** et **F** déjà énoncées, et on représentera la barque par ⚓.

Un exemple de situation intermédiaire est alors, **L M / V F** ⚓ qui représente une situation acceptable où **L** et **M** sont sur la rive A, et où **V** et **F** sont sur la rive B, ainsi que la barque.

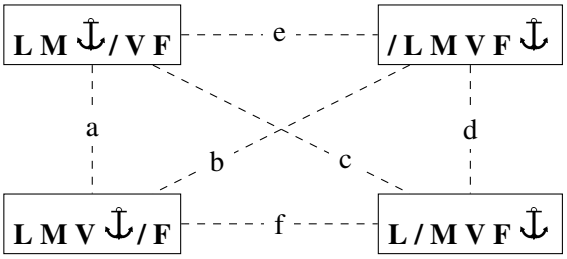
De même, **L M F** ⚓ **/ V** et **M / L V F** ⚓ représentent deux autres situations acceptables.

On représente les mouvements possibles de la barque en traçant un lien entre deux situations s'il est possible de passer de l'une à l'autre (et vice-versa) par un unique trajet *acceptable* en barque.

Par exemple, on tracera : **L M / V F** ⚓ —— **L M F** ⚓ **/ V** car, depuis **L M / V F** ⚓, **V** peut emprunter la barque pour aller de la rive B à la rive A, ce qui donne la situation **L M F** ⚓ **/ V**. Ces deux situations sont bien acceptables, ainsi que le trajet.

Par contre, on ne tracera pas de lien entre **L M / V F** ⚓ et **M / L V F** ⚓ car l'une ne peut pas être obtenue à partir de l'autre en un seul trajet de barque.

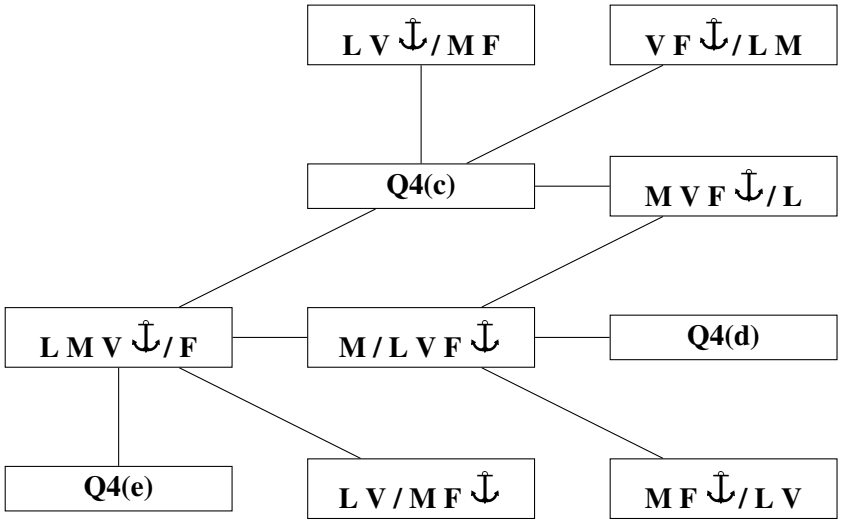
Dans la représentation graphique ci-dessous, on a indiqué 4 situations acceptables, et on tracé en pointillés toutes les possibilités de liens entre deux situations (nommés a, b,. . . , f). Seuls certains de ces liens sont corrects. Lesquels ?



Q4(b) [2 pts] **Zéro, une ou plusieurs lettres**

. . . . . .

Voici un extrait d'une représentation graphique du problème, dont certaines cases ont été laissées vierges (**Q4(c)**, **Q4(d)**, **Q4(e)**). Que faut-il mettre dans ces cases sachant qu'on ne représente que des situations et des trajets acceptables, et que chaque état acceptable ne peut apparaître qu'une seule fois ?



Q4(c) [2pts] **une situation acceptable**

. . . . . .

Q4(d) [2pts] **une situation acceptable**

. . . . . .

Q4(e) [2 pts] **une situation acceptable**

. . . . . .

Question 5 – Exclusion mutuelle (10 min – 11 pts)

Sur les ordinateurs modernes, il est possible *d'exécuter plusieurs programmes en même temps*. En pratique, l'ordinateur ne peut exécuter qu'une seule ligne de programme à la fois, et on s'arrange donc pour qu'il exécute quelques lignes d'un programme, puis quelques lignes d'un autre programme, et ainsi de suite… La rapidité du passage d'un programme à l'autre est imperceptible pour l'utilisateur, ce qui donne l'illusion que les programmes fonctionnent *en même temps*.

Si cette technique permet d'augmenter le confort d'utilisation, elle pose de grosses difficultés quand on utilise des *variables globales*, c'est-à-dire des variables qui peuvent être lues et écrites par plusieurs programmes différents (et qui sont souvent utilisées à des fins de communication entre ces programmes).

Pour illustrer les difficultés, considérons les deux programmes P_1 et P_2 ci-dessous, qui s'exécutent *en même temps*, où `i` est une variable globale (aussi bien P_1 que P_2 peuvent y lire et écrire), alors que `j` et `k` ne peuvent être lues et écrites respectivement que par P_1 et P_2 (remarquez que nous avons ajouté des numéros de ligne pour assurer une meilleure lisibilité). Clairement, le but de chacun des deux programmes est d'ajouter 1 à la variable globale `i`. En supposant que `i` vaut 0 initialement, on souhaiterait donc que la valeur de `i` soit `2` à la fin de l'exécution des deux programmes.

Initialement : `i` $= 0$.

P_1 :

```
1  j ← i
2  j ← j+1
3  i ← j
```

P_2 :

```
4  k ← i
5  k ← k+1
6  i ← k
```

Sur cet exemple, le problème vient du fait qu'on ne sait pas *dans quel ordre les instructions des deux programmes vont être exécutées*. Clairement, l'instruction 1 du programme P_1 sera exécutée avant son instruction 2. Mais nous ne pouvons pas prévoir si l'instruction 1 de P_1 sera exécutée *avant* ou *après* l'instruction 4 de P_2, par exemple.

Malheureusement, cet ordre peut avoir une influence sur le résultat de l'exécution des programmes. Si l'ordinateur exécute les instructions dans l'ordre 1, 2, 3, 4, 5, 6, la valeur de `i` sera 2. Par contre, si l'ordre d'exécution est 1, 4, 2, 3, 5, 6, la valeur finale dans `i` sera 1 !

En supposant toujours que la valeur initiale de `i` est 0, quelle sera sa valeur après avoir exécuté les lignes dans l'ordre : 4, 5, 6, 1, 2, 3 ?

Q5(a) [1 pts] **une valeur**

. . . . . .

Même question pour l'ordre : 4, 5, 1, 6, 2, 3 ?

Q5(b) [1 pts] **une valeur**

. . . . . .

Afin d'éviter ces problèmes, on suggère d'ajouter des variables globales `b1`, `b2` et `tour` qui permettent aux programmes de se coordonner, pour éviter d'alterner entre les instructions des deux programmes. Les variables `b1` et `b2` valent initialement 0, et la variable `tour` vaut initialement 1. L'instruction `ne_rien_faire` est une instruction qui n'a pas d'effet :

Initialement : `tour` $= 1$, `b1` $=$ `b2` $= 0$, `i` $= 1$.

P_1 :

```
1  b1 ← 1
2  while ([...]) // (w)
3  {
4    b1 ← 0
5    while ([...]) // (x)
6    {
7      ne_rien_faire
8    }
9    b1 ← 1
10 }
11 j ← i
12 j ← j+1
13 i ← j
14 tour ← 2
15 b1 ← 0
```

P_2 :

```
16 b2 ← 1
17 while ([...]) // (y)
18 {
19   b2 ← 0
20   while ([...]) // (z)
21   {
22     ne_rien_faire
23   }
24   b2 ← 1
25 }
26 k ← i
27 k ← k+1
28 i ← k
29 tour ← 1
30 b2 ← 0
```

Les conditions des **`while`** (w, x, y et z) doivent permettre d'interdire les exécutions jugées problématiques. Supposons, cette fois, que `i` vaut 1 initialement. Pour chacune des propositions suivantes, on vous demande d'indiquer si les conditions données permettent de garantir que, **dans tous les cas, la variable `i` contiendra la valeur 3 à l'issue de l'exécution complète des deux programmes** :

	oui	non	`(w)`	`(x)`	`(y)`	`(z)`
Q5(c) [3 pts]	☐	☐	`b1`$= 1$	`tour`$\neq 1$	`b2`$= 1$	`tour`$\neq 2$.
Q5(d) [3 pts]	☐	☐	`tour`$\neq 1$	`b2`$= 1$	`tour`$\neq 2$	`b1`$= 1$
Q5(e) [3 pts]	☐	☐	`b2`$= 1$	`tour`$\neq 1$	`b1`$= 1$	`tour`$\neq 2$

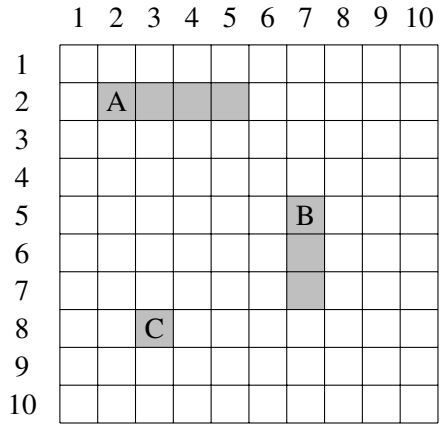
Question 6 – Bataille navale (20 min – 7 + 2 pts)

Le jeu de *bataille navale* se joue sur une grille de 10×10 cases. Sur cette grille, on place des *bateaux*. Un bateau *horizontal* se présente comme une série de cases consécutives sur la même ligne. Un bateau *vertical* est constitué d'une série de cases consécutives sur la même colonne. Un bateau horizontal a donc une hauteur d'une ligne, et un bateau vertical a une largeur d'une colonne.

Sur l'image ci-dessous, A est un bateau horizontal, B est un bateau vertical, et C est un bateau aussi bien horizontal que vertical. Pour indiquer un bateau sur la grille, nous utilisons quatre attributs :

- Le numéro de ligne r de la case supérieure gauche du bateau.
- Le numéro de colonne c de la case supérieure gauche du bateau.
- La longueur ℓ du bateau, c'est-à-dire le nombre de cases dont le bateau est constitué.
- La direction hor du bateau : **`true`** si le bateau est horizontal, **`false`** s'il est vertical.

(Les numéros de lignes et de colonnes commencent à 1)



Les bateaux sur l'image ci-contre ont donc les attributs suivants :

bateau	r	c	ℓ	hor
A	2	2	4	**`true`**
B	5	7	3	**`false`**
C	8	3	1	**`true`**†

† **`false`** est possible également

Aidez-nous à écrire deux fonctions dont nous avons besoin pour programmer le jeu *bataille navale*.

La première fonction '`estTouché`' doit permettre de savoir si une case donnée (de numéro de ligne rr et de numéro de colonne cc) fait partie d'un bateau dont les attributs sont r, c, ℓ, hor. (Dans la terminologie du jeu : savoir si on a *touché* un bateau si on *tire* dans la case de coordonnées rr, cc.)

Cela n'est pas difficile à programmer en parcourant toutes les cases d'un bateau, mais nous voulons obtenir une version plus efficace, *sans* boucle.

Remplissez les parties manquantes dans ce fragment :

```
function estTouché (rr, cc, r, c, ℓ, hor)
{
   if (not hor)
   {
       return [...]       // (a)
   }
   else
   {
       return [...]       // (b)
   }
}
```

Q6(a) [1pt] **une expression logique**

. . . . . .

. . . . . .

Q6(b) [1pt] **une expression logique**

. . . . . .

. . . . . .

Remarque. Une expression *logique* a comme valeur uniquement **`true`** ou **`false`**. Les expression logiques sont des comparaisons (à l'aide de $<$, $>$, $\leq$, $\geq$, $=$ ou $\neq$) que l'on peut éventuellement combiner à l'aide de **`and`** (et), **`or`** (ou) et **`not`** (non). Utilisez des parenthèses () pour plus de clarté. Quelques exemples : '$c \leq r$ **`and`** $cc \geq rr - 5$' ou '$(c \neq 10$ **`and`** $c > 5)$ **`or`** $r = rr/2$'.

Attention ! Nous n'acceptons que les expressions contenant **moins de 6** comparaisons.

La seconde fonction '`collision`' doit servir à vérifier que deux bateaux ne se chevauchent pas sur le damier. Le premier bateaux a pour attributs $r1$, $c1$, $\ell 1$ et $hor1$, et le second $r2$, $c2$, $\ell 2$ et $hor2$. À nouveau, cela ne pose pas de difficulté si l'on peut utiliser des boucles, mais nous souhaitons cette fois encore une solution sans boucle, et sans appeler la fonction `estTouché` !

Remplissez les deux parties manquantes dans le fragment suivant. (Vous ne devez pas remplir les deux autres parties manquantes).

```
function collision (r1, c1, ℓ1, hor1, r2, c2, ℓ2, hor2)
{
   if (not hor1)
   {
        if (hor2)
        {
            return [...]       // (c)
        }
        else
        {
            return [...]       // ne pas compléter!
        }
   }
   else
   {
        if (hor2)
        {
            return [...]       // (d)
        }
        else
        {
            return [...]       // ne pas compléter!
        }
   }
}
```

Q6(c) [3+1 pts] **une expression logique**

. . . . . .

Q6(d) [2+1 pts] **une expression logique**

. . . . . .

Nous n'acceptons à nouveau que les expressions avec moins de 6 comparaisons. Vous obtenez un **bonus** quand vous trouvez la réponse la plus courte (c'est-à-dire avec le nombre minimum de comparaisons).