be-OI 2025 **Finale - JUNIOR** samedi 22 mars 2025	**Remplissez ce cadre en MAJUSCULES et LISIBLEMENT svp** PRÉNOM : .... NOM : .... ÉCOLE :	O **Réservé**

Finale de l'Olympiade belge d'Informatique 2025 (durée : **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 **cadet**.
   - 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 cette page**.
3. Indiquez **vos réponses** sur les pages prévues à cet effet. Écrivez de façon **bien lisible** à l'aide d'un **bic ou stylo** bleu ou noir.
4. Utilisez un crayon et une gomme lorsque vous travaillez au brouillon sur les feuilles de questions.
5. Vous ne pouvez avoir que de quoi écrire avec vous; les calculatrices, GSM, smartphone, ... sont **interdits**.
6. Vous pouvez toujours demander des feuilles de brouillon supplémentaires à un surveillant.
7. Quand vous avez terminé, **remettez la première page (avec votre nom) et les pages avec les réponses**, vous pouvez conserver les autres pages.
8. Tous les extraits de code de l'énoncé sont en **pseudo-code**. Vous trouverez, sur les pages suivantes, une **description** du pseudo-code que nous utilisons.
9. Si vous devez répondre en code, vous devez utiliser le **pseudo-code** ou un **langage de programmation courant** (Java, C, C++, Pascal, Python, ...). Les erreurs de syntaxe ne sont pas prises en compte pour l'évaluation.

Bonne chance !

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

©2025 Olympiade Belge d'Informatique (beOI) ASBL
Cette oeuvre 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 l'opérateur $\%$. 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 $a\%b = 2$.

Voici un premier exemple de code, dans lequel la variable age reçoit 17.

```
anneeNaissance ← 2008
age ← 2025 − anneeNaissance
```

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 $prix$. Observez les commentaires dans le code.

```
if (age ≥ 18)
{
    prix ← 8 // Ceci est un commentaire.
}
else
{
    prix ← 6 // moins cher !
}
```

Parfois, quand une condition est fausse, on doit en vérifier une autre. Pour cela on peut utiliser **`else if`**, qui revient à exécuter un autre **`if`** à l'intérieur du **`else`** du premier **`if`**. Dans l'exemple suivant, il y a 3 catégories d'âge qui correspondent à 3 prix différents pour le ticket de cinéma.

```
if (age ≥ 18)
{
    prix ← 8 // Prix pour une personne majeure.
}
else if (age ≥ 6)
{
    prix ← 6 // Prix pour un enfant de 6 ans ou plus.
}
else
{
    prix ← 0 // Gratuit pour un enfant de moins de 6 ans.
}
```

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 $tab[0]= 5$, $tab[1]= 9$, $tab[2]= 12$. Le tableau est de taille 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 augmenté 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 jusqu'à ce que $n < 10$).

```
d ← 2
while (n ≥ 10)
{
    n ← n/d
    d ← d+1
}
```

Souvent les algorithmes seront dans un cadre et précédés d'explications. Après **Input**, on définit chacun des arguments (variables) donnés en entrée à l'algorithme. Après **Output**, on définit l'état de certaines variables à la fin de l'exécution de l'algorithme et éventuellement la valeur retournée. Une valeur peut être retournée avec l'instruction **return**. Lorsque cette instruction est exécutée, l'algorithme s'arrête et la valeur donnée est retournée.

Voici un exemple en reprenant le calcul de la somme des éléments d'un tableau.

```
Input  : tab[ ], un tableau de n nombres.
         n, le nombre d'éléments du tableau.
Output : sum, la somme de tous les nombres contenus dans le tableau.

sum ← 0
for (i ← 0 to n−1 step 1)
{
    sum ← sum + tab[i]
}
return sum
```

Remarque: dans ce dernier exemple, la variable i est seulement utilisée comme compteur pour la boucle **for**. Il n'y a donc aucune explication à son sujet, ni dans **Input** ni dans **Output**, et sa valeur n'est pas retournée.

Question 1 – Alcuin et son bateau

Le moine Alcuin de York est l'heureux propriétaire d'une souris, d'un chat et d'un gros morceau de fromage. Il doit faire traverser la rivière à tout cela, mais son bateau est trop petit. Il ne peut transporter qu'un passager à la fois: soit le chat, soit la souris, soit le fromage. Malheureusement, si la souris est laissée seule sur la berge avec le fromage, elle le mangera (si Alcuin est présent sur la berge, il peut surveiller la souris). De même, le chat ne peut pas être laissé seul avec la souris, sans quoi il la croquera. Ces deux situations sont appelées des conflits. Il n'y a pas de conflit entre le chat et le fromage, ils peuvent rester ensemble sans surveillance.

Alcuin veut faire passer le chat, la souris, le fromage et lui-même, de la rive A à la rive B de la rivière. Il doit faire des allers-retours entre les deux rives, en commençant sur la rive A. Il explore plusieurs scénarios. Voici le premier, où il indique la direction de chaque étape et ce qu'il transporte avec lui dans son bateau: le chat ( ), la souris ( ), le fromage ( ), ou bien rien du tout (rien).

Scénario 1

1. rive A → rive B:
2. rive B → rive A: rien
3. rive A → rive B:
4. rive B → rive A: rien
5. rive A → rive B:

Afin de vérifier si son scénario est valide, Alcuin remplit un tableau pour indiquer sur quelle rive (A ou B) se trouvent Alcuin, le chat, la souris et le fromage lors de chaque étape. Le tableau commence à l'étape '0' qui représente la situation initiale (avant le premier transport).
Les deux premières lignes du tableau sont déjà remplies. Vous devez le compléter et tracer une croix dans la dernière colonne s'il y a un conflit à l'étape correspondante.

Q1(a) /5	**Complétez le tableau d'Alcuin pour le scénario 1.**

	Alcuin				conflit?
étape 0 :	A	A	A	A	
étape 1 :	B	A	B	A	
étape 2 :	A	A	B	A	
étape 3 :	B	A	B	B	
étape 4 :	A	A	B	B	**X**
étape 5 :	B	B	B	B	

Alcuin envisage ensuite un nouveau scénario. On vous demande à nouveau de compléter le tableau.

Scénario 2

1. rive A → rive B:
2. rive B → rive A: rien
3. rive A → rive B:
4. rive B → rive A:
5. rive A → rive B:
6. rive B → rive A: rien
7. rive A → rive B:

Q1(b) /7	**Complétez le tableau d'Alcuin pour le scénario 2.**

	Alcuin				conflit?
étape 0 :	A	A	A	A	
étape 1 :	B	A	B	A	
étape 2 :	A	A	B	A	
étape 3 :	B	A	B	B	
étape 4 :	A	A	A	B	
étape 5 :	B	B	A	B	
étape 6 :	A	B	A	B	
étape 7 :	B	B	B	B	

Alcuin hérite ensuite d'un bateau plus grand, qui lui permet de transporter deux passagers en plus de lui-même. On dit qu'il a un bateau de taille 2 (avant, il avait un bateau de taille 1).

Q1(c) /3	**Donnez un scénario en trois étapes permettant de faire passer le chat, la souris et le fromage de la rive A à la rive B, avec un bateau de taille 2.** **Pour chaque étape, vous devez nommer 0, 1 ou 2 passagers transportés par Alcuin.**

1. rive A → rive B :
2. rive B → rive A : rien
3. rive A → rive B : ,

Il y a plusieurs autres solutions.

Supposons maintenant que le chat décide d'aimer le fromage: il y a un **nouveau conflit**, on ne peut plus laisser le chat et le fromage ensemble sans surveillance.

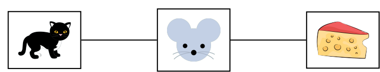
Q1(d) /3	Donnez un scénario en 3 étapes avec un bateau de taille 2. Vous devez tenir compte du nouveau conflit entre le chat et le fromage. La souris doit traverser lors de la première étape. Pour chaque étape, vous devez nommer 0, 1 ou 2 passagers transportés par Alcuin.

1. rive A → rive B :
2. rive B → rive A :
3. rive A → rive B : ,

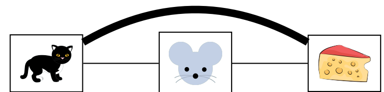
Il y a plusieurs autres solutions.

Fort de son expérience, Alcuin décide de quitter la vie monastique et de fonder une entreprise spécialisée dans les traversées compliquées de la rivière. Ses clients lui remettent n passagers (objets ou animaux) qui doivent traverser la rivière, ainsi qu'un diagramme indiquant ceux qui sont en conflit. Ce diagramme est appelé un 'graphe' et est composé de rectangles étiquetés, appelés 'nœuds', un pour chaque passager. Il contient aussi des liens entre les nœuds, appelés 'arêtes' : on trace une arête entre deux nœuds si les deux passagers correspondants sont en conflit. Par contre si deux passagers peuvent rester ensemble sans surveillance, on ne met pas d'arête entre les nœuds correspondants.

Voici le graphe de la situation initiale quand le chat ne mangeait pas de fromage. Il y avait un conflit entre la souris et le chat, un conflit entre la souris et le fromage, mais pas de conflit entre le chat et le fromage.



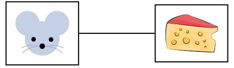
Q1(e) /1	Modifiez ce graphe (en dessinant ou barrant des arêtes) pour ajouter un conflit entre le chat et le fromage.



Alcuin veut maintenant connaître la taille du bateau dont il a besoin pour faire traverser un ensemble de passagers dont on donne le graphe de conflits. Naturellement, pour transporter n passagers, un bateau de taille n est toujours suffisant. Mais Alcuin veut utiliser le plus petit bateau possible.

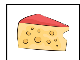
Il tient le raisonnement suivant:
“ *Je dois répartir les passagers en 2 groupes. Ceux du premier groupe doivent rester en permanence sous surveillance dans le bateau. Les autres peuvent rester sans surveillance car il n'y a aucun conflit entre-eux. Les passagers à surveiller doivent être tels que si j'enlève du graphe tous les nœuds qui leur correspondent et les arêtes qui les touchent, il ne doit plus rester aucune arête dans le graphe.*”.

Par exemple, retirer le nœud ’chat’ n'est pas une bonne solution, car il reste un conflit entre la souris et le fromage.



Autrement dit, on ne peut pas garder le chat seul dans le bateau et laisser la souris et le fromage sur la berge.

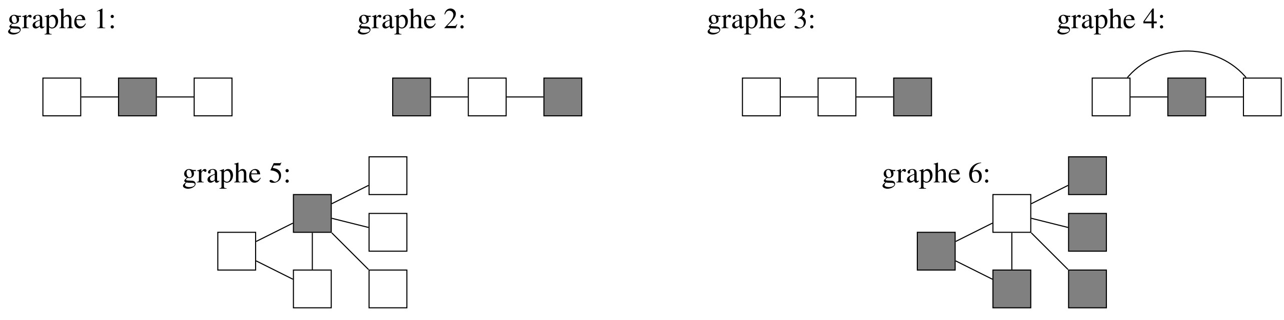
Par contre, à l'époque où le chat n'aimait pas le fromage, retirer la souris donne le graphe suivant qui ne contient aucun conflit:



Autrement dit, on pouvait (quand le chat n'aimait pas le fromage) charger la souris seule dans le bateau et laisser le chat et le fromage sur la berge.

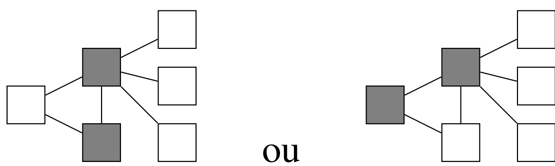
Ce qu'Alcuin veut calculer s'appelle une *couverture*: **un ensemble de nœuds tels que, si on supprime du graphe ces nœuds et les arêtes qui les touchent, il ne reste plus aucune arête**.

Voici six graphes, dont certains nœuds ont été coloriés en gris :



	Oui	Non	Indiquez si les nœuds gris forment oui ou non une couverture dans chaque graphe.
Q1(f) /1	☒	☐	Les nœuds gris du graphe 1 forment une couverture.
Q1(g) /1	☒	☐	Les nœuds gris du graphe 2 forment une couverture.
Q1(h) /1	☐	☒	Les nœuds gris du graphe 3 forment une couverture.
Q1(i) /1	☐	☒	Les nœuds gris du graphe 4 forment une couverture.
Q1(j) /1	☐	☒	Les nœuds gris du graphe 5 forment une couverture.
Q1(k) /1	☒	☐	Les nœuds gris du graphe 6 forment une couverture.

Q1(l) /4	**Coloriez des nœuds du graphe ci-dessous. Les nœuds coloriés doivent former une couverture contenant un nombre minimal de nœuds.**

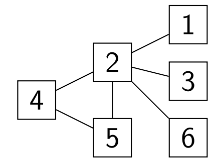


Alcuin veut disposer d'un algorithme qui calcule une couverture d'un graphe. Après en avoir longuement discuté avec le chat, il trouve la stratégie suivante (qui n'est pas forcément très intelligente, mais c'est l'idée du chat).
Les nœuds sont numérotés de 1 à n et représentés par leur numéro dans l'algorithme.
S'il y a une arête entre les nœuds de numéros i et j, on la note (i, j).
L'algorithme considère successivement toutes les arêtes (i, j) du graphe et 'traite' ses extrémités, c'est-à-dire les nœuds de numéros i et j.
Traiter un nœud consiste à :

1. mettre le nœud dans la couverture,
2. enlever du graphe le nœud et toutes les arêtes dont il est une extrémité.

On continue jusqu'à ce qu'il ne reste plus aucune arête dans le graphe.

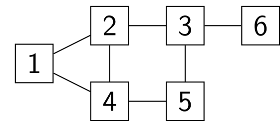
Alcuin observe qu'il n'y a pas de direction aux arêtes, par exemple l'arête $(3, 1)$ est la même que $(1, 3)$.
Il peut donc se contenter d'examiner les arêtes (i, j) où $i < j$ (pour éviter de traiter 2 fois la même arête).
Alcuin décide de traiter les arêtes dans l'ordre suivant (si elles existent):
d'abord les arêtes $(1, 2)$, $(1, 3)$,... $(1, n)$; puis les arêtes $(2, 3)$, $(2, 4)$,... $(2, n)$; puis les arêtes $(3, 4)$, $(3, 5)$,... $(3, n)$; et ainsi de suite jusqu'à la dernière arrête $(n - 1, n)$.

Q1(m) /3	**Dessinez le graphe obtenu au début de l'algorithme après avoir considéré uniquement la première arête $(1, 2)$ et donc après avoir traité les nœuds 1 et 2 du graphe ci-dessous.** 

Solution:

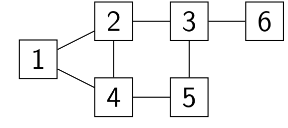
Q1(n) /4	**Quelle est la couverture calculée après exécution complète de l'algorithme sur le graphe de la question précédente ? Vous devez donner la liste des nœuds dans la couverture calculée.**

Solution : $\{1, 2, 4, 5\}$

Q1(o) /4	**Quelle est la couverture calculée par l'algorithme sur le graphe ci-dessous ? Vous devez donner la liste des nœuds dans la couverture calculée.** 

Solution : $\{1, 2, 3, 5\}$

Pour implémenter l'algorithme, Alcuin utilise un tableau `G` de taille $n \times n$ pour stocker les arêtes.
Comme les arêtes n'ont pas de direction, il considère uniquement les arêtes (i, j) avec $i < j$.
Le tableau contient **`true`** dans la case `G[i][j]` (avec $i < j$) si et seulement si il y a une arête (i, j) dans le graphe.
Les autres cases du tableau contiennent **`false`**.

Q1(p) /4	**Donnez le tableau `G` qui correspond au graphe ci-dessous, en considérant que la case `G[i][j]` se trouve à la rangée `i`, colonne `j`. Écrivez un T dans les cases qui contiennent `true`.** 

	1	2	3	4	5	6
1		T		T		
2			T	T		
3					T	T
4					T	
5						
6						

Alcuin a écrit son algorithme sur un morceau de parchemin mais malheureusement la souris en a grignoté certaines parties. Alcuin a besoin de votre aide pour retrouver les parties manquantes.

L'algorithme commence par deux fonctions :

- `InitCountEdges` compte le nombre d'arêtes du graphe et stocke ce nombre dans la variable `countEdges`,
- `ProcessNode(i)` traite le nœud `i` (comme décrit ci-dessus).

En plus du tableau `G[][]`, on utilise un tableau de Booléens `InCover[]` de taille `n` dont tous les éléments sont initialisés à **`false`**. L'algorithme signale que le nœud `i` fait partie de la couverture en modifiant la valeur de `InCover[i]` qui passe à **`true`**.

Q1(q) /4	**Complétez les ____ dans la fonction `InitCountEdges` (réponses aussi courtes que possible).**

```
function InitCountEdges()
{
  CountEdges = 0
  for (i ← 1 to n-1 step 1)
  {
    for (j ← i+1 to n step 1)
    {
      if (G[i][j]) CountEdges ← CountEdges + 1
    }
  }
}
```

Q1(r) /6	**Complétez les ____ dans la fonction `ProcessNode` (réponses aussi courtes que possible).**

```
function ProcessNode(i)
{
  InCover[i] ← true
  for (k ← 1 to i-1 step 1)
  {
    if (G[k][i])
    {
      G[k][i] ← false
      CountEdges ← CountEdges - 1
    }
  }
  for (k ← i+1 to n step 1)
  {
    if (G[i][k])
    {
      G[i][k] ← false
      CountEdges ← CountEdges - 1
    }
  }
}
```

L'algorithme est complété par la fonction `Cover`.
Cette fonction reçoit un tableau `G[][]` de taille `n` × `n` qui représente les arêtes existantes entre les *n* nœuds.
Le tableau `InCover[]` (de taille `n`) est initialisé à **`false`** comme expliqué plus haut avant d'exécuter la fonction `Cover`.
La couverture calculée est composée de tous les nœuds `i` tels que `InCover[i]` vaut **`true`** après l'exécution de la fonction `Cover`.

Q1(s) /6	Complétez les ▭ dans la fonction **`Cover`** (réponses aussi courtes que possible).

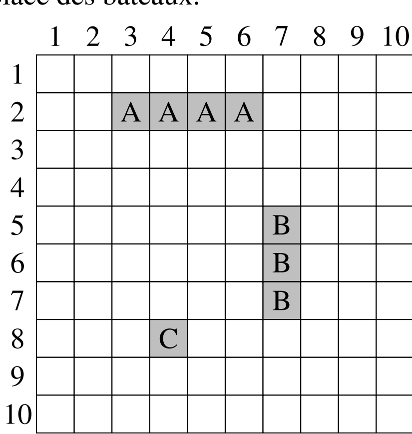
```
function Cover()
{
  InitCountEdges()
  i ← 1
  j ← 2
  while (CountEdges != 0)
  {
    if (G[i][j])
    {
      ProcessNode(i)
      ProcessNode(j)
    }
    j ← j+1
    if (j = n+1){
      i ← i+1

      j ← i+1
    }
  }
}
```

Question 2 – Combat naval

On veut programmer le jeu de combat naval qui se joue sur une grille 10×10 où on place des bateaux.

- Il y a 10 rangées numérotées de 1 à 10 de haut en bas.
- Il y a 10 colonnes numérotées de 1 à 10 de gauche à droite.
- Un bateau horizontal occupe des cases consécutives sur la même rangée.
- Un bateau vertical occupe des cases consécutives sur la même colonne.
- On note `(r,c)` les coordonnées de la case à l'intersection de la rangée `r` et de la colonne `c`.



Exemple :

– 3 bateaux **A**, **B** et **C** sont dessinés en gris sur l'image.
– **A** est horizontal, **B** est vertical, **C** est aussi bien horizontal que vertical.
– **B** occupe les cases de coordonnées `(5,7)`, `(6,7)` et `(7,7)`.

On hésite entre 2 systèmes pour représenter les bateaux dans notre programme.
Chaque système utilise 4 attributs pour indiquer la position d'un bateau sur la grille.

Attributs dans le système **Orientation**.
- La rangée `r` de la case supérieure gauche du bateau.
- La colonne `c` de la case supérieure gauche du bateau.
- La longueur `L` du bateau.
- La direction `hor` du bateau :
  `true` s'il est horizontal, **`false`** s'il est vertical.

Attributs **Orientation** des bateaux sur l'image.

Bateau	`r`	`c`	`L`	`hor`
A	2	3	4	**`true`**
B	5	7	3	**`false`**
C	8	4	1	**`true`** ou **`false`** au choix

Attributs dans le système **Rectangle**.
- La rangée `r1` de la case supérieure gauche du bateau.
- La colonne `c1` de la case supérieure gauche du bateau.
- La rangée `r2` de la case inférieure droite du bateau.
- La colonne `c2` de la case inférieure droite du bateau.

Attributs **Rectangle** des bateaux sur l'image.

Bateau	`r1`	`c1`	`r2`	`c2`
A	2	3	2	6
B	5	7	7	7
C	8	4	8	4

On va écrire quelques fonctions dans les 2 systèmes afin de décider lequel choisir pour notre programme.

Conversions entre les 2 systèmes

Entraînons-nous d'abord à passer d'un système à l'autre.

Orientation $\longrightarrow$ Rectangle

Un bateau est décrit par ses 4 attributs `(r,c,L,hor)` dans le système **Orientation**.
Donnez ses attributs `(r1,c1,r2,c2)` dans le système **Rectangle**.

Q2(a) /1	**`(4, 5, 2, True)`** $\longrightarrow$ **`( 4 , 5 , 4 , 6 )`**
Q2(b) /1	**`(3, 6, 3, False)`** $\longrightarrow$ **`( 3 , 6 , 5 , 6 )`**
Q2(c) /1	**`(7, 5, 1, True)`** $\longrightarrow$ **`( 7 , 5 , 7 , 5 )`**
Q2(d) /1	**`(1, 9, 5, False)`** $\longrightarrow$ **`( 1 , 9 , 5 , 9)`**

Rectangle ⟶ Orientation
Un bateau est décrit par ses 4 attributs `(r1,c1,r2,c2)` dans le système **Rectangle**.
Donnez ses attributs `(r,c,L,hor)` dans le système **Orientation**.

Q2(e) /1	`(2, 7, 5, 7)` ⟶ `( 2 , 7 , 4 , False )`

Q2(f) /1	`(4, 3, 4, 3)` ⟶ `( 4 , 3 , 1 , True )`

Q2(g) /1	`(8, 3, 8, 6)` ⟶ `( 8 , 3 , 4 , True )`

Q2(h) /1	`(1, 4, 1, 10)` ⟶ `( 1 , 4 , 7 , True )`

Créons des fonctions pour automatiser ces conversions.

Fonction O2R
La fonction `O2R` convertit des attributs du système **Orientation** en attributs équivalents du système **Rectangle**.
Par exemple, en utilisant le bateau **A** de l'image, `O2R(2,3,4,true)` retourne le résultat `(2,3,2,6)`.

Q2(i) /5	**Complétez les ____ dans la fonction `O2R`.**

```
function O2R(r,c,L,hor)
{
  if (hor)
    { return ( r , c , r , c+L-1 ) }
  else
    { return ( r , c , r+L-1 , c ) }
}
```

Fonction R2O
La fonction `R2O` convertit des attributs du système **Rectangle** en attributs équivalents du système **Orientation**.
Il faut utiliser une expression logique (voir page suivante) pour calculer la valeur de `hor`.
Par exemple, en utilisant le bateau **A** de l'image, `R2O(2,3,2,6)` retourne le résultat `(2,3,4,true)`.

Q2(j) /5	**Complétez les ____ dans la fonction `R2O`.**

```
function R2O(r1,c1,r2,c2)
{
  L ← (r2-r1) + (c2-c1) + 1

  hor ← (r1==r2)

  return ( r1 , c1 , L, hor)
}
```

Tests de validité

Le programme doit vérifier que les attributs décrivent un bateau qu'on peut placer dans la grille 10x10.
Il le fait en évaluant des expressions logiques, dont le résultat est soit **`true`** soit **`false`**.
Ce type d'expression utilise les opérateurs de comparaison (`==`, `!=`, `>`, `>=`, `<`, `<=`) et les opérateurs logiques (**`and`**, **`or`**, **`not`**) présentés dans les exemples ci-dessous.

Expression logique	Valeur
`c==4`	**`true`** si `c` est égal à 4, **`false`** sinon.
`r!=5`	**`true`** si `r` est différent de 5.
`r1>3`	**`true`** si `r1` est plus grand que 3.
`c2>=c1`	**`true`** si `c2` est plus grand ou égal à `c1`.
`r+L<9`	**`true`** si `r+L` est plus petit que `9`.
`c2<=c1+2`	**`true`** si `c2` est plus petit ou égal à `c1+2`.
`(c<2) and (r>3)`	**`true`** si les 2 inégalités sont vraies.
`(c1==3) or (r1==2)`	**`true`** si au moins une des 2 égalités est vraie.
`not(hor)`	**`true`** si `hor` est égal à **`false`** (et **`false`** si `hor` est égal à **`true`**).

Fonction Rok

Dans le système **Rectangle**, il faut que les cases `(r1,c1)` et `(r2,c2)` soient dans la même rangée ou dans la même colonne, que toutes les coordonnées soient entre 1 et 10 et que la première case soit au-dessus ou à gauche de la deuxième.
Complétez la fonction `Rok` qui retourne **`true`** si les attributs `(r1,c1,r2,c2)` vérifient toutes ces conditions.
Comme l'expression est très longue, elle est évaluée en plusieurs étapes en utilisant la variable logique `ok`.

Q2(k) /5	Complétez les ____ dans la fonction **`Rok`**.

```
function Rok(r1, c1, r2, c2)
{
  ok ← ( r1==r2 ) or ( c1==c2 )
  ok ← ok and (1<=r1 and r1<=r2 and r2<=10 )
  ok ← ok and (1<=c1 and c1<=c2 and c2<=10 )
  return ok
}
```

Fonction Ook

Dans le système **Orientation**, la longueur doit être supérieure ou égale à 1.
Complétez la fonction `Ook` qui retourne **`true`** si le bateau `(r,c,L,hor)` peut être placé sur la grille 10×10.

Q2(l) /5	Complétez les ____ dans la fonction **`Ook`**.

```
function Ook(r,c,L,hor)
{
  ok ← ( 1<=L and 1<=r and 1<=c)
  if (hor)
    { return (ok and r<=10 and c+L-1<=10 ) }
  else
    { return (ok and r+L-1<=10 and c<=10 ) }
}
```

Bateau touché

Le programme doit vérifier si un bateau est touché quand on tire sur une case dont on donne les coordonnées.
C'est le cas si le bateau occupe la case visée.

Dans les questions suivantes, essayez de trouver les réponses les plus courtes et simples possibles en utilisant un minimum de comparaisons et d'opérateurs logiques.

Fonction hitR (système Rectangle)

La fonction `hitR(rr,cc, r1,c1,r2,c2)` retourne **`true`** si le bateau d'attributs `(r1,c1,r2,c2)` occupe la case de coordonnées `(rr,cc)` et retourne **`false`** si ce n'est pas le cas.

Avec la situation décrite sur l'image:

- `hitR(2,5, 2,3,2,6)` retourne **`true`** car le bateau **A** occupe la case de coordonnées `(2,5)`.
- `hitR(7,6, 5,7,7,7)` retourne **`false`** car le bateau **B** n'occupe pas la case de coordonnées `(7,6)`.

Q2(m) /6	**Complétez les ____ dans la fonction `hitR` (réponse aussi courte que possible).**

```
function hitR(rr,cc, r1,c1,r2,c2)
{
  return (r1<=rr) and (rr<=r2) and (c1<=cc) and (cc<=c2)
}
```

Fonction hitO (système Orientation)

La fonction `hitO(rr,cc, r,c,L,hor)` retourne **`true`** si le bateau d'attributs `(r,c,L,hor)` occupe la case de coordonnées `(rr,cc)` et retourne **`false`** si ce n'est pas le cas.

Avec la situation décrite sur l'image:

- `hitO(2,5, 2,3,4,true)` retourne **`true`** car le bateau **A** occupe la case de coordonnées `(2,5)`.
- `hitO(7,6, 5,7,3,false)` retourne **`false`** car le bateau **B** n'occupe pas la case de coordonnées `(7,6)`.

Q2(n) /6	**Complétez les ____ dans la fonction `hitO` (réponses aussi courtes que possible).**

```
function hitO(rr,cc, r,c,L,hor)
{
  if (hor)
    { return (rr==r) and (c<=cc) and (cc<c+L) }
  else
    { return (cc==c) and (r<=rr) and (rr<r+L) }
}
```

Collision

Deux bateaux ne peuvent pas occuper une même case sur la grille, sinon il y a collision.

Fonction collisionR (système Rectangle)
La fonction `collisionR` retourne **`true`** si deux bateaux sont en collision et **`false`** si ce n'est pas le cas.
Les attributs des bateaux sont `(r1A,c1A,r2A,c2A)` et `(r1B,c1B,r2B,c2B)`.
Il faut utiliser des expressions logiques aussi courtes que possible. C'est malgré tout très long et l'expression est évaluée en plusieurs étapes en utilisant les variables logiques `rbool` et `cbool`.

Q2(o) /7	Complétez les ____ dans la fonction **`collisionR`** (réponses aussi courtes que possible).

```
function collisionR(r1A,c1A,r2A,c2A, r1B,c1B,r2B,c2B)
{
    rbool ← (r1A<=r1B and r1B<=r2A) or (r1B<=r1A and r1A<=r2B)

    cbool ← (c1A<=c1B and c1B<=c2A) or (c1B<=c1A and c1A<=c2B)
    return rbool and cbool
}
```

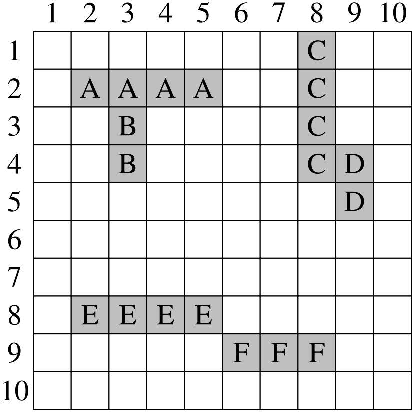
Fonction collisionO (système Orientation)
La fonction `collisionO` retourne **`true`** si deux bateaux sont en collision et **`false`** si ce n'est pas le cas.
Les attributs des bateaux sont `(rA,cA,LA,horA)` et `(rB,cB,LB,horB)`.
Pour éviter des expressions logiques encore plus longues, on utilise la fonction `hitO` définie plus haut.

Q2(p) /7	Complétez les ____ dans la fonction **`collisionO`** en utilisant la fonction **`hitO`**.

```
function collisionO(rA,cA,LA,horA, rB,cB,LB,horB)
{
  if (horA==horB)
    { return hitO(rB,cB, rA,cA,LA,horA) ) or hitO(rA,cA, rB,cB,LB,horB ) }
  else
  {
    if (horA)
      { return hitO(rA,cB, rA,cA,LA,horA ) and hitO(rA,cB, rB,cB,LB,horB ) }
    else
      { return hitO(rB,cA, rA,cA,LA,horA ) and hitO(rB,cA, rB,cB,LB,horB ) }
  }
}
```

Risque de collision

Les bateaux ne peuvent pas être trop proches.
Plus précisément, deux cases occupées par deux bateaux différents ne peuvent pas avoir un côté ou un coin en commun.
Par exemple, tous les bateaux dans l'image ci-dessous sont trop proches d'un autre bateau.
A et **B** se touchent, **C** et **D** se touchent, **E** et **F** se touchent par un coin.



	1	2	3	4	5	6	7	8	9	10
1								C		
2		A	A	A	A			C		
3			B					C		
4			B					C	D	
5									D	
6										
7										
8		E	E	E	E					
9						F	F	F		
10										

Fonction riskR (système Rectangle)

La fonction `riskR` retourne **`true`** si deux bateaux sont trop proches et **`false`** si ce n'est pas le cas.
Les attributs des bateaux sont `(r1A,c1A,r2A,c2A)` et `(r1B,c1B,r2B,c2B)`.
Le plus simple est d'utiliser la fonction `collisionR` définie plus haut.

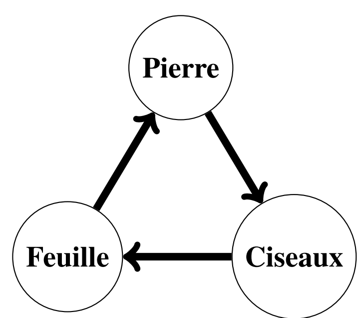
Q2(q) /6	Complétez les ____ dans la fonction **`riskR`** en utilisant la fonction **`collisionR`**.

```
function riskR(r1A,c1A,r2A,c2A,r1B,c1B,r2B,c2B)
{
  return collisionR( r1A,c1A,r2A,c2A,  r1B-1,c1B-1,r2B+1,c2B+1 )
}
```

Fonction riskO (système Orientation)

Cette fonction est plus compliquée à écrire.
On n'insiste pas et on décide d'utiliser le système Rectangle dans notre programme.

Question 3 – Pierre-Ciseaux-Feuille



Dans un célèbre jeu, les 2 joueurs miment simultanément une arme avec leur main : **Pierre** ou **Ciseaux** ou **Feuille**.

Le résultat du combat est déterminé grâce au graphe circulaire ci-contre:
Pierre bat **Ciseaux** qui bat **Feuille** qui bat **Pierre**.
Il y a égalité si les joueurs choisissent la même arme.

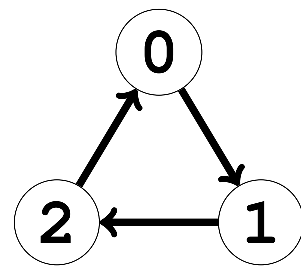
Un fabriquant de jouets veut commercialiser des petits robots capables de jouer entre-eux à **Pierre-Ciseaux-Feuille**.

Deux robots placés à proximité l'un de l'autre communiquent sans fil. Un joueur appuie sur un bouton de son robot pour proposer un match, l'autre joueur appuie sur un bouton de son robot pour accepter le match.
Lors d'un match, les robots jouent 10 parties de **Pierre-Ciseaux-Feuille**, chaque victoire rapporte 1 point au robot vainqueur.
Les parties et les scores peuvent être suivis sur les écrans des robots.
Celui qui a remporté le plus de parties gagne le match.
Il peut y avoir égalité si les deux robots remportent le même nombre de parties.

Les robots ne jouent pas au hasard: chacun suit une **stratégie** déterminée par un programme.
Le fabriquant veut proposer un grand nombre de robots de couleurs, de formes et surtout de stratégies différentes.

Codages

Dans les programmes, **Pierre** est codé par `0`, **Ciseaux** par `1` et **Feuille** par `2`.
Donc `0` bat `1` qui bat `2` qui bat `0` et le graphe du jeu devient celui ci-contre.



Dans la suite, on dira qu'un robot joue le coup `0`, `1` ou `2`.

Stratégies

La stratégie d'un robot est déterminée par 2 éléments.

- La variable `init` qui contient le coup à jouer lors de la première partie.
- Le tableau `strat`, de 3 rangées et 3 colonnes, utilisé pour déterminer les coups des autres parties.
  Si le robot a joué `r` et si son adversaire a joué `s`, alors il va jouer `strat[r][s]` à la partie suivante.

Un exemple de tableau `strat` est donné à droite.
Ses numéros de rangées et de colonnes commencent à `0` et sont notés en gris.

	0	1	2
0	1	2	0
1	2	1	0
2	2	2	1

Avec ce tableau et en fonction des coups de la partie précédente, le robot doit jouer :

- `strat[1][2]=0` si le robot a joué `1` et son adversaire `2`.
- `strat[2][0]=2` si le robot a joué `2` et son adversaire `0`.
- `1` s'il y a eu match-nul car `strat[0][0]=strat[1][1]=strat[2][2]=1`.

Encoder des stratégies

Dans les questions suivantes, on ne s'occupe pas de la variable `init`.
Une stratégie est décrite par un texte expliquant ce que le robot doit jouer en fonction de ce que lui et son adversaire ont joué lors de la partie précédente.
Vous devez encoder le tableau `strat` qui correspond à la stratégie décrite par le texte.
Vous pouvez utiliser les grilles de la page suivante comme brouillon.
N'oubliez pas de **recopier vos solutions sur les feuilles de réponses**.

Q3(a) /4	Remplir le tableau `strat` de la stratégie décrite par cette phrase. "Jouer le coup précédent de l'adversaire."

	0	1	2
0	0	1	2
1	0	1	2
2	0	1	2

Q3(b) /4	Remplir le tableau `strat` de la stratégie décrite par cette phrase. "Jouer le coup qui gagne contre le coup précédent de l'adversaire."

	0	1	2
0	2	0	1
1	2	0	1
2	2	0	1

Q3(c) /4	Remplir le tableau `strat` de la stratégie décrite par cette phrase. "S'il y a eu égalité, jouer le coup précédent du robot; sinon jouer le coup du gagnant précédent."

	0	1	2
0	0	0	2
1	0	1	1
2	2	1	2

Q3(d) /4	Remplir le tableau `strat` de la stratégie décrite par cette phrase. "S'il y a eu égalité, jouer le coup qui gagne contre le coup précédent; sinon rejouer le coup précédent du robot."

	0	1	2
0	2	0	0
1	1	0	1
2	2	2	1

Dans la suite, on utilise les notations décrites ci-dessous.

- `r` : le coup joué par le robot à la partie précédente.
- `s` : le coup joué par l'adversaire à la partie précédente.
- `w` : le coup joué par le gagnant de la partie précédente (ou par les 2 joueurs en cas d'égalité).
- `x` : le coup joué par le perdant de la partie précédente (ou par les 2 joueurs en cas d'égalité).
- `W(c)` : le coup qui gagne contre le coup `c` (par exemple `W(0)=2` car `2` bat `0`).
- `X(c)` : le coup qui perd contre le coup `c` (par exemple `X(0)=1` car `0` bat `1`).

Q3(e) /4	**Remplir le tableau `strat` de la stratégie décrite par cette phrase.** **"S'il y a eu égalité, jouer `X(r)`; sinon jouer le coup qui n'a pas été joué."**

	0	1	2
0	1	2	1
1	2	2	0
2	1	0	0

Q3(f) /4	**Remplir le tableau `strat` de la stratégie décrite par cette phrase.** **"Si `s=0`, jouer `r`; si `s=1` et qu'il n'y a pas eu égalité, jouer `w`; si `s=2` et qu'il n'y a pas eu égalité, jouer `x`; dans les autres cas, jouer `W(r)`."**

	0	1	2
0	0	0	0
1	1	0	2
2	2	1	1

Battre une stratégie

Dans les questions suivantes, une stratégie A est complètement donnée à l'aide de `initA` et du tableau `stratA`.
Trouvez une stratégie B qui gagne avec un score de 10 à 0 dans un match contre la stratégie A.
Vous devez donner le **premier coup `initB`** et **un minimum de valeurs dans le tableau `stratB`**.
Laissez vides les cases de `stratB` qui ne seront jamais utilisées par votre stratégie dans un match contre la stratégie A.
Les numéros de rangées et de colonnes ne sont plus notés, ajoutez-les si vous en ressentez le besoin.

Q3(g) /4	**Donner une stratégie B qui gagne 10 à 0 contre la stratégie A.** **Ne précisez que les valeurs indispensables dans `stratB`.**

`initA=0, stratA=`

2	0	0
1	0	1
2	2	1

⟶ `initB=` 2 `, stratB=`

2		

Q3(h) /4	**Donner une stratégie B qui gagne 10 à 0 contre la stratégie A.** **Ne précisez que les valeurs indispensables dans `stratB`.**

`initA=1, stratA=`

2	0	1
0	1	2
1	2	0

⟶ `initB=` 0 `, stratB=`

	2	
0		

Q3(i) /4	**Donner une stratégie B qui gagne 10 à 0 contre la stratégie A.** **Ne précisez que les valeurs indispensables dans `stratB`.**

`initA=2, stratA=`

2	0	1
2	0	1
2	0	1

⟶ `initB=` 1 `, stratB=`

	1	
		2
0		

Q3(j) /4	**Donner une stratégie B qui gagne 10 à 0 contre la stratégie A.** **Ne précisez que les valeurs indispensables dans `stratB`.**

`initA=1, stratA=`

1	1	2
0	1	0
0	2	2

⟶ `initB=` 0 `, stratB=`

	2	
		1
1		

N'oubliez pas de **recopier vos solutions sur les feuilles de réponses**.