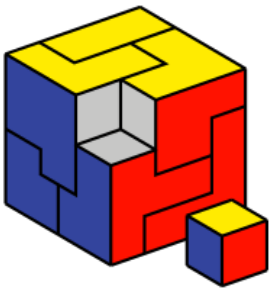
Prologin 2021



Panda Race

Championnat Des Pandas

Sujet de la finale en ligne du Concours National d'Informatique
Vendredi 9 Juillet 2021

Table des matières

1. périodicité de 200 ans


Prologin 2021

1 Contexte

1.1 Tournoi bicentennal [2] des Pandas

Comme vous le savez très certainement, la Chine est un grand pays [3] regorgeant de traditions vieilles comme le monde [4]. L'une d'entre elles est pratiquée, encore à ce jour, par les Pandas Géants et les Pandas Roux. En effet, des observations récentes nous montrent qu'ils pratiquent un rituel, lors duquel la meilleure race de panda est reconnue pour les deux siècles à venir. Par souci d'équité et de justice, le jury n'est pas composé de pandas.

1.2 Le championnat

Le but du championnat est de trouver quelle race compte le meilleur couple de parents [5]. Des bébés pandas seront disposés sur la rivière, dans le grand danger des torrents. Le but des parents est d'aller récupérer tous leurs petits. Le premier couple à avoir récupéré tous les pandas a gagné le match ! [6]

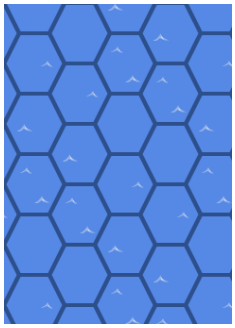
1.3 Champ de bataille

Le champ de bataille sera cette année la rivière de *Hulun*, au Nord-Ouest du pays oriental, hôte de ce championnat depuis toujours : la Chine. La rivière, à l'apparence rectangulaire est en fait composée d'hexagones, à travers lesquels des pandas de toutes sortes [7] devront se mouvoir.

2. périodicité de 200 ans
3. 9,597 millions km²
4. Nous considèrerons l'arrivée des Ailuropoda melanoleuca (Panda) comme la génèse de ce bas-monde
5. On remarquera un grand nombre de pédiatres au sein des deux communautés pandesques
6. Vous jouerez en effet les deux parents, et non pas un seul panda
7. Vous et vos adversaires

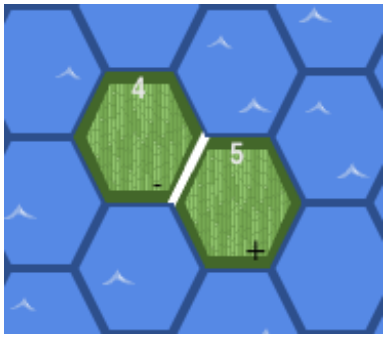
1.3.1 Eau

L'eau est surprenamment l'élément le plus présent dans la rivière. Il est le premier obstacle. Les pandas ne peuvent pas aller sur l'eau. Ils doivent marcher sur les ponts. Les bébés pandas se trouvent uniquement sur des cases d'eau.



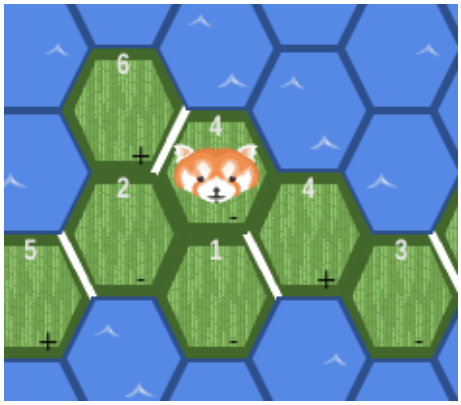
1.3.2 Ponts

Les ponts sont le seul moyen de déplacement des pandas. Les ponts sont constitués de deux cases distinctes : une case "+" et une case "-" (en bas à droite des cases de pont). Ces deux cases sont toujours adjacentes [8] et reliées par une barre blanche dans l'affichage du match. Chaque case de pont a également une hauteur ou valeur : les nombres en blanc en haut des cases.

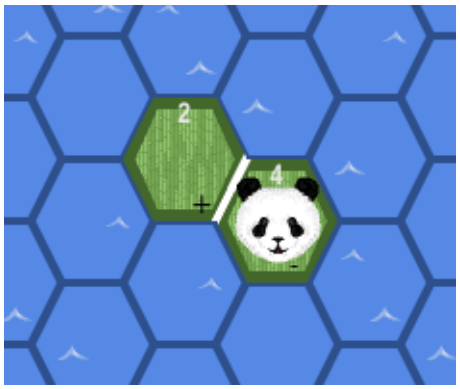


Un panda peut se déplacer sur un même pont (à travers le trait blanc) sans contrainte. Pour passer d'un pont à l'autre, il faut que la hauteur du pont duquel le panda part soit la même que celle du pont sur lequel le panda veut aller. C'est le même fonctionnement que les dominos.

8. Sinon, ça ne fait pas un pont.



Lorsqu'un panda quitte une case d'un pont, la hauteur du pont change. Si c'était une case "+", la hauteur du pont augmente (plus 1 modulo 6). Si la case quittée était une case "-", la hauteur du pont diminue (moins 1 modulo 6). Les valeurs des cases de ponts sont donc toujours incluses entre 1 et 6.



Une fois quelques ponts construits, on peut obtenir ce genre de configuration :

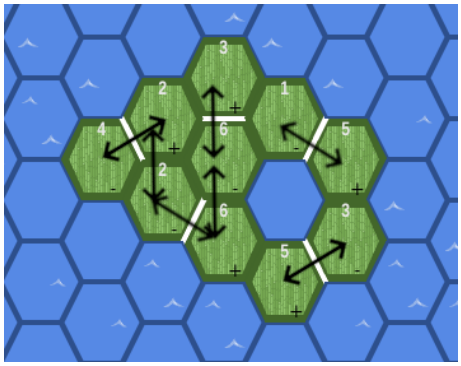


FIGURE 1 : Les flèches montrent les déplacements possibles

1.3.3 Rochers

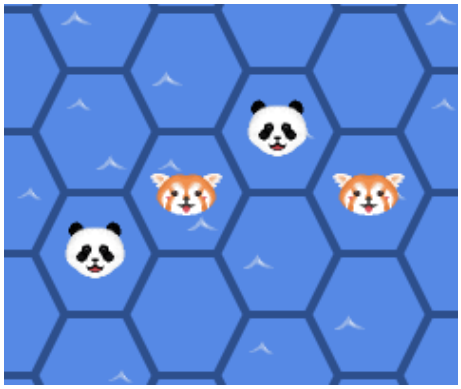
Les rochers sont les obstacles naturels de la rivière. Rien ne peut traverser ni modifier une case de rocher.

1.3.4 Pandas

Les pandas sont ceux dont vous contrôlez le mouvement. Ils font toutes les actions sur la rivière. Ils peuvent poser des ponts, se déplacer et ramasser les bébés (cette dernière action est automatique). Il y les pandas géants (en noir et blanc), et les pandas roux (en blond vénitien). Ils sont toujours sur un pont.

1.3.5 Bébés pandas

Les bébés pandas sont votre objectif ! C'est eux que vous devez sauver et qui vous feront gagner la partie. Ils sont toujours sur une case d'eau et ne peuvent pas se déplacer.



1.4 Déplacements

Les pandas n'étant pas de grands amateurs des étendues d'eau [9], ils préfèrent se déplacer sur des ponts. Ils devront dont construire des ponts en bambou sur la rivière. C'est là qu'ils rencontreront la première grande difficulté : les fonds marins n'ont pas tous la même hauteur et sont très (très) sensibles. Le sable chinois au fond de la rivière de *Hulun* se meut [10] à la moindre perturbation. En plus de construire le pont, marcher dessus changera sa hauteur. Pouvoir passer d'un pont à l'autre pourrait ne plus être garanti à 100% !

2 Règles

Revoyons ici les règles et modalités plus en détail. Vous êtes lancés sur une rivière constituée de cases hexagonales, deux pandas par joueurs. Les deux pandas sont les parents et se trouvent toujours forcément sur des ponts.

2.1 Début d'une partie

Au début, on place deux pandas par race à 4 endroits distincts (p_1j_1, p_2j_1, p_1j_2, p_2j_2) [11]. Les bébés pandas et les ponts sont également placés, dépendamment de la carte chargée.

9. Ils savent bien nager, mais préfèrent éviter de mouiller leur magnifique pelage tout fluffy
10. Du verbe "mouvoir"
11. p = panda, j = joueur

2.2 Déroulement d'un tour

L'ordre des tours est comme suit : panda 1 joueur 1, panda 1 joueur 2, panda 2 joueur 1, panda 2 joueur 2. Pour chaque panda, il est possible d'effectuer deux types d'actions par tour :

— Pose d'un pont
— Déplacement

2.2.1 Pose d'un pont

Pour poser un pont, il faut de la place. Vous ne pouvez pas poser un pont sur un bébé panda [12], ni sur un autre pont. Vous ne pouvez pas poser un pont par-dessus un panda, car les pandas sont forcément sur un bout de pont. La valeur d'un pont (ou sa hauteur) est uniquement modifiée lorsqu'un panda *rentre* sur le pont. Lorsqu'un panda quitte une case de pont, sa valeur n'est pas modifiée, mais la valeur de la case de pont sur laquelle il va sera modifiée.

2.2.2 Déplacements

Vous pouvez toujours vous déplacer d'un côté à l'autre d'un même pont. Ce qui sera la grande difficulté, c'est de passer d'un pont à l'autre. Les ponts ont tous des hauteurs et pour passer d'un pont à l'autre, il faut que la partie du pont sur laquelle vous êtes soit à la même hauteur que la partie du pont où vous voulez aller. Il n'y a pas de nombre de déplacements limités, mais vous ne pouvez pas revenir en arrière lors d'un déplacement atomique [13] !

2.3 Les bébés pandas

Les bébés pandas sont ce qui vous rapporte des points [14]. Le but est de ramasser tous les bébés. La répartition des bébés par parent panda n'est pas importante. Vous pouvez avoir un seul parent avec tous les bébés, comme les répartir sur les deux parents.

Pour ramasser un bébé panda, il suffit d'être sur une case adjacente. Le ramassage se fera automatiquement.

12. Voir la section Bébés pandas pour comment ramasser un panda
13. déplacement atomique = déplacement d'une seule case. Vous pouvez en faire plusieurs, tant que c'est possible. Puis vous vous arrêtez.
14. Voir la partie "Points et Fin de partie"

2.4 Points et Fin de partie

Une partie peut s'arrêter pour deux raisons :
— une des races de panda a récupéré tous ses bébés
— la partie a atteint 100 tours

Le calcul se fait comme suit : lorsqu'un bébé panda est ramassé par un parent, il sera porté par ce parent jusqu'à la fin de la partie. Voici ce qu'on fait chaque tour [15] pour calculer les points du tour, qui seront additionnés aux points du joueur (classique). Pour chaque parent, on compte son nombre de bébés ramassés. Ce nombre est ensuite multiplié par 5. C'est le nombre de point que rapporte ce parent. Voici un exemple :

Disons qu'un parent panda géant porte un bébé depuis 2 tours et un autre bébé depuis 3 tours. Il aura rapporté au total 3 * 5 + 2 * 5 points depuis le début de la partie. Au tour suivant [16], il rapportera 10 points au total [17], car il porte 2 bébés. Ce sont donc 2 * 5 = 10 points.

15. Un tour étant à la fin d'un cyce de 4 ponts posés et déplacements faits par les 4 parents pandas sur la map.
16. Il faut attendre que les 3 autres aient joué leur tour
17. A condition qu'il ne ramasse pas d'autre bébé pendant ce tour

3 API

Constante : NB_TOURS
Valeur : 200
Description : Nombre de tours à jouer avant la fin de la partie.

Constante : NB_PANDAS
Valeur : 2
Description : Nombre de pandas par joueur.

Constante : NB_TOURS_PERTE_BEBE
Valeur : 3
Description : Nombre de tours nécessaires pour faire tomber un bébé panda.

Constante : VALEUR_MAX_PONT
Valeur : 6
Description : Valeur max d'un pont (les valeurs sont comprises entre 1 et cette constante inclus).

Constante : NB_POINTS_CAPTURE_BEBE
Valeur : 10
Description : Nombre de points obtenus à la capture d'un bébé pandas.

- **case_type**

Description : Types de cases
Valeurs :

LIBRE :	Case libre
OBSTACLE :	Obstacle
PONT :	Pont
BEBE :	Bébé panda

- **direction**

Description : Directions cardinales
Valeurs :

NORD_EST :	Direction : nord-est
SUD_EST :	Direction : sud-est
SUD :	Direction : sud
SUD_OUEST :	Direction : sud-ouest
NORD_OUEST :	Direction : nord-ouest
NORD :	Direction : nord

• **erreur**

Description : Erreurs possibles

Valeurs :

OK :	L'action s'est effectuée avec succès.
POSITION_INVALIDE :	La position spécifiée n'est pas sur la rivière.
POSITION_OBSTACLE :	La position spécifiée est un obstacle.
MAUVAIS_NOMBRE :	La hauteur de la position spécifiée ne correspond pas.
DEPLACEMENT_HORS_LIMITES :	Ce déplacement fait sortir un panda des limites de la rivière.
DIRECTION_INVALIDE :	La direction spécifiée n'existe pas.
MOUVEMENT_INVALIDE :	Le panda ne peut pas se déplacer dans cette direction.
POSE_INVALIDE :	Le pont ne peut pas être placé a cette position et dans cette direction.
ID_PANDA_INVALIDE :	Le panda spécifié n'existe pas.
ACTION_DEJA_EFFECTUEE :	Une action a déjà été effectuée ce tour.
DRAPEAU_INVALIDE :	Le drapeau spécifié n'existe pas.
DEPLACEMENT_EN_ARRIERE :	La panda c'est déjà déplacé sur cette case.

• **action_type**

Description : Types d'actions

Valeurs :

ACTION_DEPLACER :	Action "deplacer".
ACTION_POSER :	Action "poser".

• **debug_drapeau**

Description : Types de drapeau de débug

Valeurs :

AUCUN_DRAPEAU :	Aucun drapeau, enlève le drapeau présent
DRAPEAU_BLEU :	Drapeau bleu
DRAPEAU_VERT :	Drapeau vert
DRAPEAU_ROUGE :	Drapeau rouge

• **position**

```
struct position {
    int x;
    int y;
};
```

Description : Position du panda.

Champs :

x :	Coordonnée : x
y :	Coordonnée : y

• **pont_type**

```
struct pont\_type {
```

```
    position debut_pos;
    position fin_pos;
    int debut_val;
    int fin_val;
};
```

Description : Case type pont, contient la case de début et de fin. La case de début a une valeur se décrémentant, celle de fin s'incrémente.

Champs :
- *debut_pos* : Position de la case de début
- *fin_pos* : Position de la case de fin
- *debut_val* : Valeur de la case de début
- *fin_val* : Valeur de la case de début

- **panda_info**

```
struct panda\_info {
    position panda_pos;
    int id_joueur;
    int id_panda;
    int num_bebes;
};
```

Description : Panda et son joueur

Champs :
- *panda_pos* : Position du panda
- *id_joueur* : Identifiant du joueur qui contrôle le panda
- *id_panda* : Identifiant du panda relatif au joueur
- *num_bebes* : Nombre de bébés qui sont portés par le panda parent

- **bebe_info**

```
struct bebe\_info {
    position bebe_pos;
    int id_bebe_joueur;
};
```

Description : Bébé panda à ramener.

Champs :
- *bebe_pos* : Position du bébé panda
- *id_bebe_joueur* : Identifiant du joueur qui peut saver le bébé

- **tour_info**

```
struct tour\_info {
    int id_panda_joue;
    int id_tour;
```

```
};
```

Description : Information sur un tour particulier.
Champs :
- *id_panda_joue* : Identifiant du panda qui joue
- *id_tour* : Identifiant unique du tour (compteur)

- **carte_info**

```
struct carte\_info {
    int taille_x;
    int taille_y;
};
```

Description : Information sur la carte de la partie en cours.
Champs :
- *taille_x* : La taille de la carte pour les coordonnées x [0; taille_x[
- *taille_y* : La taille de la carte pour les coordonnées y [0; taille_y[

- **action_hist**

```
struct action\_hist {
    action_type type_action;
    int action_id_panda;
    direction dir;
    int valeur_debut;
    int valeur_fin;
    position pos_debut;
    position pos_fin;
};
```

Description : Action représentée dans l'historique.
Champs :
- *type_action* : Type de l'action
- *action_id_panda* : Identifiant du panda concerné par l'action
- *dir* : Direction visée par le panda durant le déplacement
- *valeur_debut* : Valeur au début du pont posé (de 1 à 6 inclus)
- *valeur_fin* : Valeur à la fin du pont posé (de 1 à 6 inclus)
- *pos_debut* : Position du début du pont posé
- *pos_fin* : Position de la fin du pont posé

• **deplacer**

```
erreur deplacer(direction dir)
```

Description : Déplace le panda “id_panda“ sur le pont choisi.
Paramètres : *dir* : Direction visée

• **poser**

```
erreur poser(position position_debut, direction dir, int pont_debut,
    ↪ int pont_fin)
```

Description : Pose un pont dans la direction choisie à partir du panda “id_panda“.
Paramètres : *position_debut* : Position de début du pont
dir : Direction visée
pont_debut : Début du pont placé
pont_fin : Fin du pont

• **debug_afficher_drapeau**

```
erreur debug_afficher_drapeau(position pos, debug_drapeau drapeau)
```

Description : Affiche le drapeau spécifié sur la case indiquée
Paramètres : *pos* : Case ciblée
drapeau : Drapeau à afficher sur la case

• **type_case**

```
case_type type_case(position pos)
```

Description : Renvoie le type d’une case donnée.
Paramètres : *pos* : Case choisie

• **panda_sur_case**

```
int panda_sur_case(position pos)
```

Description : Renvoie le numéro du joueur à qui appartient panda sur la case indiquée. Renvoie -1 s’il n’y a pas de panda ou si la position est invalide.
Paramètres : *pos* : Case choisie

• **bebe_panda_sur_case**

```
int bebe_panda_sur_case(position pos)
```

Description : Renvoie le numéro du joueur à qui appartient le bébé panda sur la case indiquée. Renvoie -1 s'il n'y a pas de bébé panda ou si la position est invalide.

Paramètres : *pos* : Case choisie

• **position_panda**

```
position position_panda(int id_joueur, int id_panda)
```

Description : Indique la position du panda sur la rivière désigné par le numéro "id_panda" appartenant au joueur "id_joueur". Si la description du panda est incorrecte, la position (-1, -1) est renvoyée.

Paramètres : *id_joueur* : Numéro du joueur
id_panda : Numéro du panda

• **info_pont**

```
pont_type info_pont(position pos)
```

Description : Renvoie les informations relatives au pont situé à cette position. Le pont est constitué de deux cases. Si aucun pont n'est placé à cette position ou si la position est invalide, les membres debut_val et fin_val de la structure "pont_type" renvoyée sont initialisés à -1.

Paramètres : *pos* : Case choisie

• **info_panda**

```
panda_info info_panda(position pos)
```

Description : Renvoie la description d'un panda en fonction d'une position donnée. Si le panda n'est pas présent sur la carte, ou si la position est invalide, tous les membres de la structure "panda_info" renvoyée sont initialisés à -1.

Paramètres : *pos* : Case choisie

• **liste_pandas**

```
panda_info array liste_pandas()
```

Description : Renvoie la liste de tous les pandas présents durant la partie.

• **liste_bebes**

```
bebe_info array liste_bebes()
```

Description : Renvoie la liste de tous les bébés présents sur la carte, et et pas encore sauvés.

• **positions_adjacentes**

```
position array positions_adjacentes(position pos)
```

Description : Renvoie la liste de toutes les positions adjacentes à la position donnée.
Paramètres : *pos* : Case choisie

• **position_dans_direction**

```
position position_dans_direction(position pos, direction dir)
```

Description : Renvoie la position relative à la direction donnée par rapport à une position d'origine. Si une telle position serait invalide, la position {-1, -1} est renvoyée.
Paramètres : *pos* : Position d'origine
dir : Direction

• **direction_entre_positions**

```
int direction_entre_positions(position origine, position cible)
```

Description : Renvoie la direction telle que position_dans_direction(origine, cible) == direction. Si aucune telle direction n'existe, -1 est renvoyée.
Paramètres : *origine* : Position d'origine
cible : Position relative à l'origine

• **historique**

```
action_hist array historique()
```

Description : Renvoie la liste des actions effectuées par l'adversaire durant son tour, dans l'ordre chronologique. Les actions de débug n'apparaissent pas dans cette liste.

• **score**

```
int score(int id_joueur)
```

Description : Renvoie le score du joueur "id_joueur". Renvoie -1 si le joueur est invalide.
Paramètres : *id_joueur* : Numéro du joueur

• **moi**

```
int moi()
```

Description : Renvoie votre numéro de joueur.

• **adversaire**

```
int adversaire()
```

Description : Renvoie le numéro de joueur de votre adversaire.

• **info_tour**

```
tour_info info_tour()
```

Description : Renvoie le tour actuel.

• **info_carte**

```
carte_info info_carte()
```

Description : Renvoie la carte pour la partie en cours.

• **afficher_case_type**

```
void afficher_case_type(case_type v)
```

Description : Affiche le contenu d'une valeur de type case_type
Paramètres : v : The value to display

• **afficher_direction**

```
void afficher_direction(direction v)
```

Description : Affiche le contenu d'une valeur de type direction
Paramètres : v : The value to display

• **afficher_erreur**

```
void afficher_erreur(erreur v)
```

Description : Affiche le contenu d'une valeur de type erreur
Paramètres : v : The value to display

• **afficher_action_type**

```
void afficher_action_type(action_type v)
```

Description : Affiche le contenu d'une valeur de type action_type
Paramètres : v : The value to display

• **afficher_debug_drapeau**

```
void afficher_debug_drapeau(debug_drapeau v)
```

Description : Affiche le contenu d'une valeur de type debug_drapeau
Paramètres : v : The value to display

• **afficher_position**

```
void afficher_position(position v)
```

Description : Affiche le contenu d'une valeur de type position
Paramètres : v : The value to display

• **afficher_pont_type**

`void afficher_pont_type(pont_type v)`

Description : Affiche le contenu d'une valeur de type pont_type
Paramètres : v : The value to display

• **afficher_panda_info**

`void afficher_panda_info(panda_info v)`

Description : Affiche le contenu d'une valeur de type panda_info
Paramètres : v : The value to display

• **afficher_bebe_info**

`void afficher_bebe_info(bebe_info v)`

Description : Affiche le contenu d'une valeur de type bebe_info
Paramètres : v : The value to display

• **afficher_tour_info**

`void afficher_tour_info(tour_info v)`

Description : Affiche le contenu d'une valeur de type tour_info
Paramètres : v : The value to display

• **afficher_carte_info**

`void afficher_carte_info(carte_info v)`

Description : Affiche le contenu d'une valeur de type carte_info
Paramètres : v : The value to display

• **afficher_action_hist**

`void afficher_action_hist(action_hist v)`

Description : Affiche le contenu d'une valeur de type action_hist
Paramètres : v : The value to display

4 Notes sur l'utilisation de l'API

4.1 C

— Les booléens sont représentés par le type `bool`, défini par le standard du C99, et que l'on retrouve dans le header `stdbool.h`;
— Les fonctions prenant des tableaux en paramètres et retournant des tableaux utilisent à la place de ces tableaux une structure `type_array`, où `type` est le type des données dans le tableau. Ces structures contiennent deux éléments : les données, `type* items`, et la taille, `size_t length`. Dans tous les cas, la libération des données est laissée au soin du candidat;
— Tout le reste est comme indiqué dans le sujet.

4.2 C++

— Les tableaux sont représentés par des `std::vector<type>`;
— Le reste est identique au sujet.

4.3 C#

— Les fonctions à utiliser sont des méthodes statiques de la classe `Api`. Ainsi, pour utiliser la fonction `Foo`, il faut faire `Api.Foo`;
— Les noms des fonctions, structures et énumérations sont en `CamelCase`. Ainsi, une fonction nommée `foo_bar` dans le sujet s'appellera `FooBar` en C#.

4.4 Haskell

— L'API est fournie par le module `Api`.
— Les énumérations sont représentées par des types sommes, les structures par des records. Seule la première lettre des noms de types et de constructeurs est en majuscule. Le nom du constructeur d'une structure est son nom de type.
— La commande `make doc` permet de générer la documentation dans le fichier `doc/index.html` pour votre code ainsi que pour l'API.
— Pour pouvoir conserver des valeurs entre différents appels à vos fonctions à compléter, il faut utiliser des variables mutables :

```
import Data.IORef
import System.IO.Unsafe (unsafePerformIO)

-- La pragma NOINLINE est importante !
-- MonType ne doit pas etre polymorphe !
{-# NOINLINE maVariable #-}
```

```
maVariable :: IORef MonType
maVariable = unsafePerformIO (newIORef maValeurInitiale)

fonctionACompleter :: IO ()
fonctionACompleter = do
  maValeur <- readIORef maVariable
  ...
  writeIORef maVariable maValeur'
```

4.5 Java

— Les fonctions à utiliser sont des méthodes statiques de la classe `Interface`. Ainsi, pour utiliser la fonction `foo`, il faut faire `Interface.foo;`
— Les structures sont représentées par des classes dont tous les attributs sont publics.

4.6 OCaml

— L'API est fournie par le fichier `api.ml`, qui est `open` par défaut par le fichier à compléter ;
— Les énumérations sont représentées par des types sommes avec des constructeurs sans paramètres. Seule la première lettre des noms des constructeurs est en majuscule ;
— Les structures sont représentées par des records, sauf pour la structure `position` qui est représentée par un couple `int * int;`
— Les tableaux sont représentés par des `array` Caml classiques.

4.7 PHP

— Les constantes sont définies via des `define` et doivent donc être utilisées sans les précéder d'un signe dollar ;
— Les énumérations sont définies comme des séries de constantes. Se référer à la puce au-dessus ;
— Les structures sont gérées sous forme de tableaux associatifs. Ainsi, une structure contenant un champ `x` et un champ `y` sera créée comme ceci : `array('x' => 42, 'y' => 1337).`

4.8 Python

— L'API est fournie par le module `api`, dont tout le contenu est importé par défaut par le code à compléter ;

— Les énumérations sont représentées par des `IntEnum` Python, qui peuvent être utilisées comme ceci : `nom_enum.CHAMP`.;
— Les structures sont représentées par des `NamedTuple` Python, dont on peut accéder aux champs via la notation pointée habituelle, et qui peuvent être créés comme ceci : `foo(bar=42, x=3)`, sauf pour la structure `position` qui est représentée par un couple `(x, y)`.

4.9 Rust

— L'API est fournie par le module `api`, dont tout le contenu est importé par défaut par le code à compléter.;
— Les noms des structures et énumérations sont en `CamelCase`. Ainsi, une structure nommée `foo_bar` dans le sujet s'appellera `FooBar` en Rust.
— Les tableaux sont représentés par des `Vec<T>` et les strings par des `String`. Les fonctions prennent leurs primitives empruntées `&[T]` et `&str` en entrée.

Bonne chance !