université PARIS-SACLAY

Projet de programmation
Cours de programmation modulaire
—Licence MI/IM/MNSI - Info 3—

Types de coordonnées et ensemble de coordonnées (préliminaire au projet)

Le code écrit dans ce TP sera utilisé dans le projet. Il est donc impératif de le tester de manière très poussée pour ne pas introduire de bug dans le projet.
Vous commencerez à travailler en binôme lors de cette séance, et vous commencerez également à utiliser `git` pour travailler en binôme.
Les méthodes des classes seront publiques, sauf précision du contraire. Les attributs seront toujours privés.

Tapez la commande `./course.py fetch Projet` afin d'initialiser votre dépôt `git` pour le projet. Tous vos fichiers seront à créer dans le dossier ainsi obtenu.

▸ **Exercice 1. (Makefile)** Dans cette partie vous allez mettre en place votre structure de fichiers pour faire votre Makefile. Pour créer et gérer les nouvelles classes qui vous permettront de manipuler les coordonnées, vous aurez les fichiers suivants :

- coord.cpp
- coord.hpp
- test.cpp
- Makefile
- doctest.h

Les fichiers coord.hpp et coord.cpp vous permettront de déclarer et décrire les classes et fonctions. Le fichier test.cpp vous permettra d'activer les tests dans un fichier à part (voir slide 16 du cours 5). Ce fichier de test ne contiendra que deux lignes :

```
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
#include "doctest.h"
```

Vous écrirez les tests à côté des fonctions qu'ils testent dans les fichiers `.cpp`. Après les avoir compilé, il faudra lier les fichiers `.o` avec le fichier de test. Ainsi, la commande décrivant la fabrication de l'exécutable de test dans le `Makefile` pourra être :

```
tests: tests.o coords.o ants.o places.o game.o
————→$(CXX) -o $@ $^ $(LDFLAGS)
```

Au début du projet vous n'aurez pas tous ces `.o` mais seulement `test.o` et `coords.o`. La commande est donc plus courte, et sera complétée au fur et à mesure de votre projet.

Quand vous créez de nouveaux fichiers, **pensez, dès la création du fichier, à faire la commande** `git add` *nomDuFichier* pour que `git` le gère

1. Créez vos différents fichiers en n'oubliant pas les `#include`.
2. Ajoutez dans `git` vos fichiers avec `git add`.
3. Créez votre `Makefile` en n'oubliant pas de lier les différents fichiers entre eux.
4. Ajoutez `Makefile` dans `git` avec `git add`.
5. Faites un `prog-mod submit Projet` *MonGroupe* pour vérifier que tout va bien.
6. Si votre binôme est connu et présent vous pouvez vous reporter au document `AideGit.pdf` d'aide sur `Git` pour lui donner l'accès à votre travail (n'y passez pas plus de 10 minutes).

▸ **Exercice 2. (La classe Coord)** On cherche à manipuler des coordonnées dans une grille, c'est-à-dire la paire constituée d'un numéro de ligne et d'un numéro de colonne.

1. Dans `coord.hpp`, définir une constante `TAILLEGRILLE` qui décrira la taille de la grille (comme la grille sera carrée, `TAILLEGRILLE` est simplement un entier).
2. Créer la classe `Coord` selon la description ci-dessus. Rappel : Les attributs seront toujours privés.
3. Coder un constructeur de Coord qui prend en paramètre un numéro de ligne *lig* et un numéro de colonne *col*. Retourner une exception si les coordonnées ne sont pas correctes.
   Dans ce constructeur, la taille de la grille est connue. Ainsi le constructeur de coordonnées **vérifie qu'on ne construit pas une coordonnée en dehors de la grille**. D'autre part, on ne met que des getters et pas de setters dans l'interface de coordonnées. Ainsi, il n'est pas possible de modifier une coordonnée. Il n'y a donc jamais de coordonnées incorrectes ce qui évitera les erreurs de segmentation causées par un dépassement des bornes du tableau. C'est aussi pour cette raison qu'il n'y a pas de constructeur par défaut de coordonnées.
   Rappel : les méthodes de vos classes doivent être publiques, sauf précision du contraire.
   Remarque : Réfléchir à ce qui doit être mis dans le fichier `.hpp` et ce qui doit être mis dans le fichier `.cpp`.
4. Écrire les getters permettant de lire les coordonnées d'une `Coord`.
5. Pour tester votre constructeur et vos getters, faites un premier test qui construit un objet `Coord`, puis récupère son numéro de ligne et vérifie qu'on obtient bien ce qu'il faut. Même chose pour le numéro de colonne.
6. Surcharger l'opérateur `<<` qui affiche les coordonnées sous la forme : (lig, col).
7. Surcharger les opérateurs `==` et `!=`.
8. Tester scrupuleusement cette première classe, en réfléchissant bien à tous les cas limites possibles, afin d'éviter qu'elle n'induise des bugs dans votre projet.

▸ **Exercice 3. (Ensemble de coordonnées)**

1. On souhaite maintenant représenter un ensemble de coordonnées. Coder la classe `EnsCoord` contenant un tableau de taille variable de coordonnées.
2. Surcharger l'opérateur `<<` pour `EnsCoord`.
3. Coder un constructeur de `EnsCoord` qui prend en paramètre un vecteur de `Coord`.
4. Coder la méthode `position` qui prend en paramètre un objet de la classe `Coord` et retourne sa position dans un objet `EnsCoord`. La méthode retournera -1 si la `Coord` n'est pas contenue dans l'`EnsCoord`. Cette méthode est une méthode auxiliaire privée, on n'en aura besoin que dans des méthodes de la classe (`contient` et `supprime`).
5. Coder la méthode `contient` qui prend en paramètre un objet de la classe `Coord` et renvoie `true` s'il est dans un `EnsCoord`.
6. Coder la méthode `ajoute` qui prend en paramètre un objet de la classe `Coord` et l'ajoute à un `EnsCoord` (s'il n'y est pas déjà).
7. Coder la méthode `supprime` qui prend en paramètre un objet de la classe `Coord` et le supprime d'un `EnsCoord`. Retourner une exception si l'`EnsCoord` ne contient pas la `Coord`.
8. Coder la méthode `estVide` qui renvoie `true` si un `EnsCoord` est vide.
9. Coder la méthode `taille` qui retourne le nombre d'éléments d'un `EnsCoord`.
10. Coder la méthode `ieme` qui prend en paramètre un entier n et retourne l'élément qui est à la position n dans un `EnsCoord`.
11. Tester minutieusement toutes ces fonctions, car s'il y a une erreur, c'est beaucoup plus facile de la détecter à ce stade que quand elle aura comme effet de rendre incohérent le comportement d'une fourmi à l'écran.

▸ **Exercice 4. (Création de la grille)**

1. On s'intéresse maintenant à trouver les coordonnées des voisines d'une case dans une grille de taille `TAILLEGRILLE` lignes et `TAILLEGRILLE` colonnes. Par exemple, considérons la grille ci-dessous où `TAILLEGRILLE` $= 5$:

$(0,0)$	$(0,1)$	$(0,2)$	$(0,3)$	$(0,4)$
$(1,0)$	$(1,1)$	$(1,2)$	$(1,3)$	$(1,4)$
$(2,0)$	$(2,1)$	$(2,2)$	$(2,3)$	$(2,4)$
$(3,0)$	$(3,1)$	$(3,2)$	$(3,3)$	$(3,4)$
$(4,0)$	$(4,1)$	$(4,2)$	$(4,3)$	$(4,4)$

   — Les voisines de $(2,1)$ sont $(1,0)$, $(1,1)$, $(1,2)$, $(2,2)$, $(3,2)$, $(3,1)$, $(3,0)$, $(2,0)$.

   — Les voisines de $(3,4)$ sont $(2,3)$, $(2,4)$, $(3,3)$, $(4,3)$, $(4,4)$.

   — Les voisines de $(0,0)$ sont $(0,1)$, $(1,0)$, $(1,1)$.

   Coder la fonction `voisines` qui prend en paramètre une coordonnée c et retourne un `EnsCoord` des voisines de c. (Remarquer que ce n'est pas une méthode.)

   Si lig est la ligne de c et col est la colonne de c, on peut collecter les coordonnées des voisines de c dans un ensemble ev de la manière suivante :

```
pour i allant de i_min à i_max faire
    pour j allant de j_min à j_max faire
        si (i, j) ≠ (lig, col) alors
            ajouteEnsCoord (ev, (i, j))
        finsi
    finpour
finpour
```

   avec : $i_{min} = max(lig - 1,\ 0)$
   $i_{max} = min(lig + 1,\ \text{TAILLEGRILLE} - 1)$
   $j_{min} = max(col - 1,\ 0)$
   $j_{max} = min(col + 1,\ \text{TAILLEGRILLE} - 1)$

2. Vérifier avec plusieurs exemples au milieu et sur les bords de la grille que votre fonction est correcte.

Attention ! ! ! Cette fonction est cruciale pour le reste de votre projet ! Il est impératif de la tester avec un maximum de cas limites possibles.

3. Coder une méthode `choixHasard` qui retourne une coordonnée au hasard parmi un ensemble de coordonnées.

   Aide : la fonction `rand()` (voir l'explication à la fin du TP2), retourne un nombre aléatoire. Pour obtenir un nombre aléatoire entre 0 et n (compris), il faut donc appliquer un modulo $(n + 1)$ au résultat de l'appel à `rand()`.