TP n° 9

Types 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.

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. Coder le type structure `Coord`.
2. Coder la fonction `nouvCoord` qui prend en paramètre un numéro de ligne *lig* et un numéro de colonne *col* et retourne une nouvelle paire de type `coord` initialisée à la coordonnée (lig, col).
3. Coder la procédure `afficheCoord` qui prend en paramètre une valeur de type `coord` et affiche cette valeur sous la forme : (lig, col).
4. Tester ces fonctions et procédures en utilisant le programme principal suivant :

```
int main(){
  coord c1 = nouvCoord(2,1);
  afficheCoord(c1);
  cout << endl;
  return 0;
}
```

5. Coder la fonction `egalCoord` retournant vrai si deux coordonnées sont égales. Tester cette fonction avec la command `ASSERT` vue en TP dans une fonction `testEgalCoord`.
6. On souhaite maintenant représenter un ensemble d'au maximum `MAXENSCOORD` coordonnées. Coder le type `EnsCoord` avec une structure contenant un champ `tab` et un champ `nbElts`.
7. Coder une procédure `afficheEnsCoord` qui affiche un ensemble de coordonnées.
8. Coder la fonction `nouvEnsCoord` qui renvoie un ensemble de coordonnées ne contenant aucun élément.
9. Coder la procédure `ajouteEnsCoord` qui prend en paramètre un ensemble de coordonnées *ec*, une coordonnée *c* et ajoute *c* à l'ensemble *ec*.
10. On vous donne le programme principal suivant qui permet de voir si les choses fonctionnent.

```
int main(){
  coord c1 = nouvCoord(2,1);
  coord c2 = nouvCoord(3,4);
  coord c3 = nouvCoord(0,0);

  // on construit un exemple d'ensemble
  EnsCoord exemple = nouvEnsCoord();
  ajouteEnsCoord(exemple, c1);
  ajouteEnsCoord(exemple, c2);
  ajouteEnsCoord(exemple, c3);
  afficheEnsCoord(exemple);

  // on ajoute encore un element
  cout << "ajout d'un element :" << endl;
```

```
  ajouteEnsCoord (exemple, nouvCoord(4,0));
  afficheEnsCoord(exemple);

  return 0;
}
```

11. Tester ces procédures et fonctions de manière plus précise.
12. On s'intéresse maintenant à trouver les coordonnées des voisines d'une case dans une grille de `TAILLE` lignes et `TAILLE` colonnes. Par exemple, considérons la grille ci-dessous (`TAILLE` $= 5$) :

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

— 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 l'ensemble des coordonnées des voisines de c.

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 $(i,j) \neq (lig, col)$ alors
      ajouteEnsCoord (ev, (i,j))
    finsi
  finpour
finpour

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

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

```
// on teste la fonction voisines
EnsCoord v;
int i;
for (i = 0; i < exemple.nbElts; i++){
  v = voisines(exemple.tab[i]);
  cout << "les voisines de ";
  afficheCoord(exemple.tab[i]);
  cout << "sont ";
  afficheEnsCoord(v);
}
```

14. Coder une fonction `choixCoordHasard` qui prend un ensemble de coordonnées en paramètre et retourne une coordonnée au hasard parmi cet ensemble.
Aide : la fonction `rand()` 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()`.

Ajouter un appel à cette fonction à la fin du programme principal pour la tester.