Dans ce TP, on suppose déjà connu le type abstrait Ensemble. Les éléments d'un Ensemble sont de type Objet supposé lui aussi défini par ailleurs. Par exemple, le type Objet pourrait être le type int, ou une structure contenant deux entiers... ou n'importe quoi d'autre.

Nous utilisons une version du type abstrait Ensemble la plus simple possible et qui ne possède que les constructeurs et les opérateurs suivants:

Constructeur

- void ensVide(Ensemble &e) // Crée l'ensemble vide e

Opérateurs d'accès

- void ajoute(Ensemble &e, Objet o)// l'objet o donné est ajouté à l'ensemble e s'il n'y est pas, sinon rien n'est changé

- void retire(Ensemble &e, Objet o)// l'objet o donné est retiré de l'ensemble e s'il y est, sinon rien n'est changé

- objet element(Ensemble e)// retourne un objet quelconque de l'ensemble e si celui-ci n'est pas vide ; // comportement non spécifié si l'ensemble e est vide.

Opérateurs de test

- bool estVide(Ensemble e)// retourne true si e est l'ensemble vide, false sinon

- bool estDans(Ensemble e, Objet o)// retourne true si l'objet o est dans l'ensemble e, false sinon

À partir de ces 6 primitives, on souhaite programmer d'autres constructeurs et d'autres fonctions plus élaborées, que l'on utilisera finalement pour vérifier une propriété ensembliste.

Remarque : le fait d'être obligé de passer par ces six primitives impose de programmer dans un style qui n'est pas efficace : par exemple pour calculer le cardinal d'un ensemble, on doit sortir les éléments un par un de l'ensemble, en incrémentant simultanément un compteur. De même, pour réaliser l'union ou l'intersection.

Un vrai type abstrait sera en général composé de plus de 6 primitives, pour améliorer les performances. Nous proposons ce jeu restreint, pour que le TP soit court, tout en illustrant comment l'utilisation d'un type abstrait permet la répartition des tâches : en aval, une personne peut réaliser les 6 primitives en C ou en Java par exemple, tandis qu'en amont, une autre personne programmera ce TP. L'exploitation conjointe des deux parties réalisées sera ensuite immédiate puisque l'interface entre ces deux parties a été définie par l'explicitation du type abstrait, i.e. les signatures des constructeurs et des opérateurs du type.

Exercice 1.

1. Réaliser une fonction cardinal qui retourne le cardinal d'un ensemble passé en paramètre.

2. Réaliser la fonction inclus qui teste si un ensemble est inclus dans un autre (elle renvoie true si le premier ensemble d'objets donné est inclus dans le deuxième, false sinon).

3. Réaliser la fonction égal qui renvoie true si deux ensembles d'objets donnés sont égaux (i.e. ont les mêmes éléments), false sinon.

4. Réaliser la fonction unionEns qui retourne l'union de deux ensembles donnés. Les deux ensembles donnés ne doivent pas être modifiés.

5. Réaliser la fonction interEns qui retourne l'intersection de deux ensembles donnés, sans les modifier.

6. En supposant qu'il existe une procédure ecrireObj(Objet o), qui permet d'imprimer un objet, réaliser la procédure ecrireEns qui imprime tous

les éléments d'un ensemble passé en paramètre.

7. En supposant qu'il existe une procédure : randomObj(Objet &o) qui renvoie un objet o aléatoire, réaliser la procédure randomEns qui crée un ensemble de n éléments aléatoires, où n est un entier positif.

8. Principe d'inclusion-exclusion : Écrire le programme principal qui vérifie si la propriété ensembliste suivante est vraie :

   si A, B, C sont des ensembles, alors on a

$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$ où $|A|$ est le nombre cardinal de A. On vérifiera expérimentalement cette propriété en la testant sur des ensembles construits de façon aléatoire et dont le cardinal est lui-même aléatoire (mais compris entre 8 et 12).

Exercice 2. On s'intéresse maintenant à la réalisation de ce type abstrait. Plusieurs types concrets peuvent alors être choisis/utilisés. On supposera ici que les objets sont des entiers et que le nombre d'objets d'un ensemble est borné par une constante MaxE.

Réalisez la fonction estDans et les procédures ensVide, ajoute et retire, avec chacun des types concrets suivants :

Réalisation 1 Dans cette première réalisation, on ne stocke que des ensembles d'entier qui sont entre 0 et MaxE – 1. On peut donc utiliser comme type concret un tableau T de booléen où T [i] vaut true si l'entier i est dans l'ensemble et faux sinon. Pour éviter les problèmes spécifiques au passage de tableaux en paramètre, on met le tableau dans une struct.

Réalisation 2 On utilise comme type concret une structure contenant un tableau d'entiers et un champ indiquant le cardinal (nombre d'éléments de l'ensemble) + un entier indiquant la taille actuelle allouée du tableau.

Réalisation 3 On utilise comme type concret une structure contenant un tableau d'entiers triés et un champ indiquant le nombre d'éléments de l'ensemble.