TP C++ n°1 : Classe simple

1. Spécification détaillée de la classe

Votre classe possède 3 caractéristiques essentielles : sa cardinalité maximale, sa cardinalité actuelle et les éléments de l'ensemble. Les cardinalités de l'ensemble sont gérées par la relation d'ordre :

`Cardinalité Maximale ≥ Cardinalité Actuelle ≥ 0`

Pour gérer les éléments de l'ensemble, il faut obligatoirement s'appuyer sur un tableau dynamique même si d'autres implémentations sont clairement possibles pour répondre à cette spécification détaillée.

`Ensemble ( unsigned int cardMax = CARD_MAX );`

`cardMax` est la cardinalité maximale de l'ensemble en cours construction. Ce constructeur construit un ensemble vide (c'est-à-dire ne comportant aucun élément entier) et de cardinalité maximale fixée par le paramètre **`cardMax`**. Si le paramètre est absent, c'est la constante **`CARD_MAX`** qui est utilisée avec comme valeur par défaut **`5`**. Si **`cardMax`** vaut **`0`**, l'ensemble construit est nul, c'est-à-dire que sa cardinalité courante et sa cardinalité maximale sont nulles.

`Ensemble ( int t [ ], unsigned int nbElements );`

`t` est un tableau d'entiers au sens du langage C/C++ et **`nbElements`** correspond au nombre d'éléments significatifs dans le tableau. Ces éléments significatifs sont obligatoirement contigus dans le tableau et en tête de ce tableau (les premiers indices). Ce nombre d'éléments peut être différent de la dimension du tableau.

Ce constructeur construit un ensemble d'entiers à partir des entiers contenus dans le tableau **`t`** en respectant la définition mathématique d'un ensemble. A la fin de la construction, la cardinalité maximale de l'ensemble construit est égale au paramètre **`nbElements`** et la cardinalité actuelle correspond au nombre d'éléments effectivement rajoutés dans l'ensemble.

Par définition : cardinalité courante ≤ cardinalité maximale.

`virtual ~Ensemble ( );`

Ce destructeur doit libérer la totalité des ressources (mémoire) occupées par l'ensemble.

`void Afficher ( void );`

Cette méthode se charge d'afficher sur la sortie standard (l'écran) la valeur d'un objet ensemble en respectant la syntaxe suivante :

- **`{}`** si l'ensemble est vide ;
- **`{x}`** si l'ensemble est composé d'un seul élément **`x`** ;
- **`{x,y,z}`** si l'ensemble est composé de plusieurs éléments (**`x`**, **`y`** et **`z`** pour l'exemple).

S'il y a plusieurs éléments, ils devront toujours être affichés par ordre croissant même si, à la base, un ensemble mathématique est non ordonné. Cette exigence est liée aux tests automatiques de votre classe. Un retour à la ligne termine toujours l'affichage de l'ensemble. Pour faciliter le test de votre classe, les 2 informations suivantes sont rajoutées **avant** l'affichage des valeurs de l'ensemble, en respectant la syntaxe donnée :

`n`
`m`

où **`n`** représente la cardinalité actuelle de l'ensemble et **`m`** sa cardinalité maximale.

`bool EstEgal ( const Ensemble & unEnsemble );`

`unEnsemble` donne l'ensemble qui est utilisé dans le test d'égalité avec l'ensemble qui invoque la méthode. L'ensemble **`unEnsemble`** et l'ensemble qui invoque la méthode sont égaux si et seulement s'ils ont la même cardinalité actuelle, indépendamment de leur cardinalité maximale, et si tous les éléments de l'ensemble **`unEnsemble`** sont aussi présents dans l'ensemble qui invoque la méthode. Dans ce cas, la méthode renvoie *vrai*. Si les cardinalités actuelles sont égales et qu'il existe au moins un élément de l'ensemble qui invoque la méthode qui n'appartient pas à l'ensemble **`unEnsemble`** alors l'égalité n'est pas satisfaite et la méthode renvoie *faux*. Ce cas est indépendant des valeurs des cardinalités maximales. Si les cardinalités actuelles sont différentes, l'égalité des ensembles n'est pas vérifiée et la méthode renvoie *faux*.

`crduEstInclus EstInclus ( const Ensemble & unEnsemble );`

`unEnsemble` est l'ensemble utilisé pour vérifier l'inclusion. Cette méthode vérifie si l'ensemble qui invoque la méthode est inclus dans l'ensemble **`unEnsemble`** (non inclusion, inclusion large ou inclusion stricte). Si les 2 ensembles sont égaux, l'inclusion est vérifiée. Dans ce cas, la méthode renvoie **`INCLUSION_LARGE`**. S'il existe au moins un élément de l'ensemble qui invoque la méthode qui ne se retrouve pas dans **`unEnsemble`**, alors l'inclusion n'est pas vérifiée et la méthode renvoie **`NON_INCLUSION`**. Si tous les éléments de l'ensemble qui invoque la méthode se retrouvent dans **`unEnsemble`** et que les 2 ensembles ne sont pas égaux (premier cas de figure), alors l'inclusion est strictement vérifiée et la méthode renvoie **`INCLUSION_STRICTE`**.

Pour réaliser cette méthode, il faut définir une énumération composée de **`NON_INCLUSION`**, **`INCLUSION_LARGE`** et **`INCLUSION_STRICTE`**.

`crduAjouter Ajouter ( int aAjouter );`

`aAjouter` est l'élément entier à rajouter à l'ensemble, si cela est possible. L'élément **`aAjouter`** est rajouté à l'ensemble, si cela est nécessaire et possible. En effet, la cardinalité maximale de l'ensemble doit rester inchangée lors de cette opération. La méthode renvoie **`DEJA_PRESENT`**, si l'élément **`aAjouter`** appartient déjà à l'ensemble (l'ajout devient inutile). Dans tous les cas de figure, **`DEJA_PRESENT`** l'emporte sur **`PLEIN`**, si les deux conditions sont vraies simultanément. La méthode renvoie **`PLEIN`**, si l'élément **`aAjouter`** n'existe pas déjà dans l'ensemble et qu'il n'y a plus de place dans l'ensemble. La méthode renvoie **`AJOUTE`**, si l'élément **`aAjouter`** n'existe pas déjà dans l'ensemble et qu'il y a encore de la place dans l'ensemble. Dans ce dernier cas, la cardinalité courante est mise à jour pour refléter l'ajout de l'élément à l'ensemble.

Pour réaliser cette méthode, il faut définir une énumération composée de **`DEJA_PRESENT`**, **`PLEIN`** et **`AJOUTE`**.

`unsigned int Ajuster ( int delta );`

`delta` donne le nombre d'éléments du réajustement. Si **`delta`** est strictement positif, l'ensemble est agrandi du nombre d'éléments défini par **`delta`** : c'est un agrandissement de l'ensemble. Si **`delta`** est strictement négatif, l'ensemble est réduit du nombre d'éléments défini par **`delta`** (dans les limites possibles) : c'est une réduction de l'ensemble. Cette réduction ne peut pas s'accompagner de perte d'éléments dans l'ensemble. Si **`delta`** est nul, l'opération est sans effet et la valeur de retour est la cardinalité maximale initiale de l'ensemble. Dans tous les cas de figure, la valeur de retour est la nouvelle cardinalité maximale de l'ensemble.

`bool Retirer ( int element );`

`element` est l'élément entier à retirer de l'ensemble, si cela est possible (existence). L'élément entier est retiré de l'ensemble s'il est présent. La méthode renvoie *vrai*, si l'élément a été retiré de l'ensemble. La méthode renvoie *faux*, si le retrait a échoué. Dans tous les cas de figure, la cardinalité maximale sera égale à la cardinalité actuelle (réajustement de l'ensemble), même si aucun élément n'est retiré de l'ensemble.

`unsigned int Retirer ( const Ensemble & unEnsemble );`

`unEnsemble` contient les éléments qu'il faut retirer à l'ensemble qui invoque la méthode, si cela est possible. Cette méthode retire les différents éléments de l'ensemble **`unEnsemble`** de l'ensemble qui invoque la méthode (existence des éléments). La méthode renvoie 0, si aucun élément n'a été retiré de l'ensemble. La méthode renvoie une valeur > 0, si au moins un élément de l'ensemble **`unEnsemble`** existe bien dans l'ensemble qui invoque la méthode et qu'il a été bien retiré de l'ensemble. En fait, la valeur rendue correspond au nombre d'éléments effectivement retirés de l'ensemble ⇒ diminution de la cardinalité actuelle de cette valeur. Dans tous les cas de figure (retrait ou pas), la cardinalité maximale restera inchangée (pas de réajustement au plus juste).

`int Reunir ( const Ensemble & unEnsemble );`

`unEnsemble` est l'ensemble qui va être utilisé pour effectuer la réunion avec l'ensemble qui invoque la méthode. Cette méthode rajoute à l'ensemble qui invoque la méthode tous les éléments de l'ensemble **`unEnsemble`** qui ne sont pas déjà présents dans l'ensemble courant. La méthode renvoie une valeur strictement négative si l'ensemble qui invoque la méthode a été réajusté pour contenir la réunion. Dans

ce cas, le réajustement se fait au plus juste. La valeur absolue rendue correspond au nombre d'éléments de l'ensemble **`unEnsemble`** effectivement rajoutés. La méthode renvoie une valeur strictement positive si l'ensemble qui invoque la méthode n'a pas été réajusté pour contenir la réunion. Dans ce cas, la valeur rendue correspond au nombre d'éléments de l'ensemble **`unEnsemble`** effectivement rajoutés. La méthode renvoie 0 si aucun élément de l'ensemble **`unEnsemble`** n'a été rajouté à l'ensemble courant pour réaliser l'union. Autrement dit, l'ensemble **`unEnsemble`** est inclus dans l'ensemble courant. Dans ce dernier cas, il n'y a pas de réajustement de la cardinalité maximale.

```
unsigned int Intersection ( const Ensemble & unEnsemble );
```

`unEnsemble` est l'ensemble qui va être utilisé pour effectuer l'intersection avec l'ensemble qui invoque la méthode. Cette méthode modifie l'ensemble qui invoque la méthode en retenant uniquement les éléments en commun entre les 2 ensembles (celui qui invoque la méthode et celui qui est en paramètre). La méthode renvoie le nombre d'éléments supprimés dans l'ensemble qui invoque la méthode pour bâtir l'intersection. Après l'opération d'intersection et quel que soit le cas de figure, l'ensemble est réajusté au plus juste.