Examen AP4A – Automne 2022

Durée 1h30

Préambule :

- Les exercices 1 et 2 sont indépendants et doivent être rédigés sur des feuilles séparées.
- Pour chaque méthode étudiée, préciser sa déclaration et sa définition.
- Il est conseillé de lire l'ensemble de l'énoncé d'un exercice avant de commencer à répondre aux questions.
- Toute tentative de triche entre étudiants ou en réutilisant du code sur internet sera sanctionnée.

Exercice 1 (10 points)

On souhaite créer une classe nommée ensemble_heterogene permettant de manipuler des ensembles dont le type des éléments est inconnu de la classe et susceptible de varier d'un élément à un autre.

Pour que la chose soit possible, on imposera simplement la contrainte suivante : tous les types concernés devront dériver d'un même type de base nommé base. Le type base sera supposé connu au moment où l'on définit la classe ensemble_heterogene.

La classe base disposera au moins d'une fonction virtuelle pure nommée affiche ; cette fonction devra être redéfinie dans les classes dérivées pour afficher les caractéristiques de l'objet concerné.

La classe ensemble_heterogene disposera des fonctions membre suivantes :

- ajoute pour ajouter un nouvel élément à l'ensemble (elle devra s'assurer qu'il n'existe pas déjà) ;
- appartient pour tester l'appartenance d'un élément à l'ensemble ;
- cardinal qui fournira le nombre d'éléments de l'ensemble.

De plus, la classe devra être munie d'un « itérateur », c'est-à-dire d'un mécanisme permettant de parcourir les différents éléments de l'ensemble. On prévoira 3 fonctions :

- init pour initialiser le mécanisme d'itération ;
- suivant qui fournira en retour le prochain élément (objet d'un type dérivé de base) ;
- existe pour préciser s'il existe encore un élément non examiné.

Enfin, une fonction nommée liste permettra d'afficher les caractéristiques de tous les éléments de l'ensemble (elle fera, bien sûr, appelle aux fonctions affiche des différents objets concernés).

Exercice 2 (10 points)

Une médiathèque prête d**i**fferents jeux de sociétés à ses utilisateurs, jeux d'échec, de dames, de cartes, de plateaux... Une application est développée pour faciliter le prêt de ces jeux et vérifier que le jeu est rendu complet. La classe principale de cette application est la classe abstraite "game" (donnée ci-dessous) que devront respecter toutes les classes de jeu de société :

```
class game {
	private :
		std::string name ;
	public :
		jeu(std::string n):name(n) {}
		// check that game is full
		virtual bool isFull() = 0 ;
} ;
```

On veut définir maintenant les classes "chess_game" et "lady_game" correspondant au jeu d'échec et au jeu de dames. Le jeu d'échec est un jeu constitué d'un plateau et de 32 pièces, alors que le jeu de dame dispose d'un plateau et de 40 pièces. La méthode "isFull()" est appelée lorsque le jeu est ramené à la médiathèque. Des questions sont posées à l'utilisateur pour qu'il renseigne le nombre d'éléments pour chaque catégorie (plateau et pièces). En fonction des réponses utilisateurs la méthode "isFull()" doit renvoyer si le jeu est complet ou non.

1. Ecrire les classes "chess_game" et "lady_game" pour que l'on puisse les instancier.

2. On veut maintenant pouvoir afficher des jeux d'échec ou de dames en surchargeant l'opérateur "<<" qu'une seule fois. Modifier les classes précédentes en conséquences en essayant d'intégrer le fait que cet opérateur pourrait être utilisé pour faire également une redirection de sortie vers un fichier.

On souhaite ensuite stocker tous les jeux de la bibliothèque dans une structure de donnée dynamique de type liste générique :

```
template <class T> class listgen {
	public :
		listgen() ;
		listgen(const listgen&) ;
		listgen& operator=(const listgen&) ;
		virtual ~listgen() ;

		// add an element to the end of the list
		listgen& operator+(const T&) ;

		// get the element with a given indice from the list
		T& operator[](int) ;

		// get the number of elements of the list
		int size() const ;
} ;
```

3. Expliquer à quoi sert le mot "const" associé à la fonction "size()".

4. Ecrire une fonction "test()" qui réalise les opérations suivantes :
   - instanciation d'un jeu d'échec "e" et d'un jeu de dames "d" ,
   - instanciation d'une liste "l" de jeux de bibliothèque
   - ajout des jeux "e" et "d" à la liste "l"
   - affichage ensuite des jeux contenus dans la liste "l"

5. On se rend compte alors que la mémoire n'est pas totalement libérée lorsque l'on quitte la fonction "test()". Pouvez-vous expliquer ce phénomène et proposer une solution pour le régler sans modifier la classe "listgen" ?