Examen AP4A – Automne 2025

Durée 1h30

Préambule :

- Les exercices 1 et 2 sont indépendants et doivent être rédigés sur des feuilles séparées.
- Pas de document, ni de calculatrice autorisé. Dictionnaire papier autorisé.

Exercice 1 (10 points)

Note : L'utilisation de la STL n'est pas autorisée pour cet exercice.

Amazon Simple Storage Service (Amazon S3) est un service de stockage d'objets (ou fichiers) dans des conteneurs nommés « *bucket* ». Un *bucket* est donc un conteneur caractérisé par un nom (ou identifiant) unique et un ensemble d'objets associés. Un objet est un fichier avec toutes les métadonnées qui le décrivent (par exemple la taille de l'objet, la date de création...). Chaque objet possède un nom qui joue le rôle d'identifiant unique dans le *bucket.*

1. Ecrire la classe « objectS3 » correspondant à un objet S3, qui contiendra les attributs suivants : « name » pour le nom de l'objet, « size » pour la taille des données et « value » qui sera de type « unsigned char* » et qui correspondra aux informations stockées dans l'objet.
2. Donner le constructeur par recopie de la classe « objectS3».
3. Est-il nécessaire d'écrire l'opérateur d'affectation ? Détailler.
4. Ecrire ensuite la classe « bucket » et ses attributs correspondant à un *bucket* S3 sans préciser de méthodes.
5. Donner la déclaration et définition de l'opérateur « << » qui permet d'ajouter un objet dans un *bucket*. Exemple d'utilisation :

```
bucket buc("mybucket") ;
buc << objectS3("myobject") ; // object "myobject" is put in the bucket buc
```

6. On souhaite définir l'opérateur « << » pour pouvoir afficher un *bucket* dans un flux de sortie, notamment : son nom et la liste des objets qui sont contenus dans le *bucket* (on ne précisera que le nom de l'objet). Modifier les classes précédentes pour réaliser cette fonctionnalité. Exemple d'utilisation :

```
bucket buc("mybucket") ;
std::cout << buc ; // display the bucket buc
```

7. On envisage maintenant de définir deux types de *buckets* : le *bucket* Standard accessible sans restriction par l'utilisateur au tarif de 0.023 €/Go, et le *bucket* Archive pour du stockage à long terme, au tarif de 0,004€/Go. Concevoir une architecture de classes permettant d'atteindre cet objectif. On précisera les classes, leurs attributs et les liens entre ces classes.
8. Ecrire la méthode « computeCost() » qui calcule le coût mensuel du bucket en fonction de son type.

Exercice 2 (10 points)

Amazon Simple Storage Service (Amazon S3) illustre la nécessité de gérer des objets de données variés stockés dans les buckets. Pour répondre aux besoins de stockage de différentes formes et types d'informations, il est essentiel de concevoir des structures génériques et flexibles.

Les exercices suivant proposent d'explorer la modélisation et la manipulation d'objets et de buckets de stockage à l'aide des templates, afin de rendre le système plus adaptable et réutilisable.

1. Définition d'une classe template
   Écrire un **template de classe** StorageObject<T> représentant un objet de stockage générique contenant :
   - un attribut name pour le nom de l'objet ;
   - un attribut data de type T pour les données ;
   - un attribut size représentant la taille des données.

2. Constructeur principal
   Écrire le **constructeur principal** du template StorageObject<T> qui initialise tous les attributs (name, data et size).

3. Fonction d'affichage générique
   Écrire une **fonction template** printObject qui prend un objet StorageObject<T> en paramètre (par référence constante) et affiche son nom et sa taille.

4. Définition d'un conteneur générique
   Écrire un **template de classe** StorageBucket<T> capable de contenir plusieurs objets StorageObject<T>.
   Vous pouvez utiliser un membre de type std::vector<StorageObject<T>> pour stocker ces objets.

5. Ajout d'un objet au conteneur
   Ajouter à la classe StorageBucket<T> une **méthode** addObject permettant d'ajouter un objet de type StorageObject<T> au bucket.

6. Surcharge de l'opérateur <<
   Définir l'**opérateur d'insertion <<** pour la classe StorageBucket<T> afin d'afficher :
   - le nom du bucket ;
   - la liste des noms des objets qu'il contient.

7. Spécialisation pour std::string
   Spécialiser la classe template StorageObject<std::string> de sorte que la taille (size) soit calculée automatiquement à partir de la longueur de la chaîne contenue dans data.

8. Question de réflexion
   Expliquez en quoi l'utilisation des **templates** est avantageuse dans le contexte d'un système de stockage générique, par rapport à une implémentation basée sur des classes fixes (non génériques).