AP4A Exam – Fall 2025

Duration: 1 hour 30 minutes

Preamble:

• Exercises 1 and 2 are independent and must be written on separate sheets.
• No documents or calculators are allowed; a paper dictionary is permitted.

Exercice 1 (10 points)

Note : The use of the STL is not allowed for this exercise

Amazon Simple Storage Service (Amazon S3) is an object (or file) storage service organized into containers called *"buckets"*. A bucket is therefore a container characterized by a unique name (or identifier) and a set of associated objects. An object is a file with all the metadata that describes it (for example, the object size, creation date, etc.). Each object has a name that serves as a unique identifier within the *bucket*.

1. Write the class "objectS3" corresponding to an S3 object, containing the following attributes: "name" for the object name, "size" for the data size, and "value", of type *unsigned char**, corresponding to the data stored in the object.
2. Provide the copy constructor for the class objectS3.
3. Is it necessary to write the assignment operator? Explain.
4. Then, write the class bucket and its attributes corresponding to an S3 bucket, without specifying methods.
5. Provide the declaration and definition of the operator << that allows adding an object to a bucket. Example of use:

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

6. We now want to define the operator << to display a bucket in an output stream, showing its name and the list of objects it contains (only the object names will be displayed). Modify the previous classes to implement this functionality. Example of use:

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

7. We now consider defining two types of buckets:
   - Standard bucket, accessible without restriction to the user, at a rate of €0.023/GB.
   - Archive bucket, for long-term storage, at a rate of €0.004/GB.

   Design a class architecture to achieve this goal. Specify the classes, their attributes, and the relationships between them.
8. Write the method computeCost() that calculates the monthly cost of the bucket depending on its type.

Exercice 2 (10 points)

Amazon Simple Storage Service (Amazon S3) illustrates the need to manage various data objects stored in buckets. To meet the storage needs of different forms and types of information, it is essential to design generic and flexible structures.

The following exercises explore the modeling and manipulation of storage objects and buckets using templates, to make the system more adaptable and reusable.

1. Definition of a Template Class

   Write a **class template** StorageObject<T> representing a generic storage object containing:

   - an attribute name for the object name;
   - an attribute data of type T for the data;
   - an attribute size representing the data size.

2. Main Constructor

   Write the **main constructor** of the StorageObject<T> template that initializes all attributes (name, data, and size).

3. Generic Display Function

   Write a **template function** printObject that takes a StorageObject<T> object as a constant reference parameter and displays its name and size.

4. Definition of a Generic Container

   Write a **class template** StorageBucket<T> capable of containing several StorageObject<T> objects. You may use a member of type std::vector<StorageObject<T>> to store these objects.

5. Adding an Object to the Container

   Add to the class StorageBucket<T> a method addObject that allows adding an object of type StorageObject<T> to the bucket.

6. Overloading the Operator <<

   Define the **insertion operator** << for the class StorageBucket<T> to display:

   - the bucket name;
   - the list of names of the objects it contains.

7. Specialization for std::string

   Specialize the class template StorageObject<std::string> so that the size (size) is automatically computed from the length of the string contained in data.

8. Reflective question

   Explain why the use of **templates** is advantageous in the context of a generic storage system, compared to an implementation based on fixed (non-generic) classes.