Object–Oriented Programming

Lab **9** : On Chapter Polymorphism

ENSIA 2024-2025

Exercise 1

An automatic checkout system for a supermarket chain needs to be completed (Exercise 1 Lab 6).

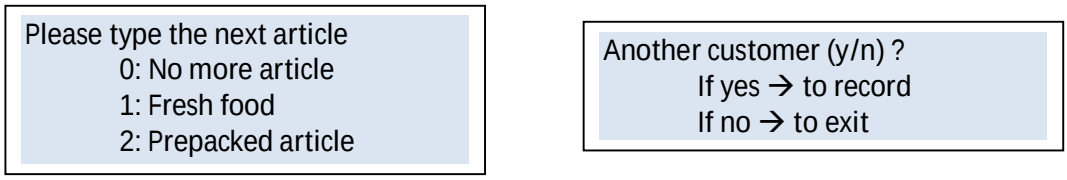
1- In the base class Product, define a virtual destructor and the virtual methods scanner () and printer ().

2- In the main, write the function record(), which registers and lists in a loop the products purchased in the store. The function creates an array of 100 pointers to the base class Product.

The checkout assistant should state if a prepacked or fresh product is going to be scanned (Fig. 1 - a):

- First, for each scanned product, the memory is dynamically allocated and referenced by the next pointer in the array.
- Then, after scanning all the purchased items, a list of the latter products is displayed.
- Finally, the prices of all the items are added, and the total amount is displayed.

3- In the main, write a program to simulate a supermarket checkout. The checkout assistant is used in a loop to state whether a new customer should be defined (Fig 1 - b). If so, the record () function is called. If not, the program ends.



(a) (b)

Fig 1. Dialog with receptionist

Exercise 2

We want to model the mathematical classes of Vector of integers and BitVector.

1- Define and implement the class Vector of integers, which is defined by the number of elements N in the vector and the array that contains the data. The member functions of this class are:

- *sumVect* which adds one vector to another (each element *i* of the result is $a_i + b_i$)
- *prodVect* which computes the product of one vector by another ($\sum_{i=0}^{n-1} a_i b_i$)
- any other needed functions

2- Define and implement the class BitVector, which is derived through public inheritance from the class Vector. The data of a BitVector can be only 1s and 0s. The member functions of this class are:

- *sumVect* which adds a BitVector to another one of same size. Each element of the resulting BitVector is computed as the logical or between a_i and b_i.

- *prodVect* which computes the "bit vector product" of one BitVector object by another. Each element of the resulting BitVector is computed as the logical and between a_i and b_i.
- A parameterized constructor.
- A print function.
- Any functions strictly needed to solve the questions below.

3- Write and test a driver program which should:

- Declare two Vector objects *vector1* and *vector2* (to be initialized with the values of your choice using setAll), print their contents, call the functions *sumVect* and *prodVect* on *vector1,* and write the resulting *vector1* after each operation.
- Declare two BitVector objects *bitVector1* and *bitVector2* (to be initialized with the values of your choice using setAll), print their contents, call the functions *sumVect* and *prodVect* on *bitVector1,* and write the resulting *bitVector1* after each operation.
- Reset all the objects to the initial values.
- Declare two pointers to Vector *vectPtr1* and *vectPtr2*, two pointers to BitVector *bitVectPtr*1 and *bitVectPtr*2, then aim them to *vector1*, *vector2*, bit*Vector1*, and bit*Vector2* respectively.
- Reset all the objects to the initial values.
- Use the pointers *vectPtr* and *bitVectPtr* to write respectively the contents of *vector1* and *bitVector1*, as well as the results of calling the functions *sumVect* and *prodVect* using pointers only.
- Reset all the objects to the initial values
- Aim *vectPtr*1 and *vectPtr*2 to *bitVector1* and *bitVector2,* write the contents of *bitVector1* and *bitVector2* using the pointers, then use these pointers to call the functions *sumVect* and *prodVect* and write their results.

4- Modify your classes by making the function *print* virtual and rerun your driver program. Notice the difference in the output when the base pointers are aimed at base class objects and when they are aimed at derived class objects.