UFPE – CENTRO DE INFORMÁTICA
ESTRUTURAS DE DADOS ORIENTADAS A OBJETOS
ATIVIDADE PRÁTICA

(Gustavo Carvalho – ghpc@cin.ufpe.br)

Exercícios selecionados de: KIRCH-PRINZ, U., PRINZ, P.
A Complete Guide to Programming in C++.
1a Edição. Editora Jones & Bartlett Learning, 2001.

Exercise 1

A supermarket chain has asked you to develop an automatic checkout system. All products are identifiable by means of a barcode and the product name. Groceries are either sold in packages or by weight. Packed goods have fixed prices. The price of groceries sold by weight is calculated by multiplying the weight by the current price per kilo. Develop the classes needed to represent the products first and organize them hierarchically. The Product class, which contains generic information on all products (barcode, name, etc.), can be used as a base class.

- The Product class contains two data members of type long used for storing barcodes and the product name. Define a constructor with parameters for both data members. Add default values for the parameters to provide a default constructor for the class. In addition to the access methods setCode() and getCode(), also define the methods scanner() and printer(). For test purposes, these methods will simply output product data on screen or read the data of a product from the keyboard.
- The next step involves developing special cases of the Product class. Define two classes derived from Product, PrepackedFood and FreshFood. In addition to the product data, the PrepackedFood class should contain the unit price and the FreshFood class should contain a weight and a price per kilo as data members. In both classes define a constructor with parameters providing default-values for all data members. Use both the base and member initializer.
- Define the access methods needed for the new data members. Also redefine the methods scanner() and printer() to take the new data members into consideration.
- Test the various classes in a main function that creates two objects each of the types Product, PrepackedFood and FreshFood. One object of each type is fully initialized in the object definition. Use the default constructor to create the other object.Test the get and set methods and the scanner() method and display the products on screen.

Exercise 2

The class hierarchy representing a supermarket chain's checkout system comprises the base class Product and the derived classes PrepackedFood and FreshFood. Your job is to test various cast techniques for this class.

- Define a global function isLowerCode() that determines which one of two products has the lower barcode and returns a reference to the product with the lower barcode.

- Define an array with three pointers to the base class Product. Dynamically create one object each of the types Product, PrepackedFood, and FreshFood. The three objects are to be referenced by the array pointers. Additionally define a pointer to the derived class FreshFood. Initialize the pointer with the address of a dynamically allocated object of the same class.
- Now call the method printer() for all four objects. Which version of printer() is executed?
- Perform downcasting to execute the correct version of printer() in every case. Display the pointer values before and after downcasting.
- Use the pointer of the derived class FreshFood to call the base class version of printer(). Perform an appropriate upcast.
- Test the function isLowerCode() by multiple calls to the function with various arguments. Output the product with the lower barcode value in each case.

Exercise 3

Implement exception handling for the Fraction class, which is used to represent fractions (see previous exercises). Dividing by 0 throws an exception that affects the constructor for the Fraction class and the operator functions / and >>.

- Define the exception class DivError, which has no data members, within the Fraction class. The exception class is of the following type Fraction::DivError. Add an appropriate exception specification to the declarations of the constructor and the operator functions / and >>.
- Change the definition of the constructor in the Fraction class. If the value of the denominator is 0, a DivisionByZero type exception should be thrown.
- Similarly modify the operator functions.
- Now write a main function to test the various exceptions.You will need to arrange three different try and catch blocks sequentially.
    - The first try/catch block tests the constructor. Create several fractions, including some with a numerator value of 0 and one with 0 as its denominator. The exception handler should issue the error message: “The denominator is 0!”.
    - The second try/catch block tests divisions. Use a statement to attempt to divide by 0. The corresponding exception handler should send the following error message to your standard output: “No division by zero!”.
    - The third try/catch block reads numerators and denominators of fractions in dialog. If the value of the denominator is 0, the denominator is read again. If the value is still 0, the following error message is output and the program terminates: “New denominator != 0: …”.