CS 474: Object Oriented Programming Languages and Environments

Spring 2017

Second C++ project

Due time: 11:59 pm on Saturday 4/29/2017

This goal of this project is to implement a portion of a library for collection classes. The library is roughly modeled after Smalltalk's abstract *Collection* class and some of its concrete subclasses. You are specifically responsible for implementing an abstract superclass *Collection*, and two concrete subclasses in C++. For sake of simplicity, your C++ classes will contain only integer numbers and no duplicate values will be allowed.

1. Abstract superclass *Collection* is the root of all concrete collection subclasses. This class defers instance implementation to its concrete subclasses; however, it does define some concrete methods that are will not be redefined in the subclasses.

2. Concrete *Collection* subclass *BST* will implement a collection as a binary search tree.

3. Concrete *Collection* subclass *Array*, will implement a collection as a fixed-length array.

Class *Collection* declares the following (public) deferred methods:

- *add()*—This method takes as input an integer element to be inserted in the receiver (i.e., an instance of a concrete *Collection* subclass). This method is defined in the subclasses; however, it will have no effect (except for printing an error message) in the *Array* subclass because arrays are fixed-length collections. The modified receiver is returned.

- *operator[]()*—This is the indexing operator. This function takes as input an integer index and returns the element at the index position in the receiver. If the index is out of bounds, an error message is printed on the standard error stream and an exception is thrown. (Do not worry about catching the exception; you can let program execution terminate as a result of this exception.) However, you must code this operator in such a way that it can be used in the left-hand side of an assignment operation to modify an element of the receiver.

In addition the *Collection* class defines the following concrete functions:

- A default constructor, a copy constructor and a virtual destructor.

- A virtual copying scheme.

- *map()*—This public function takes as input a function parameter *fn*. The parameter function *fn* takes as input an integer and returns an integer. Function *map()* applies function *fn* to all elements contained in the receiver. Each element in the receiver is replaced by the value returned by *fn* for that element. The modified receiver is returned.

- *contains()*—This public function takes as input an integer and returns a boolean indicating whether the receiver contains the argument integer or not.

- *operator=()*—This the assignment operator. This function takes as input a collection instance of the same type as the receiver, that is, the right-hand side in an assignment. The function deep copies the argument into the receiver. The modified receiver is returned.

Finally, class *Collection* declares an integer data member *size_*, which returns the number of elements contained in the receiver. *Collection* defines a public accessor for *size_*; however, only functions in the *Collection* class *and its subclasses* are allowed to modify this data member.

Abstract superclass *Collection* has two sibling subclasses named *BST* and *Array*. Class *BST* is a binary search tree implementation of abstract class *Collection*. *BST* implements all the deferred *Collection* methods, a default constructor, a copy constructor, a virtual destructor and virtual copying scheme, including method *copy()*. You are required to allow clients of *BST* to invoke the inherited methods *map()* and *contains()* but you cannot redefine these inherited methods in *BST*. The assignment operator, copy constructor and virtual copy method should always perform a deep copy of the receiver.

When coding the indexing operator for the *BST* class, return the i-th element you encounter in an in-order tree traversal. The first element you encounter will be at position 0. Finally, recall that a binary search tree is a tree subject to the following two conditions: (1) each node can have at most 2 children, and (2) nodes in the left subtree of a node n have values less than n's value and nodes in n's right subtree have values greater than n.

Class *Array* is similar to *BST*; however, it is a traditional *Array* class. This class does not define a default constructor. Instead, it defines a constructor with an integer argument *n*, which is the size of the array to be allocated. The constructor allocates an *int* array of size *n* dynamically. Inherited method *add()* should be defined to print an error message on the standard error stream.

Implementation notes. In C++ you define an abstract class by deferring the definition of one or more virtual member functions. Deferred functions are called *pure virtual member functions* and are denoted by the `=0` syntax in the function declaration appearing in the class definition. *You are not allowed to redefine either function map() or contains() in the two Collection subclasses below.* Finally, method *add()* must be coded in such a way that their invocations can be cascaded. You are not allowed to use C++ libraries, such as the Standard Template Library (STL). Needless to say, your code should avoid memory leaks and dangling pointers at all cost.

Your must work alone on this project. Students are not allowed to discuss designs or share code with each other. Make sure to test your code on the g++ compiler before you submit. However, students are encouraged to use Piazza to post or answer questions about specific aspects of the project. Just remember never, never, ever to post code on Piazza.