# CS 474: Object Oriented Programming Languages and Environments

Fall 2012

*Second C++ project*

*Due time:* 9:00 pm on Saturday 12/8/2012

*(Project specification revised on 12/5/2012)*

For this project you must implement an abstract class called *Collection* in C++. This class defines a protocol for a number of concrete *Collection* subclasses that will hold integer values. In C++ you can 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. Your abstract *Collection* class will defer the definition of functions *add()*, *remove()*, the assignment operator *operator=()*, and the indexing operator *operator[]()*. Method *add()* takes as input two integers, denoting the element to be inserted in the collection and the index position where the insertion should take place. This method is defined in the subclasses; however, it will have no effect (except for printing an error message) in the *Array* subclass. Method *remove()* takes as input an integer to be removed. No action is taken if the integer is not contained in the receiver or if the receiver is an *Array* instance. Both methods return the modified receiver. Define the methods in such a way that their invocations can be cascaded.

In addition, *Collection* will define concrete implementations of functions *iterate()*, which applies an argument function to all elements of the receiver, and *contains()*, which checks whether an argument integer is contained in the receiver. *You are not allowed to redefine either function iterate() or contains() in the two Collection subclasses below.* In addition, *Collection* defines appropriate constructors, a virtual destructor, and a virtual *copy()* method. Finally, *Collection* will have exactly one an integer data member *size_*, which holds the size of the receiver (i.e., the number of integer numbers stored therein). *Collection* defines a public accessor for *size_*; however, only functions in the *Collection* class *and its subclasses* are allowed to modify this data member.

In addition, you are required to code two concrete *Collection* subclasses named *OrderedCollection* and *Array*. *OrderedCollection and Array are sibling subclasses of Collection.* Class *OrderedCollection* is a C++ version of the homonymous class defined in Smalltalk. Thus, this class defines sequenceable collections that are stored contiguously but have variable size. The initial *capacity* of instances of this class is exactly 4 elements. *However, instances are initially empty in the sense that the none of the elements in the 4-element allocation is used to store an integer.* When the collection is full (i.e., when trying to insert a fifth integer), a new allocation is made of double the original size, the elements of the previous collection are copied, and the old collection is deallocated. You are not required to follow strictly the Smalltalk implementation. For instance, you should not implement methods *makeRoomAtFirst()* and *makeRoomAtLast()*; however, you should have instance variables *first* and *last*. *In addition, you are required to support the "grow" functionality of Smalltalk's ordered collections, although this functionality will be hidden from your class's clients.* Also, you should override method *operator[]()* to take into account that the first element in the collection is not stored in the first position of the collection. *As with Smalltalk, the first element will be stored in the index position given by data member first. Likewise, the last element of the collection will be stored in the position given by data member last. All the elements of an OrderedCollection instance are stored between first and last.* Finally, you are required to allow clients of *OrderedCollection* to invoke the inherited methods *iterate()* and *contains()* but you cannot redefine these methods in *OrderedCollection*.

Class *Array* is a traditional *Array* class. Inherited methods *add()* and *remove()* should be overridden to print an error message on the standard error stream.

The concrete subclasses must implement all the deferred functions of *Collection*, including the virtual destructor and virtual copying scheme. The copying methods should always perform a deep copy of the receiver. You must avoid all memory leaks and dangling pointers.

*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 the Blackboard discussion board to post or answer questions about specific aspects of the project.

Implementation note: I could not get the dynamic cast of the *rhs* parameter in function *operator=()* to work with a target data type *const OrderedCollection&* in g++. To make dynamic cast work, I had to declare *rhs* to be of type *Collection&*, that is, a non-const parameter. I suggest that you do the same.

Good luck!