**Assignment 1 Report**

**Purpose**

The purpose of the program was to introduce various data structures as well as practice using various features of C++, such as encapsulation and inheritance. The types of data structures used were vectors and linked-lists, which are both mutable sets of data. Abstract classes and their pure-virtual functions were used as well. This represents one feature unique to object-oriented programming: inheritance. Abstract classes are defined as those containing at least one pure-virtual function, and must be overridden. In this lab, VectorJar and LListJar were sub-classes of the parent class, Jar. There were also elements of I/O involved, namely outputting various jars and the results from merge, intersection, and difference operations.

**Program Description**

The program represents “marbles” in a “jar,” using vectors and linked-lists. The user can perform a variety of actions on a given jar using the various menu options. The program remembers one jar, which can either be created by n random marbles or from a given file. Once the jar is loaded, the user can remove marbles with specific colors and sizes. The user can also perform various set operations with two jars saved to files. Namely, they are union, intersection, and difference. Once the operation is complete, the jar is loaded into memory and output to a file. The jars behave like sets, in which no duplicates are allowed.

**Code Design – Data Structures**

Data structures used in the program were mainly vectors and linked-lists. Sets and arrays may have been used instead, but the latter would be immutable. In this program, there are several examples of linear search algorithms. For example, is\_in\_jar(const Marble&) searches through the vector/list one element at a time to determine whether a marble exists in the jar. Other examples of this algorithm are seen in the merge, intersection, and difference operations. Linear search was chosen because of code simplicity as well as the fact that a jar can contain no more than 12 marbles.

**Code Design – C++ Classes/Functions**

There are four main classes in this program: Jar, VectorJar, LListJar, and Marble. VectorJar and LListJar are both derived from Jar, which is an abstract class used as a template for the two. These classes represent vectors and linked-lists filled with Marble objects. The classes add additional functionality that is otherwise absent from vectors and lists, such as duplicate marble checking, the number of marbles in the jar of given color/size, and various set operations. In the different jar classes, everything is declared public except for the vector/linked-list itself. All of the functions within these classes are designed to be interfaced with externally. The vector/linked-list are declared private because this could otherwise allow unwanted operations to be performed on the jars. In class Marble, helper functions and member variables are declared private while everything else is public.

**Code Design – Exception Handling**

The program will provide appropriate error messages if a VectorJar is empty when the user tries to access the remove\_any() function when the jar is empty or when the user tries to remove a marble that does not exist. In LListJar, an exception is thrown if remove\_any() is called when the jar is empty. In all of these cases the errors are not caught, causing the program to crash. When handling file I/O, an error message is printed if any file operations fail. However, this will simply result in an empty jar (in the case of file input) or no file written (in the case of output), and the program will continue running.

**Algorithm Discussion**

This program has several algorithms that can perform operations on the various jars, the most intricate of which were probably the set operations. In calculating the intersection of two jars, the list of marbles in the current jar is cleared. Then, the algorithm uses nested for loops to compare each marble in jar1 to each marble in jar2, performing iterations. (If the jars are of the same size, big-O asymptotic notation of the worst-case scenario can be expressed as ). If the two elements match, the algorithm checks if the element is already in the jar, and if not, it adds the element to the jar. The resulting jar is an intersection of the two specified jars, representing , where and represent two different jars.

**C++ Programming Constructs**

The most obvious object-oriented element used in this program is inheritance. Both VectorJar and LListJar derive from Jar. Since Jar is an abstract class, it cannot be used by itself, and all pure virtual functions must be defined elsewhere. Abstract classes serve as a template for other classes, since a class that extends an abstract class will not function until the virtual functions are defined. This ensures completeness of any subclasses of Jar.

**Bugs Fixed in** LListJar

Function intersect(const Jar&, const Jar&):

The line if(jar1.get\_elem(i) == jar2.get\_elem(i)) was corrected to if(jar1.get\_elem(i) == jar2.get\_elem(j)). The wrong indexes were being used for elements in jar2, causing numerous errors.

Function add(const Marble&):

The line if(is\_in\_jar) was changed to if(!is\_in\_jar). This contradicted the specification that a jar cannot contain duplicates.

Function clear():

Simplified the code to just ll.clear();