8.1 ADT

An ADT or abstract data type defines a way of interacting with data: it specifies only how the ADT can be used and says nothing about the implementation of the structure. An ADT is conceptually more abstract than a Java interface specification or C++ list of class member function prototypes, and should be expressed in some formal language (such as mathematics).

A data structure is a way of storing data that implements certain operations. When choosing a data structure for your ADT, you might consider many issues such as whether the data is static or dynamic, whether the deletion operation is important, and whether the data is ordered. In general

A data structure should take ownership of its data.

In particular, it is responsible for recycling the storage of the data.

8.2 Basic Collections

There are three basic collections.

1. The basic collection is often called a bag. It stores objects with no ordering of the objects and no restrictions on them.

2. Another unstructured collection is a set where repeated objects are not permitted: it holds at most one copy of each item. A set is often from a predefined universe.

3. A collection where there is an ordering is often called a list. Specific examples include an array, a vector and a sequence. These have the same idea, but vary as to the methods they provide and the efficiency of those methods.

The Bag ADT might have:

- accessors methods such as size, countOccurrence, possibly an iterator (which steps through all the elements);

- modifier methods such as add, remove, and addAll; and

- also a union method which combines two bags to produce a third.
8.3 The Array Implementation

A common implementation of a collection is a partially filled array. This is often expanded every time it needs to be, but rarely shrunk. It has a pointer/counter which keeps track of where the real data ends.

```
0  1  2  3  4  5  6
Amy Bo Carl Dana ? ? ?
```

count=4

Sample Code

An array-based implementation of a set of strings.

```
StringSet.h
StringSet.cpp
TestStringSet.cpp
```