Chapter C1: Data Structures

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, whether the data is ordered, etc.

C1.1 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 accessor methods such as size, countOccurrences, and an iterator; modifier methods such as add, remove, and addAll; and also a merge method that combines two bags to produce a third.

C1.2 Stacks and Queues

A linear data structure is one which is ordered. There are two special types with restricted access: a stack and a queue.

A stack is a data structure of ordered items such that items can be inserted and removed only at one end (called the top). It is also called a LIFO structure: last-in, first-out. The standard (and usually only) modification operations are:

- push: add the element to the top of the stack
- pop: remove the top element from the stack and return it
If the stack is empty and one tries to remove an element, this is called \textit{underflow}.
Another common operation is called \textit{peek}: this returns a reference to the top element on the stack (leaving the stack unchanged).

A queue is a linear data structure that allows items to be added only to the rear of the queue and removed only from the front of the queue. Queues are \textit{FIFO} structures: first-in first-out. The two standard modification methods are:

- insert the item at the \textit{rear} of the queue
- delete and return the item at the \textit{front} of the queue

\section*{C1.3 Dictionary}

A dictionary is also known as an \textit{associative data structure} or a \textit{map}. The dictionary ADT supports:

- \textbf{insert}: Insert new item
- \textbf{lookup}: Look up item based on key; return access/boolean

There are several implementations: for example, red-black trees do both operations in $O(\log n)$ time. But one can do better by allowing the dictionary to be unsorted. See hash tables later.

\section*{C1.4 Priority Queue}

The (min)-priority queue ADT supports:

- \textbf{insert}: Insert new item
- \textbf{extractMin}: Remove and return item with minimum key

Other common methods include \texttt{decreaseKey} and \texttt{delete}.

Note that a priority queue provides a “natural” way to sort: simply

\emph{insert all elements and then repeatedly extractMin}

\section*{Exercises}

1. \texttt{ADD ME}