In the discussion of the list object in Chapter 5, it was observed that linked list representations employing pointer variables and dynamic storage allocation provide a considerable amount of flexibility over their static counterparts. Linear linked lists are useful for applications requiring fast sequential access to the data. However, for applications in need of a higher degree of random access to the data, the list object can prove to be ineffective. In this chapter, the tree object is introduced to fulfill such requirements.

The tree object is one of the most fundamental and universally applied concepts in computer science. Trees have many variations and implementations, and many more applications. In this chapter we develop the concepts of ADT trees and present their implementations as object-oriented programs in C++. We concentrate on pointer-based implementations of tree structures, relying heavily on the powerful facilities of pointer variables and dynamic memory allocation, although pointer-less variations are also presented.

8.1 Fundamental Definitions and Terminology

We begin with the development of trees by introducing the basic definitions and terminology associated with this very important abstract data type.

**Definition 8.1** A tree is a nonempty and finite collection of data objects called nodes such that:

1. there is a single node or
2. there is a node designated as the root connected by edges to a finite set of trees that are the offspring of the root.

A node is an abstract data object that contains the basic branching information of a tree and may also contain additional data attributes. Consider, for example, a tree-object based information retrieval system for the maintenance of a customer mailing list. A node in this instance might contain information (data attributes) such as the customer’s name, address, and phone number. In addition to data, a node typically contains the information linking it to other nodes in the tree. This interconnectivity information forms the edges between nodes.
The nodes in a tree have a parent-child (predecessor-successor) relationship. Each node, with the exception of the root of the entire tree, has exactly one node preceding it, which is referred to as its parent or ancestor. All nodes succeeding a given node are referred to as its children or descendants. All nodes that share the same parent are siblings. A node that has no children (subtrees) is called a leaf or terminal and is identified as the last node in a (sub)tree. All other nodes that contain at least one child are referred to as internal nodes. The degree of a node corresponds to the number of subtrees or children of a node while the degree of a tree is the maximum of the degrees of its nodes. It follows that a node of degree zero is a leaf node while a tree of degree zero is a single node.

A path is any specifiable list of successively connected nodes in a tree. An important defining property of a tree is that there exists only one path connecting the root node to any other node in the tree. In Chapter 10, we will examine the graph structure for which this restriction is lifted.

The level of a node in a tree is equivalent to the number of ancestors for that node. Alternately, the level of a node is given as the number of nodes along the path between it and the root of the entire tree (including the root). The height of a tree is the maximum of the levels of the nodes. That is, the height is the length of the longest path from the root to any leaf node. The total path length of a tree is the sum of all lengths of all paths from the root of the tree to all the leaf nodes. This is equivalent to the sum of the levels of all the leaf nodes of the tree. The balance of a node in a tree is a relative measure of the number of descendants between the children of the node. The height balance of an entire tree is the balance among the nodes through all levels of the tree.

As an illustration of these concepts, consider the tree diagram of Figure 8.1. The nodes of this tree are labeled with the alphabetic characters A through M. Node A, the root of the entire tree, has two children, B and C. Node B, itself, is the root of one of node A's subtrees and has nodes D, E, and F as children. Node A, therefore, is the parent of node B as well as node C. Node B has one sibling, node C, while node K has three siblings. Node K, like nodes G, E, H, I, J, L, and M, is a leaf node, since it has no children. Node F is at level 2 since it has two ancestors. This number corresponds to the number of nodes encountered as one travels along the path from node F to the root node A. The height of this entire tree is 3 while the total path length is 19. The degrees of nodes A, B, and C are 2, 3 and 4, respectively. The degree of node K is zero, as it is a leaf node, while the degree of the entire tree is 4.

Tree structures are naturally recursive, as evidenced herein by the choice of definition and adopted terminology. Indeed, Definition 8.1 is recursive, as it defines a tree in terms of trees. Any node may itself be a tree (a single node) or the root of a set of trees (subtrees). In Figure 8.1, node A is found as the root of a tree that corresponds to the entire tree. Root node A is connected to a set of two trees, the roots of these trees being nodes B and