15-5 Trees

Trees are used extensively in computer science to represent algebraic formulas; as an efficient method for searching large, dynamic lists; and for such diverse applications as artificial intelligence systems and encoding algorithms.

Topics discussed in this section:

Basic Tree Concepts
Terminology
Binary Trees
Binary Search Trees
Binary Tree Example

Note

A tree consists of a finite set of elements, called nodes, and a finite set of directed lines, called branches, that connect the nodes.

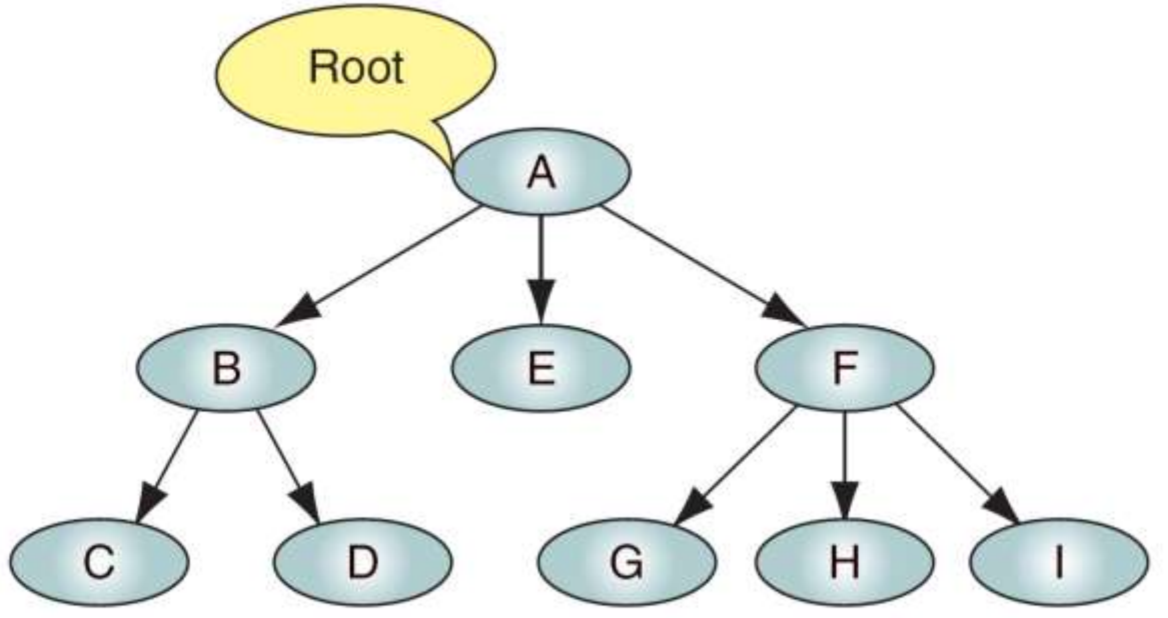
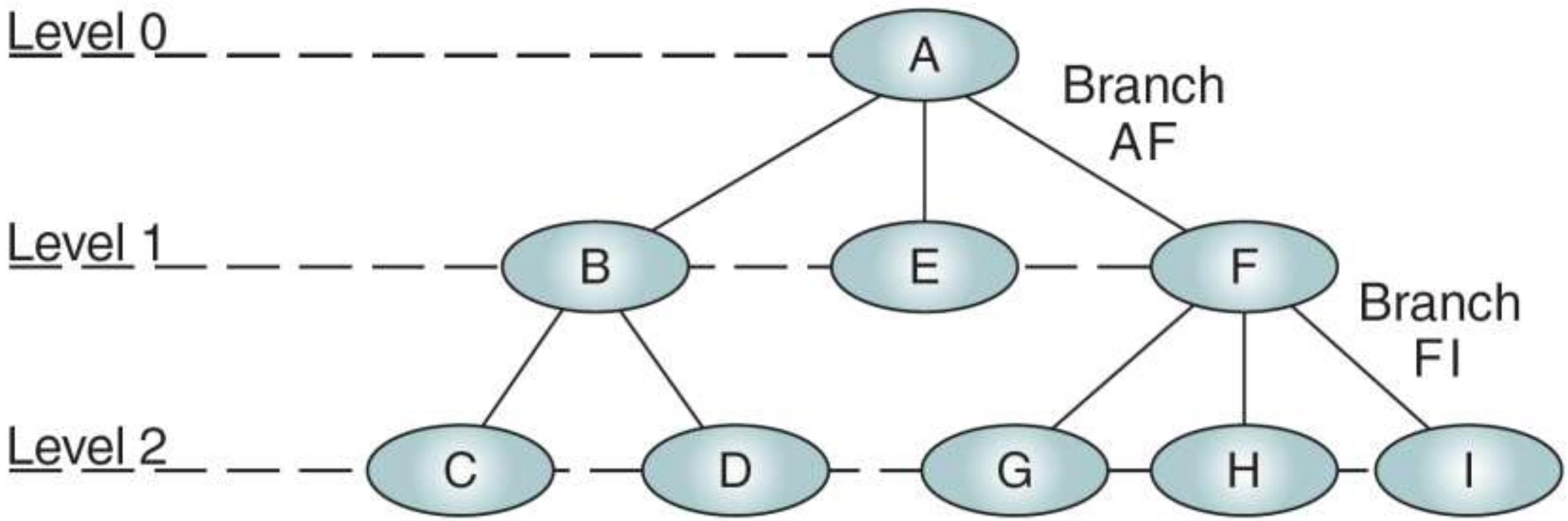


FIGURE 15-30 Tree



Root: A
Parents: A, B, F
Children: B, E, F, C, D, G, H, I

Siblings: {B, E, F}, {C, D}, {G, H, I}
Leaves: C, D, E, G, H, I
Internal nodes: B, F

FIGURE 15-31 **Tree Nomenclature**

Note

The level of a node is its distance from the root. The height of a tree is the level of the leaf in the longest path from the root plus 1.

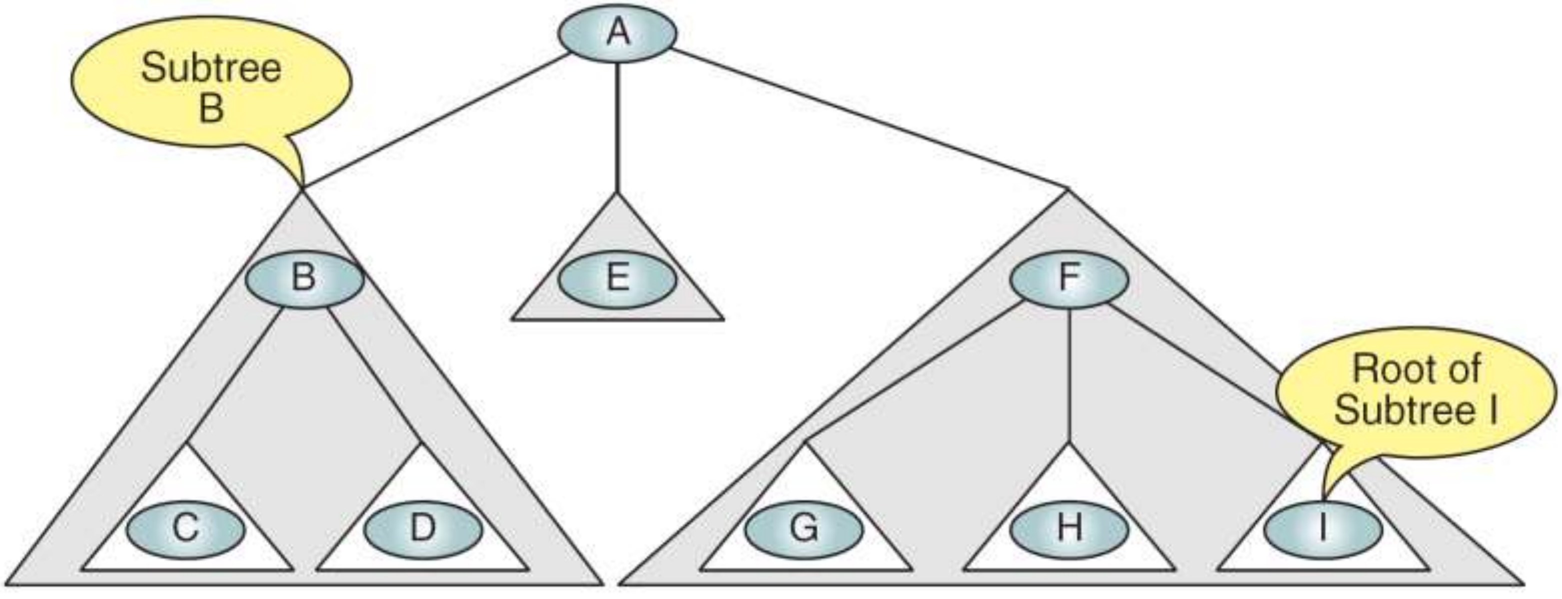


FIGURE 15-32 **Subtrees**

Note

A tree is a set of nodes that either:

1. **Is empty, or**
2. **Has a designated node, called the root, from which hierarchically descend zero or more subtrees, which are also trees**

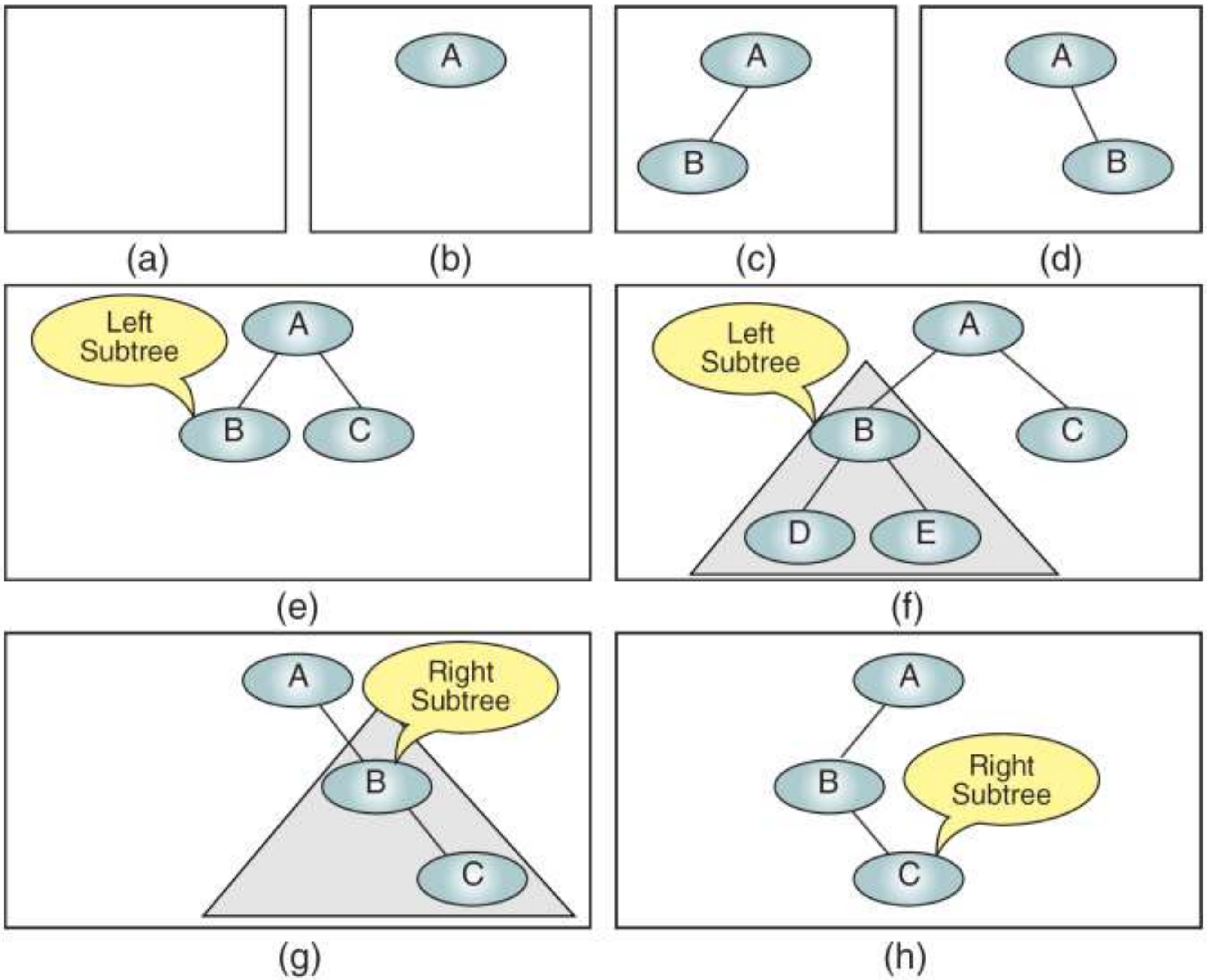
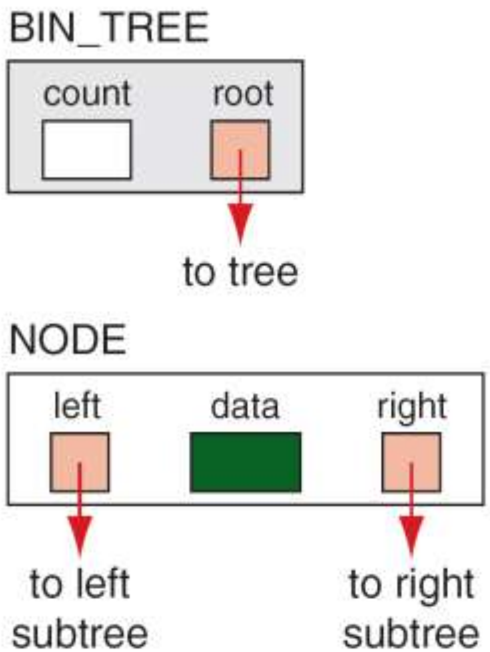


FIGURE 15-33 **Collection of Binary Trees**




```
typedef struct
  {
   int    count;
   NODE* root;
  } BIN_TREE;

typedef struct node
  {
   int            data;
   struct node* left;
   struct node* right;
  } NODE;
```

FIGURE 15-34 **Binary Tree Data Structure**

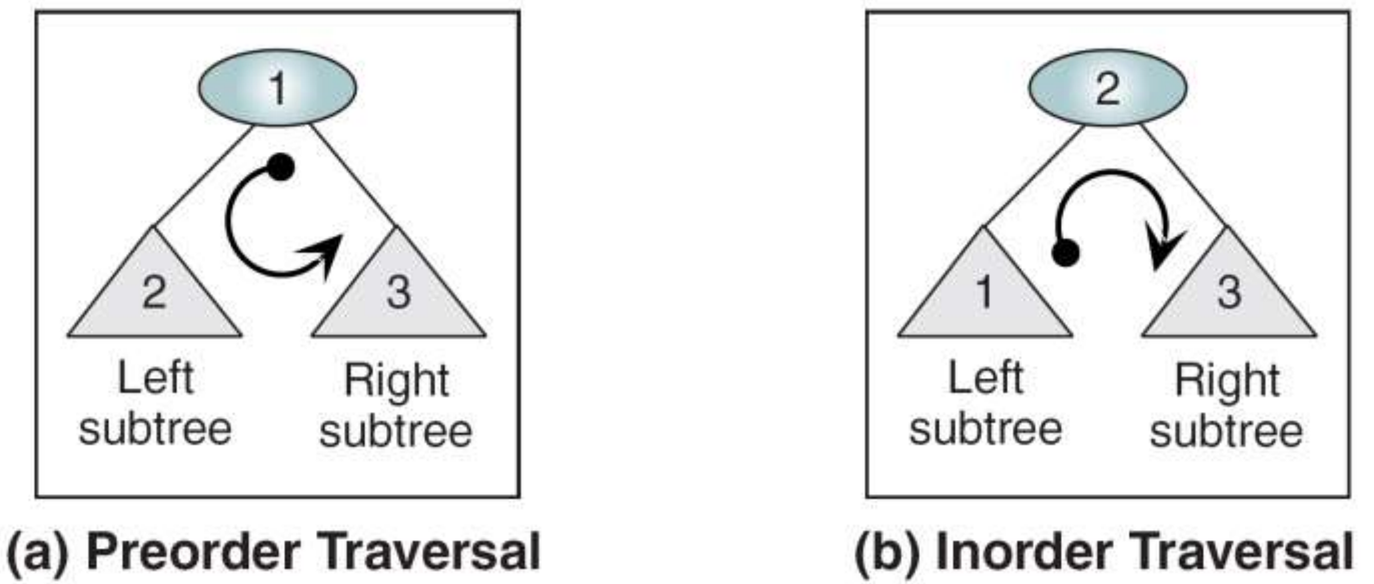


FIGURE 15-35 Binary Tree Traversals

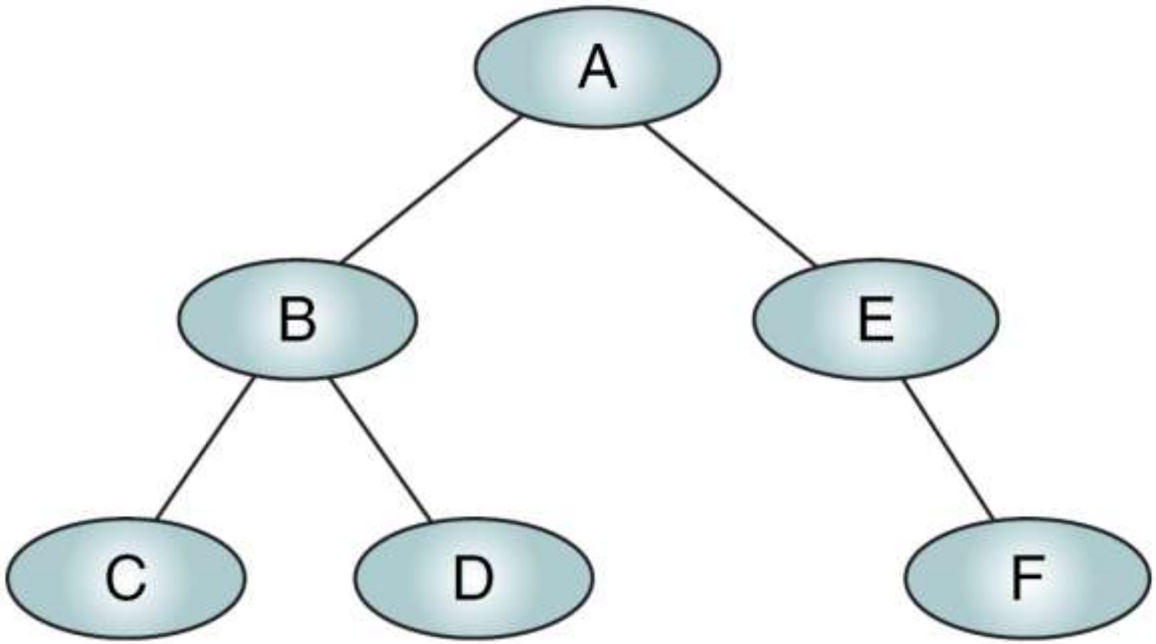


FIGURE 15-36 **Binary Tree for Traversals**

Note

In the preorder traversal, the root is processed first, before its subtrees.

PROGRAM 15-19 Preorder Traversal of a Binary Tree

```
/* Traverse a binary tree and print its data (integers)
      Pre  root is entry node of a tree or subtree
      Post each node has been printed
*/
void preOrder (NODE* root)
{
// Statements
   if (root)
      {
       printf("%4d", root->data);
       preOrder (root->left);
       preOrder (root->right);
      } // if
   return;
}  // preOrder
```

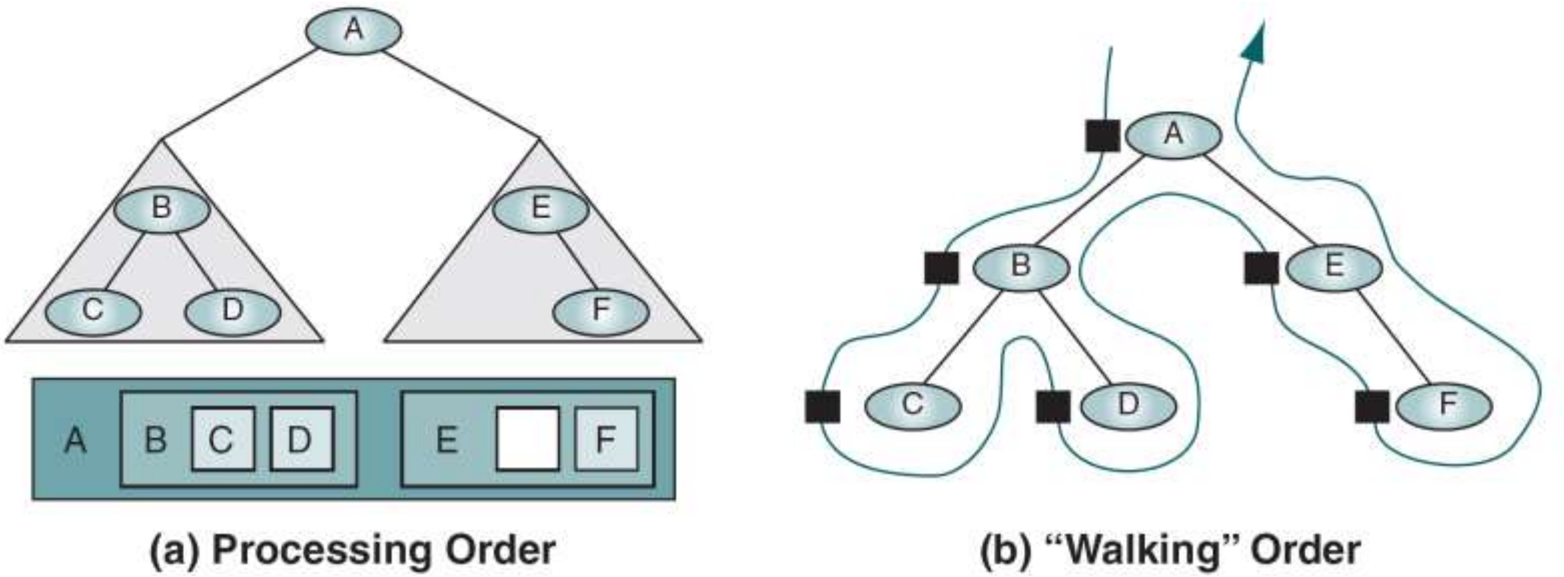


FIGURE 15-37 **Preorder Traversal—A B C D E F**

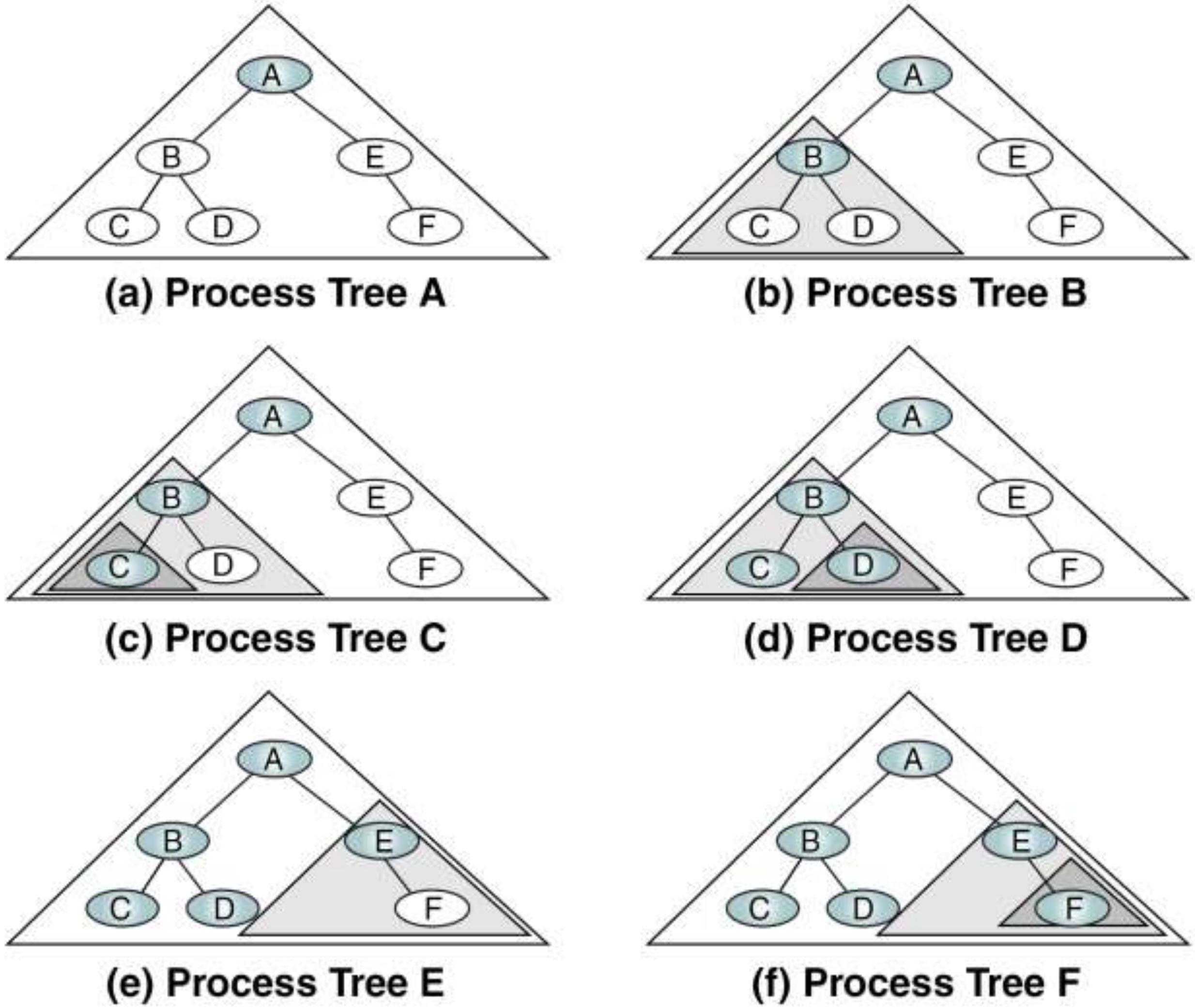


FIGURE 15-38 **Algorithmic Traversal of Binary Tree**

PROGRAM 15-20 Inorder Traversal of a Binary Tree

```c
/* Traverse a binary tree and print its data (integers)
      Pre  root is entry node of a tree or subtree
      Post each node has been printed
*/
void inOrder (NODE* root)
{
// Statements
   if (root)
      {
       inOrder (root->left);
       printf("%4d", root->data);
       inOrder (root->right);
      } // if
   return;
}  // inOrder
```

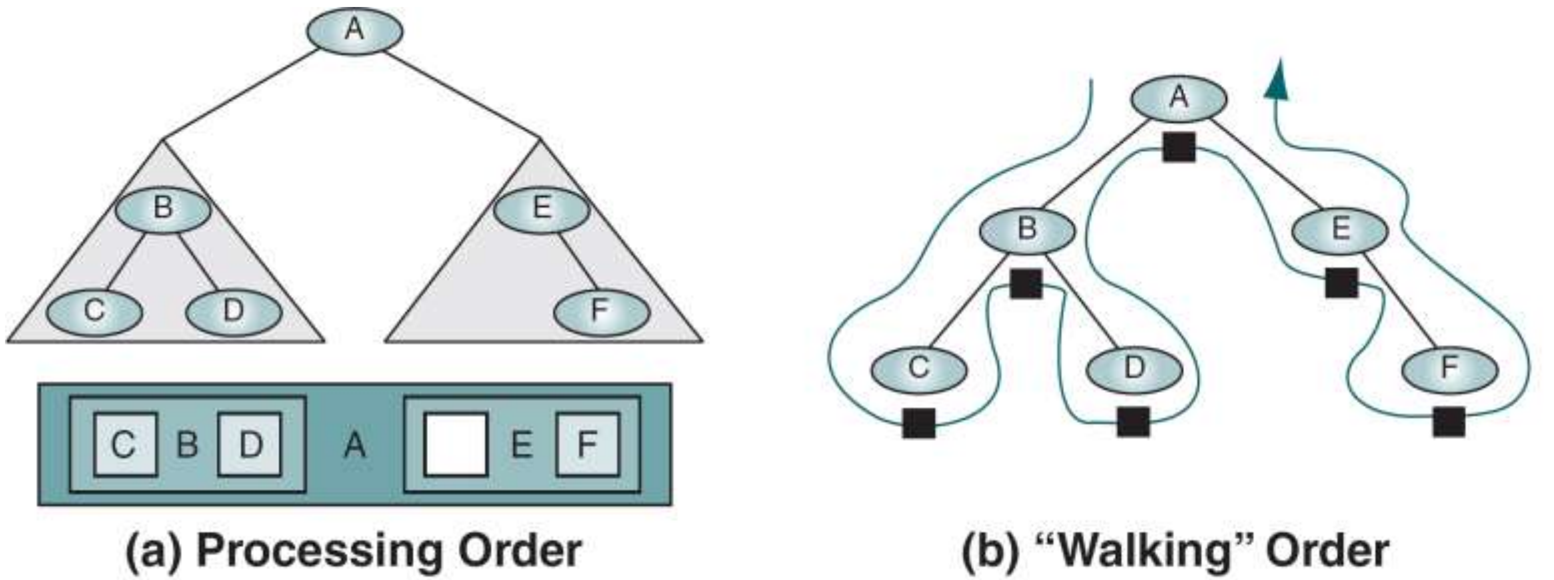


FIGURE 15-39 **Inorder Traversal—C B D A E F**

Note

In the inorder traversal, the root is processed between its subtrees.

Note

In a binary search tree, the left subtree contains key values less than the root, and the right subtree contains key values greater than or equal to the root.

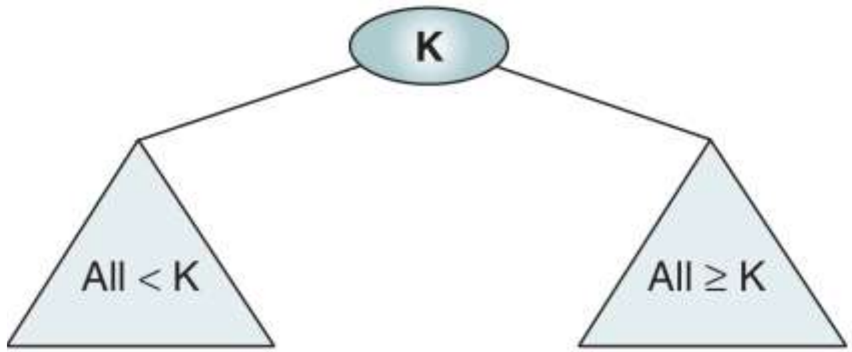


FIGURE 15-40 **Binary Search Tree**

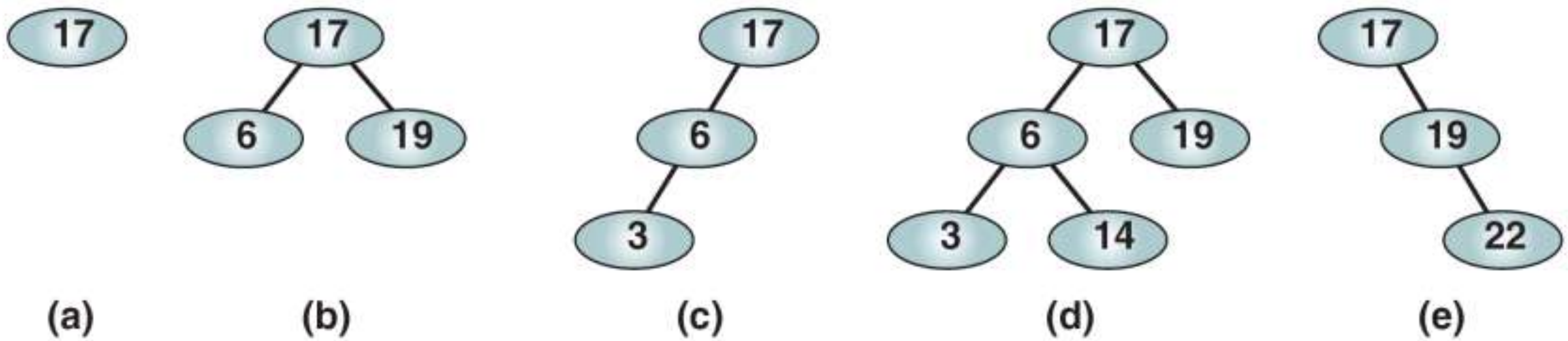


FIGURE 15-41 Valid Binary Search Trees

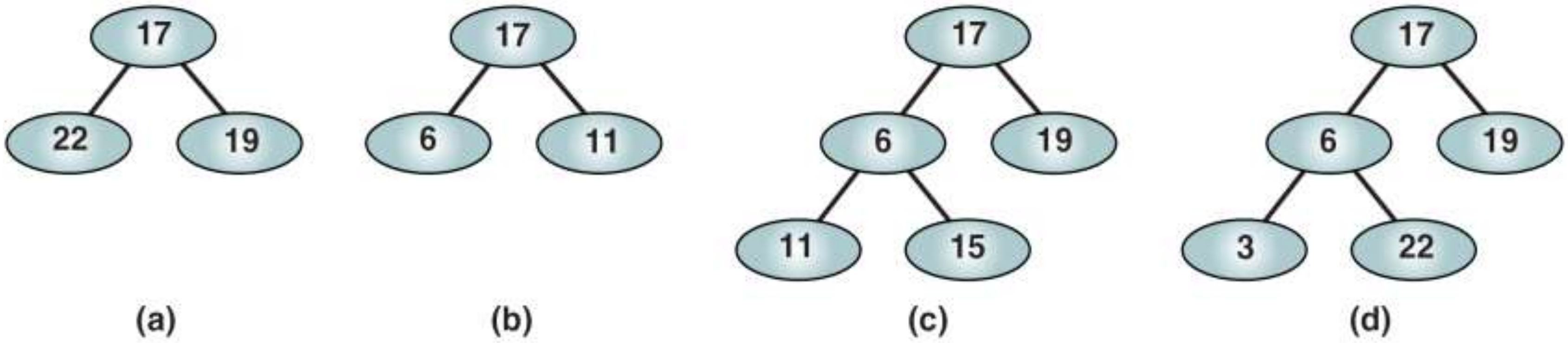


FIGURE 15-42 **Invalid Binary Search Trees**

Note

All BST insertions take place at a leaf or a leaflike node.

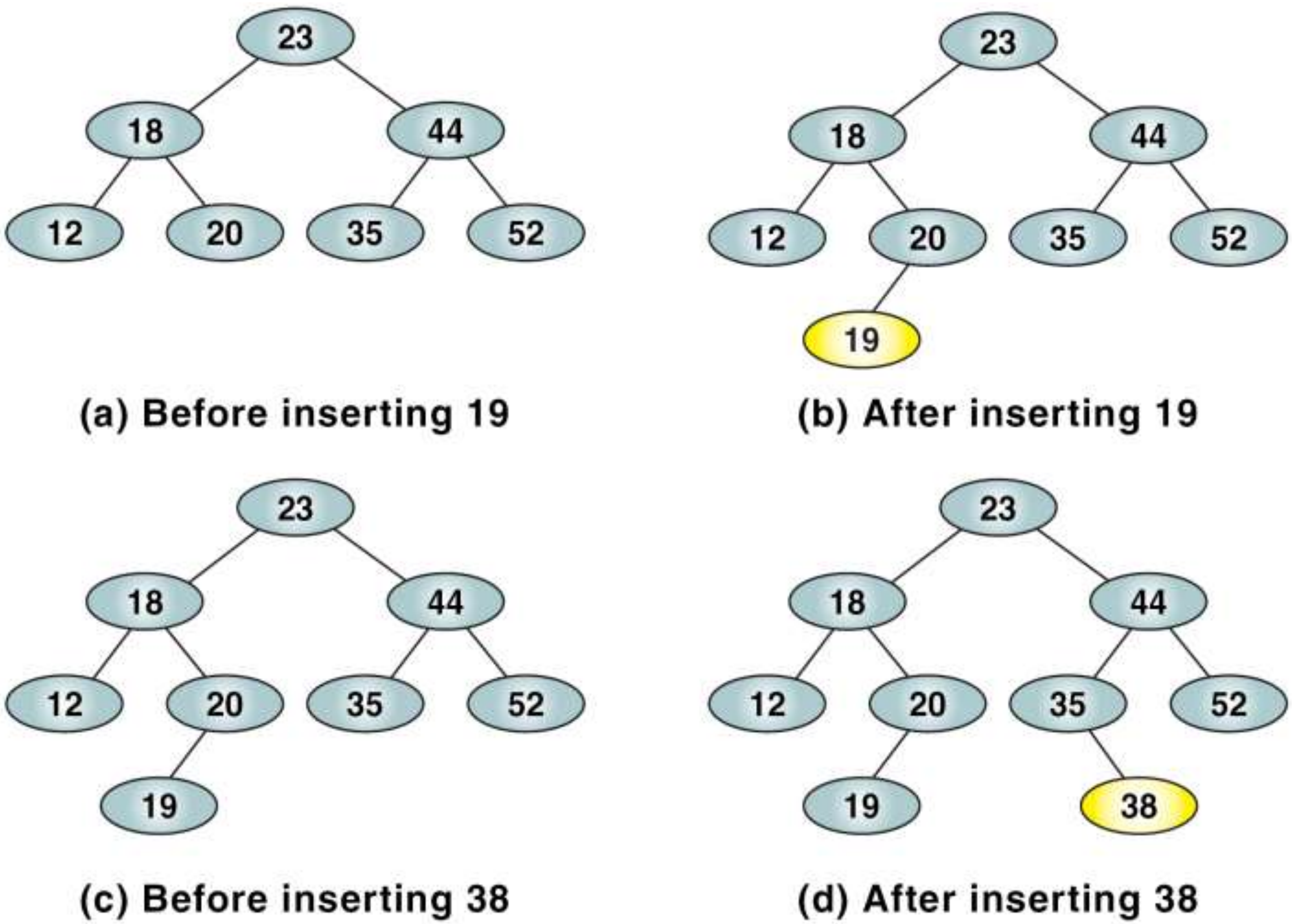


FIGURE 15-43 **BST Insertion**

PROGRAM 15-21 Binary Tree Insert Function

```
/* ==================== BST_Insert ====================
   This function uses recursion to insert the new data
   into a leaf node in the BST tree.
      Pre    Application has called BST_Insert, which
             passes root and data pointer
      Post   Data have been inserted
      Return pointer to [potentially] new root
*/
NODE* BST_Insert (BST_TREE* tree,
                  NODE* root,     int dataIn)
{
// Local Declarations
   NODE* newPtr;

// Statements

   if (!root)
      {
       // NULL tree -- create new node
       newPtr = malloc(sizeof (NODE));
```

PROGRAM 15-21 Binary Tree Insert Function

```
        if (!newPtr)
           printf("Overflow in Insert\n"), exit (100);
        newPtr->data = dataIn;
        newPtr->left = newPtr->right = NULL;
        return newPtr;
       } // if

    // Locate null subtree for insertion
    if (dataIn < root->data)
        root->left = BST_Insert(tree, root->left,
                                dataIn);
    else
       // new data >= root data
        root->right = BST_Insert(tree, root->right,
                                 dataIn);
    return root;
} // BST_Insert
```

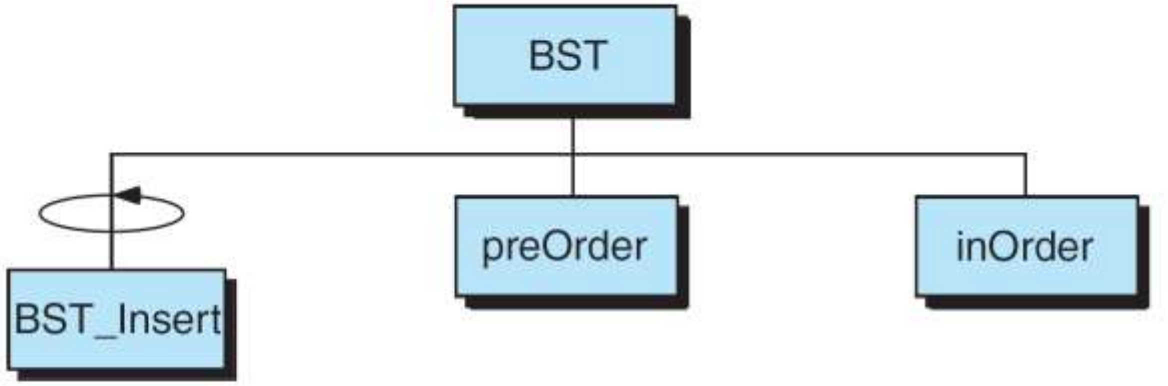


FIGURE 15-44 **Binary Tree Example**

PROGRAM 15-22 Binary Tree Example

```
/* Demonstrate the binary search tree insert and
   traversals.
      Written by:
      Date:
*/
#include <stdio.h>
#include <stdlib.h>

// Global Declarations
typedef struct node
  {
   int            data;
   struct node* left;
   struct node* right;
  } NODE;

typedef struct
  {
   int   count;
   NODE* root;
```

PROGRAM 15-22 Binary Tree Example

```
  } BST_TREE;

// Function Declarations
void  preOrder    (NODE* root);
void  inOrder     (NODE* root);
NODE* BST_Insert (BST_TREE* tree,
                  NODE* root,     int data);

int main (void)
{
// Local Declarations
   int      numIn;
   BST_TREE tree;

// Statements
   printf("Please enter a series of integers."
          "\nEnter a negative number to stop\n");

   tree.count = 0;
   tree.root  = NULL;
```

PROGRAM 15-22 Binary Tree Example

```
    do
       {
        printf("Enter a number: ");
        scanf("%d", &numIn);
        if (numIn > 0)
           {
            tree.root = BST_Insert
                (&tree, tree.root, numIn);
            tree.count++;
           } // if
       } while (numIn > 0);

    printf("\nData in preOrder: ");
    preOrder (tree.root);

    printf("\n\nData in inOrder:  ");
    inOrder (tree.root);

    printf("\n\nEnd of BST Demonstration\n");
    return 0;
 }  // main
```

PROGRAM 15-22 Binary Tree Example

```
Results
Please enter a series of integers.
Enter a negative number to stop
Enter a number: 45
Enter a number: 54
Enter a number: 23
Enter a number: 32
Enter a number: 3
Enter a number: -1

Data in preOrder:    45  23   3  32  54

Data in inOrder:      3  23  32  45  54

End of BST Demonstration
```