```
//-------------------------------------------------------------------------
//Class Tree defines a tree implemented using a dynamically allocated
//array.
//-------------------------------------------------------------------------
template <class T>
class Tree {
  struct Node {
    Node* LNode;          //Left node (subtree)
    T Key;                //Key
    Node* RNode;          //Right node (subtree);
    //-----------------------------------------------------------------
    //Constructor for struct Node
    //-----------------------------------------------------------------
    Node(T K):LNode(0),Key(K),RNode(0){}
    //-----------------------------------------------------------------
    //Prints the Key
    //-----------------------------------------------------------------
    void Print(ostream& o){o << endl; o << Key;}
    //-----------------------------------------------------------------
    //Prints the Key indented according to depth of the Node.
    //-----------------------------------------------------------------
    void Print(ostream& o,int depth)
    {  o << endl;
       for (int a=0;a<depth;a++) o << "  ";
       o << Key;
    }
  };
  Node* Root;             //Root of tree
  //-----------------------------------------------------------------
  //Reclaim storage for all Nodes using a postorder traversal
  //-----------------------------------------------------------------
  void Kill(Node* N){if(N==0) return;Kill(N->LNode);Kill(N->RNode);delete N;}
  //-----------------------------------------------------------------
  //Insert a new Key
  //-----------------------------------------------------------------
  Node* Insert(Node* N,T Key)
  {  if(N==0) return new Node(Key);
     if (Key==N->Key) return N;
     if (Key<N->Key) {
        N->LNode=Insert(N->LNode,Key);
     } else {
        N->RNode=Insert(N->RNode,Key);
     }
     return N;
  }
```

```
  //-------------------------------------------------------------
  //Print the Keys using a postorder traversal
  //-------------------------------------------------------------
  void PostOrderPrint(ostream& o,Node* N)
  {  if (N==0) return;
     PostOrderPrint(o,N->LNode);
     PostOrderPrint(o,N->RNode);
     N->Print(o);
  }
  //-------------------------------------------------------------
  //Print the Keys using a preorder traversal
  //-------------------------------------------------------------
  void PreOrderPrint(ostream& o,Node* N)
  {  if (N==0) return;
     N->Print(o);
     PreOrderPrint(o,N->LNode);
     PreOrderPrint(o,N->RNode);
  }
  //-------------------------------------------------------------
  //Print the Keys using an inorder traversal
  //-------------------------------------------------------------
  void InOrderPrint(ostream& o,Node* N)
  {  if (N==0) return;
     InOrderPrint(o,N->LNode);
     N->Print(o);
     InOrderPrint(o,N->RNode);
  }
  //-------------------------------------------------------------
  //Print the Keys using an inorder traversal and printing the keys
  //according to their depth
  //-------------------------------------------------------------
  void InOrderPrint(ostream& o,Node* N,int depth)
  {  if (N==0) return;
     InOrderPrint(o,N->LNode,depth+1);
     N->Print(o,depth);
     InOrderPrint(o,N->RNode,depth+1);
  }
public:
  //-------------------------------------------------------------
  //Constructor
  //-------------------------------------------------------------
  Tree():Root(0){}
  //-------------------------------------------------------------
  //Destructor
  //-------------------------------------------------------------
  ~Tree(){Kill(Root);}
```

```
  //-------------------------------------------------------------
  //Invoke private member function Insert to Insert the Key
  //-------------------------------------------------------------
  void Insert(T Key){ Root=Insert(Root,Key); }
  //-------------------------------------------------------------
  //Invoke private member function PostOrderPrint to print the Keys
  //stored in the tree using a postorder traversal.
  //-------------------------------------------------------------
  void PostOrderPrint(ostream& o)
  {  o << endl;
     o << "Postorder Traversal";
     PostOrderPrint(o,Root);
     o << endl;
  }
  //-------------------------------------------------------------
  //Invoke private member function PreOrderPrint to print the Keys
  //stored in the tree using a preorder traversal.
  //-------------------------------------------------------------
  void PreOrderPrint(ostream& o)
  {  o << endl;
     o << "Preorder Traversal";
     PreOrderPrint(o,Root);
     o << endl;
  }
  //-------------------------------------------------------------
  //Invoke private member function InOrderPrint to print the Keys
  //stored in the tree using an inorder traversal.
  //-------------------------------------------------------------
  void InOrderPrint(ostream& o)
  {  o << endl;
     o << "Inorder Traversal";
     InOrderPrint(o,Root);
     o << endl;
   }
  //-------------------------------------------------------------
  //Invoke private member function InOrderPrint(ostream& o,Node *N,int depth)
  //to format and print the Keys stored in the Tree using an inorder
  //traversal and indenting each node according to its depth
  //-------------------------------------------------------------
  void GraphPrint(ostream& o)
  {  o << endl;
     o << "Graphical Representation";
     InOrderPrint(o,Root,0);
     o << endl;
  }
};
```