Solution to Exercise 10.1
See Figure 10.10 on page 215 of the book by Cormen, Leiserson, Rivest, and Stein: Introduction to Algorithms (2nd Edition). MIT Press (2001).

Solution to Project 10.1
The key idea is that we can make decisions where to go next based on where we have come from. When we arrive at a node coming from its parent, then we must always go left. When we arrive at a node from its left child, then we must visit the node, and then go to its right child. When we arrive at a node coming from its right child, then we must go to the parent of the node. Special cases arise if nodes do not have certain children, or when we are done traversing the subtree. Details are in the code below.

```
#include <stdlib.h>

typedef enum {FROM_ABOVE,FROM_LEFT_CHILD,FROM_RIGHT_CHILD} State;

typedef struct Node Node;

struct Node
{
     Node* parent;
     Node* left;
     Node* right;
};

void visit(Node *v)
{
}

void inorder( Node *root )
// inorder traversal starting from the root node using constant space
{
     State state;
     Node *node;

     state = FROM_ABOVE;
     node = root;

     while( NULL != node ) {
          switch( state ){
               case FROM_ABOVE:
                    if( NULL != node->left )
                         node = node->left;
                    else
                         state = FROM_LEFT_CHILD;
                    break;
               case FROM_LEFT_CHILD:
                    visit(node);
                    if( NULL != node->right ) {
                         node = node->right;
                         state = FROM_ABOVE;
                    }
                    else
                         state = FROM_RIGHT_CHILD;
                    break;
               case FROM_RIGHT_CHILD:
                    if( node == root )
                         node = NULL;
                    else {
                         if( node->parent->left == node ) {
                              // node is the left child of its parent
                              node = node->parent;
                              state = FROM_LEFT_CHILD;
                         }
                         else {
                              // node is the right child of its parent
                              node = node->parent;
                              state = FROM_RIGHT_CHILD;
                         };
                    };
                    break;
          };
     };
}
```