```c
#include<stdio.h>
#include<stdlib.h>
#define bool int

/* A binary tree tNode has data, pointer to left child
and a pointer to right child */
struct tNode
{
int data;
struct tNode* left;
struct tNode* right;
};

/* Structure of a stack node. Linked List implementation is
stack. A stack node contains a pointer to tree node and a p
next stack node */
struct sNode
{
struct tNode *t;
struct sNode *next;
};

/* Stack related functions */
void push(struct sNode** top_ref, struct tNode *t);
struct tNode *pop(struct sNode** top_ref);
bool isEmpty(struct sNode *top);

/* Iterative function for inorder tree traversal */
void inOrder(struct tNode *root)
{
/* set current to root of binary tree */
struct tNode *current = root;
struct sNode *s = NULL; /* Initialize stack s */
bool done = 0;

while (!done)
{
    /* Reach the left most tNode of the current tNode */
    if(current != NULL)
```

```
40:     {
41:     /* place pointer to a tree node on the stack before travers
42:         the node's left subtree */
43:     push(&s, current);
44:     current = current->left;
45:     }
46:
47:     /* backtrack from the empty subtree and visit the tNode
48:     at the top of the stack; however, if the stack is empty
49:     you are done */
50:     else
51:     {
52:     if (!isEmpty(s))
53:     {
54:         current = pop(&s);
55:         printf("%d ", current->data);
56:
57:         /* we have visited the node and its left subtree.
58:         Now, it's right subtree's turn */
59:         current = current->right;
60:     }
61:     else
62:         done = 1;
63:     }
64: } /* end of while */
65: }
66:
67: /* UTILITY FUNCTIONS */
68: /* Function to push an item to sNode*/
69: void push(struct sNode** top_ref, struct tNode *t)
70: {
71: /* allocate tNode */
72: struct sNode* new_tNode =
73:             (struct sNode*) malloc(sizeof(struct sNode));
74:
75: if(new_tNode == NULL)
76: {
77:     printf("Stack Overflow \n");
78:     getchar();
```

```
 79:     exit(0);
 80: }
 81:
 82: /* put in the data */
 83: new_tNode->t = t;
 84:
 85: /* link the old list off the new tNode */
 86: new_tNode->next = (*top_ref);
 87:
 88: /* move the head to point to the new tNode */
 89: (*top_ref) = new_tNode;
 90: }
 91:
 92: /* The function returns true if stack is empty, otherwise f
 93: bool isEmpty(struct sNode *top)
 94: {
 95: return (top == NULL)? 1 : 0;
 96: }
 97:
 98: /* Function to pop an item from stack*/
 99: struct tNode *pop(struct sNode** top_ref)
100: {
101: struct tNode *res;
102: struct sNode *top;
103:
104: /*If sNode is empty then error */
105: if(isEmpty(*top_ref))
106: {
107:     printf("Stack Underflow \n");
108:     getchar();
109:     exit(0);
110: }
111: else
112: {
113:     top = *top_ref;
114:     res = top->t;
115:     *top_ref = top->next;
116:     free(top);
117:     return res;
```

```
118: }
119: }
120:
121: /* Helper function that allocates a new tNode with the
122: given data and NULL left and right pointers. */
123: struct tNode* newtNode(int data)
124: {
125: struct tNode* tNode = (struct tNode*)
126:                       malloc(sizeof(struct tNode));
127: tNode->data = data;
128: tNode->left = NULL;
129: tNode->right = NULL;
130:
131: return(tNode);
132: }
133:
134: /* Driver program to test above functions*/
135: int main()
136: {
137:
138: /* Constructed binary tree is
139:             1
140:         / \
141:         2    3
142:     / \
143:     4    5
144: */
145: struct tNode *root = newtNode(1);
146: root->left    = newtNode(2);
147: root->right   = newtNode(3);
148: root->left->left = newtNode(4);
149: root->left->right = newtNode(5);
150:
151: inOrder(root);
152:
153: getchar();
154: return 0;
155: }
156:
```