ASSIGNMENT 2

AIM:

Construct a threaded binary search tree by inserting value in the given order and traverse it in inorder traversal using threads.

OBJECTIVE:

Traverse a threaded binary search tree

THEORY:

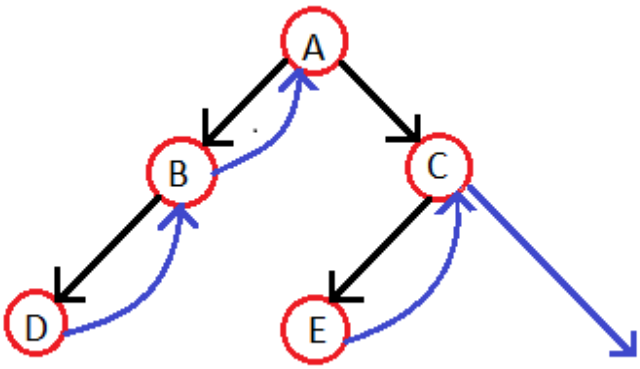
Inorder traversal of a Binary tree can either be done using recursion or with the use of a auxiliary stack. The idea of threaded binary trees is to make inorder traversal faster and do it without stack and without recursion. A binary tree is made threaded by making all right child pointers that would normally be NULL point to the inorder successor of the node (if it exists).

There are two types of threaded binary trees.

Single Threaded: Where a NULL right pointers is made to point to the inorder successor (if successor exists)

Double Threaded: Where both left and right NULL pointers are made to point to inorder predecessor and inorder successor respectively. The predecessor threads are useful for reverse inorder traversal and postorder traversal.

The threads are also useful for fast accessing ancestors of a node.



ALGORITHM:-

```
void inOrder(struct Node *root)

{
    struct Node *cur = leftmost(root);
    while (cur != NULL)
    {
        printf("%d ", cur->data);

        // If this node is a thread node, then go to
        // inorder successor
        if (cur->rightThread)
            cur = cur->right;
        else // Else go to the leftmost child in right subtree
            cur = leftmost(cur->right);
    }
}
```

CODE :

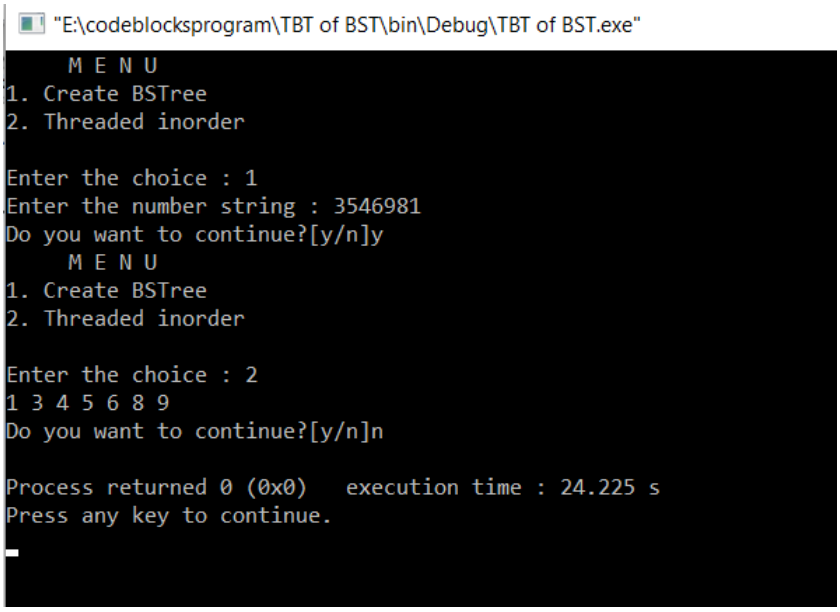
```
#include <iostream>

using namespace std;

struct TreeNode{
    int num;
    TreeNode *lptr,*rptr;
    bool lth,rth;
};

class BST{
    private:

    public:
        TreeNode *headNode = NULL;
        TreeNode * root = NULL;
        BST(){
            headNode = new TreeNode();
            headNode->lptr = root;
            headNode->rptr = headNode;
            headNode->lth = false;
            headNode->rth = false;
            headNode->num = NULL;
        }


        string input;
        void createBST(){
            cout<<"Enter the number string : ";
```

```
cin>>input;
for(int i=0; input[i]!='\0'; i++){
    TreeNode * nn = new TreeNode;
    int currNum = (int)input[i] - 48;
    nn->num = currNum;
    // cout<<nn->num<<" ";
    nn->lptr = headNode;
    nn->rptr = headNode;
    nn->lth = true;
    nn->rth = true;
    if(root == NULL){
        root = nn;
    }
    else{
        bool isleft;
        TreeNode* currNode = root;
        TreeNode* parent = currNode;
        while(currNode != headNode){
            isleft = false;
            parent = currNode;
            if(currNode->num > nn->num){
                if(currNode->lth != true){
                    currNode = currNode->lptr;
                    isleft = true;
                }
                else{
                    currNode = currNode->lptr;
                    isleft = true;
                    break;
                }

            }
            else{
                if(currNode->rth != true){currNode = currNode->rptr;}
                else{
                    currNode = currNode->rptr;
                    break;
                }
            }
        }

        if(isleft){
            nn->lptr = parent->lptr;
            parent->lth = false;
            parent->lptr = nn;
            nn->rptr = parent;
        }
        else{
            nn->rptr = parent->rptr;
            parent->rth = false;
```

```
                parent->rptr = nn;
                nn->lptr = parent;
            }
        }

    }
}
TreeNode *subRoot = root;
void RecInfix(TreeNode *subRoot){
    if(subRoot!=NULL){
        RecInfix(subRoot->lptr);
        cout<<subRoot->num;
        RecInfix(subRoot->rptr);
    }
    else
        return;
}

TreeNode* leftmost(TreeNode *node){
    //cout<<"sumedh2";
    while(node->lth != true){
        //cout<<"sumedh3  ";
        // cout<<node->num <<" test " ;
        node = node->lptr;

    }
    return node;
}

void inorder(TreeNode * rooot){
    TreeNode *current = rooot;
    //cout<<current->num<<" test ";
    current = leftmost(current);
    while(current != headNode){
        cout<<current->num<<" ";
        if(current->rth)
            current = current->rptr;
        else
            current = leftmost(current->rptr);
    }
}


void Inorder(){
    TreeNode* temp = root;
    while(temp->lth == false){
        temp = temp->lptr;
    }
    while(temp != headNode){
        cout<<temp->num;
```

```
            if(temp->rth){
                temp = temp->rptr;
            }
            else{
                temp = temp->rptr;
                while(temp->lth!=true)
                    temp = temp->lptr;
            }
        }
    }
};

int main()
{
    BST bst1;
    char ch;
    do{
        cout<<"     M E N U     "<<endl;
        cout<<"1. Create BSTree "<<endl<<"2. Threaded inorder "<<endl;
        cout<<endl<<"Enter the choice : ";
        int choice;
        cin>>choice;
        switch(choice){
            case 1 : bst1.createBST();
            break;
            case 2 :bst1.inorder(bst1.root);
                    cout<<endl;
            break;
            default :  cout<<"Sorry Wrong choice!!"<<endl;
        }
        cout<<"Do you want to continue?[y/n]";
        cin>>ch;
    }while(ch=='Y' || ch=='y');
    return 0;
}
```

OUTPUT :



CONCLUSION :

Threaded trees make in-order tree traversal a little faster, because you you have guaranteed O(1) time to access the next node. This is opposed to a regular binary tree where it would take O(log n) time, because you have to "climb" up the tree and then back down.