**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

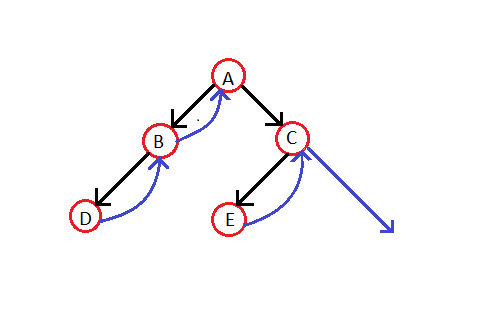
**THEORY:-**

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>

#include<stdlib.h>

using namespace std;

class TBT

{

private:

int data;

int lbit,rbit;

TBT\*lchild,\*rchild;

public:

TBT\*create\_TBT(TBT\*,TBT\*);

void inorder\_TBT(TBT\*,TBT\*);

void preorder\_TBT(TBT\*,TBT\*);

};

void TBT::inorder\_TBT(TBT\*root,TBT\*header)

{

int flag=0;

TBT\*trav;

trav=root;

while(trav!=header)

{

while(trav->lbit!=1&&flag==0)

{

trav=trav->lchild; // go to left

}

cout<<" "<<trav->data; // ptint leftmost data

if(trav->rbit==0)

{

trav=trav->rchild;

flag=0;

}

else

{

trav=trav->rchild;

flag=1;

}

}

}

TBT\*TBT::create\_TBT(TBT\*root,TBT\*header)

{

TBT\*trav,\*temp,\*p;

int attached\_flag=0;

char ans;

while(1)

{

trav=root;

if(root==NULL)

{

temp=new TBT();

cout<<"\nEnter the data ";

cin>>temp->data;

temp->lbit=1;

temp->rbit=1;

temp->lchild=header;

temp->rchild=header;

root=temp;

}

else

{

temp=new TBT();

cout<<"\nEnter the data ";

cin>>temp->data;

temp->lbit=1;

temp->rbit=1;

attached\_flag=0;

trav=root;

while(attached\_flag==0)

{

if(trav->data<temp->data&&trav->rbit==0)

{

trav=trav->rchild;

}

else

if(trav->data<temp->data&&trav->rbit==1)

{

trav->rbit=0;

p=trav->rchild;

trav->rchild=temp;

temp->rchild=p;

temp->lchild=trav;

attached\_flag=1;

}

if(trav->data>temp->data&&trav->lbit==0)

{

trav=trav->lchild;

}

else

if(trav->data>temp->data&&trav->lbit==1)

{

trav->lbit=0;

p=trav->lchild;

trav->lchild=temp;

temp->lchild=p;

temp->rchild=trav;

attached\_flag=1;

}

}

attached\_flag=0;

}

cout<<"\nDo you want to attach more nodes [y/n] ";

cin>>ans;

if(ans=='n'||ans=='N')

break;

}

return root;

}

int main()

{

int choice;

TBT\*root=NULL,obj,\*header=NULL;

do

{

cout<<"\n1.Create TBT";

cout<<"\n2.Inorder TBT";

cout<<"\n3.Exit";

cout<<"\nEnter your choice :";

cin>>choice;

switch(choice)

{

case 1: root=obj.create\_TBT(root,header);

break;

case 2: obj.inorder\_TBT(root,header);

break;

case 3: exit(0);

break;

default:

cout<<"Invalid Choice..\n";

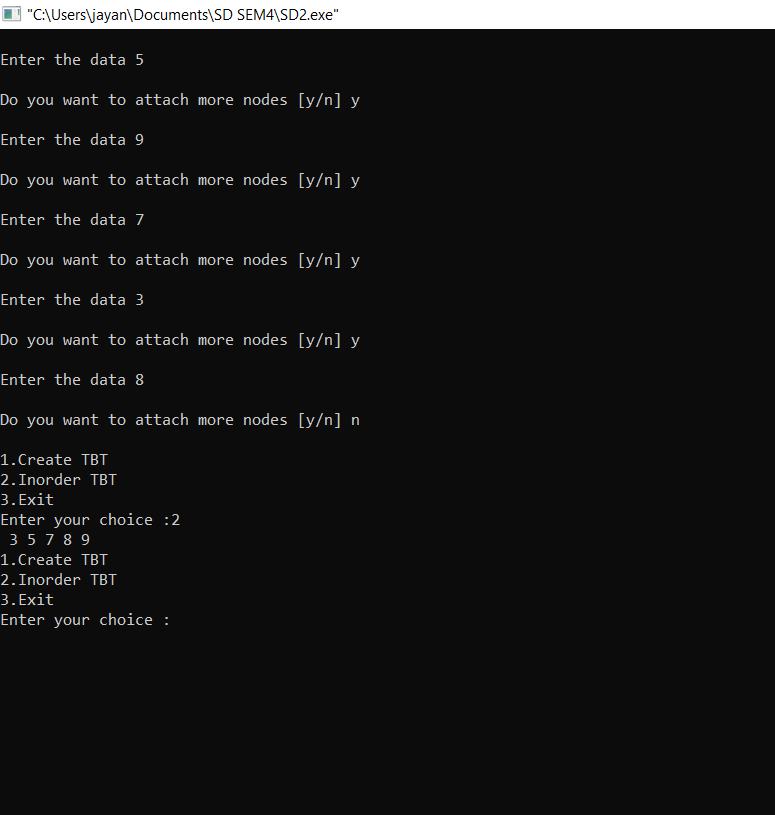
}

}

while(choice!=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(lg n) time, because you have to "climb" up the tree and then back down.