Week-9 UE20CS207 DSLAB

- Name : P K Navin Shrinivas
- SRN : PES2UG20CS237
- Section : D
- Batch : 2

Lab problem 1 : Iterative implementation of traversals

Code :

main.c

```
#include "2_1.h"
#include <stdio.h>

int main(){
    struct stack* st=(struct stack*)malloc(sizeof(struct stack));
    st->top=-1;
    struct node* root;
    root=(struct node*)malloc(sizeof(struct node));
    root->data = -1;

    while(true){
        printf("1.BST Insertion \n");
        printf("2.Iterative Postorder Traversal \n");
        printf("3.Interative Preorder Traversal \n");
        printf("4.Interative Inorder Traversal \n");
        printf("Enter choice : ");
        int choice=0;
        scanf("%d",&choice);
        if(choice == 1)
        {
            int d;
            printf("Enter data to be inserted : ");
            scanf("%d",&d);
            BSTInsertion(root , d);
        }
        else if(choice == 2){
            printf("BST in Postorder : \n");
            printf("Start of traversal ->");
            BSTIterativePostorder(root , st);
            printf("End of traversal \n");
        }
        else if(choice == 3){
            printf("BST in Preorder : \n");
            printf("Start of traversal ->");
            BSTIterativePreorder(root,st);
            printf("End of traversal \n");
        }
        else if(choice == 4){
            printf("BST in Inorder : \n");
            printf("Start of traversal ->");
            BSTIterativeInorder(root , st);
            printf("End of traversal\n");
        }
    }
}
```

2_1.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>


struct node{
    int data;
    struct node* l;
    struct node* r;
};

#define STACKSIZE 100

struct vnode{
    struct node* n;
    int vleft;
    int vright;
};

struct stack{
    int top;
    struct vnode* data[STACKSIZE];
};

void pushe(struct stack* st , struct vnode* d);
void pope(struct stack* st);
void BSTInsertion(struct node* root , int d);
void BSTIterativePostorder(struct node* root , struct stack* st);
void BSTIterativePreorder(struct node* root , struct stack* st);
void BSTIterativeInorder(struct node* root , struct stack* st);
```

2_1.c

```c
#include "2_1.h"
#include <bits/pthreadtypes.h>
#include <stdio.h>


void BSTInsertion(struct node *root , int d)
{
    if(root->data < 0){
        struct node *temp = (struct node*) malloc(sizeof(struct node));
        root->data = d;
        root->l = NULL;
        root->r = NULL;
        printf("\n----------------------------\n");
        printf("Data %d inserted to BST! :))", root->data);
        printf("\n----------------------------\n");
        return;
    }
    else if(root->data > d){
        if(root->l == NULL){
            struct node* temp = (struct node*) malloc(sizeof(struct node));
            temp->data = d;
            temp->l = NULL;
            temp->r =NULL;
            root->l = temp;
            printf("\n----------------------------\n");
            printf("Data inserted to BST! :))");
            printf("\n----------------------------\n");
            return;
        }
        else{
            BSTInsertion(root->l,d);
        }
    }
    else{
        if(root->r == NULL){
            struct node* temp = (struct node*) malloc(sizeof(struct node));
            temp->data = d;
```

```
            temp->l = NULL;
            temp->r =NULL;
            root->r = temp;
            printf("\n----------------------------\n");
            printf("Data inserted to BST! :))");
            printf("\n----------------------------\n");
            return;
        }
        else{
            BSTInsertion(root->r,d);
        }
    }
}

struct vnode* stackpeek(struct stack* st){
    return st->data[st->top];
}

void pushe(struct stack* st , struct vnode* d)
{
    if(st->top == STACKSIZE-1)
    {
        return;
    }
    else{
        (st->top)++;
        st->data[st->top]=d;
        return;
    }
}

void pope(struct stack* st)
{
    if(st->top==-1)
    {
        return;
    }
    else{
        (st->top)--;
    }
}


void BSTIterativePostorder(struct node* root , struct stack* st){

    struct vnode* tempvnode = (struct vnode*)malloc(sizeof(struct vnode));
    tempvnode->n = root;
    tempvnode->vleft = 0;
    tempvnode->vright = 0;
    pushe(st , tempvnode);
    while(st->top != -1){
        if(root->l != NULL && stackpeek(st)->vleft==0){
            stackpeek(st)->vleft = 1;
            root= root->l;
            struct vnode* templnode =  (struct vnode*)malloc(sizeof(struct vnode));
            templnode->n = root;
            templnode->vleft = 0;
            templnode->vright = 0;
            pushe(st , templnode);
            continue; //very important
        }

        if(root->r != NULL && stackpeek(st)->vright == 0){
            stackpeek(st)->vright = 1;
            root = root->r;
            struct vnode* temprnode =   (struct vnode*)malloc(sizeof(struct vnode));
            temprnode->n = root;
            temprnode->vleft = 0;
            temprnode-> vright = 0;
            pushe(st , temprnode);
            continue; //very important

        }
```

```
        //if none of both
        printf("%d->",root->data);
        pope(st);
        if(st->top != -1){
            root = stackpeek(st)->n;
        }
    }
    return;
}

void BSTIterativePreorder(struct node* root , struct stack* st){
    struct vnode* tempvnode = (struct vnode*)malloc(sizeof(struct vnode));
    tempvnode->n = root;
    pushe(st , tempvnode);
    while(st->top != -1){
        root = stackpeek(st)->n;
        printf("%d ->",root->data);
        pope(st);
        if(root->r != NULL){
            struct vnode* temprnode =  (struct vnode*)malloc(sizeof(struct vnode));
            temprnode->n = root->r;
            pushe(st , temprnode);
        }
        if(root->l != NULL){
            struct vnode* templnode =  (struct vnode*)malloc(sizeof(struct vnode));
            templnode->n = root->l;
            pushe(st , templnode);
        }
    }
}

void BSTIterativeInorder(struct node* root , struct stack* st){
    while(root != NULL){
        while(root != NULL){
            if(root->r != NULL){
                struct vnode* temprnode =  (struct vnode*)malloc(sizeof(struct vnode));
                temprnode->n = root->r;
                pushe(st , temprnode);
            }
            struct vnode* tempvnode = (struct vnode*)malloc(sizeof(struct vnode));
            tempvnode->n = root;
            pushe(st , tempvnode);
            root=root->l;
        }
        root = stackpeek(st)->n;
        pope(st);
        while(st->top != -1 && root->r == NULL){
            printf("%d ->",root->data);
            root = stackpeek(st)->n;
            pope(st);
        }
        printf("%d ->" , root->data);
        if(st->top != -1){
            root = stackpeek(st)->n;
            pope(st);
        }else{
            root = NULL; //terminating
        }
    }
}
```