BINARY SEARCH TREES

Design, Develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers

a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2

b. Traverse the BST in Inorder, Preorder and Post Order

c. Exit

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int data;
        struct node *lchild;
        struct node *rchild;
};

typedef struct node *NODE;
NODE root = NULL;


                         void insert();
                         void inorder(NODE root);
                         void preorder(NODE root);
                         void postorder(NODE root);
                         void display(NODE root,int);
void main()
{
int ch,i,n,item;

for(;;)
{
printf("\n***TREE OPERATIONS ARE***\n");
printf(" 1:Insert nodes\n 2:Inorder traversal\n 3:Preorder traversal
              \n 4:Postorder traversal\n 5:Display\n 6:Exit");
printf("\n**************************\n");
printf("\nEnter your choice:\n");
scanf("%d",&ch);
```

```
 switch(ch)
{
case 1: printf("Enter the no of elements\n");
        scanf("%d",&n);
        for(i=0;i<n;i++)
                insert();
         break;

case 2: if(root==NULL)
             printf("Empty tree\n");
        else
        {
           printf("Inorder traversal\n");
           inorder(root);
        }
       break;
```

```
case 3: if(root==NULL)
            printf("Empty tree\n");
        else
        {
            printf("preorder traversal\n");
            preorder(root);
        }   break;

case 4: if(root==NULL)
             printf("Emtpy tree\n");
        else
       {
            printf("Postorder traversal\n");
            postorder(root);
       }   break;
```

```
case 5: if(root==NULL)
            printf("Empty tree\n");
        else
                printf("The tree is:\n");
               display(root,1);   break;

Case 6: exit(0);

default: printf("Invalid choice\n");
}
}
}
```

```
void insert()
{
        NODE newnode,prev,curr;
        newnode=(NODE)malloc(sizeof(struct node));

        printf("Enter the element to be inserted in tree\n");
        scanf("%d", &newnode->data);

        newnode->lchild=newnode->rchild=NULL;

        if(root==NULL)
        {
                root=newnode;
                return;
        }
```

```
 prev=NULL;
 curr=root;
 while(curr!=NULL)
 {
        if(curr->data==newnode->data)
        {
                printf("Duplicate is not possible\n");
                free(newnode);
                return;
        }
        prev=curr;
        if(newnode->data < curr->data)
                curr=curr->lchild;
        else
                curr=curr->rchild;
}
```

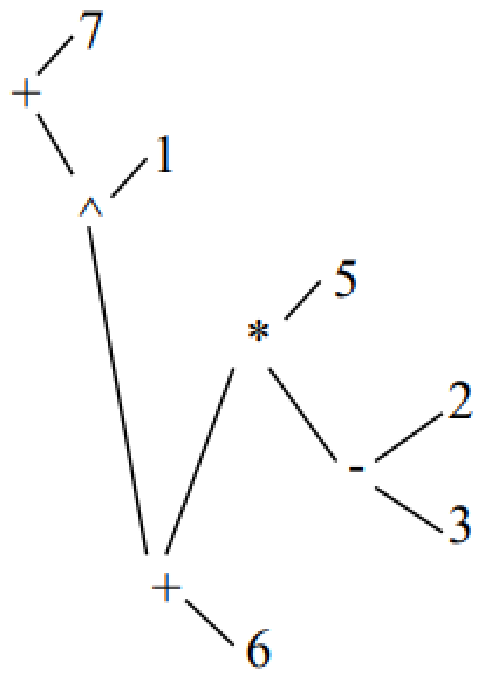
```
 if(newnode->data < prev->data)
        prev->lchild=newnode;
 else
        prev->rchild=newnode;
 return;
}
```

```
void inorder(NODE root)
{
    if(root!=NULL)
    {
        inorder(root->lchild);
        printf("%4d",root->data);
        inorder(root->rchild);
    }
    return;
}
```

```
void preorder(NODE root)
{
    if(root!=NULL)
    {
        printf("%4d", root->data);
        preorder(root->lchild);
        preorder(root->rchild);
    }
return;
}
```

```
void postorder(NODE root)
{
        if(root!=NULL)
        {
                postorder(root->lchild);
                postorder(root->rchild);
                printf("%4d",root->data);
        }
        return;
}
```

```
void display(NODE root, int level)
{
    int i;
    if(root)
    {
        display(root->rchild,level+1);
        printf("\n");
        for(i=0;i<level;i++)
            printf("\t");
        printf("%d", root->data);
        display(root->lchild,level+1);
    }
    return;
}
```



Additional Functions

```
NODE search(int item, NODE root)
{
        NODE cur;

        if (root == NULL) return NULL;              /* empty tree */

        cur = root;
        while ( cur != NULL )                       /* search for the item */
        {
                if (item == cur->info) return cur;  /* If found return the node */

                if ( item < cur->info )
                        cur = cur->llink;           /* Search towards left */
                else
                        cur = cur->rlink;           /* Search towards right */
        }

        return NULL;                                /* Key not found */
}
```

```
void maximum(NODE root)
{
        NODE cur;

        if ( root == NULL ) return root;

        cur = root;
        while ( cur->rlink != NULL )
        {
                cur = cur->rlink;            /* obtain right most node in BST */
        }

        printf("The Maximum element in the BST is %d\n", cur->info);
}
```

```
void count_node(NODE root)
{
        if ( root = = NULL ) return;
        count_node(root->llink);
        count++;
        count_node(root->rlink);
}
```

```
void count_leaf(NODE root)
{
        if ( root == NULL ) return ;
        count_leaf(root->llink);                /* Traverse recursively towards left */
        /* if a node has empty left and empty right child ? */
        if (root->llink == NULL && root->rlink == NULL ) count++;
        count_leaf(root->rlink);                /* Traverse recursively towards right */
}
```

The procedure to be followed while creating an expression tree using postfix expression is shown below:

- Scan the symbol from left to right.
- Create a node for the scanned symbol.
- If the symbol is an operand push the corresponding node onto the stack.
- If the symbol is an operator, pop one node from the stack and attach to the right of the corresponding node with an operator. The first popped node represents the right operand. Pop one more node from the stack and attach to the left. Push the node corresponding to the operator on to the stack.
- Repeat through step 1 for each symbol in the postfix expression. Finally when scanning of all the symbols from the postfix expression is over, address of the root node of the expression tree is on top of the stack.

```
NODE creat_tree(char postfix[])
{
        NODE temp, st[20];
        int  i,k;
        char symbol;

        for ( i = k = 0; ( symbol = postfix[i] ) != '\0'; i++)
        {
                temp = getnode();              /* Obtain a node for each operator */
                temp->info = symbol;
                temp->llink = temp->rlink = NULL;

                if( isalnum(symbol) )
                        st[k++] = temp;        /* Push the operand node on to the stack */
                else
                {
                        temp->rlink = st[--k];         /* Obtain 2nd  operand from stack */

                        temp->llink = st[--k];         /* Obtain 1st operand from stack */

                        st[k++] = temp;                /* Push operator node on to stack */

                }
        }

        return st[--k];                        /* Return the root of the expression tree */
}
```

Evaluation of the expression

In the expression trees, Whenever an operator is encountered, evaluate the expression in the left subtree and evaluate the expression in the right subtree and perform the operation.

The recursive definition to evaluate the expression represented by an expression tree is shown below:

$$\text{Eval (root)} = \begin{cases} \text{Eval(root->llink) } \textbf{op} \text{ Eval(root->rlink)} & \text{if root->info is operator} \\ \text{Root->info – ‘0’} & \text{if root->info is operand} \end{cases}$$

The C function for the above recurrence relation to evaluate the expression represented by an expression tree is shown below:

Example 10.25: Function to evaluate the expression represented as a binary tree

```
float eval(NODE root)
{
        float num;

        switch(root->info)
        {
                case '+': return eval(root->llink) + eval(root->rlink);

                case '-': return eval(root->llink) - eval(root->rlink);

                case '/': return eval(root->llink) / eval(root->rlink);

                case '*': return eval(root->llink) * eval(root->rlink);

                case '$':
                case '^': return pow (eval(root->llink),eval(root->rlink));

                default :
                        if ( isalpha(root->info) )
                        {
                                printf("%c = ",root->info);
                                scanf("%f",&num);
                                return num;
                        }
                        else
                                return root->info - '0';
        }
}
```