Project 2. Binary Search Tree

Experiment 1. Binary Tree

Purpose:

Understand the linked representation of binary trees, and master the design of various basic operation algorithms in binary trees.

Contents:

Use linked representation as the storage structure of a binary tree. The data type of BinTree (binary tree) is defined as bellow:

```
typedef struct node{
        char data;
        struct node *lchild,*rchild;
}BinTNode;        //define the data type of the node
  typedef BinTNode *BinTree;
```

Problem Description:

Input the pre-order traversal sequence of a binary tree, where * represents virtual node (i.e. the node does not exist, NULL link), for example ABD***CE**F**. (Also consider the example: ABC*D*E***F**.)
Complete the following tasks:

Complete the following tasks:

1. Program to create the linked representation of the binary tree base on the inputted pre-order traversal sequence.
2. Program to traverse the binary tree **iteratively** and output the traversal sequences with pre-order, in-order and post-order traversals.
3. Program to count the total number of the nodes in the tree
4. Program to count the number of the leaves of the tree
5. Output the binary tree with the string representation
6. Program to delete the tree.

Part of source codes:

```
typedef struct node{
   char data;
   struct node *lchild,*rchild;
            }BinTNode;        //define the data type of the node
```

```
typedef BinTNode *BinTree;

BinTree CreatBinTree(void){
      BinTree T;
      char ch;
      if((ch=getchar())=='*')
     return(NULL);          /*input is '*', return null vector*/
      else{
     T= (BinTNode *)malloc(sizeof(BinTNode)); /*create a node*/
     T->data=ch;
     T->lchild=CreatBinTree();          /*create the left subtree*/
     T->rchild=CreatBinTree();          /*create the right subtree*/
     return(T);
      }
}

void PostOrder(BinTree b)
{ BinTNode * St[MaxSize];
  BinTNode *p;
  int top=-1;
  bool flag;
  if(b!=NULL)
     {   do
          {   while(b!=NULL)
                {   top++;
                    St[top]=b;
                    b=b->lchild;
                }
              p=NULL;
              flag=true;
              while (top!=-1&&flag)
              {     b=St[top];
                    If(b->rchild==p)
                     {     printf("%c", b->data);
                           Top--;
                           P=b;
                      }
                    else
                      {     b=b->rchild;
                            flag=false;
                       }
              }
        } while (top!=-1);
        printf("\n");
```

```
    }
}

void DispBinTree(BinTree b)
{   if (b!=NULL)
        {     printf("%c", b->data);
              if ( b->lchild!=NULL || b->rchild!=NULL)
                {     printf("(");
                      DispBinTree(b->lchild);
                      if (b->rchild!=NULL) pintf(",");
                      DispBinTree(b->rchild);
                      printf(")");
                }
        }
}
```

Experiment 2. Binary Search Tree

Purpose:

Master the construction process of binary search tree and its algorithm design

Contents:

Define the binary search tree ADT. Program to realize the creation, search, insertion and deletion algorithms for binary search trees:

(1) Given an ordered key sequence, create corresponding binary search tree with lowest height.

(2) Given a binary tree, check whether the tree is a binary search tree.

(3) Program to realize insertion operation into a binary search tree.

(4) Program to realize deletion operation from a binary search tree.

(5) Write a main function to use the binary search tree ADT. Design many testing cases and check whether the operations are executed properly.

Tips: To create a lowest binary search tree, the closer the number of nodes in the left and right subtrees, the better. Suppose the key sequence is saved in an ordered array A[start..end], we should use the middle key A[mid] as the root and create the left subtree with A[start..mid] and right subtree with A[mid+1..left].

```
typedef KeyType int;
typedef struct node{
        KeyType data;
```

```
        struct node *lchild,*rchild;
  }BSTNode;        //define the data type of the node
    typedef BSTNode *BSTree;


BSTree CreateBST(KeyType A[ ], int start, int end)
{
        int mid=(end+start)/2;
        BSTNode *bt;
        if (end<start) return NULL;
        bt=(BSTNode *)malloc(sizeof(BSTNode));
        bt->data=A[mid];
        bt->lchild=CreateBST(A, start, mid);
        bt->rchild=CreateBST(A, mid+1, end);
        return bt;
}
```