Dr. D.Y.Patil Arts, Commerce & Science College,

Pimpri, Pune-18

S.Y.B.Sc(Comp. Sci) sem-IV 2022-23

Data Structures and Algorithms – II

Practical Assignment 1: Binary Search Tree and Traversals

Set A

a) Implement a Binary search tree (BST) library (btree.h) with operations – create, search, insert, inorder, preorder and postorder. Write a menu driven program that performs the above operations.

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

NODE *getnode()
 {
   NODE *temp;
   temp=(NODE*)malloc(sizeof(NODE));
   printf("\n\n Enter the data : ");
   scanf("%d",&temp->data);
   temp->lchild=NULL;
   temp->rchild=NULL;
    return(temp);
}
NODE *create()
 {
   NODE *temp,*ptr,*root;
   char ch;
   root=NULL;
   do
      {
        temp=getnode();
        if(root==NULL)
            root=temp;
        else
       {
        ptr=root;
        while(ptr!=NULL)
         {
              if(temp->data<ptr->data)
              {
                  if(ptr->lchild==NULL)
                  {
                          ptr->lchild=temp;
                          break;
                  }
                  else
                     ptr=ptr->lchild;
             }
             else
```

```
            {
                  if(ptr->rchild==NULL)
                  {
                        ptr->rchild=temp;
                        break;
                  }
                  else
                    ptr=ptr->rchild;
            }
         }//while
      } //else
        printf("\n Add More (Y/N)? : ");
       scanf(" %c",&ch);
    }while(ch=='Y' || ch=='y');
   return(root);
 }
int search(int num,NODE *ptr)
{
    while(ptr!=NULL)
        {
           if(num==ptr->data)
            return 1;
            if(num<ptr->data)
            {
                  ptr=ptr->lchild;

            }
           if(num>ptr->data)
           {
                  ptr=ptr->rchild;
           }


        }

   return 0;


}
NODE *insert(NODE *temp, NODE *root)
{
   NODE *ptr;
   ptr=root;
   if(ptr==NULL)
    {
      root=ptr=temp;
    }
   else
   {
       ptr=root;
       while(ptr!=NULL)
      {
        if(temp->data<ptr->data)
          {
           if(ptr->lchild==NULL)
            {
```

```
            ptr->lchild=temp;
            break;
             }
           else
            ptr=ptr->lchild;
          }
        else
          {
           if(ptr->rchild==NULL)
           {
              ptr->rchild=temp;
              break;
           }
           else
             ptr=ptr->rchild;
          }
      }
    return(root);
  }

return(root);
}


void inorder(NODE *ptr)
{
  if(ptr!=NULL)
  {
     inorder(ptr->lchild);
     printf(" %d",ptr->data);
     inorder(ptr->rchild);
  }
}


void preorder(NODE *ptr)
{
  if(ptr!=NULL)
  {
     printf(" %d",ptr->data);
     preorder(ptr->lchild);
     preorder(ptr->rchild);
  }
}
void postorder(NODE *ptr)
{
  if(ptr!=NULL)
  {
     postorder(ptr->lchild);
     postorder(ptr->rchild);
     printf(" %d",ptr->data);
  }
}

main()
  {
```

```
    int ch,num,t,abc;
    NODE *root;
    NODE *temp;
    while(1)
    {
     printf("\nMain Menu");
     printf("\n1: Create Binary search tree");
     printf("\n2: Inorder traversal");
     printf("\n3: Preorder traversal");
     printf("\n4: postorder traversal");
     printf("\n5: Search a value");
     printf("\n6: Insert a value");
     printf("\n7: Exit");
     printf("\n Enter the choice: ");
     scanf("%d",&ch);
     switch(ch)
     {
       case 1:   root=create();
            break;
       case 2:
            printf("\nInorder traversal");
            inorder(root);
            break;
       case 3:
            printf("\nPreorder traversal:  ");
            preorder(root);
            break;
       case 4:
            printf("\nPostorder Traversal: ");
            postorder(root);
            break;
       case 5:   printf("\nEnter the Value to be searched: ");
            scanf("%d",&num);
            t=search(num,root);
            if(t==1)
             printf("\nValue is found");
            else
                printf("\nValue is not found");
            break;
       case 6:
           temp=getnode();
           root=insert(temp,root);
           break;
       case 7:
           exit(1);
   }
 }
}
```

b) Write a program which uses binary search tree library and counts the total nodes and total leaf nodes in the tree.

Int count(T) – returns the total number of nodes from BST

int countLeaf(T) – returns the total number of leaf nodes from BST

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

```
int nodetotal=0,leaftotal=0;

struct node
{
      struct node *lchild;
      int data;
      struct node *rchild;
};
typedef struct node NODE;

NODE *getnode()
{
      NODE *temp;
      temp=(NODE*)malloc(sizeof(NODE));
      printf("\n\n Enter the data : ");
      scanf("%d",&temp->data);
      temp->lchild=NULL;
      temp->rchild=NULL;
      return(temp);
}

NODE *create()
{
      NODE *temp,*ptr,*root;
      char ch;
      root=NULL;
      do
      {
            temp=getnode();
            if(root==NULL)
                  root=temp;
            else
            {
                  ptr=root;
                  while(ptr!=NULL)
                  {
                        if(temp->data<ptr->data)
                        {
                              if(ptr->lchild==NULL)
                              {
                                    ptr->lchild=temp;
                                    break;
                              }
                              else
                                    ptr=ptr->lchild;
                        }
                        else
                        {
                              if(ptr->rchild==NULL)
                              {
                                    ptr->rchild=temp;
                                    break;
                              }
                              else
                                    ptr=ptr->rchild;
                        }
```

```
                }//while
            } //else
            printf("\n Add More (Y/N)? : ");
            scanf(" %c",&ch);
      }while(ch=='Y' || ch=='y');
      return(root);
}

int totalnode(NODE *ptr)
{
      if(ptr!=NULL)
      {
            nodetotal++;
            totalnode(ptr->lchild);
            totalnode(ptr->rchild);
      }
      return nodetotal;
}
int leafcount(NODE *ptr)
{
      if(ptr!=NULL)
      {
            leafcount(ptr->lchild);
            if(ptr->lchild==NULL && ptr->rchild==NULL)
                  leaftotal++;
            leafcount(ptr->rchild);
      }
      return leaftotal;
}

main()
{
      NODE *root;
      root=create();
      printf("\n");
      nodetotal=totalnode(root);
      leaftotal=leafcount(root);
      printf("\n The total no. of nodes are : %d",nodetotal);
      printf("\n\n The total no. of leaf nodes are : %d",leaftotal);
}
```

Set B

a) Write a C program which uses Binary search tree library and implements following function with recursion:
T copy(T) – create another BST which is exact copy of BST which is passed as parameter.
int compare(T1, T2) – compares two binary search trees and returns 1 if they are equal and 0 otherwise.

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

struct node
{
      struct node *lchild;
      int data;
      struct node *rchild;
};
```

```
typedef struct node NODE;

NODE *getnode()
{
      NODE *temp;
      temp=(NODE*)malloc(sizeof(NODE));
      printf("\n\n Enter the data : ");
      scanf("%d",&temp->data);
      temp->lchild=NULL;
      temp->rchild=NULL;
      return(temp);
}

NODE *create()
{
      NODE *temp,*ptr,*root;
      char ch;
      root=NULL;
      do
      {
            temp=getnode();
            if(root==NULL)
                  root=temp;
            else
            {
                  ptr=root;
                  while(ptr!=NULL)
                  {
                        if(temp->data<ptr->data)
                        {
                              if(ptr->lchild==NULL)
                              {
                                    ptr->lchild=temp;
                                    break;
                              }
                              else
                                    ptr=ptr->lchild;
                        }
                        else
                        {
                              if(ptr->rchild==NULL)
                              {
                                    ptr->rchild=temp;
                                    break;
                              }
                              else
                                    ptr=ptr->rchild;
                        }
                  }//while
            } //else
            printf("\n Add More (Y/N)? : ");
            scanf(" %c",&ch);
      }while(ch=='Y' || ch=='y');
      return(root);
}
```

```
NODE* copy(NODE *ptr)
{
        NODE *temp;
      if(ptr!=NULL)
      {
            temp=(NODE*)malloc(sizeof(NODE));
            temp->data=ptr->data;
            temp->lchild=copy(ptr->lchild);
            temp->rchild=copy(ptr->rchild);
                 return(temp);
      }
      return NULL;
}
int compare(NODE *ptr,NODE *ptr1)
{
        if(!ptr && !ptr1)
           return 1;
      if(ptr!=NULL && ptr1!=NULL && ptr->data==ptr1->data &&
compare(ptr->lchild,ptr1->lchild) && compare(ptr->rchild,ptr1->rchild))
   return 1;
else
   return 0;

}
void display(NODE *ptr)
{
 if(ptr!=NULL)
 {
    display(ptr->lchild);
    printf(" %d",ptr->data);
    display(ptr->rchild);
 }
}
main()
{
      NODE *root,*root1;
      int t;
      root=create();
      printf("\nBinary search after creation: ");
      display(root);
      root1=copy(root);
         printf("\nBinary search tree after copying: ");
         display(root1);
         root1=NULL;
         root1=create(root1);
         t=compare(root,root1);
      if(t==1)
      printf("\nBoth the trees are same.");
         else
      printf("\nTrees are not same.");
}
```

Set C

a) Write a C program which uses Binary search tree library and implements following two functions:

int sumodd(T) – returnssum of all odd numbers from BST //should be done by student

int sumeven(T) – returnssum of all even numbers from BST //should be done by student
mirror(T) – converts given tree into its mirror image

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

struct node
{
      struct node *lchild;
      int data;
      struct node *rchild;
};
typedef struct node NODE;

NODE *getnode()
{
      NODE *temp;
      temp=(NODE*)malloc(sizeof(NODE));
      printf("\n\n Enter the data : ");
      scanf("%d",&temp->data);
      temp->lchild=NULL;
      temp->rchild=NULL;
      return(temp);
}

NODE *create()
{
      NODE *temp,*ptr,*root;
      char ch;
      root=NULL;
      do
      {
            temp=getnode();
            if(root==NULL)
                  root=temp;
            else
            {
                  ptr=root;
                  while(ptr!=NULL)
                  {
                        if(temp->data<ptr->data)
                        {
                              if(ptr->lchild==NULL)
                              {
                                    ptr->lchild=temp;
                                    break;
                              }
                              else
                                    ptr=ptr->lchild;
                        }
                        else
                        {
                              if(ptr->rchild==NULL)
                              {
                                    ptr->rchild=temp;
                                    break;
```

```
                              }
                              else
                                    ptr=ptr->rchild;
                        }
                  }//while
            } //else
            printf("\n Add More (Y/N)? : ");
            scanf(" %c",&ch);
      }while(ch=='Y' || ch=='y');
      return(root);
}
/*
void mirror(NODE *ptr)
{
        NODE *temp;
      if(ptr)
      {
            if(ptr->lchild)
                    mirror(ptr->lchild);
                if(temp->rchild)
                   mirror(ptr->rchild);
                temp=ptr->lchild;
                ptr->lchild=ptr->rchild;
                ptr->rchild=temp;
        }
}
*/
void display(NODE *ptr)
{
 if(ptr!=NULL)
 {
    display(ptr->lchild);
    printf(" %d",ptr->data);
    display(ptr->rchild);
 }
}


void mirror(NODE *h)
{
  NODE *ptr=h,*temp;
  if(ptr!=NULL)
  {
    if(ptr->lchild!=NULL)
      mirror(ptr->lchild);
    if(ptr->rchild!=NULL)
      mirror(ptr->rchild);
    temp1=ptr->lchild;
    ptr->lchild=ptr->rchild;
    ptr->rchild=temp1;
  }
}

main()
{
      NODE *root,*root1;
```

```
        int t;
        root=create();
        printf("\nBinary search after creation: ");
        display(root);
        mirror(root);
           printf("\nBinary search tree after mirror image: ");
           display(root);

}
```