CSE 2003: Lab Assignment #7

Due on Thursday, March 16, 2017

Prof. Shaik Naseera 2:00pm

Jacob John

Contents

Problem 1

Implement a binary search tree using C.

Listing 1: Binary search tree using C

```c
/*Recursive operations in Binary Search Tree*/
#include<stdio.h>
#include<stdlib.h>
struct node
{
     struct node *lchild;
     int info;
     struct node *rchild;
};
struct node *search(struct node *ptr,int skey);
struct node *insert(struct node *ptr,int ikey);
struct node *del(struct node *ptr,int dkey);
struct node *Min(struct node *ptr);
struct node *Max(struct node *ptr);
void preorder(struct node *ptr);
void inorder(struct node *ptr);
void postorder(struct node *ptr);
int height(struct node *ptr);

int main()
{
     struct node *root=NULL,*ptr;
     int choice,k;
     while(1)
     {
          printf("\n");
          printf("1.Search\n");
          printf("2.Insert\n");
          printf("3.Delete\n");
          printf("4.Preorder Traversal\n");
          printf("5.Inorder Traversal\n");
          printf("6.Postorder Traversal\n");
          printf("7.Height of tree\n");
          printf("8.Find minimum and maximum\n");
          printf("9.Quit\n");
          printf("Enter your choice: ");
          scanf("%d",&choice);
          switch(choice)
          {
               case 1:
               printf("Enter the key to be searched: ");
               scanf("%d",&k);
               ptr = search(root,k);
               if(ptr==NULL)
                    printf("Key not present\n");
               else
                    printf("Key present\n");
               break;
```

Problem 1 continued on next page. . .

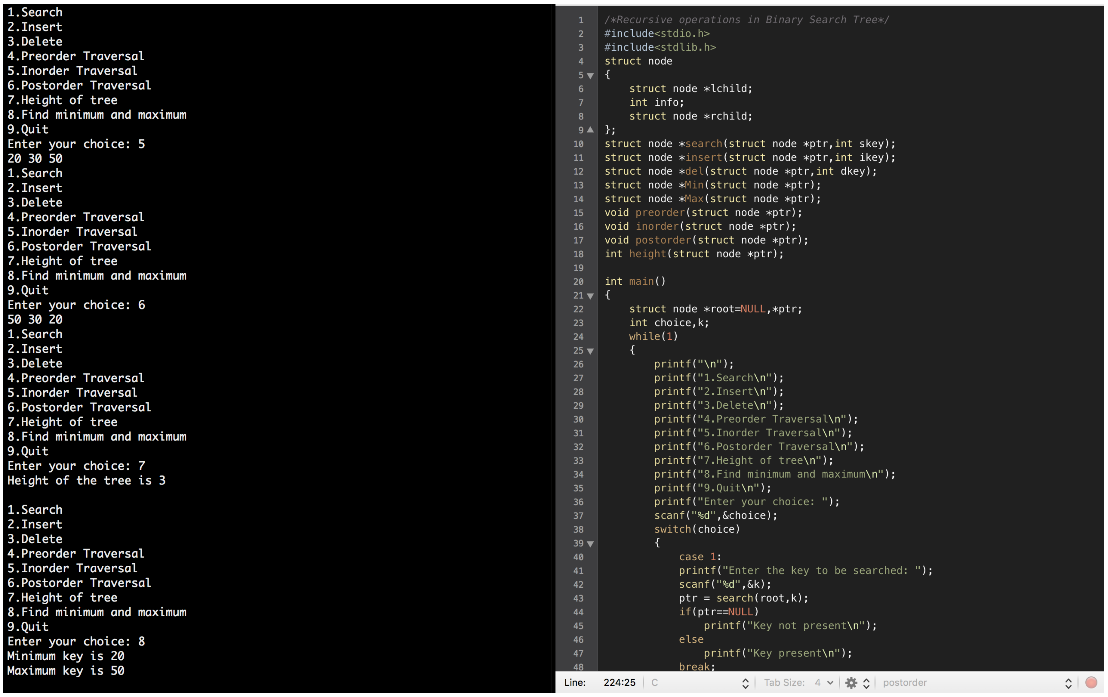
```
            case 2:
            printf("Enter the key to be inserted: ");
            scanf("%d",&k);
            root = insert(root,k);
            break;

            case 3:
            printf("Enter the key to be deleted: ");
            scanf("%d",&k);
            root = del(root,k);
            break;

            case 4:
            preorder(root);
            break;

            case 5:
            inorder(root);
            break;

            case 6:
            postorder(root);
            break;

            case 7:
            printf("Height of the tree is %d\n",height(root));
            break;

            case 8:
            ptr= Min(root);
            if(ptr!=NULL)
                printf("Minimum key is %d\n",ptr->info);
            ptr = Max(root);
            if(ptr!=NULL)
                printf("Maximum key is %d\n",ptr->info);
            break;

            case 9:
                exit(1);

            default:
                printf("Wrong choice\n");
        }/*End of switch*/
    }/*End of while*/
}/*End of main()*/

struct node *search(struct node *ptr,int skey)
{
    if(ptr == NULL)
    {
        printf("key not found\n");
        return NULL;
```

```c
     }
     else if(skey < ptr->info) /*Search in left subtree*/
          return search(ptr->lchild, skey);
     else if(skey > ptr->info) /*Search in right subtree*/
          return search(ptr->rchild, skey);
     else /*skey found*/
          return ptr;
}/*End of search()*/

struct node *insert(struct node *ptr, int ikey)
{
     if(ptr==NULL) /*Base Case*/
     {
          ptr = (struct node *)malloc(sizeof(struct node));
          ptr->info = ikey;
          ptr->lchild = NULL;
          ptr->rchild = NULL;
     }
     else if(ikey < ptr->info) /*Insertion in left subtree*/
          ptr->lchild = insert(ptr->lchild, ikey);
     else if(ikey > ptr->info) /*Insertion in right subtree*/
          ptr->rchild = insert(ptr->rchild, ikey);
     else
          printf("Duplicate key\n"); /*Base Case*/
     return ptr;
}/*End of insert*/

struct node *del(struct node *ptr,int dkey)
{
     struct node *tmp,*succ;
     if(ptr==NULL)
     {
          printf("dkey not found\n");
          return ptr;
     }
     if(dkey < ptr->info) /*Deletion from left subtree*/
          ptr->lchild = del(ptr->lchild, dkey);
     else if(dkey > ptr->info) /*Deletion from right subtree*/
          ptr->rchild = del(ptr->rchild, dkey);
     else
     {
          if(ptr->lchild!=NULL && ptr->rchild!=NULL) /*2 children*/
          {
               succ = ptr->rchild;
               while(succ->lchild)
                    succ = succ->lchild;
               ptr->info = succ->info;
               ptr->rchild = del(ptr->rchild, succ->info);
          }
          else
          {
               tmp = ptr;
               if(ptr->lchild!=NULL) /*only left child*/
```

```
                    ptr = ptr->lchild;
               else if(ptr->rchild != NULL) /*only right child*/
                    ptr = ptr->rchild;
               else /*no child*/
                    ptr = NULL;
               free(tmp);
          }
     }
     return ptr;
}/*End of del()*/

struct node *Min(struct node *ptr)
{
     if(ptr==NULL)
          return NULL;
     else if(ptr->lchild==NULL)
          return ptr;
     else
          return Min(ptr->lchild);
}/*End of Min()*/

struct node *Max(struct node *ptr)
{
     if(ptr == NULL)
          return NULL;
     else if(ptr->rchild==NULL)
          return ptr;
     else
          return Max(ptr->rchild);
}/*End of Max()*/

int height(struct node *ptr)
{
     int h_left,h_right;
     if(ptr==NULL) /*Base Case*/
          return 0;
     h_left = height(ptr->lchild);
     h_right = height(ptr->rchild);
     if(h_left > h_right)
          return 1 + h_left;
     else
          return 1 + h_right;
}/*End of height()*/

void preorder(struct node *ptr)
{
     if(ptr==NULL) /*Base Case*/
          return;
     printf("%d ",ptr->info);
     preorder(ptr->lchild);
     preorder(ptr->rchild);
}/*End of preorder()*/
```

```
void inorder(struct node *ptr)
{
     if(ptr==NULL) /*Base Case*/
          return;
     inorder(ptr->lchild);
     printf("%d ",ptr->info);
     inorder(ptr->rchild);
}/*End of inorder()*/

void postorder(struct node *ptr)
{
     if(ptr==NULL) /*Base Case*/
          return;
     postorder(ptr->lchild);
     postorder(ptr->rchild);
     printf("%d ",ptr->info);
}/*End of posteorder()*/
```

Output:



```
1.Search
2.Insert
3.Delete
4.Preorder Traversal
5.Inorder Traversal
6.Postorder Traversal
7.Height of tree
8.Find minimum and maximum
9.Quit
Enter your choice: 5
20 30 50
1.Search
2.Insert
3.Delete
4.Preorder Traversal
5.Inorder Traversal
6.Postorder Traversal
7.Height of tree
8.Find minimum and maximum
9.Quit
Enter your choice: 6
50 30 20
1.Search
2.Insert
3.Delete
4.Preorder Traversal
5.Inorder Traversal
6.Postorder Traversal
7.Height of tree
8.Find minimum and maximum
9.Quit
Enter your choice: 7
Height of the tree is 3

1.Search
2.Insert
3.Delete
4.Preorder Traversal
5.Inorder Traversal
6.Postorder Traversal
7.Height of tree
8.Find minimum and maximum
9.Quit
Enter your choice: 8
Minimum key is 20
Maximum key is 50
```

```
/*Recursive operations in Binary Search Tree*/
#include<stdio.h>
#include<stdlib.h>
struct node
{
    struct node *lchild;
    int info;
    struct node *rchild;
};
struct node *search(struct node *ptr,int skey);
struct node *insert(struct node *ptr,int ikey);
struct node *del(struct node *ptr,int dkey);
struct node *Min(struct node *ptr);
struct node *Max(struct node *ptr);
void preorder(struct node *ptr);
void inorder(struct node *ptr);
void postorder(struct node *ptr);
int height(struct node *ptr);

int main()
{
    struct node *root=NULL,*ptr;
    int choice,k;
    while(1)
    {
        printf("\n");
        printf("1.Search\n");
        printf("2.Insert\n");
        printf("3.Delete\n");
        printf("4.Preorder Traversal\n");
        printf("5.Inorder Traversal\n");
        printf("6.Postorder Traversal\n");
        printf("7.Height of tree\n");
        printf("8.Find minimum and maximum\n");
        printf("9.Quit\n");
        printf("Enter your choice: ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            printf("Enter the key to be searched: ");
            scanf("%d",&k);
            ptr = search(root,k);
            if(ptr==NULL)
                printf("Key not present\n");
            else
                printf("Key present\n");
            break;
```