Register Now!

Contact Us

Home Project Ideas » Training Programs New » Downloads » Campus Experience » Blog » Contact Us » Search... Go

Binary Search Tree

Code Id 16

Date Updated 3/7/2010

Title Binary search tree

Description

```
This program illustrates Insertion, Deletion and Traversal in Binary Search Tree.
```

Codes Snippet

```
# include
# include
struct node
{
int info;
struct node *1child;
struct node *rchild;
} *root;
main( )
{
int choice,num;
root=NULL;
while( l)
{
printf("n");
printf(" l.Insertn");
printf("2.De1eten");
printf("3.Inorder Traversaln");
printf("4.Preorder Traversaln");
printf(" 5 .Postorder Traversaln");
printf("6.Quitn");
printf("Enter your choice: ");
scanf("%d" ,&choice);
switch( choice)
{
case 1:
printf("Enter the number to be inserted: ");
 scanf("%d" ,&num);
insert(num);
break;
case 2:
printf("Enter the number to be deleted: ");
 scanf("%d" ,&num);
del(num);
break;
case 3:
inorder(root);
break;
case 4:
preorder(root);
break;
case 5:
postorder(root);
break;
        case 6:
        exit( );
        default:
        printf(�Wrong choicen�);
        } /*End of switch * /
        }/*End of while */
}/*End of main( )*/
find(int item,struct node **par,struct node **loc)
{
struct node *ptr, *ptrsave;
if(root==NULL) /*tree empty*/
{       *loc=NULL;
*par=NULL;
        return;
}
 if(item==root->info) /*item is at root*/
```

Recent Posts

Types of Cloud Computing

What is Cloud Computing ?

How to pass a multi-dimensional array to a function?

Memory Layout of a C Program

PHP and Its Advantages

```
                {       *loc=ptr;
                        *par=ptrsave;
                        return;
}
ptrsave=ptr;
 if(iteminfo)
                        ptr=ptr->1child;
else
ptr=ptr->rchild;
}/*End of while */
*loc=NULL; /*item not found*/
 *par=ptrsave;
}/*End of find( )*/
insert(int item)
        {        struct node *tmp, *parent, *location;
        find( item,&parent,&location);
        if(location!=NULL)
        {
printf("Item already present");
 return;
}
tmp=(struct node *)malloc(sizeof(struct node));
tmp->info=item;
tmp->1child=NULL;
tmp->rchild=NULL;
if(parent== NULL)
        root=tmp;
else
if(iteminfo)
                parent->lchild=tmp;
else
parent->rchild=tmp;
}/*End of insert( )*/
del(int item)
{
struct node *parent, *location;
if(root==NULL)
{
printf("Tree empty");
 return;
}
find( item,&parent,&location);
if(location== NULL)
{       printf("Item not present in tree");
return;
        }
        if(location->1child ==NULL && location->rchild==NULL,)
        case_a(parent,location);
        if(location->lchild!=NULL && location->rchild NULL)
        case_b(parent,location);
        if(location->lchild== NULL && location->rchild!=NULL)
        case_b(parent,location);
        if(location->lchild!=NULL && location->rchild!=NULL)
        case_c(parent,location);
        free(location);
}/*End of del( )*/
case_a(struct node *par,struct node *loc )
{       if(par==NULL)/*item to be deleted is root node*/
                root=NULL;
else
if(1oc== par->1child)
                        par->1child=NULL;
else
par->rchild=NULL;
}/*End of case_a( )*/
case_b(struct node *par,struct node *loc)
{       struct node *child;
/*Initialize child * /
if(1oc->lchild!=NULL) /*item to be deleted has lchild */
                child=loc->lchild;
else    /*item to be deleted has rcbild */
        child=loc->rchild;
if(par==NULL)/*Item to be deleted is root node*/
                root=child;
else
if( loc==par->1child) /*item is 1child of its parent*/
                        par->1child=child;
        else    /*item is rchild of its parent*/
                        par->rchild=child;
```

```
suc=ptr;
parsuc=ptrsave;
        if(suc->lchild==NULL&& suc->rchild NULL)
                case _a(parsuc,suc);
else
case_b(parsuc,suc);
if(par==NULL) /*if item to be deleted is root node */
                root=sue;
else
if(1oc==par->1child)
                        par->1child=suc;
else
par->rchild=sue;
sue-> lchild=loe-> lchild;
sue->rchild= loc->rchild;
} /*End of case_c( )* /
preorder(struct node *ptr)
{
if(root=-NULL)
{
printf("Tree is empty");
return;
}
if(ptr!=NULL)
{       printf("%d ",ptr->info);
preorder(ptr-> 1child);
preorder(ptr ->rchild);
}
} /*End of preorder( )* /
inorder(struct node *ptr)
{
if(rool--NULL)
{
printf("Tree is empty");
return;
}
if(ptr!=NULL)
{
                        inorder(ptr -> lchild);
printf("%d ",ptr->info);
                        inorder(ptr->rchild);
}
}/*End of inorder( )*/
postorder( struct node *ptr)
{
if(root==NULL)
{
                printf("Tree is empty");
return;
}
if(ptr!=NULL)
{       postorder(ptr -> IchiId); postorder(ptr ->rchild);
 printf("%d ",ptr->info);
}
}/*End of postorder( )*/
```