|  |
| --- |
| /\* Description : Implementation of Binary Search tree |
|  | \*Learner : ARSHEE QURESHI |
|  | \*Created on : 14th september 2017 |
|  | \*/ |
|  |  |
|  | #include<stdio.h> |
|  | #include<stdlib.h> |
|  |  |
|  | typedef struct bst{ |
|  | int data; |
|  | struct bst \*left,\*right; |
|  | }node; |
|  |  |
|  |  |
|  |  |
|  | void traverse\_inorder(node \*q){ |
|  | if(q!=NULL){ |
|  | traverse\_inorder(q->left); |
|  | printf("%d\t",q->data); |
|  | traverse\_inorder(q->right); |
|  | } |
|  |  |
|  |  |
|  | } |
|  |  |
|  |  |
|  | void insert(node \*\*r,int num) |
|  | { |
|  | node \*temp,\*ptr; |
|  | temp=\*r; |
|  |  |
|  | if(temp==NULL){ |
|  | ptr=(node\*)malloc(sizeof(node)); |
|  | ptr->data=num; |
|  | ptr->left=NULL; |
|  | ptr->right=NULL; |
|  | \*r = ptr; |
|  | } |
|  | else |
|  | { |
|  | if(num > temp->data) |
|  | { |
|  | insert(&temp->right,num); |
|  | } |
|  | else{ |
|  | insert(&temp->left,num); |
|  | } |
|  |  |
|  | } |
|  | } |
|  |  |
|  |  |
|  |  |
|  | int search\_bst(node \*q, int num){ |
|  | if(q==NULL){ |
|  | return -1; |
|  | } |
|  | else{ |
|  | if(q->data==num) |
|  | return 1; |
|  | else{ |
|  | if(num > q->data) |
|  | { |
|  | search\_bst(q->right,num); |
|  | } |
|  | else{ |
|  | search\_bst(q->left,num); |
|  | } |
|  | } |
|  | } |
|  | return num; |
|  | } |
|  |  |
|  | void search\_node(node \*\*x,node \*root,node \*\*parent,int num,int \*f) |
|  | { |
|  | node \*temp; |
|  | temp=root; |
|  | if(temp==NULL) |
|  | return; |
|  | if(temp->data==num){ |
|  | \*f=1; |
|  | \*x=temp; |
|  | return ; |
|  | } |
|  | \*parent=temp; |
|  | while(temp!=NULL) |
|  | { |
|  | if(temp->data==num){ |
|  | \*f=1; |
|  | \*x=temp; |
|  | return ; |
|  | } |
|  | \*parent=temp; |
|  | if(num>temp->data) |
|  | temp=temp->right; |
|  | else |
|  | temp=temp->left; |
|  | } |
|  | } |
|  |  |
|  | void delete(node \*\*q,int num){ |
|  | node \*temp,\*parent,\*xsucc,\*x; |
|  | int f=0; |
|  | parent=NULL; |
|  | x=NULL; |
|  | xsucc=NULL; |
|  | temp=\*q; |
|  | search\_node(&x,temp,&parent,&xsucc,num,&f); |
|  | if(f==0){ |
|  | printf("\n The element %d is not found",num); |
|  | return ; |
|  | } |
|  | //x has no child |
|  | if(x->left==NULL && x->right== NULL){ |
|  | if(x->data > parent->data) |
|  | parent->right=NULL; |
|  | else |
|  | parent->left=NULL; |
|  |  |
|  | } |
|  | //x has left child |
|  | else if(x->left!=NULL && x->right==NULL){ |
|  | if(x->data > parent->data) |
|  | parent->right=x->left; |
|  | else |
|  | parent->left=x->left; |
|  |  |
|  | } |
|  | //x has right child |
|  | else if(x->right!=NULL && x->left==NULL){ |
|  | if(x->data > parent->data) |
|  | parent->right=x->right; |
|  | else |
|  | parent->left=x->right; |
|  |  |
|  | } |
|  | //x has both left and right child |
|  | else if(x->left!=NULL && x->right!=NULL){ |
|  |  |
|  | parent=x; |
|  | xsucc=x->right; |
|  | while(xsucc->left!=NULL) |
|  | { |
|  | parent=xsucc; |
|  | xsucc=xsucc->left; |
|  | } |
|  | if(xsucc->data > parent->data) |
|  | parent->right=NULL; |
|  | else |
|  | parent->left=NULL; |
|  | x->data=xsucc->data; |
|  | x=xsucc; |
|  | } |
|  | free (x); |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | int main() |
|  | { |
|  | node \*root; |
|  | root=NULL; |
|  | insert(&root,4); |
|  | insert(&root,5); |
|  | insert(&root,7); |
|  | traverse\_inorder(root); |
|  | if(search\_bst(root,7)==1) |
|  | printf("\nThe number %d is present in the tree",7); |
|  | else |
|  | printf("\nThe number %d is not present in the tree",7); |
|  | if(search\_bst(root,8)==1) |
|  | printf("\nThe number %d is present in the tree",8); |
|  | else |
|  | printf("\nThe number %d is not present in the tree",7); |
|  | delete(&root,19); |
|  | traverse\_inorder(root); |
|  |  |
|  | return 0; |
|  |  |
|  |  |
|  | } |