**PROGRAM CODE:**

#include<stdio.h>

#include<stdlib.h>

#include "avltree.h"

void main()

{

avltree t=NULL;

position pos;

intchoice,element;

do{

printf("\n\*\*\*AVL SEARCH TREE MENU\*\*\*:\n\n1.Insert element\n2.Find element\n3.Find minimum\n4.Find maximum\n5.Display in ascending order\n6.Exit\n");

printf("\nEnter choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Enter element to be inserted: ");

scanf("%d",&element);

t=insert(element,t);

break;

case 2:

printf("Enter element to be found: ");

scanf("%d",&element);

if(find(element,t)==NULL)

printf("\nElement %d not found!\n",element);

else

printf("\nElement %d found!\n",element);

break;

case 3:

pos=findmin(t);

printf("Minimum element is %d\n",pos->element);

break;

case 4:

pos=findmax(t);

printf("Maximum element is %d\n",pos->element);

break;

case 5:

printf("\nAscending order of integers: \n");

inorder(t);

printf("\n");

break;

case 6:

break;

default:

printf("Invalid choice\n");

}

}while(choice!=6);

}

***avltree header file***

typedefstructavlnode \*position;

typedefstructavlnode \*avltree;

typedefintelementtype;

structavlnode

{

elementtype element;

avltree left;

avltree right;

int height;

};

static int height (position p)

{

if(p==NULL)

return -1;

else

return p->height;

}

int max(int a, int b)

{

if (a>b)

return a;

else

return b;

}

static position singlerotatewithleft(position k2)

{

position k1;

k1=k2->left;

k2->left=k1->right;

k1->right=k2;

k2->height=max(height(k2->left),height(k2->right))+1;

k1->height=max(height(k1->left),k2-> height)+1;

return k1;

}

static position singlerotatewithright(position k2)

{

position k1;

k1=k2->right;

k2->right=k1->left;

k1->left=k2;

k2->height=max(height(k2->right),height(k2->left))+1;

k1->height=max(height(k1->right),k2-> height)+1;

return k1;

}

static position doublerotatewithleft(position k3)

{ //rotate between k1 and k2

k3->left=singlerotatewithright(k3->left);

//rotate between k3 and k2

return singlerotatewithleft(k3);

}

static position doublerotatewithright(position k3)

{ //rotate between k1 and k2

k3->right=singlerotatewithleft(k3->right);

//rotate between k3 and k2

return singlerotatewithright(k3);

}

avltree insert(elementtype x, avltree t)

{

if(t==NULL)

{

t=(avltree)malloc(sizeof(structavlnode));

if(t==NULL)

printf("Out of Space");

else

{

t->element=x;

t->height=0;

t->left=t->right=NULL;

}

}

else if(x<t->element)

{

t->left=insert(x,t->left);

if(height(t->left) - height(t->right) ==2)

if(x<t->left->element)

t=singlerotatewithleft(t);

else

t=doublerotatewithleft(t);

}

else if(x>t->element)

{

t->right=insert(x,t->right);

if(height(t->right) - height(t->left) == 2)

if(x>t->right->element)

t=singlerotatewithright(t);

else

t=doublerotatewithright(t);

}

t->height=max(height(t->left),height(t->right))+1;

return t;

}

position find(elementtype x, avltree t)

{ static intcnt=0;

if(x==31)

{

cnt++;

if (t==NULL)

return NULL;

if (x<t->element)

return find (x, t->left);

else if (x>t->element)

return find(x,t->right);

else

{printf("No of times find function is called to find 31 is: %d",cnt); return t; }

}

else //x not 31

{

if (t==NULL)

return NULL;

if (x<t->element)

return find (x, t->left);

else if (x>t->element)

return find(x,t->right);

else

return t;

}

}

position findmin(avltree t)

{

position p=t;

if(p!=NULL)

while(p->left!=NULL)

p=p->left;

return p;

}

position findmax(avltree t)

{

position p=t;

if(p!=NULL)

while(p->right!=NULL)

p=p->right;

return p;

}

void inorder(avltree t)

{ if ( t == NULL)

return;

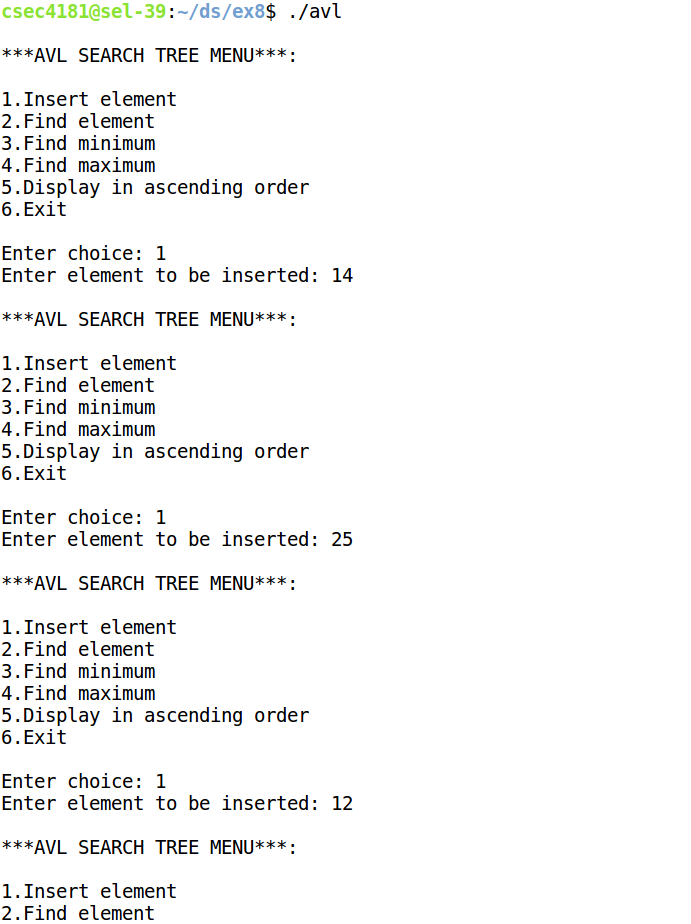
inorder( t->left);

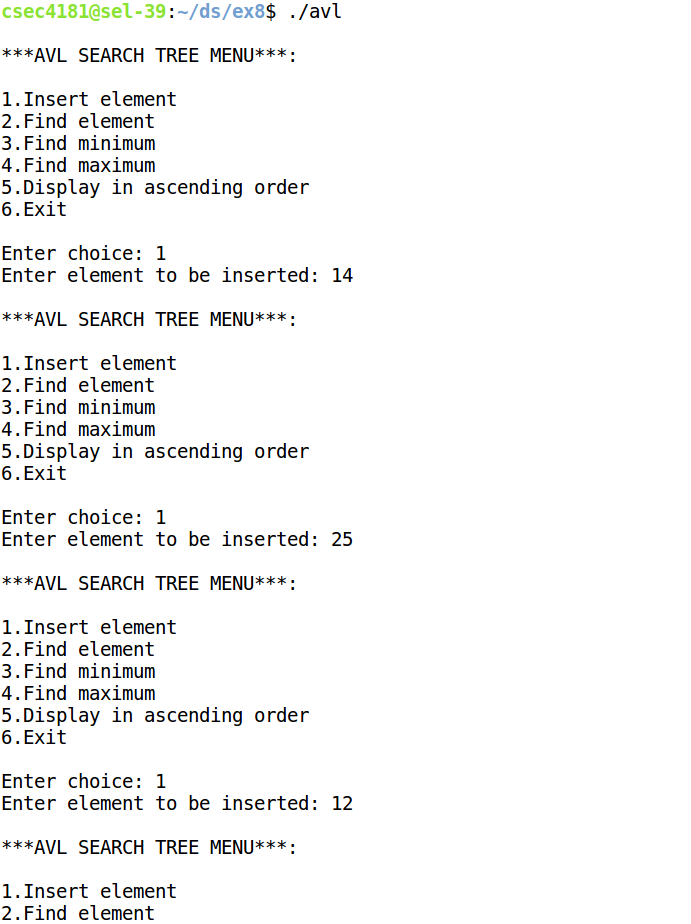
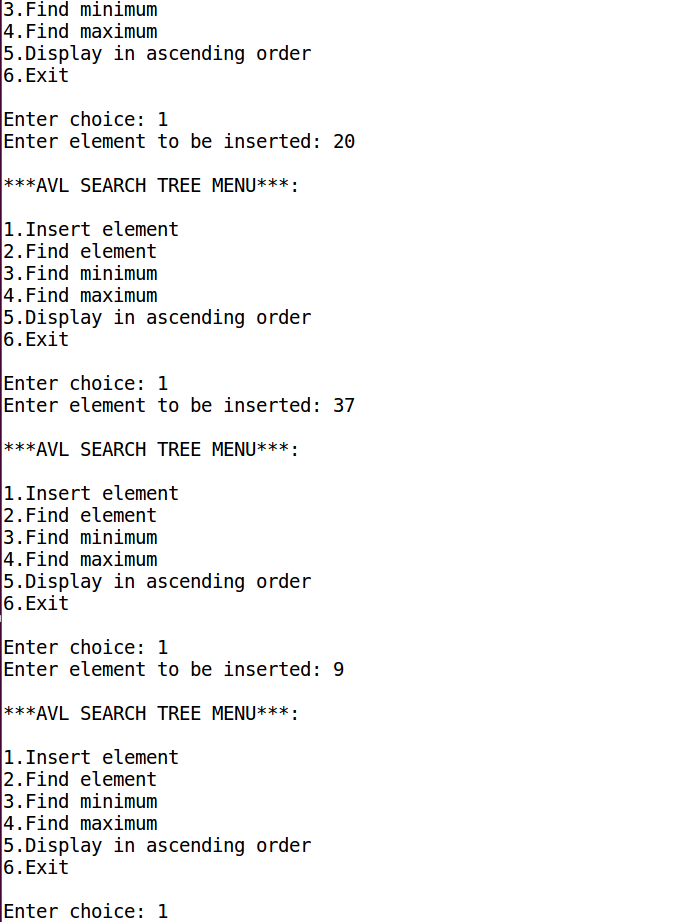
printf("%d ", t->element);

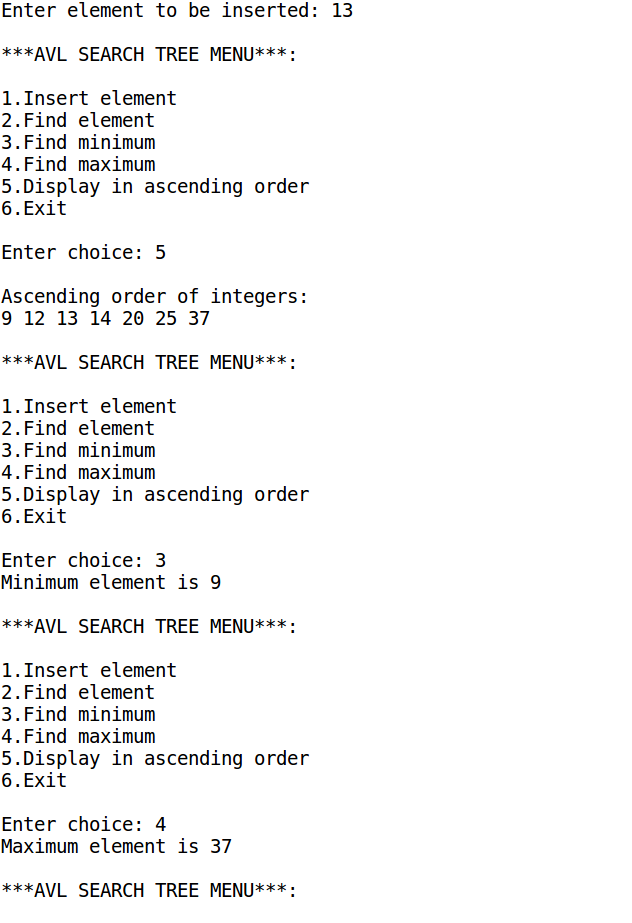
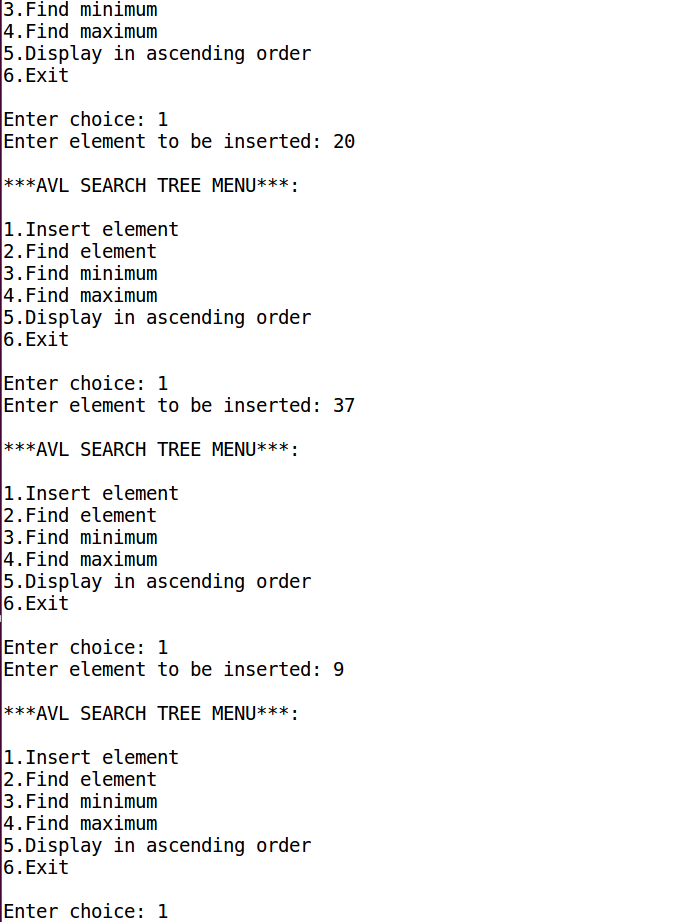
inorder( t->right);

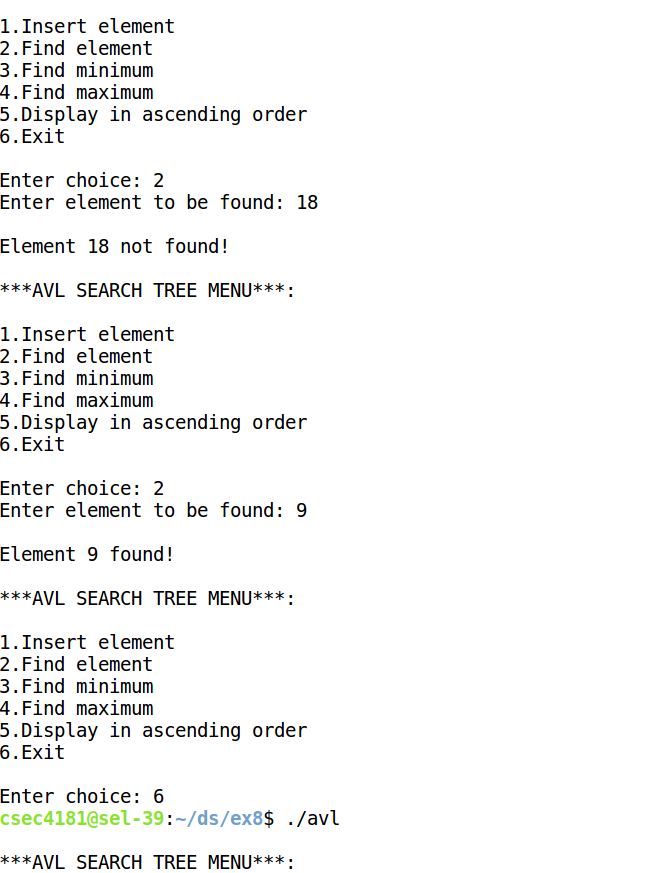
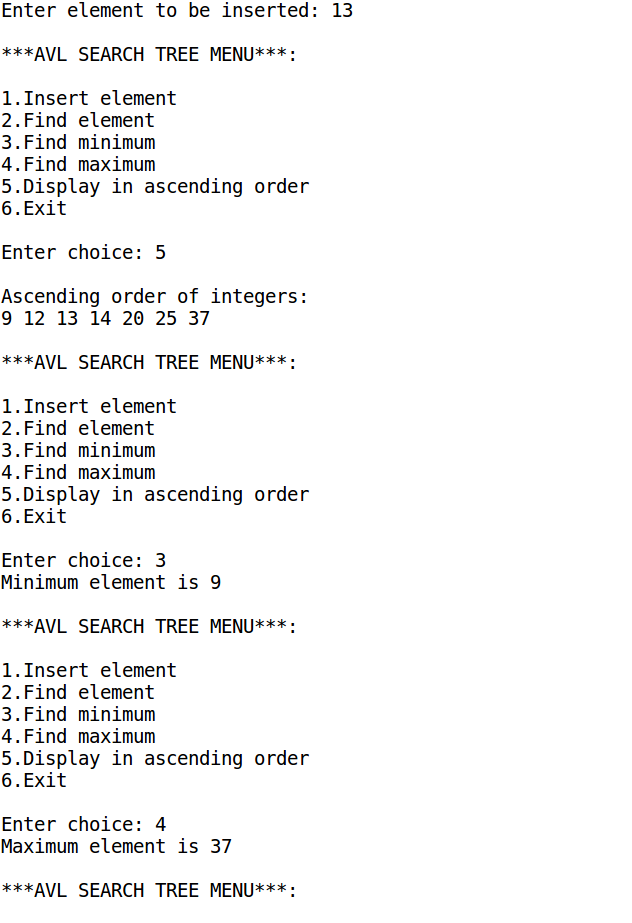
}

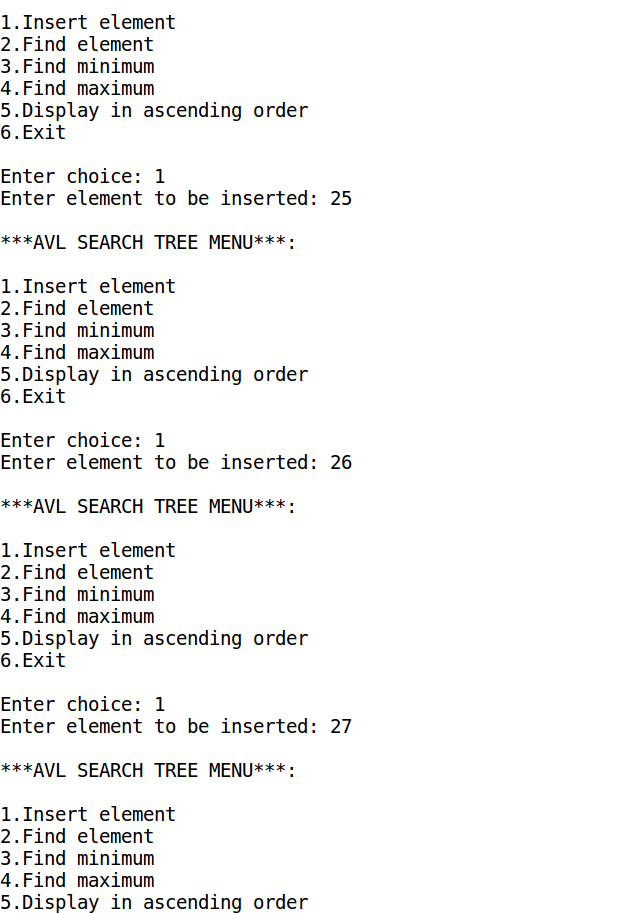
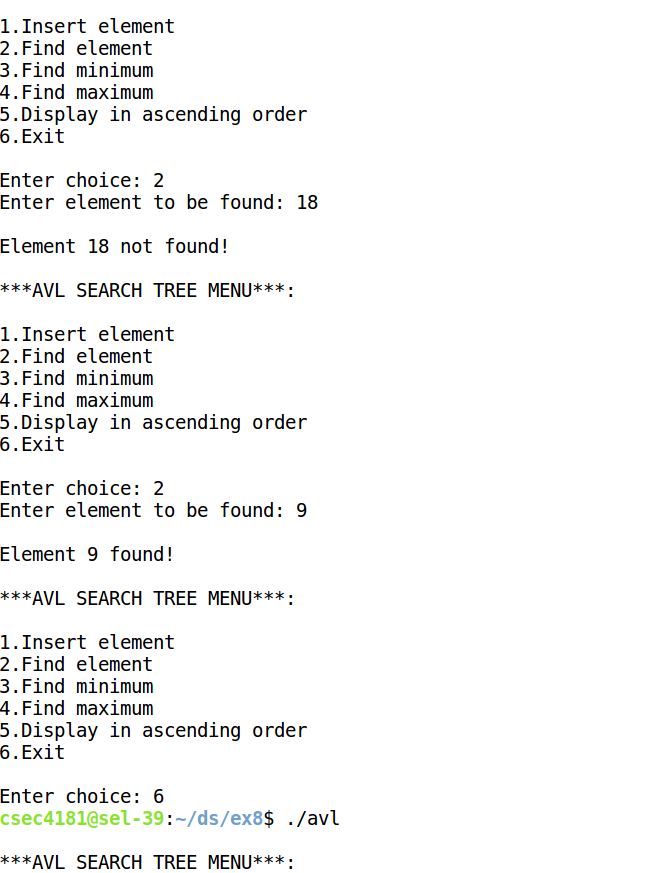
**OUTPUT:**

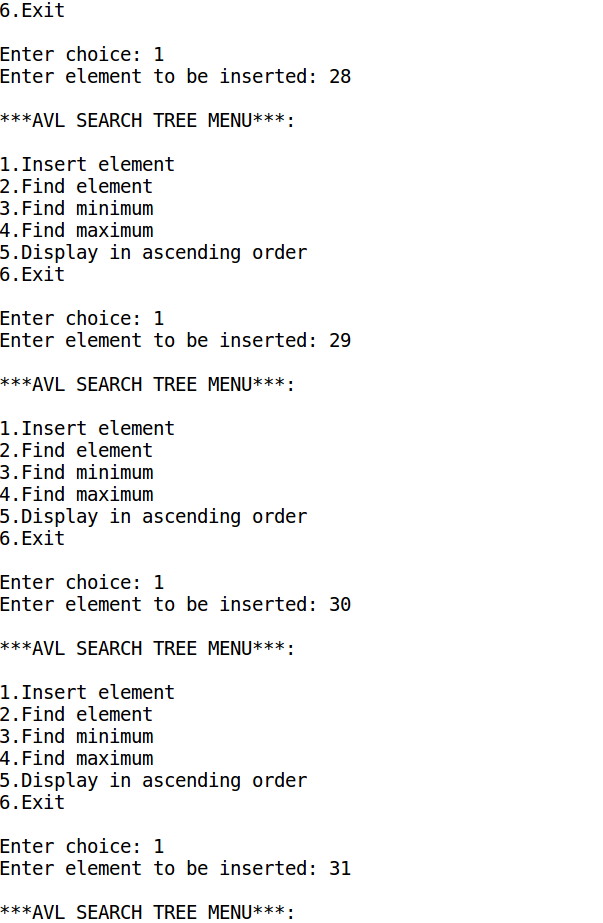
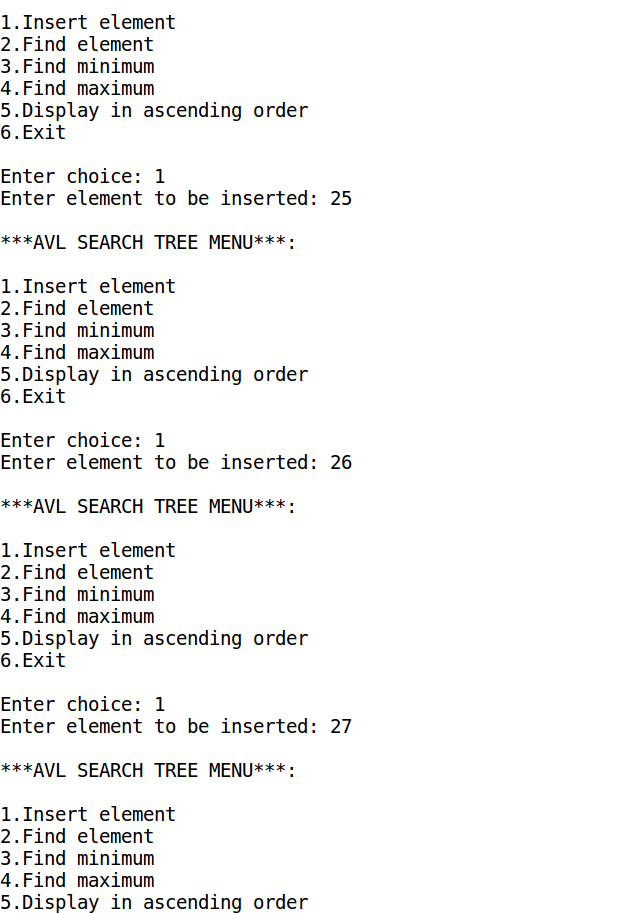
****

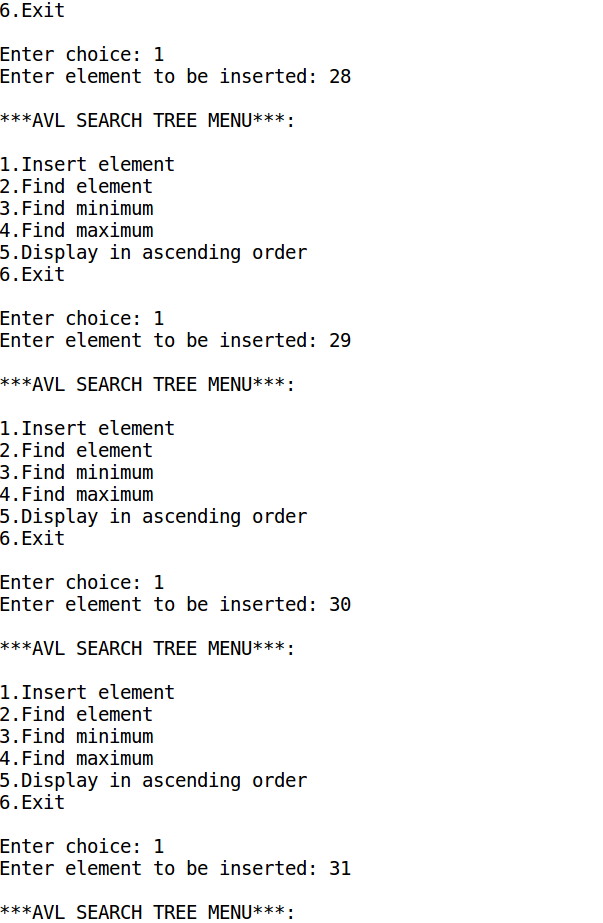
****

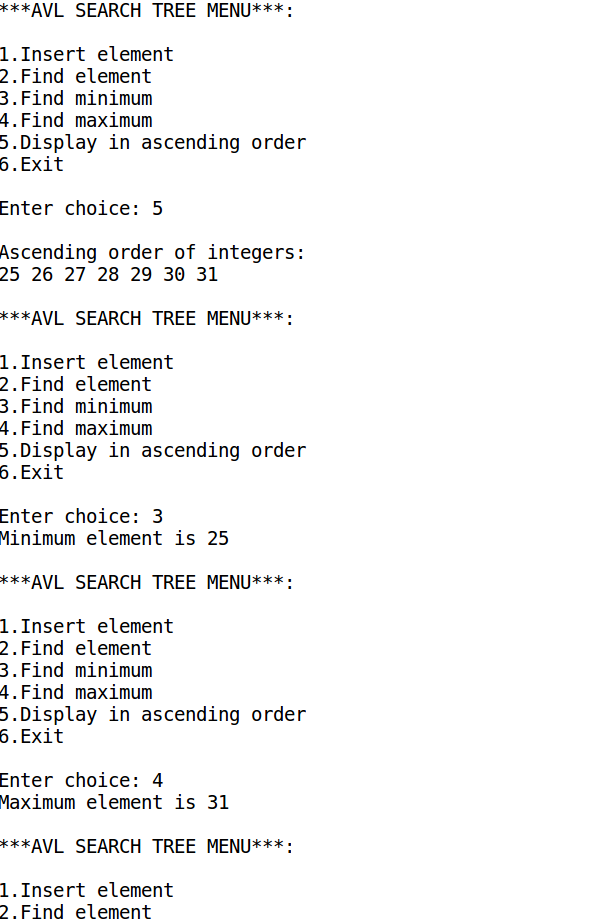


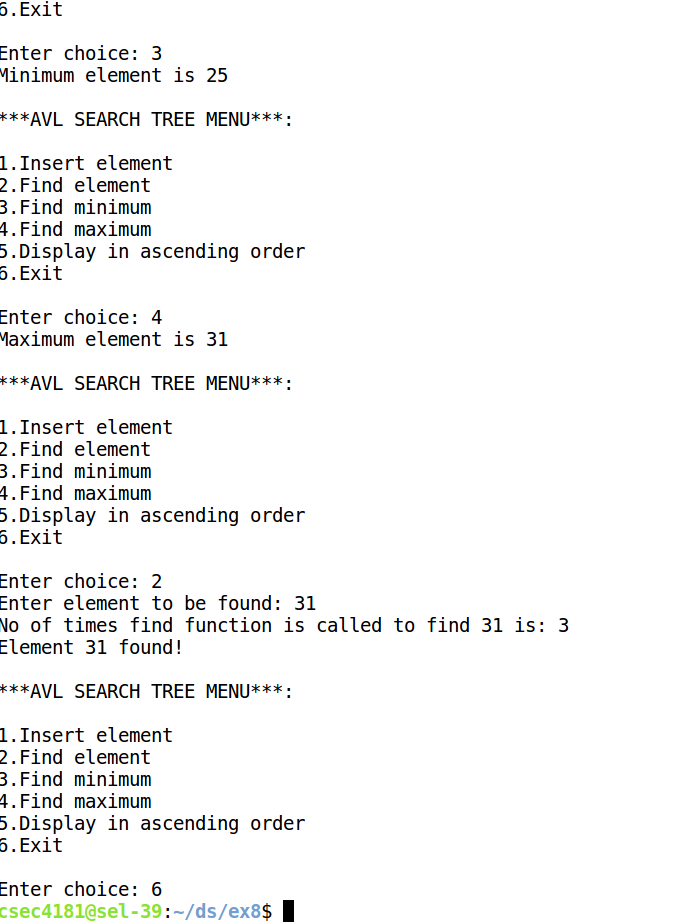












**PROGRAM CODE:**

#include<stdio.h>

#include<stdlib.h>

#include "bstADT.h"

void main()

{

searchtree t=NULL;

position pos;

intchoice,element;

do{

printf("\n\*\*\*BINARY SEARCH TREE MENU\*\*\*:\n\n1.Insert element\n2.Delete element\n3.Find element\n4.Find minimum\n5.Find maximum\n6.Display in ascending order\n7.Exit\n");

printf("\nEnter choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Enter element to be inserted: ");

scanf("%d",&element);

t=insert(element,t);

break;

case 2:

printf("Enter element to be deleted: ");

scanf("%d",&element);

t=delete(element,t);

break;

case 3:

printf("Enter element to be found: ");

scanf("%d",&element);

if(find(element,t)==NULL)

printf("\nElement %d not found!\n",element);

else

printf("\nElement %d found!\n",element);

break;

case 4:

pos=findmin(t);

printf("Minimum element is %d\n",pos->element);

break;

case 5:

pos=findmax(t);

printf("Maximum element is %d\n",pos->element);

break;

case 6:

printf("\nAscending order of integers: \n");

inorder(t);

printf("\n");

break;

case 7:

break;

default:

printf("Invalid choice\n");

}

}while(choice!=7);

}

***bstADT header file***

typedefstructtreenode \*position;

typedefstructtreenode \*searchtree;

typedefstructtreenode \*ptrtonode;

typedefintelementtype;

structtreenode

{

elementtype element;

searchtree left;

searchtree right;

};

searchtreemakeempty(searchtree t)

{

if (t!=NULL)

{

makeempty(t->left);

makeempty(t->right);

free(t);

}

return NULL;

}

position find(elementtype x, searchtree t)

{ static intcnt=0;

if(x==31)

{

cnt++;

if (t==NULL)

return NULL;

if (x<t->element)

return find (x, t->left);

else if (x>t->element)

return find(x,t->right);

else

{printf("No of times find function is called to find 31 is: %d",cnt); return t; }

}

else //x not 31

{

if (t==NULL)

return NULL;

if (x<t->element)

return find (x, t->left);

else if (x>t->element)

return find(x,t->right);

else

return t;

}

}

position findmin(searchtree t)

{

if(t==NULL)

return NULL;

else if(t->left==NULL)

return t;

else

return findmin(t->left);

}

/\*position findmin(searchtree t)

{

position p=t;

if(p!=NULL)

while(p->left!=NULL)

p=p->left;

return p;

intfindmin(searchtree t)

{

position p=t;

if(p!=NULL)

while(p->left!=NULL)

p=p->left;

return p->element;

}\*/

position findmax(searchtree t)

{

position p=t;

if(p!=NULL)

while(p->right!=NULL)

p=p->right;

return p;

}

searchtree insert(elementtype x, searchtree t)

{

if(t==NULL)

{

t=(searchtree)malloc(sizeof(structtreenode));

if(t==NULL)

printf("Out of space");

else

{

t->element=x;

t->left=t->right=NULL;

}

}

else if(x<t->element)

t->left=insert(x,t->left);

else if(x>t->element)

t->right=insert(x,t->right);

return t;

}

searchtree delete(elementtype x, searchtree t)

{

position tmpcell;

if(t==NULL)

printf("\nElement %d not found\n",x);

else if(x<t->element)

t->left=delete(x,t->left);

else if(x>t->element)

t->right=delete(x,t->right);

else if(t->left && t->right)

{

tmpcell=findmin(t->right);

t->element=tmpcell->element;

t->right=delete(t->element,t->right);

}

else

{

tmpcell=t;

if(t->left==NULL)

t=t->right;

else if(t->right==NULL)

t=t->left;

free(tmpcell);

}

return t;

}

void inorder(searchtree t)

{ if ( t == NULL)

return;

inorder( t->left);

printf("%d ", t->element);

inorder( t->right);

}

**OUTPUT:**

