**UCS 2312 Data Structures Lab**

**Assignment 7: Implementation of AVL Tree**

**Date of Assignment: 30.10.2023**

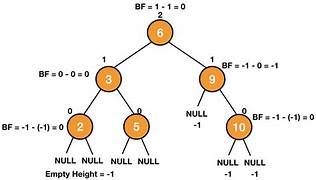
Design an ADT for the AVL Tree data structure with the following functions. Each node consists of a character data, address of left, right and parent nodes

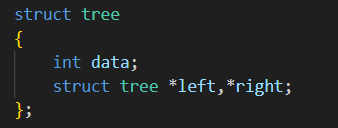
1. insertAVL(t, data) – insert data into BST
2. hierarchical(t) – display the tree in hierarchical fashion
3. findParent(t, key) – will return the parent of the given data

Demonstrate the AVL ADT with the insertion of the following character data one at a time.

**H, I, J, B, A, E, C, F, D, G, K, L**

**Data Structure – AVL Tree:**



****

**Algorithm –**

**Algorithm: Insert data into BST**

Input – Pointer to tree, data to be added to tree

Output – struct tree \*

1. if (t==NULL)

t=(struct tree \*)malloc(sizeof(struct tree));

t->data=data;

t->right=NULL;

t->left=NULL;

1. else if(data<t->data)

t->left=insert(t->left,data)

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

if(data<t->left->data)

t=singlerotateleft(t)

else

t=doublerotateleft(t)

1. else if(data>t->data)

t->right=insert(t->right,data)

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

if(data<t->left->data)

t=singlerotateright(t)

else

t=doublerotateright(t)

1. t->height=max(heigth(t->left),height(t->right))+1
2. return t;

**Algorithm: display the tree in hierarchical fashion**

Input – Pointer to tree, space

Output – void

1. if (t==NULL)

return

1. space+=1
2. hierarchical(t->right,space)
3. print \n
4. for i from 0 till space-1

print \t

1. print data and \n
2. hierarchical(t->left,space)

**Algorithm: will return the parent of the given data**

Input – Pointer to tree, key

Output – struct tree \*

1. if (t->left==NULL && t->right==NULL)

return NULL

1. else if(t->left->data==key || t->right->data==key)

return t

1. else if(t->data>key)

findParent(t->left,key)

1. else if(t->data<key)

findParent(t->right,key)

**AVLtree.h code:**

struct tree

{

int data;

struct tree \*left,\*right;

int height;

};

#define c 1

int height(struct tree \*t)

{

if(t==NULL)

return -1;

else

return t->height;

}

int max(int a,int b)

{

if(a>b)

return a;

return b;

}

struct tree\* singlerotateleft(struct tree \*k2)

{

struct tree \*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),height(k1->right)) + 1;

return k1;

}

struct tree\* singlerotateright(struct tree \*k1)

{

struct tree \*k2 = k1->right;

k1->right = k2->left;

k2->left = k1;

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

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

return k2;

}

struct tree\* findParent(struct tree\* t,int key)

{

if (t->left==NULL && t->right==NULL)

return NULL;

else if (t->left->data==key || t->right->data==key)

{

return t;

}

else if (t->data>key)

{

findParent(t->left, key);

}

else if (t->data<key)

{

findParent(t->right, key);

}

}

void hierarchical(struct tree \*t, int space)

{

if(t == NULL)

return;

space+=c;

hierarchical(t->right, space);

printf("\n");

for(int i = 0 ; i < space; i++)

{

printf("\t");

}

printf("%d\n", t->data);

hierarchical(t->left, space);

}

struct tree \*doublerotateleft(struct tree \* k1)

{

k1->left = singlerotateright(k1->left);

return singlerotateleft(k1);

}

struct tree \*doublerotateright(struct tree \* k1)

{

k1->right = singlerotateleft(k1->right);

return singlerotateright(k1);

}

struct tree\* insert(struct tree \*t,int x)

{

if(t==NULL)

{

t=(struct tree \*)malloc(sizeof(struct tree));

t->data=x;

t->height=0;

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

}

else if(x<t->data)

{

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

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

{

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

t=singlerotateleft(t);

else

t=doublerotateleft(t);

}

}

else if(x>t->data)

{

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

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

{

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

t=singlerotateright(t);

else

t=doublerotateright(t);

}

}

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

return t;

}

**AVLtree.c code:**

#include<stdio.h>

#include<stdlib.h>

#include"AVLtree.h"

void main()

{

struct tree\* t=NULL;

int choice=100;

int el;

while(choice!=4)

{

printf("\n\n1.Insert\n2.Print\n3.Find Parent\n4.Exit\nChoice = ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Enter the element: ");

scanf("%d",&el);

t=insert(t,el);

break;

case 2:

hierarchical(t,0);

printf("\n\n");

break;

case 3:

printf("Enter the element: ");

scanf("%d",&el);

struct tree \*parent=findParent(t,el);

if(parent!=NULL)

printf("Parent = %d",parent->data);

else

printf("Element Not Found");

break;

case 4:

exit(0);

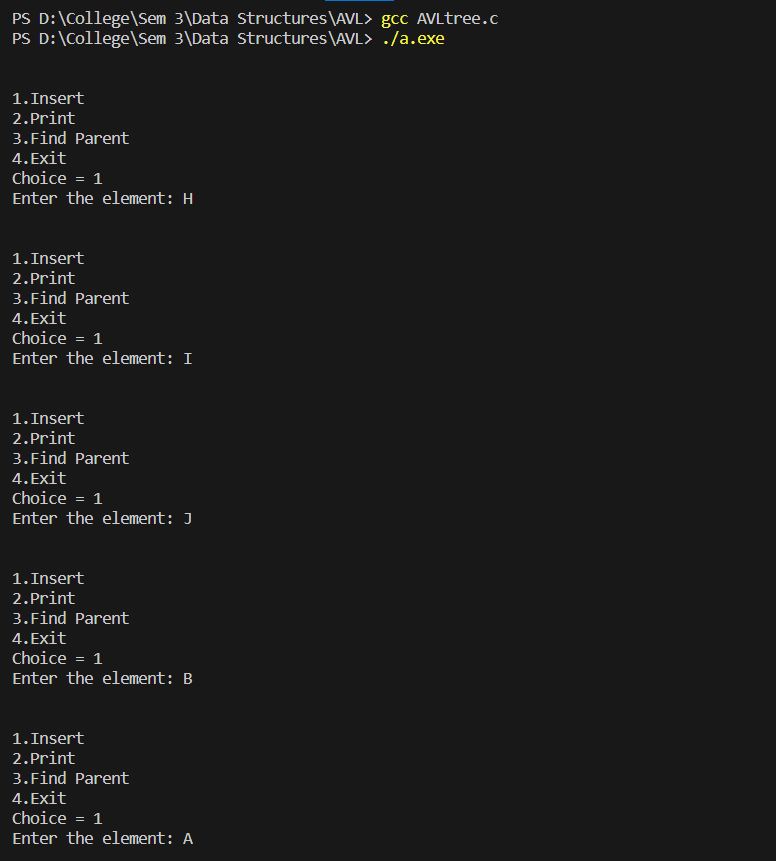
break;

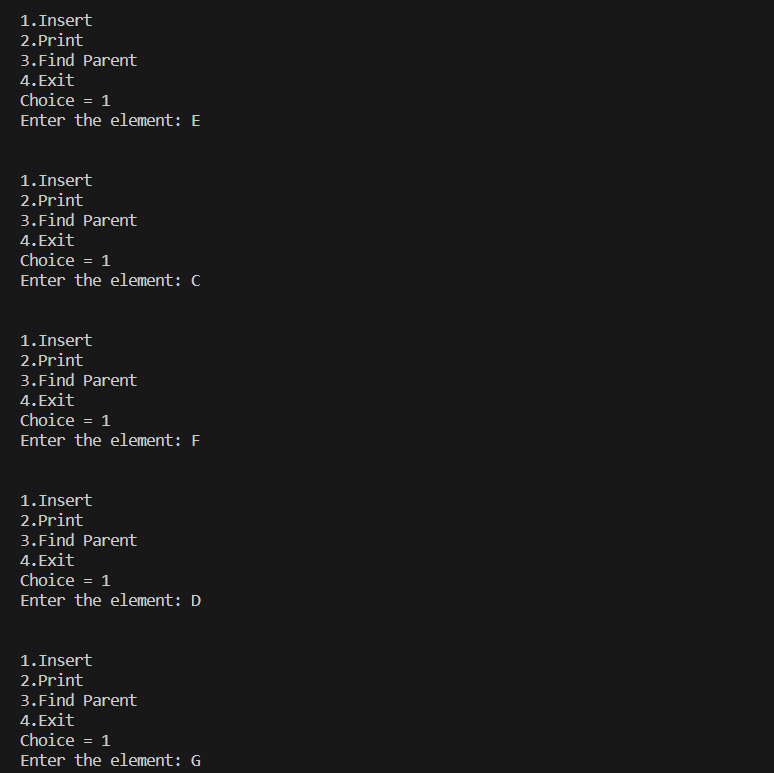
}

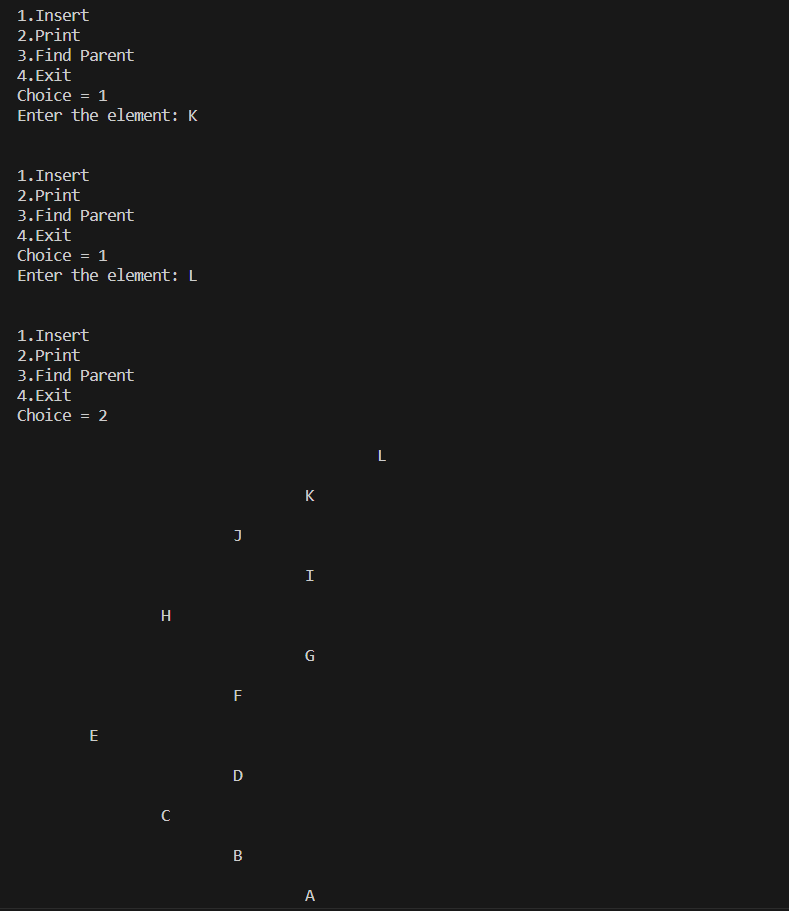
}

}

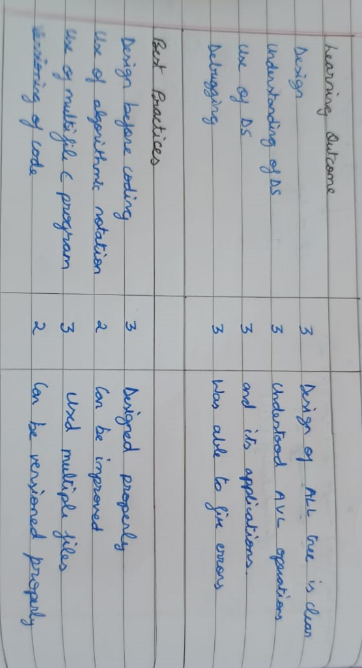
**Output Screen:**

****

****

****

**Learning Outcome:**

****