

LAB 7 :LAB EXCERSIZE :

/*1)Modify the solved exercise to find the balance factor for every node in the binary search tree.*/

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <stdbool.h>
```

```
#include <limits.h>
```

```
struct node
```

```
{
```

```
int data;
```

```
int balance_factor;
```

```
struct node *rightLink;
```

```
struct node *leftLink;
```

```
};
```

```
struct node *newNode(int item)
```

```
{
```

```
struct node *root = (struct node *)malloc(sizeof(struct node));
```

```
root->data = item;
```

```
root->balance_factor = 0;
```

```
root->leftLink = NULL;
```

```
root->rightLink = NULL;
```

```
return root;
```

```
}
```

```
void inorder(struct node *root)
```

```
{
```

```
if (root != NULL)
```

```
{
```

```
inorder(root->leftLink);
```

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

```
inorder(root->rightLink);
```

```
}
```

```
}
```

```
void preorder(struct node *root)
```

```
{  
if (root != NULL)  
{  
printf("%d\t", root->data);  
preorder(root->leftLink);  
preorder(root->rightLink);  
}  
}
```

```
void postorder(struct node *root)  
{  
if (root != NULL)  
{  
postorder(root->leftLink);  
postorder(root->rightLink);  
printf("%d\t", root->data);  
}  
}
```

```
bool search(struct node *root, int key)  
{  
if (root == NULL || root->data == key)  
{  
return true;  
}  
if (root->data < key)  
{  
return search(root->rightLink, key);  
}  
else if (root->data > key)  
{  
return search(root->leftLink, key);  
}  
return false;  
}
```

```
struct node *insert(struct node *node, int key)  
{  
if (node == NULL)  
return newNode(key);  
if (key < node->data)  
{
```

```
node->leftLink = insert(node->leftLink, key);
}
else if (key > node->data)
{
node->rightLink = insert(node->rightLink, key);
}
return node;
}
```

```
int maxVal (int val1, int val2)
{
if (val1 > val2)
{
return val1;
}
return val2;
}
```

```
int findHeight(struct node *root)
{
int count_nodes1 = 1;
int count_nodes2 = 1;

if (root == NULL)
{

return 0;
}
else
{
if (root->leftLink != NULL)
{
count_nodes1 += findHeight(root->leftLink);
}
if (root->rightLink != NULL)
{
count_nodes2 += findHeight(root->rightLink);
}
}
return maxVal(count_nodes1, count_nodes2);
}

void balance_assign(struct node *root)
```

```

{
if (root != NULL)
{

root->balance_factor = findHeight(root->leftLink) - findHeight(root->rightLink);
balance_assign(root->leftLink);
balance_assign(root->rightLink);
}
}

```

```

void balance_factor_traversal(struct node *root)
{
if (root != NULL)
{
balance_factor_traversal(root->leftLink);
printf("%d --> %d \n", root->data, root->balance_factor);
balance_factor_traversal(root->rightLink);
}
}

```

```

int main()
{
printf("Enter the number of Case : ");
int temp;
scanf("%d", &temp);
while (temp-- > 0)
{
printf("Enter the size of an array : ");
int size;
scanf("%d", &size);
int *array = (int *)malloc(sizeof(int) * size);

printf("Enter the element of an array :");
for (int i = 0; i < size; i++)
{
scanf("%d", array + i);
}
struct node *root = NULL;
root = insert(root, *(array + 0));
for (int i = 1; i < size; i++)
{

```

```

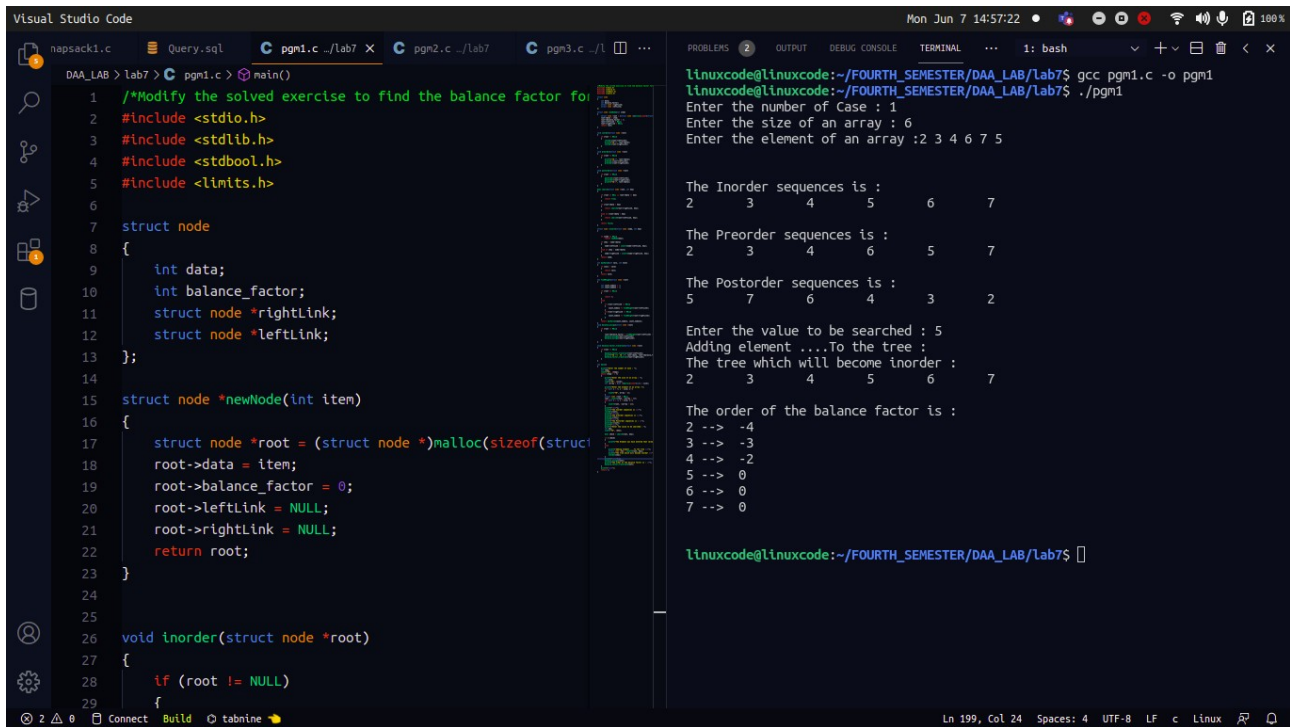
insert(root, *(array + i));
}
printf("\n\n");
printf("The Inorder sequences is :\n");
inorder(root);
printf("\n\n");
printf("The Preorder sequences is :\n");
preorder(root);
printf("\n\n");
printf("The Postorder sequences is : \n");
postorder(root);
printf("\n\n");
printf("Enter the value to be searched : ");
int key;
scanf("%d", &key);

bool check = search(root, key);

if (!check)
{
printf("The Element you have entered that laready present in Array \n");
}
else
{
printf("Adding element ....To the tree :\n");
struct node *new = insert(root, key);
printf("The tree which will become inorder :\n");
inorder(new);
}
printf("\n\n");
balance_assign(root);
printf("The order of the balance factor is : \n");
balance_factor_traversal(root);
}
printf("\n\n");
return 0;
}

```

OUTPUT :



The screenshot shows a Visual Studio Code editor with a C program in the left pane and its execution output in the terminal on the right. The program calculates the balance factor for an AVL tree and displays the inorder, preorder, and postorder sequences. The terminal output shows the program being compiled and run, with user input for the number of cases, array size, and elements, followed by the calculated sequences and balance factors.

```
Visual Studio Code
DAA_LAB > lab7 > pgm1.c > main()
1  /*Modify the solved exercise to find the balance factor for
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <stdbool.h>
5  #include <limits.h>
6
7  struct node
8  {
9      int data;
10     int balance_factor;
11     struct node *rightLink;
12     struct node *leftLink;
13 };
14
15 struct node *newNode(int item)
16 {
17     struct node *root = (struct node *)malloc(sizeof(struct node));
18     root->data = item;
19     root->balance_factor = 0;
20     root->leftLink = NULL;
21     root->rightLink = NULL;
22     return root;
23 }
24
25
26 void inorder(struct node *root)
27 {
28     if (root != NULL)
29     {
30         inorder(root->leftLink);
31         printf("%d ", root->data);
32         inorder(root->rightLink);
33     }
34 }
35
36 int main()
37 {
38     int n;
39     scanf("%d", &n);
40     int size;
41     scanf("%d", &size);
42     int arr[size];
43     for (int i = 0; i < size; i++)
44     {
45         scanf("%d", &arr[i]);
46     }
47     struct node *root = NULL;
48     for (int i = 0; i < size; i++)
49     {
50         root = newNode(arr[i]);
51     }
52     inorder(root);
53     printf("\n");
54     return 0;
55 }
```

Terminal Output:

```
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab7$ gcc pgm1.c -o pgm1
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab7$ ./pgm1
Enter the number of Case : 1
Enter the size of an array : 6
Enter the element of an array :2 3 4 6 7 5

The Inorder sequences is :
2 3 4 5 6 7

The Preorder sequences is :
2 3 4 6 5 7

The Postorder sequences is :
5 7 6 4 3 2

Enter the value to be searched : 5
Adding element ...To the tree :
The tree which will become inorder :
2 3 4 5 6 7

The order of the balance factor is :
2 --> -4
3 --> -3
4 --> -2
5 --> 0
6 --> 0
7 --> 0

linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab7$
```

/*2) Write a program to create the AVL tree by iterative insertion.*/

```
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
    int info;
    struct node *left, *right;
} NODE;

struct Stack
{
    int top;
    unsigned capacity;
    NODE **array;
};

struct Stack *createStack(unsigned capacity)
{
    struct Stack *stack = (struct Stack *)malloc(sizeof(struct Stack));
```

```
stack->capacity = capacity;
stack->top = -1;
stack->array = (NODE **)malloc(stack->capacity * sizeof(NODE *));
return stack;
}
```

```
int isFull(struct Stack *stack)
{
return stack->top == stack->capacity - 1;
}
```

```
int isEmpty(struct Stack *stack)
{
return stack->top == -1;
}
```

```
void push(struct Stack *stack, NODE *item)
{
if (isFull(stack))
return;
stack->array[++stack->top] = item;
// printf("%d pushed to stack\n", item);
}
```

```
NODE *pop(struct Stack *stack)
{
if (isEmpty(stack))
return NULL;
return stack->array[stack->top--];
}
```

```
NODE *peek(struct Stack *stack)
{
if (isEmpty(stack))
return NULL;
return stack->array[stack->top];
}
```

```
int max(int x, int y)
{
return x > y ? x : y;
}
```

```
int height(NODE *root)
{
    if (root == NULL)
        return 0;
    return 1 + max(height(root->left), height(root->right));
}
```

```
int getBalFactor(NODE *root)
{
    return height(root->left) - height(root->right);
}
```

```
NODE *rightRotate(NODE *y)
{
    NODE *x = y->left;
    NODE *T2 = x->right;
    x->right = y;
    y->left = T2;
    return x;
}
```

```
NODE *leftRotate(NODE *x)
{
    NODE *y = x->right;
    NODE *T2 = y->left;
    y->left = x;
    x->right = T2;
    return y;
}
```

```
NODE *create(NODE *root, int x)
{
    struct Stack *stack = createStack(100);
    NODE *newnode = (NODE *)malloc(sizeof(NODE));
    newnode->info = x;
    newnode->right = NULL;
    newnode->left = NULL;
    NODE *curr = root;
    NODE *trail = NULL;
    while (curr != NULL)
    {
```



```

trail = curr;
push(stack, trail);
if (x < curr->info)
curr = curr->left;
else if (x > curr->info)
curr = curr->right;
else
{
printf("Duplicate element\n");
exit(0);
}
}
if (trail == NULL)
{
trail = newnode;
return trail;
}
else if (x < trail->info)
trail->left = newnode;
else
trail->right = newnode;
NODE *newRoot = root;
while (!isEmpty(stack))
{
NODE *toBalance = pop(stack);
NODE *prev = peek(stack);
int balance = getBalFactor(toBalance);
if (balance > 1 && x < toBalance->left->info)
{
toBalance = rightRotate(toBalance);
}
else if (balance < -1 && x > toBalance->right->info)
{
toBalance = leftRotate(toBalance);
}
else if (balance > 1 && x > toBalance->left->info)
{
toBalance->left = leftRotate(toBalance->left);
toBalance = rightRotate(toBalance);
}
else if (balance < -1 && x < toBalance->right->info)
{

```

```
toBalance->right = rightRotate(toBalance->right);
toBalance = leftRotate(toBalance);
}
if (prev != NULL && prev->info > toBalance->info)
{
prev->left = toBalance;
}
else if (prev != NULL)
{
prev->right = toBalance;
}
newRoot = toBalance;
}
return newRoot;
}
```

```
void inorder(NODE *root)
{
if (root != NULL)
{
inorder(root->left);
printf("%5d", root->info);
inorder(root->right);
}
}
```

```
void postorder(NODE *root)
{
if (root != NULL)
{
postorder(root->left);
postorder(root->right);
printf("%5d", root->info);
}
}
```

```
void preorder(NODE *root)
{
if (root != NULL)
{
printf("%5d", root->info);
preorder(root->left);
}
```

```

preorder(root->right);
}
}

int printBalanceFactor(NODE *root)
{
if (root != NULL)
{
printf("\nBalance factor of node with value %d : %d", root->info,
getBalFactor(root));
printBalanceFactor(root->left);
printBalanceFactor(root->right);
}
}

void main()
{
int n, x, ch, i;
NODE *root;
root = NULL;
printf("-----Menu-----\n");
printf(" 1. Insert\n 2. All traversals\n 3. Get Balance Factor\n 4. Exit\n");
while (1)
{
printf("Enter your choice : ");
scanf("%d", &ch);
switch (ch)
{
case 1:
printf("Enter node (do not enter duplicate nodes) : ");
scanf("%d", &x);
root = create(root, x);
break;
case 2:
printf("\n*****");
printf("\nInorder traversal : ");
inorder(root);
printf("\nPreorder traversal : ");
preorder(root);
printf("\nPostorder traversal : ");
postorder(root);

```

```

printf("\n\n*****\n\n");
break;
case 3:
printf("\n*****");
printBalanceFactor(root);
printf("\n\n*****\n\n");
break;
case 4:
exit(0);
default:
printf("Invalid Choice\n");
}}}

```

OUTPUT :

The screenshot shows the Visual Studio Code editor with a C program for AVL tree operations. The code includes headers, defines a node structure, a stack, and functions for creating a stack, checking if it's full, and performing insertions. The terminal output shows the program being compiled and run, displaying a menu with options: 1. Insert, 2. All traversals, 3. Get Balance Factor, and 4. Exit. The user enters choice 1, then enters nodes 2 through 7. The program then displays the Inorder, Preorder, and Postorder traversals of the tree. Next, the user enters choice 3, and the program displays the balance factor for each node. Finally, the user enters choice 4, and the program exits.

```

Visual Studio Code
DAA_LAB > lab7 > C pgm2.c > ...
1  /*2) Write a program to create the AVL tree by iterative tr
2
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  typedef struct node
7  {
8      int info;
9      struct node *left, *right;
10 } NODE;
11
12 struct Stack
13 {
14     int top;
15     unsigned capacity;
16     NODE **array;
17 };
18
19 struct Stack *createStack(unsigned capacity)
20 {
21     struct Stack *stack = (struct Stack *)malloc(sizeof(sti
22     stack->capacity = capacity;
23     stack->top = -1;
24     stack->array = (NODE **)malloc(stack->capacity * sizeof
25     return stack;
26 }
27
28 int isFull(struct Stack *stack)
29 {

```

```

Mon Jun 7 15:49:05
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab7$ gcc pgm2.c -o pgm2
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab7$ ./pgm2
-----Menu-----
1. Insert
2. All traversals
3. Get Balance Factor
4. Exit
Enter your choice : 1
Enter node (do not enter duplicate nodes) : 2
Enter your choice : 1
Enter node (do not enter duplicate nodes) : 3
Enter your choice : 1
Enter node (do not enter duplicate nodes) : 4
Enter your choice : 1
Enter node (do not enter duplicate nodes) : 6
Enter your choice : 1
Enter node (do not enter duplicate nodes) : 7
Enter your choice : 1
Enter node (do not enter duplicate nodes) : 5
Enter your choice : 2
*****
Inorder traversal : 2 3 4 5 6 7
Preorder traversal : 4 3 2 6 5 7
Postorder traversal : 2 3 5 7 6 4
*****
Enter your choice : 3
*****
Balance factor of node with value 4 : 0
Balance factor of node with value 3 : 1
Balance factor of node with value 2 : 0
Balance factor of node with value 6 : 0
Balance factor of node with value 5 : 0
Balance factor of node with value 7 : 0
*****
Enter your choice : 4
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB/lab7$

```

ANALYSIS PART :

1.

Most of the BST operations (e.g., search, max, min, insert, delete.. etc) take $O(h)$ time where h is the height of the BST. The cost of these operations may become $O(n)$ for a skewed Binary tree. So best case time complexity is $O(\log n)$ and worst case time complexity is $O(n)$.

2.

The height of an AVL tree is always $O(\log n)$ where n is the number of nodes in the tree. Thus most of the BST operations (e.g., search, max, min, insert, delete.. etc) take $O(\log n)$ time.

