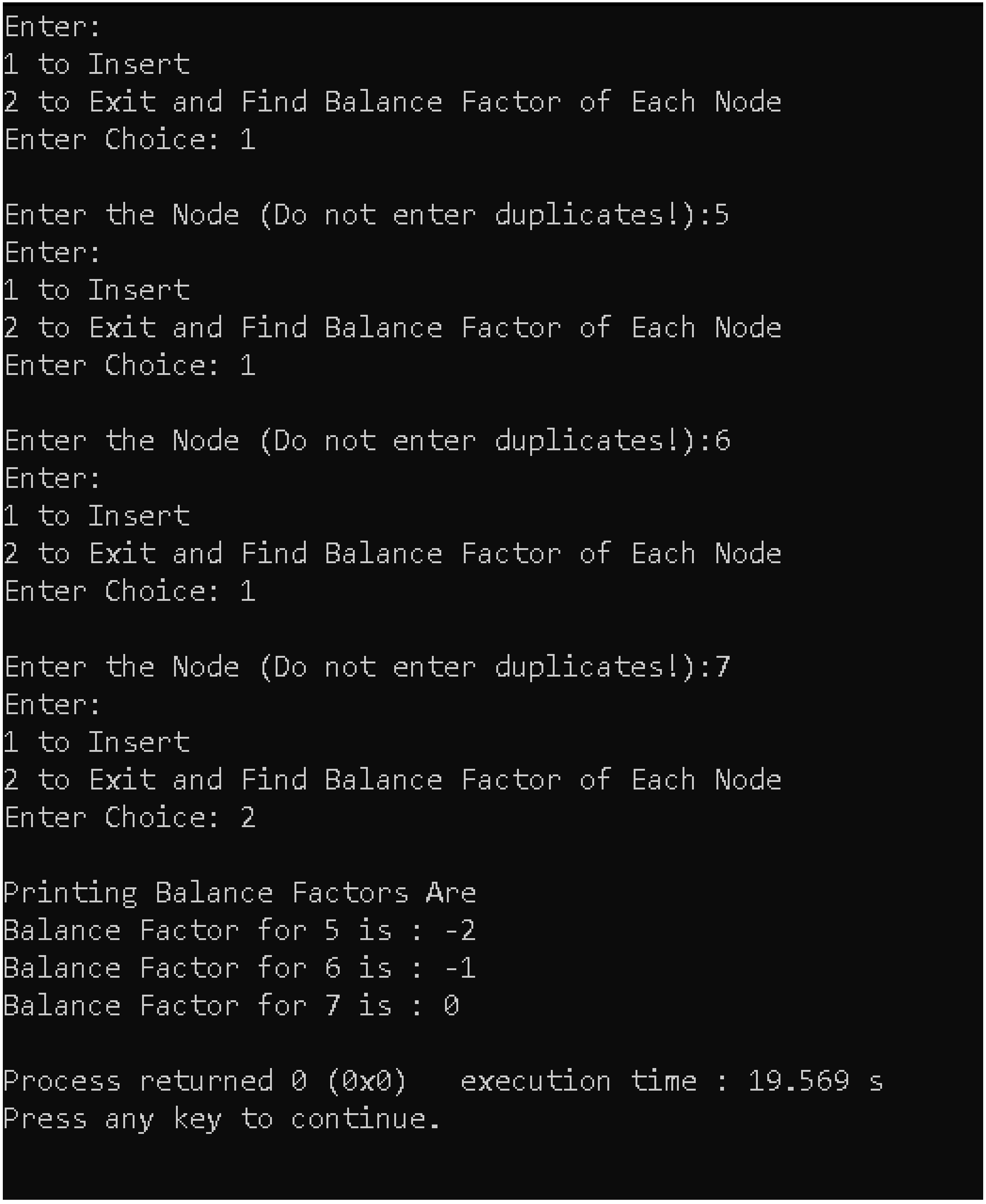
Lab Exercises:

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

Program:

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

struct Node
{
    int data;
    struct Node* left;
    struct Node* right;
};

typedef struct Node Node;

int max(int a, int b)
{
    return (a>b)?a:b;
}

Node* getNode(Node* t, int data)
{
            if(t == NULL)
            {
            t = (Node*)malloc(sizeof(Node));
            t->data = data;
            t->right = t->left = NULL;
    }
    else if(data > t->data)
            t->right = getNode(t->right, data);
    else if(data < t->data)
            t->left = getNode(t->left, data);
    else
    {
            printf("Duplicate Node Inserted!\n");
            exit(0);
    }
    return t;
}

int getHeight(Node* root)
{
    if(root == NULL)
        return 0;
    else
        return 1 + max(getHeight(root->left), getHeight(root->right));
}

void balfact(Node* root)
{
    if(root != NULL)
    {
        balfact(root->left);
        printf("Balance Factor for %d is : %d\n", root->data, (getHeight(root->left) - getHeight(root->right)));
        balfact(root->right);
    }
}

int main()
{
            int n, x, ch, i;
            Node *root;
            root = NULL;
            while(1)
            {
                        printf("Enter: \n1 to Insert\n2 to Exit and Find Balance Factor of Each Node\n");
                        printf("Enter Choice: ");
                        scanf("%d", &ch);
                        if(ch == 1)
                        {
                                    printf("\nEnter the Node (Do not enter duplicates!):");
                                    scanf("%d", &x);
                                    root = getNode(root, x);
                        }
                        else if(ch == 2)
                        {
                                    printf("\nPrinting Balance Factors Are\n");
                                    balfact(root);
                                    break;
                        }
                        else
                        {
                                    printf("\nInvalid Option");
                                    exit(0);
                        }
            }

            return 0;
}
```

Output:



```
Enter:
1 to Insert
2 to Exit and Find Balance Factor of Each Node
Enter Choice: 1

Enter the Node (Do not enter duplicates!):5
Enter:
1 to Insert
2 to Exit and Find Balance Factor of Each Node
Enter Choice: 1

Enter the Node (Do not enter duplicates!):6
Enter:
1 to Insert
2 to Exit and Find Balance Factor of Each Node
Enter Choice: 1

Enter the Node (Do not enter duplicates!):7
Enter:
1 to Insert
2 to Exit and Find Balance Factor of Each Node
Enter Choice: 2

Printing Balance Factors Are
Balance Factor for 5 is : -2
Balance Factor for 6 is : -1
Balance Factor for 7 is : 0

Process returned 0 (0x0)   execution time : 19.569 s
Press any key to continue.
```

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

Program:

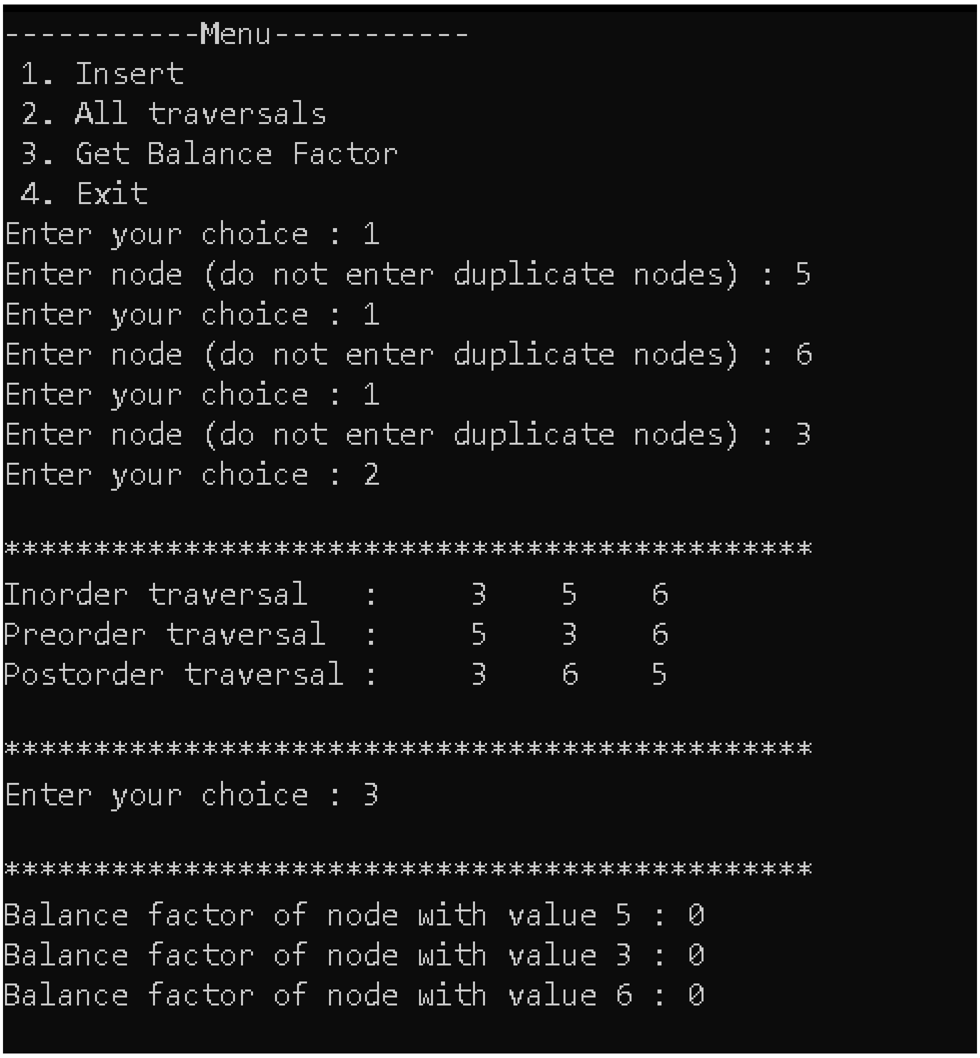
```
#include <stdio.h>
#include <stdlib.h>

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

Output:



```
-----------Menu-----------
 1. Insert
 2. All traversals
 3. Get Balance Factor
 4. Exit
Enter your choice : 1
Enter node (do not enter duplicate nodes) : 5
Enter your choice : 1
Enter node (do not enter duplicate nodes) : 6
Enter your choice : 1
Enter node (do not enter duplicate nodes) : 3
Enter your choice : 2

*********************************************
Inorder traversal   :     3    5    6
Preorder traversal  :     5    3    6
Postorder traversal :     3    6    5

*********************************************
Enter your choice : 3

*********************************************
Balance factor of node with value 5 : 0
Balance factor of node with value 3 : 0
Balance factor of node with value 6 : 0
```