Assignment – 5

23. Implement an binary search tree and write functions for insertion & deletion.

Source Code: main()

```
#include <iostream>
using namespace std;
typedef struct Node
{
   Node *left, *right;
   int data;
   Node(int data)
   {
      this->left = NULL;
      this->data = data;
      this->right = NULL;
   }
} Node;
Node *root = NULL;
void createBST();
Node *createNode(int);
void freeTree(Node *root);
void printTree(Node *, int);
void insertBST(Node *&, int);
Node *deleteNodeBST(Node *, int);
Node *inOrderPredecessor(Node *ptr);
int main(){
   createBST(); // Initialize the BST with some predefined values.
   while (1)    {
      cout << "\nMenu:\n";
      cout << "1. Insert Node in the tree\n";
      cout << "2. Delete Node from tree\n";
      cout << "0. Exit\n";
      printTree(root, 0);
      cout << "\nEnter your choice: ";
      int choice, value;
      cin >> choice;
      switch (choice)      {
      case 1:
         cout << "Enter value to insert: ";
         cin >> value;
         insertBST(root, value);
         break;
      case 2:
         cout << "Enter value to delete: ";
         cin >> value;
         root = deleteNodeBST(root, value);
         break;
      case 0:
         freeTree(root);
         cout << "\nExiting...\n\n";
         exit(0);
      default:
         cout << "Invalid choice. Try again." << endl;
      }
   }
```

Source Code: insertBST()

```
void insertBST(Node *&ptr, int value)
{
    if (ptr == NULL)
    {
        ptr = createNode(value);
        return;
    }

    if (value < ptr->data)
        insertBST(ptr->left, value);

    else if (value > ptr->data)
        insertBST(ptr->right, value);

    else
        cout << "\nDuplication is not allowed! " << value << " already exists.\n";
}
```

Source Code: createBST()

```
void createBST()
{
    root = NULL; // Reset root for initialization

    insertBST(root, 8);
    insertBST(root, 3);
    insertBST(root, 10);
    insertBST(root, 1);
    insertBST(root, 6);
    insertBST(root, 14);
    insertBST(root, 4);
    insertBST(root, 7);
    insertBST(root, 13);
}
```

Source Code: freeTree()

```
void freeTree(Node *root)
{
    if (root == nullptr)
        return;

    freeTree(root->left);
    freeTree(root->right);

    delete root;
}
```

Source Code: printTree()

```
void printTree(Node *root, int space = 0)
{
   if (root == nullptr)
      return;

   space += 5;

   printTree(root->right, space);

   cout << '\n';
   for (int i = 5; i < space; i++)
   {
      cout << ' ';
   }
   out << root->data;

   printTree(root->left, space);
}
```

Source Code: inOrderPredecessor ()

```
Node *inOrderPredecessor(Node *ptr)
{
   ptr = ptr->left;
   while (ptr->right != NULL)
      ptr = ptr->right;
   return ptr;
}
```

Source Code: createNode()

```
Node *createNode(int data)
{
   return new Node(data);
}
```

Source Code: deleteNodeBST()

```
Node *deleteNodeBST(Node *ptr, int value)
{
   if (ptr == NULL)
   {
      cout << "Value " << value << " not found in the tree.\n";
      return NULL;
   }

   if (value < ptr->data)
   {
      ptr->left = deleteNodeBST(ptr->left, value);
   }
   else if (value > ptr->data)
   {
      ptr->right = deleteNodeBST(ptr->right, value);
   }
   else
   {
      // Node with one child or no child
      if (ptr->left == NULL)
      {
         Node *temp = ptr->right;
         delete ptr;
         return temp;
      }
      else if (ptr->right == NULL)
      {
         Node *temp = ptr->left;
         delete ptr;
         return temp;
      }

      // Node with two children
      Node *inPre = inOrderPredecessor(ptr);
      ptr->data = inPre->data;
      ptr->left = deleteNodeBST(ptr->left, inPre->data);
   }
   return ptr;
}
```

Output

```
Menu:
1. Insert Node in the tree
2. Delete Node from tree
0. Exit
        14
            13
    10
8
            7
        6
            4
    3
        1
Enter your choice: 1
Enter value to insert: 5

Menu:
1. Insert Node in the tree
2. Delete Node from tree
0. Exit
        14
            13
    10
8
            7
        6
                5
            4
    3
        1
Enter your choice: 2
Enter value to delete: 6

Menu:
1. Insert Node in the tree
2. Delete Node from tree
0. Exit
        14
            13
    10
8
            7
        5
            4
    3
        1
Enter your choice: 0
Exiting...
```

24. Write a program to find the height of a binary tree.

Source Code: main()

```
#include <iostream>
using namespace std;

typedef struct Node
{
   Node *left, *right;
   int data;

   Node(int data)
   {
      this->left = NULL;
      this->data = data;
      this->right = NULL;
   }
} Node;

Node *root = NULL;

void createBinaryTree();
Node *createNode(int);
void freeTree(Node *root);
void printTree(Node *, int);
int heightCalculate(Node *);

int main()
{
   createBinaryTree(); // Initialize the BinaryTree with some predefined
values.

   printTree(root, 0);

   cout << "\n\n"
        << "The height of the current tree is: " << heightCalculate(root)
        << "\n\n"
        << endl;

   return 0;
}
```

Source Code: insertBST()

```
void insertBST(Node *&ptr, int value)
{
    if (ptr == NULL)
    {
        ptr = createNode(value);
        return;
    }

    if (value < ptr->data)
        insertBST(ptr->left, value);

    else if (value > ptr->data)
        insertBST(ptr->right, value);

    else
        cout << "\nDuplication is not allowed! " << value << " already exists.\n";
}
```

Source Code: createBST()

```
void createBST()
{
    root = NULL; // Reset root for initialization

    insertBST(root, 8);
    insertBST(root, 3);
    insertBST(root, 10);
    insertBST(root, 1);
    insertBST(root, 6);
    insertBST(root, 14);
    insertBST(root, 4);
    insertBST(root, 7);
    insertBST(root, 13);
}
```

Source Code: freeTree()

```
void freeTree(Node *root)
{
    if (root == nullptr)
        return;

    freeTree(root->left);
    freeTree(root->right);

    delete root;
}
```

Source Code: printTree()

```
void printTree(Node *root, int space = 0)
{
    if (root == nullptr)
        return;

    space += 5;

    printTree(root->right, space);

    cout << '\n';
    for (int i = 5; i < space; i++)
    {
        cout << ' ';
    }
    out << root->data;

    printTree(root->left, space);
}
```

Source Code: heightCalculate()

```
int heightCalculate(Node *root)
{
    if (root == NULL)
        return 0;
    int left_height = heightCalculate(root->left);
    int right_height = heightCalculate(root->right);

    return max(left_height, right_height) + 1;
}
```

Source Code: createNode()

```
Node *createNode(int data)
{
    return new Node(data);
}
```

Output

```
        14
            13
    10
8
            7
        6
            4
    3
        1
```

The height of the current tree is: 4