Q 1)

```
#include<iostream>
#include<stdlib.h>
using namespace std;

template <class temp>
class Node
{
   public :
      temp data;
      Node *parent, *left, *right;
};

template <class temp>
class BinarySearchTree
{
   public :
      Node<temp> *root, *ptr;

      BinarySearchTree()
      {
         root = NULL;
      }

      void insertNode()
      {
         temp d;
         cout<<"Enter the data to be inserted into the tree : ";
         cin>>d;

         if(root==NULL)
         {
            root = new Node<temp>();
            root->data = d;
            root->parent = NULL;
            root->left = NULL;
            root->right = NULL;
         }
         else
         {
            ptr = root;
            bool flag = true;

            Node<temp> *newnode = new Node<temp>();
            newnode->data = d;
            newnode->left = NULL;
            newnode->right = NULL;

            while(ptr!=NULL && flag==true)
            {
               if(d == ptr->data)
               {
                  cout<<"The element entered is already present in the tree."<<endl;
                  flag = false;
                  break;
               }
               else if(d > ptr->data)
               {
```

```
                if(ptr->right == NULL)
                {
                    ptr->right = newnode;
                    newnode->parent = ptr;
                    cout<<"The entered element has been inserted into the tree."<<endl;
                    flag = false;
                }
                else
                {
                    ptr = ptr->right;
                }
            }
            else
            {
                if(ptr->left == NULL)
                {
                    ptr->left = newnode;
                    newnode->parent = ptr;
                    cout<<"The entered element has been inserted into the tree."<<endl;
                    flag = false;
                }
                else
                {
                    ptr = ptr->left;
                }
            }

        }
    }
}

void preorder(Node<temp> *ptr)
{
    if(ptr!=NULL)
    {
        cout<<ptr->data<<'\t';
        preorder(ptr->left);
        preorder(ptr->right);
    }
    return;
}

void inorder(Node<temp> *ptr)
{
    if(ptr!=NULL)
    {
        inorder(ptr->left);
        cout<<ptr->data<<'\t';
        inorder(ptr->right);
    }
    return;
}

void postorder(Node<temp> *ptr)
{
    if(ptr!=NULL)
    {
        postorder(ptr->left);
        postorder(ptr->right);
        cout<<ptr->data<<'\t';
```

```
    }
    return;
}

bool search(Node<temp> *ptr, temp d)
{
    if(ptr==NULL)
    {
        return(false);
    }
    else
    {
        if(ptr->data == d)
        {
            return(true);
        }
        else if(ptr->data > d)
        {
            return(search(ptr->left, d));
        }
        else
        {
            return(search(ptr->right, d));
        }
    }
}

void deleteNode(Node<temp> *ptr, temp d)
{
    if(ptr==NULL)
    {
        cout<<"The entered element cannot be found."<<endl;
        return;
    }
    else
    {
        if(ptr->data > d)
        {
            deleteNode(ptr->left, d);
            return;
        }
        else if(ptr->data < d)
        {
            deleteNode(ptr->right, d);
            return;
        }
        else
        {
            Node<temp> *ptr1;
            if(ptr->right!=NULL)
            {
                ptr1 = ptr->right;
                while(ptr1->left!=NULL)
                {
                    ptr1 = ptr1->left;
                }
                ptr->data = ptr1->data;
                deleteNode(ptr->right, ptr->data);
                return;
            }
```

```
                else if(ptr->left!=NULL)
                {
                    ptr1 = ptr->left;
                    while(ptr1->right!=NULL)
                    {
                        ptr1 = ptr1->right;
                    }
                    ptr->data = ptr1->data;
                    deleteNode(ptr->left, ptr->data);
                    return;
                }
                else
                {
                    cout<<"Reached the place !!!!!!!!"<<endl;
                    ptr1 = ptr;
                    ptr = ptr->parent;
                    if(ptr->left == ptr1)
                    {
                        ptr->left = NULL;
                    }
                    else
                    {
                        ptr->right = NULL;
                    }
                    cout<<"The entered element has been deleted from the tree."<<endl;
                    free(ptr1);
                }
            }
        }
    }

    void printOptions()
    {
        cout<<"The avalible options are : "<<endl;
        cout<<"1) Insert a node into the tree."<<endl;
        cout<<"2) Delete a node from the tree."<<endl;
        cout<<"3) In-Order traversal of the tree."<<endl;
        cout<<"4) Pre-Order traversal of the tree."<<endl;
        cout<<"5) Post-Order traversal of the tree."<<endl;
        cout<<"6) Search for an element in the tree."<<endl;
        cout<<"7) Exit"<<endl;
    }
};

int main()
{

cout<<"-----------------------------------------------------------------------------------------------------"<<e
ndl;
    cout<<"\t\t\t\t\tWelcome\n";

cout<<"-----------------------------------------------------------------------------------------------------"<<e
ndl;

    BinarySearchTree<int> tree;
    int cho, num;
    do
    {
        tree.printOptions();
        cout<<"Enter your option : ";
```

```
        cin>>cho;
        switch(cho)
        {
            case 1 :
                tree.insertNode();
                break;
            case 2 :
                cout<<"Enter the element to be deleted : ";
                cin>>num;
                tree.deleteNode(tree.root, num);
                break;
            case 3 :
                tree.inorder(tree.root);
                cout<<endl;
                break;
            case 4 :
                tree.preorder(tree.root);
                cout<<endl;
                break;
            case 5 :
                tree.postorder(tree.root);
                cout<<endl;
                break;
            case 6 :
                cout<<"Enter the element to be searched : ";
                cin>>num;
                if(tree.search(tree.root, num))
                {
                    cout<<"The entered element is present in the tree."<<endl;
                }
                else
                {
                    cout<<"The entered element is not present in the tree."<<endl;
                }
                break;
            case 7 :
                cout<<"End of program."<<endl;
                break;
            default :
                cout<<"Invalid Option"<<endl;
        }

cout<<"-------------------------------------------------------------------------------------------------"<<e
ndl;
    } while (cho!=7);
}
```

```
Aadhityas-MacBook-Air:DSA-Course aadhitya$ g++ BST.cpp
Aadhityas-MacBook-Air:DSA-Course aadhitya$ ./a.out
--------------------------------------------------------------------------------------------------------
                                        Welcome
--------------------------------------------------------------------------------------------------------
The avalible options are :
1) Insert a node into the tree.
2) Delete a node from the tree.
3) In-Order traversal of the tree.
4) Pre-Order traversal of the tree.
5) Post-Order traversal of the tree.
6) Search for an element in the tree.
7) Exit
Enter your option : 1
Enter the data to be inserted into the tree : 5
--------------------------------------------------------------------------------------------------------
The avalible options are :
1) Insert a node into the tree.
2) Delete a node from the tree.
3) In-Order traversal of the tree.
4) Pre-Order traversal of the tree.
5) Post-Order traversal of the tree.
6) Search for an element in the tree.
7) Exit
Enter your option : 1
Enter the data to be inserted into the tree : 3
The entered element has been inserted into the tree.
--------------------------------------------------------------------------------------------------------
The avalible options are :
1) Insert a node into the tree.
2) Delete a node from the tree.
3) In-Order traversal of the tree.
4) Pre-Order traversal of the tree.
5) Post-Order traversal of the tree.
6) Search for an element in the tree.
7) Exit
Enter your option : 1
Enter the data to be inserted into the tree : 5
The element entered is already present in the tree.
--------------------------------------------------------------------------------------------------------
The avalible options are :
1) Insert a node into the tree.
2) Delete a node from the tree.
3) In-Order traversal of the tree.
4) Pre-Order traversal of the tree.
5) Post-Order traversal of the tree.
6) Search for an element in the tree.
7) Exit
Enter your option : 1
Enter the data to be inserted into the tree : 7
The entered element has been inserted into the tree.
--------------------------------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
The avalible options are :
1) Insert a node into the tree.
2) Delete a node from the tree.
3) In-Order traversal of the tree.
4) Pre-Order traversal of the tree.
5) Post-Order traversal of the tree.
6) Search for an element in the tree.
7) Exit
Enter your option : 1
Enter the data to be inserted into the tree : 11
The entered element has been inserted into the tree.
--------------------------------------------------------------------------------
The avalible options are :
1) Insert a node into the tree.
2) Delete a node from the tree.
3) In-Order traversal of the tree.
4) Pre-Order traversal of the tree.
5) Post-Order traversal of the tree.
6) Search for an element in the tree.
7) Exit
Enter your option : 3
3       5       7       11
--------------------------------------------------------------------------------
The avalible options are :
1) Insert a node into the tree.
2) Delete a node from the tree.
3) In-Order traversal of the tree.
4) Pre-Order traversal of the tree.
5) Post-Order traversal of the tree.
6) Search for an element in the tree.
7) Exit
Enter your option : 4
5       3       7       11
--------------------------------------------------------------------------------
The avalible options are :
1) Insert a node into the tree.
2) Delete a node from the tree.
3) In-Order traversal of the tree.
4) Pre-Order traversal of the tree.
5) Post-Order traversal of the tree.
6) Search for an element in the tree.
7) Exit
Enter your option : 5
3       11      7       5
--------------------------------------------------------------------------------
```

```
The avalible options are :
1) Insert a node into the tree.
2) Delete a node from the tree.
3) In-Order traversal of the tree.
4) Pre-Order traversal of the tree.
5) Post-Order traversal of the tree.
6) Search for an element in the tree.
7) Exit
Enter your option : 6
Enter the element to be searched : 12
The entered element is not present in the tree.
```

```
The avalible options are :
1) Insert a node into the tree.
2) Delete a node from the tree.
3) In-Order traversal of the tree.
4) Pre-Order traversal of the tree.
5) Post-Order traversal of the tree.
6) Search for an element in the tree.
7) Exit
Enter your option : 6
Enter the element to be searched : 11
The entered element is present in the tree.
```

```
The avalible options are :
1) Insert a node into the tree.
2) Delete a node from the tree.
3) In-Order traversal of the tree.
4) Pre-Order traversal of the tree.
5) Post-Order traversal of the tree.
6) Search for an element in the tree.
7) Exit
Enter your option : 2
Enter the element to be deleted : 12
The entered element cannot be found.
```

```
The avalible options are :
1) Insert a node into the tree.
2) Delete a node from the tree.
3) In-Order traversal of the tree.
4) Pre-Order traversal of the tree.
5) Post-Order traversal of the tree.
6) Search for an element in the tree.
7) Exit
Enter your option : 2
Enter the element to be deleted : 11
Reached the place !!!!!!!!
The entered element has been deleted from the tree.
```

```
The avalible options are :
1) Insert a node into the tree.
2) Delete a node from the tree.
3) In-Order traversal of the tree.
4) Pre-Order traversal of the tree.
5) Post-Order traversal of the tree.
6) Search for an element in the tree.
7) Exit
Enter your option : 3
3       5       7
```

```
The avalible options are :
1) Insert a node into the tree.
2) Delete a node from the tree.
3) In-Order traversal of the tree.
4) Pre-Order traversal of the tree.
5) Post-Order traversal of the tree.
6) Search for an element in the tree.
7) Exit
Enter your option : 7
End of program.
```

```
Aadhityas-MacBook-Air:DSA-Course aadhitya$
```

Q 2)

```
#include<iostream>
#include<stdlib.h>
using namespace std;

template <class temp>
class Node
{
    public :
        temp data;
        Node *left, *right;

        Node(temp d)
        {
            data = d;
            left = NULL;
            right = NULL;
        }
};

template <class temp>
class BinarySearchTree
{
    public :
        Node<temp> *root, *ptr;

        BinarySearchTree()
        {
            root = NULL;
        }

        void formTree()
        {
            root = new Node<temp>('-');

            root->left = new Node<temp>('-');
            root->left->left = new Node<temp>('+');
            root->left->left->left = new Node<temp>('x');
            root->left->left->right = new Node<temp>('y');
            root->left->right = new Node<temp>('*');
            root->left->right->left = new Node<temp>('2');
            root->left->right->right = new Node<temp>('x');

            root->right = new Node<temp>('+');
            root->right->left = new Node<temp>('*');
            root->right->right = new Node<temp>('/');
            root->right->left->left = new Node<temp>('x');
            root->right->left->right = new Node<temp>('z');
            root->right->right->left = new Node<temp>('y');
            root->right->right->right = new Node<temp>('z');
        }

        void preorder(Node<temp> *ptr)
        {
            if(ptr!=NULL)
            {
```

```
            cout<<ptr->data<<'\t';
            preorder(ptr->left);
            preorder(ptr->right);
        }
        return;
    }

    void inorder(Node<temp> *ptr)
    {
        if(ptr!=NULL)
        {
            inorder(ptr->left);
            cout<<ptr->data<<'\t';
            inorder(ptr->right);
        }
        return;
    }

    void postorder(Node<temp> *ptr)
    {
        if(ptr!=NULL)
        {
            postorder(ptr->left);
            postorder(ptr->right);
            cout<<ptr->data<<'\t';
        }
        return;
    }
};

int main()
{
    BinarySearchTree<char> tree;
    tree.formTree();

    cout<<"In-Order traversal : ";
    tree.inorder(tree.root);
    cout<<endl;

    cout<<"Pre-Order traversal : ";
    tree.preorder(tree.root);
    cout<<endl;

    cout<<"Post-Order traversal : ";
    tree.postorder(tree.root);
    cout<<endl;
}
```

```
Aadhityas-MacBook-Air:DSA-Course aadhitya$ g++ ParseTree.cpp
Aadhityas-MacBook-Air:DSA-Course aadhitya$ ./a.out
In-Order traversal : x  +       y       -       2       *       x       -       x       *       z       +       y       /       z
Pre-Order traversal : - -       +       x       y       *       2       x       +       *       x       z       /       y       z
Post-Order traversal : x        y       +       2       x       *       -       x       z       *       y       z       /       +       -
Aadhityas-MacBook-Air:DSA-Course aadhitya$
```

Q 3)

```
#include<iostream>
#include<stdlib.h>
using namespace std;

template <class temp>
class Node
{
    public :
        temp data;
        Node *parent, *left, *right;
};

template <class temp>
class BinarySearchTree
{
    public :
        Node<temp> *root, *ptr;

        BinarySearchTree()
        {
            root = NULL;
        }

        void insertNode(temp d)
        {
            if(root==NULL)
            {
                root = new Node<temp>();
                root->data = d;
                root->parent = NULL;
                root->left = NULL;
                root->right = NULL;
            }
            else
            {
                ptr = root;
                bool flag = true;

                Node<temp> *newnode = new Node<temp>();
                newnode->data = d;
                newnode->left = NULL;
                newnode->right = NULL;

                while(ptr!=NULL && flag==true)
                {
                    if(d == ptr->data)
                    {
                        cout<<"The element entered is already present in the tree."<<endl;
                        flag = false;
                        break;
                    }
                    else if(d > ptr->data)
                    {
                        if(ptr->right == NULL)
                        {
                            ptr->right = newnode;
```

```
                newnode->parent = ptr;
                flag = false;
            }
            else
            {
                ptr = ptr->right;
            }
        }
        else
        {
            if(ptr->left == NULL)
            {
                ptr->left = newnode;
                newnode->parent = ptr;
                flag = false;
            }
            else
            {
                ptr = ptr->left;
            }
        }

    }
  }
}

void printPaths(Node<temp> *root, temp *arr, int itr)
{
  if(root->left!=NULL)
  {
    arr[itr] = root->data;
    printPaths(root->left, arr, itr+1);
    arr[itr] = 0;
  }
  if (root->right!=NULL)
  {
    arr[itr] = root->data;
    printPaths(root->right, arr, itr+1);
    arr[itr] = 0;
  }
  if(root->right==NULL && root->left==NULL)
  {
    int i = 0;
    cout<<"Path : ";
    arr[itr] = root->data;
    for(i=0;i<=itr;i++)
    {
      cout<<arr[i]<<'\t';
    }
    cout<<endl;
    arr[itr] = 0;
  }
  return;
}

bool pathFinder(Node<temp> *root, temp k, temp sum)
{
  bool flag;
  if(root->left!=NULL)
  {
```

```
            flag = pathFinder(root->left, k, sum+root->data);
            if(flag)
            {
                return(flag);
            }
        }
        if (root->right!=NULL)
        {
            flag = pathFinder(root->right, k, sum+root->data);
            if(flag)
            {
                return(flag);
            }
        }
        if(root->right==NULL && root->left==NULL)
        {
            if((sum+root->data)==k)
            {
                flag = true;
            }
            else
            {
                flag = false;
            }
        }
        return(flag);
    }
};

int main()
{
    BinarySearchTree<int> tree;
    int num, k, i, n;
    cout<<"Enter the number of elements to be entered : ";
    cin>>n;
    cout<<"Enter the value of k : ";
    cin>>k;
    cout<<"Enter the elements : ";
    for(i=0;i<n;i++)
    {
        cin>>num;
        tree.insertNode(num);
    }
    int *arr = new int[50];
    cout<<"The various paths are : "<<endl;
    tree.printPaths(tree.root, arr, 0);
    bool flag = tree.pathFinder(tree.root, k, 0);
    if(flag)
    {
        cout<<"There exists a root to leaf path with the specfied conditions."<<endl;
    }
    else
    {
        cout<<"There does not exist a root to leaf path with the specfied conditions."<<endl;
    }
}
```

```
Aadhityas-MacBook-Air:DSA-Course aadhitya$ g++ RootLeafPath.cpp
Aadhityas-MacBook-Air:DSA-Course aadhitya$ ./a.out
Enter the number of elements to be entered : 5
Enter the value of k : 3
Enter the elements : 2
1
4
3
5
The various paths are :
Path : 2        1
Path : 2        4       3
Path : 2        4       5
There exists a root to leaf path with the specfied conditions.
Aadhityas-MacBook-Air:DSA-Course aadhitya$
```

Q 4)

```
#include<iostream>
#include<stdlib.h>
using namespace std;

template <class temp>
class Node
{
   public :
      temp data;
      Node *parent, *left, *right;
};

template <class temp>
class BinarySearchTree
{
   public :
      Node<temp> *root, *ptr;

      BinarySearchTree()
      {
         root = NULL;
      }

      void insertNode(temp d)
      {
         if(root==NULL)
         {
            root = new Node<temp>();
            root->data = d;
            root->parent = NULL;
            root->left = NULL;
            root->right = NULL;
         }
         else
         {
            ptr = root;
```

```
        bool flag = true;

        Node<temp> *newnode = new Node<temp>();
        newnode->data = d;
        newnode->left = NULL;
        newnode->right = NULL;

        while(ptr!=NULL && flag==true)
        {
            if(d > ptr->data)
            {
                if(ptr->right == NULL)
                {
                    ptr->right = newnode;
                    newnode->parent = ptr;
                    flag = false;
                }
                else
                {
                    ptr = ptr->right;
                }
            }
            else
            {
                if(ptr->left == NULL)
                {
                    ptr->left = newnode;
                    newnode->parent = ptr;
                    flag = false;
                }
                else
                {
                    ptr = ptr->left;
                }
            }

        }
    }
}

void inorder(Node<temp> *ptr)
{
    if(ptr!=NULL)
    {
        inorder(ptr->left);
        cout<<ptr->data<<'\t';
        inorder(ptr->right);
    }
    return;
}

void mirrorTree(Node<temp> *node)
{
    if(node->left==NULL && node->right==NULL)
    {
        return;
    }
    else
    {
        ptr = node->left;
```

```
            node->left = node->right;
            node->right = ptr;
            if(node->left!=NULL)
            {
                mirrorTree(node->left);
            }
            if(node->right!=NULL)
            {
                mirrorTree(node->right);
            }
        }
    }
};

int main()
{
    BinarySearchTree<int> tree;
    int num, i, n;
    cout<<"Enter the number of elements to be entered : ";
    cin>>n;
    cout<<"Enter the elements : ";
    for(i=0;i<n;i++)
    {
        cin>>num;
        tree.insertNode(num);
    }
    cout<<"The in-order traversal of the entered tree is as follows : ";
    tree.inorder(tree.root);
    cout<<endl;
    tree.mirrorTree(tree.root);
    cout<<"The in-order traversal of the mirror tree is as follows : ";
    tree.inorder(tree.root);
    cout<<endl;
}
```

```
Aadhityas-MacBook-Air:DSA-Course aadhitya$ g++ mirrorBST.cpp
Aadhityas-MacBook-Air:DSA-Course aadhitya$ ./a.out
Enter the number of elements to be entered : 5
Enter the elements : 3
1
2
5
4
The in-order traversal of the entered tree is as follows : 1    2       3       4       5
The in-order traversal of the mirror tree is as follows : 5     4       3       2       1
Aadhityas-MacBook-Air:DSA-Course aadhitya$
```