Tree

1. Binary tree

```
#include<iostream>
#include<string>
using namespace std;
namespace collection
{
    class Exception
    {
    private:
        string message;
    public:
        Exception( string messgae ) throw( ) : message( messgae )
        {   }
        string getMessage( void )const throw( )
        {
            return this->message;
        }
    };
    class BSTree;   //Forward declaration
    class Node
    {
    private:
        Node *left;
        int data;
        Node *right;
    public:
        Node( int data  = 0 ) : left( NULL ) ,data( data ), right( NULL )
        {   }
        friend class BSTree;
    };
    class BSTree
    {
    private:
        Node *root;
    public:
        BSTree( void ) throw( ) : root( NULL )
        {   }
        bool empty( void )const throw( )
        {
            return this->root == NULL;
        }
        void addNode( int data )
        {
            Node *newNode = new Node( data );
            if( this->empty( ) )
                this->root = newNode;
            else
```

```
            {
                Node *trav = this->root;
                while( true )
                {
                    if( data < trav->data )
                    {
                        if( trav->left == NULL )
                        {
                            trav->left = newNode;
                            break;
                        }
                        trav = trav->left;
                    }
                    else
                    {
                        if( trav->right == NULL )
                        {
                            trav->right = newNode;
                            break;
                        }
                        trav = trav->right;
                    }
                }
            }
        }
        void preOrder( void)
        {
            this->preOrder( this->root );
        }
        void inOrder( void)
        {
            this->inOrder( this->root );
        }
        void postOrder( void )
        {
            this->postOrder(this->root );
        }
        ~BSTree( void )
        {
            this->clear( this->root );
            this->root = NULL;
        }
    private:
        void preOrder( Node *trav )
        {
            if( trav == NULL )
                return;
            cout<<trav->data<<" ";
            this->preOrder( trav->left );
            this->preOrder( trav->right );
        }
        void inOrder( Node *trav )
        {
            if( trav == NULL )
```

```
                return;
            this->inOrder(trav->left);
            cout<<trav->data<<" ";
            this->inOrder(trav->right);
        }
        void postOrder( Node *trav )
        {
            if( trav == NULL )
                return;
            this->postOrder( trav->left );
            this->postOrder( trav->right );
            cout<<trav->data<<" ";
        }
        void clear( Node *trav )
        {
            if( trav == NULL )
                return;
            this->clear( trav->left );
            this->clear( trav->right );
            delete trav;
        }
    };
}
int main( void )
{
    using namespace collection;
    BSTree tree;
    tree.addNode( 70 );
    tree.addNode( 50 );
    tree.addNode( 40 );
    tree.addNode( 60 );
    tree.addNode( 55 );
    tree.addNode( 90 );
    tree.addNode( 80 );
    tree.addNode( 100 );

    //tree.preOrder( );
    //tree.inOrder( );
    tree.postOrder( );
    return 0;
}
```

2. Deletion of Node In Binery Search Tree

```
#include<iostream>
#include<string>
using namespace std;
namespace collection
{

class Exception
```

```
{
  private:
   string message;
  public :
  Exception(string message) throw() : message(message)
  { }

  string getMessage(void) const throw()
  {
      return this->message;
  }

};

class BSTree;
class Node
{
  private:
  Node *left;
  int data;
  Node *right;
  public :
   Node (int data = 0): data(data), left(NULL),right(NULL)
   { }

  friend class BSTree;
};

class BSTree
{
    private:
    Node* root;
    public:

    BSTree(void) throw() : root(NULL)
    { }

    bool empty(void) const throw()
    {
        return this->root == NULL;

    }

    void addNode(int data)
    {
        Node *newNode = new Node(data);

        if(this->empty())
        this->root = newNode;
        else
        {
            Node *trav = this->root;
            while(true)
            {
```

```
                if(data < trav->data )
                {
                    if(trav->left == NULL )
                    {
                     trav->left = newNode;
                      break;
                    }
                   trav = trav->left;
                } else
                {
                    if(trav->right == NULL)
                    {
                        trav->right = newNode;
                        break;
                    }
                    trav = trav->right;
                }



            }

        }
    }

    void preOrder(void)
    {
        this->preOrder(this->root);
    }
    void InOrder(void)
    {
        this->InOrder(this->root);
    }
     void postOrder(void)
    {
        this->postOrder(this->root);
    }

    Node* searchNode(int data,Node *&parent)
    {
        Node * trav = this->root;
        parent = NULL;
        while(trav != NULL)
        {
            if(data == trav->data)
            return trav;
           parent = trav;
            if(data < trav->data)
            trav = trav->left;
            else
            {
                trav->right;
            }
```

```
        }

        parent = NULL;
        return NULL;
    }

    void deleteNode(int data) throw(Exception)
    {
        Node *parent;
       Node* target =  this->searchNode(data, parent) ;

      if(target->left != NULL && target->right != NULL  )
      {
          parent = target;
          Node * successor = target->right;

          while(successor->left != NULL)
          {
              parent = successor;
              successor = successor->left;

          }
          target->data = successor->data;
          target = successor;



      }



        if(target != NULL)
        {
            if(target->left == NULL)
            {
                if(target == this->root)
                {
                    this->root = target->right;

                } else if(target == parent->left)
                {
                    parent->left = target->right;


                } else
                {
                    parent->right = target->right;
                }

            } else
            {
                if(target == this->root)
                {
                   this->root = target->left;
```

```
            } else if(target == parent->left)
            {
              parent->left = target->left;


            } else
            {
                parent->right = target->left;
            }

        }


    }
  delete target;
}
private:
void preOrder(Node *trav)
{
      if(trav  == NULL )
      return;
    cout<<trav->data <<" ";
    this->preOrder(trav->left);
    this->preOrder(trav->right);
}

void InOrder(Node *trav)
{
      if(trav  == NULL )
      return;

    this->InOrder(trav->left);
     cout<<trav->data <<" ";
    this->InOrder(trav->right);
}
 void postOrder(Node *trav)
{
      if(trav  == NULL )
      return;

    this->postOrder(trav->left);
    this->postOrder(trav->right);
       cout<<trav->data <<" ";
}

public:

 void clear(Node *trav)
{
      if(trav  == NULL )
      return;

    this->postOrder(trav->left);
```

```
        this->postOrder(trav->right);
           //cout<<trav->data <<" ";
           delete trav;
    }

    ~BSTree(void)
    {
        //this->clear(this->root);
        //this->root = NULL;
    }

};



}

int main(void)
{

 using namespace collection;
 BSTree tree;
  tree.addNode(150);
    tree.addNode(80);
    tree.addNode(180);
    tree.addNode(50);
    tree.addNode(135);
    tree.addNode(175);
    tree.addNode(200);
    tree.addNode(30 );
     tree.addNode(120 );
      tree.addNode(145 );
 tree.addNode(100 );
  tree.addNode(125 );
   tree.addNode(115 );

cout<< " in order"<<endl;
    tree.InOrder( );  // 30 50 80 100 115 120 125 135 145 150 175 180 200
     cout<<endl;

    tree.deleteNode(80);
cout<< " in order : after deletion of 80 "<<endl;
    tree.InOrder( );
     cout<<endl;


//  cout<<" Pre order" <<endl;
//     tree.preOrder( );  // 150 80 50 30 135 120 100 115 125 145 180 175
200
//      cout<<endl;

//     cout << "post order"<<endl;
//     tree.postOrder(); // 30 50 115 100 125 120 145 135 80 175 200 180
150
```

```
//      cout<<endl;
    // predecessor and successor changes based on traversel , can be seen
for 80
    return 0;
}
```