```
#include <iostream>
#include <cstdlib>
#include <string>
#include <sstream>
#include <stdio.h>
#include <stdlib.h>
#include "bst.h"
using namespace std;



//========================================================================
// Insert()
// Use Recursion to Properly Insert an Item into the BST
//========================================================================

template <class KeyType>
void BinarySearchTree<KeyType>::insert(KeyType *x)
{
        if (root)
        {
                return insert_helper(root, x);
        }


        root = new node<KeyType>(x);
}


template <class KeyType>
void BinarySearchTree<KeyType>::insert_helper(*root, KeyType* key)
{
        //create new node
        if (root->key >= key)
        {

                if (root->left)
                        return insert_helper(root->left,key);

                else
                        root->left = new node<KeyType>(key);

        }

        else
        {
                if(root->right)
                {
                        return insert_helper(root->right, key);
                }

                else
                {
                        root->right = new node<KeyType>(key);
                }
        }
}

//========================================================================
// Get()
// Use Recursion to Properly Return First Element with Key Equal to k
//========================================================================

template <class KeyType>
KeyType *BinarySearchTree<KeyType>::get(const KeyType& x)
{
        return search_helper(root,key);
}
```

```
template <class KeyType>
KeyType *BinarySearchTree<KeyType>::get_helper(node* root, const KeyType& key)
{
        if(root->key == key)
                return root;

        else
        {
                if(root->key > key)
                {
                        root = root->right;

                        if(root == NULL)
                        {
                                cout << "not in tree" << endl;
                                return -1;
                        }

                        else
                                return get_helper(root,key);
                }

                else
                {
                        root = root->left;
                        if(root == NULL)
                        {
                                cout << "not in tree" << endl;
                                return -1;
                        }

                        else
                                return get_helper(root,key);

                }
        }
}



//====================================================================
// Minimum()
// Use Recursion to Properly Return the Minimum Element in the BST
//====================================================================

template <class KeyType>
KeyType* BinarySearchTree<KeyType>::minimum()
{
        return minimum_helper(root);
}


template <class KeyType>
KeyType *BinarySearchTree<KeyType>::minimum_helper(node* root)
{
        if(root == NULL)
        {
                cout << "empty tree" << endl;
                return NULL;
        }

        else if(root->left)
                return minimum_helper(root->left);

        else
                return root->key;
}

//====================================================================
```

```
// Maximum()
// Use Recursion to Properly Return the Maximum Element in the BST
//=====================================================================

template <class KeyType>
KeyType* BinarySearchTree<KeyType>::maximum()
{
        return maximum_helper(root);
}


template <class KeyType>
KeyType *BinarySearchTree<KeyType>::maximum_helper(node* root)
{
        if(root == NULL)
        {
                cout << "empty tree" << endl;
                return NULL;
        }

        else if(root->right)
                return maximum_helper(root.right);
        else
                return root->key;
}


//=====================================================================
// inOrder()
// Use Recursion to Properly Return String of Elements From an Inorder Traversal
//=====================================================================


template <class KeyType>
void BinarySearchTree<KeyType>::inOrder()
{
        return inOrder_helper(root);
}


template <class KeyType>
void BinarySearchTree<KeyType>::inOrder_helper(node* root)
{
        if( root == NULL)
        {
                cout << "empty tree" << endl;
                return -1;
        }

        inOrder_helper(root->left);
        cout << root << key;
        inOrder_helper(root->right);
}




//=====================================================================
// preOrder()
// Use Recursion to Properly Return String of Elements From an Preorder Traversal
//=====================================================================



template <class KeyType>
void BinarySearchTree<KeyType>::preOrder()
{
        return preOrder_helper(root);
}
```

```
template <class KeyType>
void BinarySearchTree<KeyType>::preOrder_helper(node* root)
{
        if( root == NULL)
        {
                cout << "empty tree" << endl;
                return -1;
        }

        cout << root << key;
        preOrder_helper(root->left);
        preOrder_helper(root->right);
}


//=====================================================================
// inOrder()
// Use Recursion to Properly Return String of Elements From an Postorder Traversal
//=====================================================================


template <class KeyType>
void BinarySearchTree<KeyType>::postOrder()
{
        return postOrder_helper(root);
}


template <class KeyType>
void BinarySearchTree<KeyType>::postOrder_helper(node* root)
{
        if( root == NULL)
        {
                cout << "empty tree" << endl;
                return -1;
        }

        postOrder_helper(root->left);
        postOrder_helper(root->right);
        cout << root << key;
}


//=====================================================================
// Predecessor()
// Use Recursion to Properly Return the Predecessor of x
//=====================================================================

template <class KeyType>
KeyType* BinarySearchTree<KeyType>::predecessor(const KeyType& x)
{
        predecessor_helper(x, root);
}


template <class KeyType>
KeyType* BinarySearchTree<KeyType>::predecessor_helper(const KeyType& x, node* root)
{
        if (x->left)
                return maximum_helper(x->left);

        else
        {
                struct node *p = x->parent;
                while( p != NULL && x == p->left)
                {
                        x = p;
                        p = p->parent;
                }
```

```
                return p;

        }
}

//=====================================================================
// Successor()
// Use Recursion to Properly Return the Successor of x
//=====================================================================


template <class KeyType>
KeyType* BinarySearchTree<KeyType>::successor(const KeyType& x)
{
        successor_helper(x, root);
}


template <class KeyType>
KeyType* BinarySearchTree<KeyType>::successor_helper(const KeyType& x, node* root)
{
        if (x->right)           //x has a right child
                return minimum_helper(x->right);

        else                                    // successor is above x in tree
        {
                struct node *p = x->parent;
                while( p != NULL && x == p->right)
                {
                        x = p;
                        p = p->parent;
                }

                return p;
        }
}
```