```
// Marius Rejdak
// Informatyka, mgr, OS1
//
// BST.h

#ifndef BST_H_INCLUDED
#define BST_H_INCLUDED

#include <memory>
#include <functional>

template <class _Key, class _Compare = std::less<_Key> >
class BST {
public:
  typedef _Key key_type;
  typedef _Compare key_compare;

  BST(const key_compare& comp = key_compare());
  bool insert(key_type value);
  bool find(key_type value);
  bool erase(key_type value);
  key_type& min(void);
  key_type& max(void);
private:
  const key_compare& comp;
  class Node {
  public:
    Node(key_type x);
    key_type& min();
    key_type& max();
    key_type value;
    std::shared_ptr<Node> parent;
    std::shared_ptr<Node> left;
    std::shared_ptr<Node> right;
  };
  typedef std::shared_ptr<Node> shared_ptr;
  shared_ptr root;

  inline bool compe(key_type v1, key_type v2) { return !(comp(v1,v2) ||
comp(v2,v1)); }
  shared_ptr next(shared_ptr x);
  shared_ptr getNode(key_type value);
};

#endif // BST_H_INCLUDED
```

```
// Marius Rejdak
// Informatyka, mgr, OS1
//
// BST.cpp

#include "bst.h"

template <class _Key, class _Compare>
BST<_Key,_Compare>::BST(const _Compare& comp) : comp(comp),
root(nullptr)
{
    //empty
}

template <class _Key, class _Compare>
bool BST<_Key,_Compare>::insert(_Key value)
{
    if (find(value))
        return false;

    shared_ptr e = shared_ptr(new Node(value)), y = nullptr, x =
root;

    while (x != nullptr)
    {
        y = x;
        if (comp(e->value, x->value))
            x = x->left;
        else
            x = x->right;
    }
    e->parent = y;

    if (y == nullptr)
    {
        root = e;
    }
    else
    {
        if (comp(e->value, y->value))
            y->left = e;
        else
            y->right = e;
    }
    return true;
}

template <class _Key, class _Compare>
bool BST<_Key,_Compare>::find(_Key value)
```

```
{
    shared_ptr x = root;
    while ((x != nullptr) && !compe(value, x->value))
    {
        if (comp(value, x->value))
            x = x->left;
        else
            x = x->right;
    }
    return !((x == nullptr) || !compe(x->value, value));
}

template <class _Key, class _Compare>
bool BST<_Key,_Compare>::erase(_Key value)
{
    shared_ptr x = nullptr;
    x = getNode(value);
    if ((x != nullptr) && compe(x->value, value))
    {
        shared_ptr y = nullptr;
        if ((x->left == nullptr) || (x->right == nullptr))
        {
            y = x;
        }
        else
        {
            y = next(x);
        }

        if (y->left != nullptr)
        {
            x = y->left;
        }
        else
        {
            x = y->right;
        }

        if (x != nullptr)
        {
            x->parent = y->parent;
        }

        if (y->parent == nullptr)
        {
            root = x;
        }
        else
        {
            if (y == y->parent->left)
            {
                y->parent->left = x;
            }
            else
            {
                y->parent->right = x;
            }
        }

        return true;
    }
    else
    {
        return false;
    }
}

template <class _Key, class _Compare>
typename BST<_Key,_Compare>::key_type& BST<_Key,_Compare>::min(void)
{
    return root->min();
}

template <class _Key, class _Compare>
typename BST<_Key,_Compare>::key_type& BST<_Key,_Compare>::max(void)
{
    return root->max();
}

template <class _Key, class _Compare>
typename BST<_Key,_Compare>::shared_ptr
BST<_Key,_Compare>::next(shared_ptr x)
{
    if (x->right != nullptr)
    {
       return getNode(x->right->min());
    }
    shared_ptr y = x->parent;
    while ((y != nullptr) && (x == y->right))
    {
        x = y;
        y = y->parent;
   }
   return y;
}

template <class _Key, class _Compare>
typename BST<_Key,_Compare>::shared_ptr
```

```
BST<_Key,_Compare>::getNode(key_type value)
{
    shared_ptr x = root;
    while ((x != nullptr) && !compe(value, x->value))
    {
        if (comp(value, x->value))
        {
            x = x->left;
        }
        else
        {
            x = x->right;
        }
    }
   return x;
}

template <class _Key, class _Compare>
BST<_Key,_Compare>::Node::Node(key_type x) : value(x)
{
    //empty
}

template <class _Key, class _Compare>
typename BST<_Key,_Compare>::key_type&
BST<_Key,_Compare>::Node::min()
{
    if (left != nullptr)
        return left->min();
    else
        return value;
}

template <class _Key, class _Compare>
typename BST<_Key,_Compare>::key_type&BST<_Key,_Compare>::Node::max()
{
    if (right != nullptr)
        return right->max();
    else
        return value;
}
```