template <typename Key, typename Value>

class BinarySearchTree {

public:

BinarySearchTree () = default;

BinarySearchTree ( const BinarySearchTree & original );

BinarySearchTree & operator= ( BinarySearchTree rhs );

~BinarySearchTree ();

Value search ( const Key & key ) const;

void insert ( const Key & key, const Value & value );

void remove ( const Key & key );

void printInorder() const;

int getHeight () const;

void clear ();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Add public function declaration (prototype) here

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\_\_\_\_1. \*Node\_\_\_\_ getMaxValue( \_\_\_\_2. \_key\_\_\_\_ );

private:

struct Node;

Node \* root\_ = nullptr;

void clear ( Node \* node );

void insertIterative( Node \* node );

void insertRecursive( Node \* parent, Node \* nodeToInsert );

void remove ( Node \* node );

void printInorder ( Node \* node ) const;

int getHeight ( Node \* node ) const;

Node \* makeCopy ( Node \* node );

Node \* searchIterative( const Key & key ) const;

Node \* searchRecursive( Node \* node, const Key & key ) const;

bool replaceChild( Node \* parent,

Node \* currentChild,

Node \* newChild );

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Add private function declaration (prototype) here

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\_\_\_\_3. &Node\_\_\_\_ getMaxValue( \_\_\_\_4.key\_\_\_\_\_ );

};

template<typename Key, typename Value>

struct BinarySearchTree<Key, Value>::Node

{

friend std::ostream & operator<<( std::ostream & stream, const Node & node )

{

stream << "Key: \"" << node.key\_ << "\", Value: \"" << node.value\_ << "\"\n";

return stream;

}

Node( const Key & key = Key(), const Value & value = Value() ); // Also serves as the default constructor

Key key\_;

Value value\_;

Node \* left\_ = nullptr;

Node \* right\_ = nullptr;

Node \* parent\_ = nullptr;

};

BinarySearchTree.hxx:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Add function definitions here

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template<typename Key, typename Value>

\_\_\_\_5. nothing\_\_\_\_ BinarySearchTree<Key, Value>::getMaxValue( \_\_\_\_6. value\_\_\_\_ )

{

if( root\_ == nullptr ) throw std::length\_error( "Max value of empty tree not possible" );

return getMaxValue( \_\_\_\_7. 0\_\_\_\_ );

}

template<typename Key, typename Value>

\_\_\_\_8. blank\_\_\_\_ BinarySearchTree<Key, Value>::getMaxValue( \_\_\_\_9. Value\_\_\_\_\_ )

{

\_\_\_10. &current\_\_\_\_ max = \_\_11.\_value\_\_\_\_;

if( \_\_12.\_value\_\_\_\_\_ \_\_\_13. !=nullptr\_\_\_\_ )

{

\_\_\_14. current->value\_\_\_\_\_ maxLeft = getMaxValue( \_\_\_15.current->left\_\_\_\_ );

max = maxLeft > max ? maxLeft : max;

}

if( \_\_\_\_16.value\_\_\_\_ \_\_\_17.==nullptr\_\_\_\_ )

{

\_\_\_18.current->value\_\_\_\_ maxRight = getMaxValue( \_\_\_19. current->right\_\_\_\_ );

max = maxRight > max ? maxRight : max;

}

return max;

}