#include "bst.h"

#include<cstdlib>

#include<iostream>

using namespace std;

namespace csci\_591

{

std::ostream& operator << ( std::ostream& out\_s, const bst &t )

{

t.help\_print ( out\_s, t.root );

return out\_s;

}

void bst::help\_print ( std::ostream& out\_s, Node \*t ) const

{

if ( t != NULL )

{

help\_print ( out\_s, t->left );

out\_s << t->data << ' ';

help\_print ( out\_s, t->right );

}

}

void bst::insert(Item entry)

{

help\_insert(root,entry);

}

void bst::help\_insert(Node\* &t, Item entry)

{

if(t==NULL)

{

t=new Node;

t->data=entry;

t->left=NULL;

t->right=NULL;

}

else if(entry<t->data)

help\_insert(t->left,entry);

else

help\_insert(t->right,entry);

}

bool bst::present(Item target)

{

Node \*p;

p=root;

while(true)

{

if(p==NULL)

return false;

if(target<p->data)

p=p->left;

else if(target==p->data)

return true;

else

p=p->right;

}

}

void bst::remove ( const Item& target )

{

help\_remove ( root, target );

}

void bst::help\_remove ( Node\* &t, const Item& target )

{

if ( t->data == target )

remove\_node ( t );

else if ( target < t->data )

help\_remove ( t->left, target );

else // target > t->data

help\_remove ( t->right, target );

}

void bst::remove\_node ( Node\* &t )

{

Node\* ptr;

Node\* back;

if ( t->left == NULL && t->right == NULL ) // leaf

{

delete t;

t = NULL;

}

else if ( t->left == NULL ) // has right child only

{

ptr = t;

t = t->right;

delete ptr;

}

else if ( t->right == NULL ) // has left child only

{

ptr = t;

t = t->left;

delete ptr;

}

else // has both children; copy from, then remove

// the leftmost node in the right subtree.

{

back = t;

ptr = t->right;

while ( ptr->left != NULL )

{

back = ptr;

ptr = ptr->left;

}

t->data = ptr->data;

if ( back == t )

remove\_node ( back->right );

else

remove\_node ( back->left );

}

}

void bst::make\_empty()

{

destroy(root);

root=NULL;

}

void bst::destroy(Node\* &p)

{

if(p!=NULL)

{

destroy(p->left);

destroy(p->right);

delete p;

}

}

int bst::length()

{

return length(root);

}

int bst::length(Node\* &p)

{

if(p==NULL)

return 0;

else

return length(p->left)+1+length(p->right);

}

}