```
#include <iostream>
#include <cstdlib>
#include <sys/time.h>

#include "pairs.h"
#include "tree.h"

int        main();

int main()
{
        int             num;
        tree_t<int>     tree;
        tree_t<pairs_t> ptree;

        struct timeval  tp;

  // get time of day
  gettimeofday(&tp,NULL);

  // use microseconds as the seed
//srand((unsigned int)tp.tv_usec);
  srand((unsigned int)1337);

  // sort while inserting, thus maintaining list as priority queue
  std::cout << "Inserting:" << std::endl;
  for(int i=0;i<10;i++) {
    num = static_cast<int>((float)rand()/(float)RAND_MAX*100.0);
    std::cout << num << " ";
    tree.insert(num);
  }
  std::cout << std::endl;

  tree.insert(57);

  std::cout << "Tree min: " << tree.min() << std::endl;
  std::cout << "Tree max: " << tree.max() << std::endl;

  std::cout << "Tree contains 57? " << tree.contains(57) << std::endl;

  std::cout << "Tree: " << tree << std::endl;

  ptree.insert(pairs_t(45.0,'b'));
  ptree.insert(pairs_t(37.2,'c'));
  ptree.insert(pairs_t(42.6,'a'));
  ptree.insert(pairs_t(53.7,'d'));

  std::cout << "Pair Tree: " << ptree << std::endl;
}
```

```
#include <iostream>

#include "pairs.h"

std::ostream& operator<<(std::ostream& s, const pairs_t& rhs)
{
  s << "(’" << rhs.c << "’," << rhs.x << ")";

  return s;
}

pairs_t::pairs_t(const pairs_t& rhs)
{
  // this constructor creates the pairs_t object given a constant
  // reference to another pairs_t object (result is a copy of
  // the other object)
  c = rhs.c;
  x = rhs.x;
}

const pairs_t& pairs_t::operator=(const pairs_t& rhs)
{
  if(this == &rhs)      // standard alias test
    return *this;

  c = rhs.c;
  x = rhs.x;

  return *this;
}

bool pairs_t::operator<(const pairs_t& rhs) const
{
  // a pairs_t is smaller than another pairs_t iff its character, c, is smaller
  return c < rhs.c;
}

bool pairs_t::operator>(const pairs_t& rhs) const
{
  // a pairs_t is larger than another pairs_t iff its character, c, is larger
  return c > rhs.c;
}
```

```
#include <iostream>

#include "pairs.h"
#include "tree.h"

template <typename T>
std::ostream& operator<<(std::ostream& s, const tree_t<T>& rhs)
{
  if(rhs.empty()) s << "empty" << std::endl;
  else rhs.inorder(s,rhs.root);

  return s;
}

template <typename T>
void tree_t<T>::inorder(std::ostream& s, node_t* const &t) const
{
  // fill this in
}

template <typename T>
tree_t<T>::tree_t()
{
  // this constructor creates an emply tree_t object
  root = NULL;
}

template <typename T>
const tree_t<T>& tree_t<T>::operator=(const tree_t<T>& rhs)
{
  if(this != &rhs) {       // standard alias test
    clear();
    root = clone(rhs.root);
  }

  return *this;
}

template <typename T>
typename tree_t<T>::node_t* tree_t<T>::clone(node_t* t) const
{
  if(t == NULL) return NULL;

  return new node_t(t->data,clone(t->left),clone(t->right));
}

template <typename T>
void tree_t<T>::clear(node_t* &t)
{
  // fill this in
}

template <typename T>
typename tree_t<T>::node_t* tree_t<T>::min(node_t* t) const
{
  return t->left == NULL ? t : min(t->left);
}

template <typename T>
typename tree_t<T>::node_t* tree_t<T>::max(node_t* t) const
{
  return t->right == NULL ? t : max(t->right);
```

```
}

template <typename T>
bool tree_t<T>::contains(const T& x, node_t* t) const
{
  if(t == NULL) return false;
  else if(x < t->data) return contains(x,t->left);
  else if(x > t->data) return contains(x,t->right);
  else return true;
}

template <typename T>
void tree_t<T>::insert( const T& x, node_t* & t)
{
  // fill this in
}

template <typename T>
void tree_t<T>::erase(const T& x, node_t* & t)
{
  // fill this in
}

////////////////////////// specializations ////////////////////////////////
template class tree_t<int>;
template std::ostream& operator<<(std::ostream&, const tree_t<int>&);

template class tree_t<float>;
template std::ostream& operator<<(std::ostream&, const tree_t<float>&);

template class tree_t<double>;
template std::ostream& operator<<(std::ostream&, const tree_t<double>&);

template class tree_t<pairs_t>;
template std::ostream& operator<<(std::ostream&, const tree_t<pairs_t>&);
```