```
//−*−Mode: C++;−*−

/*
  $Id: node.template,v 1.2 2006/05/18 10:22:10 jmbs Exp $
*/
/*
  $Log: node.template,v $
  Revision 1.2  2006/05/18 10:22:10  jmbs
  Corregida funciï¿½ bintree<T>::copy

  Revision 1.1.1.1  2006/05/17 12:25:32  jmbs
  Versiï¿½ Inicial

*/


#include <cassert>

//
// Operaciones de nodewrapper
//
template <typename T>
inline
bintree<T>::nodewrapper::nodewrapper()
//  : pad(0), izda(0), dcha(0)
{
}


template <typename T>
inline
bintree<T>::nodewrapper::nodewrapper(const T & e)
  : etiqueta(e)//, pad(0), izda(0), dcha(0)
{
}


//
// Operaciones de node
//

template <typename T>
inline
bintree<T>::node::node()
{
  elnodo = 0;
};


template <typename T>
inline
bintree<T>::node::node(const T & e)
{
  elnodo = new nodewrapper(e);
}


template <typename T>
inline
bintree<T>::node::node(const typename bintree<T>::node & n)
  : elnodo(n.elnodo)
{
```



```
}

/*
template <typename T>
inline
bintree<T>::node::node(typename bintree<T>::node *n)
{
  if (n != 0)
    elnodo = n->elnodo;
  else
    elnodo = 0;
};
*/

template <typename T>
inline
typename bintree<T>::node &
bintree<T>::node::operator=(const typename bintree<T>::node & n)
{
  if (&n != this)
    if (n.null())
      elnodo = 0;
    else
      elnodo = n.elnodo;

  return *this;
};

template <typename T>
inline
bool bintree<T>::node::null() const
{
  return elnodo == 0;
};

template <typename T>
inline
void bintree<T>::node::parent(typename bintree<T>::node n)
{
  elnodo->pad = n;
};

template <typename T>
inline
void bintree<T>::node::left(typename bintree<T>::node n)
{
  elnodo->izda = n;
};

template <typename T>
inline
void bintree<T>::node::right(typename bintree<T>::node n)
{
  elnodo->dcha = n;
};

template <typename T>
inline
typename bintree<T>::node bintree<T>::node::parent() const
{
  return (elnodo->pad);
};
```

```
template <typename T>
inline
typename bintree<T>::node bintree<T>::node::left() const
{
  return (elnodo->izda);
};

template <typename T>
inline
typename bintree<T>::node bintree<T>::node::right() const
{
  return (elnodo->dcha);
};

template <typename T>
inline
T & bintree<T>::node::operator*()
{
  return elnodo->etiqueta;
};

template <typename T>
inline
const T & bintree<T>::node::operator*() const
{
  return elnodo->etiqueta;
};

template <typename T>
void bintree<T>::node::remove()
{
  delete elnodo;
  elnodo = 0;
};

template <typename T>
inline
bool bintree<T>::node::operator==(const node & n) const
{
  return elnodo == n.elnodo;
};

template <typename T>
inline
bool bintree<T>::node::operator!=(const node & n) const
{
  return elnodo != n.elnodo;
};
```