Subject: Re: NEW: Tree<T> container
Posted by mrjt on Fri, 30 Jul 2010 11:31:47 GMT
View Forum Message <> Reply to Message

You know that won't work at all right? You've completely mixed up the data type and the storage type, there's mis-casts from Tree<T> to T all over the place.

I think I can see what you're trying to do, but this would work better IMO:

```
template <class T>
class Tree : public One<T>, public Moveable<Tree<T> >
{
private:
 Tree<T> *  parent;
 Vector<Tree<T> >  children;
public:
 Tree<T>       *GetParent()           { return parent; }
 const Tree<T> *GetParent() const     { return parent; }
 Tree<T>       *GetRoot()             { return Tree<T> *p = this; while (p->GetParent()) p = GetParent(); return p; }
 const Tree<T> *GetRoot() const       { return const Tree<T> *p = this; while (p->GetParent()) p = GetParent(); return p; }

 // All the necessary add/insert stuff goes here

 Tree<T>& operator[](int i)  { return children[i]; }
 const Tree<T>& operator[](int i) const  { return children[i]; }
};
```

You'll have to add as much of the Array interface as required but there isn't any way round this. If you inherit from Array then you end up having to do the same for One<> anyway and I think it makes more sense this way personally.

I've also attached a templated tree implementation that I wrote a while ago. It uses a different approach and has some different problems (notably the traversal algortihms are broken) but may be interesting to you.

And Link<T, N> isn't suitable for trees IMO, its really for multiply linked lists (such as an indexed database).

File Attachments

1) Tree.zip, downloaded 227 times