Subject: NEW: Tree<T> container
Posted by kohait00 on Fri, 30 Jul 2010 06:05:53 GMT
View Forum Message <> Reply to Message

hi folks, what about a Tree container? there is still none in U++, though there is a Link<T> with which one easyly could implement binary trees (what still should be done anyway, they are quite usefull). But for trees with variable numbers of elements there is nothing. so here comes a first shot. what do you think of the following:

///

```
template <class T>
class Tree
 : protected Array<T>
{
protected:
 typedef Array<T> B;
 T * parent;

public:
 T       *GetPtr()                { return (T *) this; }
 const T *GetPtr() const          { return (const T *) this; }
 T       *GetParent()             { return parent; }
 const T *GetParent() const       { return parent; }

// Array interface

 T&      Add()                    { T & t = B::Add(); t.parent = (T *)this; return t; }
 void    Add(const T& x)          { T & t = B::Add(DeepCopyNew(x)); t.parent = (T *)this; }// return t;
 }
 void    AddPick(pick_ T& x)      { T & t = B::Add(new T(x)); t.parent = (T *)this; }// return t; }
 T&      Add(T *newt)             { ASSERT(newt->parent == NULL); T & t = B::Add(newt); t.parent =
(T *)this; return t; }

 using B::operator[];
 using B::GetCount;
 using B::IsEmpty;

 using B::Trim;
 void    SetCount(int n)          { B::SetCount(n); for(int i = 0; i < B::GetCount(); i++)
B::operator[](i).parent = (T *)this; }
 void    SetCountR(int n)         { B::SetCountR(n); for(int i = 0; i < B::GetCount(); i++)
B::operator[](i).parent = (T *)this; }
 using B::Clear;

 using B::Remove;
```

```
 T&      Insert(int i)             { T & t = B::Insert(i); t.parent = (T *)this; return t; }
 void    InsertPick(int i, pick_ T& x) { x.parent = (T *)this; B::InsertPick(i, x); }

 using B::GetIndex;
 using B::Swap;
 using B::Move;

 T       *Detach(int i)            { T *t = B::Detach(i); t->parent = NULL; return t; }
 T&      Set(int i, T *newt)       { ASSERT(newt->parent == NULL); T & t = B::Set(i, newt); parent =
(T *)this; return t; }
 void    Insert(int i, T *newt)    { ASSERT(newt->parent == NULL); B::Insert(i, newt); newt->parent
= (T *)this; }

 using B::Drop;
 using B::Top;

 T       *PopDetach()              { T * t = B::PopDetach(); t->parent = NULL; return t; }

 void    Swap(Tree& b)             { B::Swap(b); for(int i = 0; i < b.GetCount(); i++) b[i].parent = (T
*)this; for(int i = 0; i < B::GetCount(); i++) B::operator[](i).parent = &b; }

 Tree& operator<<(const T& x)      { Add(x); return *this; }
 Tree& operator<<(T *newt)         { Add(newt); return *this; }
 Tree& operator|(pick_ T& x)       { AddPick(x); return *this; }

// Array Interface end

 Tree()
  : parent(NULL)
 {}

private:
 Tree(const Tree&);
 void operator=(const Tree&);

public:
#ifdef _DEBUG
 void Dump() {
  for(int i = 0; i < GetCount(); i++)
   LOG((*this)[i]);
  LOG("----------------------------------");
 }
#endif
};

//
```

the Tree is actually a partially hidden Array of the same type elements, with a parent pointer. some methods from Array are free to access, some are overblended. the protected inheritance ensures that the overblended base methods are inaccessible. some methods are not critical though and can be exposed, in some, the parent ref is to be ensured. this thing can be thought in Vector and the Map flavours as well. in general i might need to a extended templated version where to specify which container to use. but this will grow in complicity at the beginning.

i'll try to provide a binary tree idea.

please post enhancments . i have not tested the whole thing very much though, first wanted to get sure that the model is right.

cheers

Subject: Re: NEW: Tree<T> container
Posted by kohait00 on Fri, 30 Jul 2010 07:11:23 GMT
View Forum Message <> Reply to Message

do i understand the Link<T, N=1> right?
is it for multidimensional linkage?

if so, the linkage is reasoably / logically not trivial (tried to draw the dependancies on a sheet of paper, freak

maybe that's why i havent seen any use of N>1 so far..any examples?

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 152 times

Subject: Re: NEW: Tree<T> container
Posted by kohait00 on Fri, 30 Jul 2010 12:00:12 GMT
View Forum Message <> Reply to Message

Quote: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.

well i've forgotten to provide how to use it , sorry..just like that, youre perfectly right. the aproach is leaning on how it's done in Link<T> , so here comes how to setup a node.

```
class Node
 : public Tree<Node>
{
public:
 typedef Node CLASSNAME;
 ///.... your data, maybe this
 String name;
 Value value;
};
```

then using it, maybe like that

```
root.name = "/";
root.SetCount(3);
Node & child = root[2];
child.name = "any child";
rood.Add().name = "another child";
RLOG(child.GetParent()->name);
RLOG(root.GetCount());
```

this way it works good. the pointer casts do their job well, just like in Link<T>, i'll take a look at your proposal as well.

Subject: Re: NEW: Tree<T> container
Posted by mrjt on Fri, 30 Jul 2010 12:15:05 GMT
View Forum Message <> Reply to Message

That does make more sense Though IMO it's not much use as a general storage class since it will only work with classes specifically designed for it. If I want a tree of Rects I don't really want to have to wrap them in a Node class.

Subject: Re: NEW: Tree<T> container
Posted by kohait00 on Fri, 30 Jul 2010 12:20:05 GMT
View Forum Message <> Reply to Message

here you're right

Subject: Re: NEW: Tree<T> container
Posted by kohait00 on Fri, 30 Jul 2010 14:59:31 GMT
View Forum Message <> Reply to Message

i'looked inside your code and borrowed the GetRoot idea
i personally prefer the Node storage beeing pure Array, without the way complicated next/prev stuff of linked lists. if i really need to iterate through them, i could GetParent and access all children (todo: know one's own position

now there is 2 flavors of the hole thing.

Tree<T>, beeing organized through a Node<T> as root (where T can easyly be One<T> if needed). making direct deriving of the object unneeded, but still possible if one wants to one can setup own Node with the needed elements directly stored.

Branch<T>, beeing organized directly from a root instance of Branch<T>. having a T instance inside. but this reduces the flexibility a bit (in terms of pick_ Add()'s to the Array, compared to Tree<T>, where T *are* the elements of Tree, here, the Branch only has a T memeber. so Branch is actually a subset of Tree and could be esiely done with Tree<T>. i'll provide another example soon.

thanks for help. if that all works, the algorithms are to be implemented, binary, binary balanced, ...

File Attachments

1) TreeTest.rar, downloaded 153 times

Subject: Re: NEW: Tree<T> container
Posted by mirek on Fri, 13 Aug 2010 07:28:51 GMT
View Forum Message <> Reply to Message

Ah, the "tree container" problem.

I have to say I have spent quite some time trying to figure this one out.

My current position is that it is not worth the effort. In almost all cases, you need some specialized handling of trees and for those, existing containers are quite fine.

So now I usually use some sort of Array or ArrayMap to represent trees.

I guess the most simple method is something like:

```
struct Node {
    int parent;
....
};

Array<Node> tree;
```

where parent is either directly index of parent node in 'tree', or some mode sophisticated thing, like database id, in that case

```
ArrayMap<int, Node> tree;
```

is perhaps better. Of course, at this point get varied by actual requirements.

Subject: Re: NEW: Tree<T> container
Posted by kohait00 on Fri, 13 Aug 2010 07:45:17 GMT
View Forum Message <> Reply to Message

yes, you are right about the specialized trees..
nevertheless exist a lot of cases where 'simple' (to be defined again) linking is needed, without the user having to spend times and times on working out a structure. beeing this the case, they can at least work out the their own implementation when comparing to this one..and analyzing drawbacks or benefits. (i.e. should Vector be used instead, or even VectorMap)

would you mind to add it to bazaar ? is it worth anyway, as a reference implementation / idea base?

Subject: Re: NEW: Tree<T> container
Posted by mirek on Fri, 13 Aug 2010 07:59:37 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Fri, 13 August 2010 03:45yes, you are right about the specialized trees..
nevertheless exist a lot of cases where 'simple' (to be defined again)

Well, if simple linking is what you aim at:

```
struct Tree {
   Array<Tree> tree;
   ... // any data associated with node
}
```

However, I guess I see no problem about bazaar - but I do not manage bazaar either

---

Subject: Re: NEW: Tree<T> container
Posted by kohait00 on Fri, 13 Aug 2010 08:10:06 GMT
View Forum Message <> Reply to Message

this is almost what Tree<T> is about
i'll put it there..
