Subject: Interesting struggle with "Moveable<T>" usage in GCC
Posted by mr_ped on Thu, 17 Jul 2008 13:13:59 GMT
View Forum Message <> Reply to Message

Hello, I did hit recently a compilation error I didn't understand fully, still I did find a workaround, so I'm fine, but maybe some compiler guru will be able to explain it better to me.

Let's consider (I always end with OOP design like this somewhere, I'm not sure whether it's a curse or good thing, but my mind loves constructs like this one):

```
class CT {
public:
  struct ST : Moveable<ST> {
    int x, y;
  };
};
//Nice little embedded struct inside a class.
//Perfectly Moveable too, just two ints.

//But now you want to initialize it like with classic struct:
/* 1) */     CT::ST a = { 0, 1 };
/* 2) */     CT::ST b[2] = { { 0, 1 }, { 3, 4 } };
```

During compilation with GCC (didn't try it with MSC, sorry) you get errors:
1) error: braces around initializer for non-aggregate type 'CT::ST'
2) error: braces around scalar initializer for type 'CT::ST'

I was like, what's wrong with simple struct initializer? As this really does save me lot of time while I'm writing unit tests, I decided to investigate it a bit more, and by hopeless trying to change every piece of that source I figured out how to make it work. The "correct" way to define such struct to be both Moveable, and allow you to write direct initialization with braces is like this:

```
//classic struct definition
class CT {
public:
  struct ST {
    int x, y;
  };
};

//make it also moveable, so NTL will store it in Vector container
namespace Upp {
  NTL_MOVEABLE( CT::ST );
}
```

(I tried to compile the "fixed" source under MSC, now it's complaining about missing DeepCopyConstruct ... fortunately after adding that one both compilers are happy (the manual page http://www.ultimatepp.org/srcdoc$Core$pick_$en-us.html did told me how to fix deep copy))

So, I have almost no idea why the compiler is such b*tch about basically the same source just written in different way, but in case somebody hits this problem, I'm posting my "work around".

Also I welcome any reasonable explanation, or eventually fix to "Moveable<T>" if possible (I think it's impossible, maybe due to template nature?), because it's much more neat to read in source ": Moveable<T>" than the NTL_MOVEABLE macro followed by 4 deepcopy functions in worst case surrounded by yet another namespace definition :/. I prefer when the source contains minimum of "accident" content, and looks almost like pure "essence" thing, if you know what I mean.

Subject: Re: Interesting struggle with "Moveable<T>" usage in GCC
Posted by mrjt on Tue, 22 Jul 2008 10:40:51 GMT
View Forum Message <> Reply to Message

The explanation for this is quite straightforward I think.

Structs with just public data members are essentially C aggregates, which can be initialised using the {,} style list.

As soon as you add anything that stops it being a 'pure' aggregate - such as a constructor, inheritance (even from an empty class as is the case with Moveable<T>) or virtual functions - the compiler starts treating the struct as a C++ class. As classes cannot be initialised like aggregates you get the error.

For example:

```
// This works with {,} initialisation
struct ST {
  Point Get() { return Point(x, y); }
  int x, y;
};

// These don't
struct BC {
};
struct ST : BC {
  int x, y;
};

struct ST {
  ST() : x(0), y(0) { }
  int x, y;
};
```

```
struct ST {
  virtual Point Get() { return Point(x, y); }
  int x, y;
};
```As far as I can tell this is correct compiler behaviour, although it would be nice if it was a bit more forgiving.

I'm not sure about your problem with NTL_MOVEABLE though, I just tried it with MingW and MSC8 and had no problem with missing DeepCopyConstructs, but presumably you're using a more complex structure than my test.

---

Subject: Re: Interesting struggle with "Moveable<T>" usage in GCC
Posted by mr_ped on Tue, 22 Jul 2008 13:36:31 GMT
View Forum Message <> Reply to Message

I was not aware of the aggregate/class flavors, that was the missing piece for me. (it's funny I use C/C++ for quite some years, but I never really bothered to study the language itself extensively to a point how well I did study ASM or Pascal, I just learn new things as I hit them during programming)
I would expect this behavior with virtual functions, as then you have to init vtab pointer for every instance, but the constructor and inheritance took me by surprise.

And yes, I would love a bit more intelligent and forgiving compiler, as from ASM point of view there's no true different between struct with 2 ints, and same struct with constructor, I will keep wishing.

Thank you for explanation, and about the deepcopy, yes, my real source is more complex, with some "typedef Vector<almost_struct_class> TmyVector;" probably being the culprit.
This was just a bare minimum source to show how Moveable prevents you from braces initialization. None of these are big problems, they just clutter my sources a bit more than it would be necessary in ideal world, so I had to ask...

Edit:
And another syntax sugar which would make my sources look better would be direct initialization of Vector container. I think with some clever C++ operator overloading this may be eventually possible, or something which would be quite close, but I don't have time+will to look into it, and heavy usage of operators makes me always to shiver a bit, as you have to never forget about them when you read the source.

---

Subject: Re: Interesting struggle with "Moveable<T>" usage in GCC
Posted by mr_ped on Wed, 30 Jul 2008 10:14:47 GMT
View Forum Message <> Reply to Message

Another funny thing, if you mess up the optional deep copy functions ( namely

DeepCopyConstruct to call the implicit copy constructor instead of your T( T, int ) deep copy constructor ) for your class defined to be moveable trough that NTL_MOVEABLE macro:
you can end with Vector source which works with MSC8 (the implicit copy constructor is not called at all), but does break under GCC. (with the wrong pick behavior assert)

I can't provide example because the source is too complex and basically it was my bug in DeepCopyConstruct, so it's not problem of UPP.

It just made me to scratch my head a bit, that two compilers do use quite different way to construct the final Vector related code for my class, MSC avoiding my bugged functions completely.

Of course using "class A : MoveableAndDeepCopyOption<A> {};" fixed that and made my code cleaner (and right now I didn't use direct initialization for that A class anymore), so if you are new to UPP+pick behavior (like me), try to avoid making things by hand, and use rather those Moveable<> and similar things whenever possible, it makes code clean and easy to read, and will prevent you from doing silly mistakes (which may go undetected with one compiler and show just later on another one).