Subject: PolyDeepCopyNew: MSC / GCC differ in behaviour
Posted by kohait00 on Thu, 14 Oct 2010 11:59:22 GMT
View Forum Message <> Reply to Message

hi people

i've got some issue using the PolyDeepCopyNew feature, where GCC and MSC behave differently, while this time MSC seems to do it right, or i did sth wrong. MSC compiles well, GCC doesnt. I use MSC9 and TDMGCC.

the problemarising is:
GCC seems not to be able to properly recognize the friend DeepCopyNew from PolyDeepCopyNew as an alternative to the template DeepCopyNew.

any help on this one is really appriciated.
i've added a testcase down there.

```
#include <Core/Core.h>

using namespace Upp;

//without this one beeing in namespace upp it doesnt work eaither
//suppose because of friend DeepCopyNew not beeing placed / referenced to template definition
NAMESPACE_UPP

template<class T, class B = EmptyClass>
class Copyable : public B
{
public:
 virtual ~Copyable() {}
 virtual T* Copy() const                 { return PartialCopy(); }
 virtual T* PartialCopy() const            = 0; //{ NEVER(); return NULL; }
};

template<class C, class B = EmptyClass>
class CopyableC : public Copyable<CopyableC<C,B>, B>
{
public:
 virtual const C& GetC() const             = 0;
 virtual C& GetC()                       = 0;

 operator const C&() const                { return GetC(); }
 operator C&()                           { return GetC(); }
};

template<class B, class C, class CB = EmptyClass>
class PolyCopyableC : public B, public PolyDeepCopyNew<CopyableC<C, CB>, CopyableC<C,
```

```
CB> >
{
public:
 virtual const C& GetC() const             { return *this; }
 virtual C& GetC()                        { return *this; }
};

END_UPP_NAMESPACE

//OWN CLASSES

//this one doesnt help
//NAMESPACE_UPP

//a common base interface, that should be available after copying, (i.e Offer() )
//is meant as an extension interface on very bottom
//should be implemented at very top
class CBase
{
public:
 virtual void Offer(int a) = 0;
};

//a common base class, which should be accessible via GetC (could be i.e Ctrl)
class Base {};

//a Base version (could be any Ctrl derive, i.e StaticText)
//neither Base nor Derived are to be changed, they are foreign
//thats why so complicated
class Derived : public Base {};

//extension of the whole thing, to make everything cloneable..
class Master : public PolyCopyableC<Derived, Base, CBase>
{
public:
 Master* PartialCopy() const { return new Master(); }
 virtual void Offer(int a) {}
};

//END_UPP_NAMESPACE

//with this one both work, but i'd like to avoid that and to have it automatic in PolyCopyableC
#if 0
NAMESPACE_UPP
template<>
inline CopyableC<Base, CBase>* DeepCopyNew(const CopyableC<Base, CBase>& x) {
 return x.Copy();
}
```

```
END_UPP_NAMESPACE
#endif

CONSOLE_APP_MAIN
{
 Array<CopyableC<Base, CBase> > a1, a2;
 a1.Add(new Master());
 a1 <<= a2;
}
```

File Attachments

1) Blate.rar, downloaded 249 times

Subject: Re: PolyDeepCopyNew: MSC / GCC differ in behaviour
Posted by mirek on Fri, 15 Oct 2010 11:50:08 GMT
View Forum Message <> Reply to Message

I have just tried with Mingw, compiles fine. Which GCC version then?

Subject: Re: PolyDeepCopyNew: MSC / GCC differ in behaviour
Posted by kohait00 on Fri, 15 Oct 2010 13:02:29 GMT
View Forum Message <> Reply to Message

tdm-mingw-1.908.0-4.4.1-2.exe

seems weired..

maybe i should try with a more recent version

EDIT: my current version of MINGW (4.4.0) doesnt compile it either. which MINGW version did u use?

EDIT: i tried the current TDM release, 4.5.1, same issue. doesnt compile. dont get how you made it..

Quote:

```
cd C:\dfm-git-cl\Blate
Blate.cpp
c++ -c  -I"C:\dfm-git-cl" -I"C:\uppsvn\bazaar" -I"C:\uppsvn\uppsrc" -I"C:\MinGW32\include"
-DflagMAIN -DflagGCC -DflagDEBUG -DflagDEBUG_FULL -DflagBLITZ -D
 flagWIN32 -DbmYEAR=2010 -DbmMONTH=10 -DbmDAY=15 -DbmHOUR=16 -DbmMINUTE=2
-DbmSECOND=31 -g2 -static   -fexceptions  -D_DEBUG -O0 -gstabs -x c++ "C:\dfm-
 git-cl\Blate\Blate.cpp" -o "C:/uppsvn/out/Blate/TDM451.Debug.Debug_full.Main\Blate.o"
```

```
compiled in (0:00.01)
In file included from C:\uppsvn\uppsrc/Core/Core.h:219:0,
                 from C:\dfm-git-cl\Blate\Blate.cpp:1:
C:\uppsvn\uppsrc/Core/Topt.h: In function 'T* Upp::DeepCopyNew(const T&) [with T =
Upp::CopyableC<Base, CBase>]':
C:\uppsvn\uppsrc/Core/Vcont.hpp:363:3:   instantiated from 'void
Upp::Array<T>::__DeepCopy(const Upp::Array<T>&) [with T = Upp::CopyableC<Base, CBase>]'
C:\uppsvn\uppsrc/Core/Vcont.h:229:40:   instantiated from 'Upp::Array<T>::Array(const
Upp::Array<T>&, int) [with T = Upp::CopyableC<Base, CBase>, Upp::Arra
 y<T> = Upp::Array<Upp::CopyableC<Base, CBase> >]'
C:\uppsvn\uppsrc/Core/Topt.h:235:42:   instantiated from 'Upp::Array<Upp::CopyableC<Base,
CBase> >& Upp::operator<<=(Upp::Array<Upp::CopyableC<Base, CBase>
  >&, const Upp::Array<Upp::CopyableC<Base, CBase> >&)'
C:\dfm-git-cl\Blate\Blate.cpp:85:9:   instantiated from here
C:\uppsvn\uppsrc/Core/Topt.h:142:16: error: cannot allocate an object of abstract type
'Upp::CopyableC<Base, CBase>'
C:\dfm-git-cl\Blate\Blate.cpp:20:1: note:   because the following virtual functions are pure within
'Upp::CopyableC<Base, CBase>':
C:\dfm-git-cl\Blate\Blate.cpp:50:15: note:  virtual void CBase::Offer(int)
C:\dfm-git-cl\Blate\Blate.cpp:15:13: note:  T* Upp::Copyable<T, B>::PartialCopy() const [with T =
Upp::CopyableC<Base, CBase>, B = CBase]
C:\dfm-git-cl\Blate\Blate.cpp:22:19: note:  const C& Upp::CopyableC<C, B>::GetC() const [with C
= Base, B = CBase]
C:\dfm-git-cl\Blate\Blate.cpp:23:13: note:  C& Upp::CopyableC<C, B>::GetC() [with C = Base, B =
CBase]
C:\MinGW32\bin\c++.exe -c  -I"C:\dfm-git-cl" -I"C:\uppsvn\bazaar" -I"C:\uppsvn\uppsrc"
-I"C:\MinGW32\include" -DflagMAIN -DflagGCC -DflagDEBUG -DflagDEBUG_
 FULL -DflagBLITZ -DflagWIN32 -DbmYEAR=2010 -DbmMONTH=10 -DbmDAY=15
-DbmHOUR=16 -DbmMINUTE=2 -DbmSECOND=31 -g2 -static   -fexceptions  -D_DEBUG -O0
-gst
 abs -x c++ "C:\dfm-git-cl\Blate\Blate.cpp" -o
"C:/uppsvn/out/Blate/TDM451.Debug.Debug_full.Main\Blate.o"
Blate: 1 file(s) built in (0:00.93), 937 msecs / file, duration = 969 msecs, parallelization 0%
Error executing C:\MinGW32\bin\c++.exe -c  -I"C:\dfm-git-cl" -I"C:\uppsvn\bazaar"
-I"C:\uppsvn\uppsrc" -I"C:\MinGW32\include" -DflagMAIN -DflagGCC -DflagDE
 BUG -DflagDEBUG_FULL -DflagBLITZ -DflagWIN32 -DbmYEAR=2010 -DbmMONTH=10
-DbmDAY=15 -DbmHOUR=16 -DbmMINUTE=2 -DbmSECOND=31 -g2 -static   -fexceptions  -
 D_DEBUG -O0 -gstabs -x c++ "C:\dfm-git-cl\Blate\Blate.cpp" -o
"C:/uppsvn/out/Blate/TDM451.Debug.Debug_full.Main\Blate.o"

There were errors. (0:01.18)
```

Subject: Re: PolyDeepCopyNew: MSC / GCC differ in behaviour
Posted by mirek on Sat, 16 Oct 2010 06:21:33 GMT
View Forum Message <> Reply to Message

First of all:

template<class B, class C, class CB = EmptyClass>
class PolyCopyableC : public B, public PolyDeepCopyNew<CopyableC<C, CB>, CopyableC<C, CB> >

should perhaps be

template<class B, class C, class CB = EmptyClass>
class PolyCopyableC : public B, public PolyDeepCopyNew<PolyCopyableC<C, CB>, CopyableC<C, CB> >

anyway, without going too much into what your code is supposed to do:

Array<CopyableC<Base, CBase> > a1, a2;

What makes you think CopyableC<Base, CBase> is somehow connected to PolyDeepCopyNew?!

Subject: Re: PolyDeepCopyNew: MSC / GCC differ in behaviour
Posted by kohait00 on Mon, 18 Oct 2010 07:35:46 GMT
View Forum Message <> Reply to Message

hi mirek

it's connected because i used PolyDeepCopyNew to specify the DeepCopyNew function to use the t.Copy() function, which is obligatoryly defined in Copyable<> (to make sure it is implemented) and accessible from this interfaces).

the weired point is, in MSC it works.. gcc does not recognize the friend DeepCopyNew as the specialisation from template DeepCopyNew, or the containers (here Array) doent get to understand that a global DeepCopyNew is available (from PolyDeepCopyNew, as friend), withtout the need to use the template DeepCopyNew..

with which mingw version did it work for you?

EDIT:

maybe it's the same problem that led you or the guys to implement the Moveable<> in the way it is done now.

```
template <class T>
inline void AssertMoveable(T *t = 0) { if(t) AssertMoveable0(t); }

#if defined(COMPILER_MSC) || defined(COMPILER_GCC) && (__GNUC__ < 4 ||
__GNUC_MINOR__ < 1)
 #define NTL_MOVEABLE(T) inline void AssertMoveable0(T *) {}
#else
 #define NTL_MOVEABLE(T) template<> inline void AssertMoveable<T>(T *) {}
#endif
```

maybe because there are problems with recognition of the template / friend function same names ...

Subject: Re: PolyDeepCopyNew: MSC / GCC differ in behaviour
Posted by kohait00 on Mon, 18 Oct 2010 13:07:02 GMT
View Forum Message <> Reply to Message

Quote:

should perhaps be

```
template<class B, class C, class CB = EmptyClass>
class PolyCopyableC : public B, public PolyDeepCopyNew<PolyCopyableC<C, CB>,
CopyableC<C, CB> >
```

if i choose PolyCopyableC instead of CopyableC, the DeepCopyNew from PolyDeepCopyNew will be instantiated PolyCopyableC, and using my Array<CopyableC....> will try to envoke DeepCopyNew<CopyableC<..> >, which will end up using the templated one, and not the one using the t.Copy() function..

short: i need Array<CopyableC<..> > to use the DeepCopyNew<CopyableC<..> > invoking Copy()

Subject: Re: PolyDeepCopyNew: MSC / GCC differ in behaviour
Posted by mirek on Mon, 18 Oct 2010 14:03:19 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Mon, 18 October 2010 03:35hi mirek

the weired point is, in MSC it works.. gcc does not recognize the friend DeepCopyNew as the specialisation from template

Well, the C++ rules as applied to your code are so complicated that even compiler writers interpret them differently

Frankly, the code is quite complicated for me too. Resolving the issue whether MSC or GCC is right might require several hours spend with C++ standard, which is something I am not going to sacrifice now:) If you want to have fun, concentrate on chapters that deal with template class instantiation...

Personally, I do not see any usefulness in what your are trying to do. Even current version of PolyDeepCopyNew is in fact "purely academic" - we know we are able to create a copy of polymorphic container, but in practice, it is never needed. I would even say that code that would benefit from it might have some serious design flaw

---

Subject: Re: PolyDeepCopyNew: MSC / GCC differ in behaviour
Posted by kohait00 on Mon, 18 Oct 2010 14:16:29 GMT
View Forum Message <> Reply to Message

i've got some polymorphic object instantiation factories in development. that's why the need. nevermind. thanks for your patience. basicly, the example provided is an extract of the current work reduced to the 'most simple' to reproduce the compile error.

nevertheless i found the thing to compile under GCC as well, moving the PolyDeepCopyNew<CopyableC<> > to the CopyableC itself, instead of specifying it somewhere in upper hierarchy. but it's actually not what i wanted.. maybe you will have a hint. if not i dont mind.

```
template<class C, class B = EmptyClass>
class CopyableC : public PolyDeepCopyNew<CopyableC<C,B>, Copyable<CopyableC<C,B>, B>
>
{
public:
 virtual const C& GetC() const             = 0;
 virtual C& GetC()                        = 0;

 operator const C&() const                { return GetC(); }
 operator C&()                           { return GetC(); }
};

template<class B, class C, class CB = EmptyClass>
class PolyCopyableC : public B, public CopyableC<C, CB>
{
public:
 virtual const C& GetC() const             { return *this; }
 virtual C& GetC()                        { return *this; }
};
```

Subject: Re: PolyDeepCopyNew: MSC / GCC differ in behaviour
Posted by kohait00 on Tue, 19 Oct 2010 13:53:50 GMT
View Forum Message <> Reply to Message

and i probably can explain why it did not compile...

since PolyDeepCopyNew::DeepCopyNew(const CopyableC&) was provided as of the hierarchy level / namespace of PolyCopyableC, and the compiler tried to match its existence only from CopyableC, it didnt find it there.

instatiating it explicitly helped it, but was no right cure..
i admit the approach is not easy.. but i hope to be able to provide a testcase showing all of what has been 'invented' so far..

it's a generic factory helper, for polymorphic elements, specifying a common handling interface and beeing able to extend it eith some additional functions from very bottom (in case common interface like i.e. Ctrl cant be extended from inside..)