Subject: Clang vs. GCC
Posted by dolik.rce on Sat, 06 Feb 2010 22:10:46 GMT
View Forum Message <> Reply to Message

Hello,

I'm not sure if this is the right place to post this, but I couldn't find any category where it would fit. I installed fresh svn version of Clang (C++ frontend for LLVM) inspired by a news that it finally can build itself. I was playing with it a bit and of course I tried to compile some U++ code.

I'm not going to talk about the compilation problems, but about interesting error that popped out. In topt.h there is this template:template <class T>

```
inline void DestroyArray(T *t, const T *lim) {
 while(t < lim) {
  t->T::~T();
  t++;
 }
}
```

Clang complained about using this template with T=unsigned int, since "type 'unsigned int' cannot be used prior to '::' because it has no members". After this I was curious about how GCC solves this problem. Well, the answer is simple: GCC ignores it.

This template is being called with T equal to types like int, const char*, void* without any problems. When I stepped through this part of code in debugger, the destructor line was simply skipped, the loop run through the given range of pointers doing nothing usefull.

My knowledge of C++ is quite limited, so I have few questions: How is that possible? Is it a GCC, Clang or U++ bug? Or is it just me, missing some deep knowledge about inlined functions, templates or some other dark corner of C++?

I hope someone can enlighten me a bit... I really like Clangs verbosity an it would be great if U++ supported it once (But rewriting half of the Core is just too high price).

Best regards,
Honza

Subject: Re: Clang vs. GCC
Posted by Novo on Sun, 07 Feb 2010 01:01:07 GMT
View Forum Message <> Reply to Message

IMHO in this particular case a Vector should be used instead of an Array. "unsigned int" is moveable.

Subject: Re: Clang vs. GCC
Posted by mirek on Sun, 07 Feb 2010 08:55:45 GMT

View Forum Message <> Reply to Message

Novo wrote on Sat, 06 February 2010 20:01IMHO in this particular case a Vector should be used instead of an Array. "unsigned int" is moveable.

Has nothing to do with that. This support routine is in fact used for Vector implementation.

Subject: Re: Clang vs. GCC
Posted by mirek on Sun, 07 Feb 2010 09:13:20 GMT

View Forum Message <> Reply to Message

I believe it is Clang bug. By C++ standard, all types, including fundemantal types, have constructor and destructor.

At the moment, I am unable to find corresponding definition in C++ language definition, closest info I have found is this:

http://www.informit.com/guides/content.aspx?g=cplusplus& seqNum=431

BTW, it is in fact impossible to create container templates without this - STL has to do the same thing. Which is strange, considering Clang refusal to compile it.

Mirek

Subject: Re: Clang vs. GCC
Posted by gprentice on Sun, 07 Feb 2010 10:49:42 GMT

View Forum Message <> Reply to Message

Yep it's a clang bug and a dark corner of C++.

As mentioned in that article, the standard defines a pseudo destructor (5.2.4) one form of which looks like this
::opt nested-name-specifier opt type-name :: ~ type-name

where the first :: and the nested-name-specifier are optional and type-name is a non-class type. The only effect is the evaluation of the post-fix expression before the arrow.

There's no such thing as a constructor for a fundamental type but the standard defines (5.2.3 / 2) that the expression T() for simple type specifier T creates an rvalue of the specified type whose value is determined by default initialization.

Graeme

Subject: Re: Clang vs. GCC
Posted by Didier on Sun, 07 Feb 2010 11:11:54 GMT
View Forum Message <> Reply to Message

This can be used to optimize the 'DestroyArray()' function by adding specialized versions for internal types.

This function could be, for example:

```
template <>
inline void DestroyArray<int>(T *t, const T *lim) {
  }
}
```

This would then get optimized out by the compiler.

This could be generalized to all internal types and factored by using a IsInternaType class:

```
// general case for all complex types
template<typename T>
struct IsInternalType
{
  enum { value = 0 };
};

// specialized classes for internal types
template<>
struct IsInternalType<int>
{
  enum { value = 1 };
};

template<>
struct IsInternalType<unsigned int>
{
  enum { value = 1 };
};

template<>
struct IsInternalType<float>
{
  enum { value = 1 };
};

// .....  and so on for all other types you want


// ==============================================================
//the generalized function would then become:
```

```
template <int I, class T>
 inline void _DestroyArray(T *t, const T *lim) {
  while(t < lim) {
   t->T::~T();
   t++;
  }
 }

// the specialized version (for internal types) does nothing
template <class T>
static inline void _DestroyArray(T *t, const T *lim) {}



// FINALLY THE ORIGINAL METHOD becomes this
// it automatically selects, AT COMPIL TIME, the wright function depending on it's type
template <class T>
inline void DestroyArray(T *t, const T *lim) {
  _DestroyArray< IsInternalType<T>::value, T >(t, lim);
};
```

NB: this could be easily extended to any custom type by writeing you're own specialized IsInternalType classe dedicated to you're type

Edit: maybe the 'IsInternalType()' function would be better named by 'HasDestructor()'

Subject: Re: Clang vs. GCC
Posted by mirek on Sun, 07 Feb 2010 13:35:30 GMT
View Forum Message <> Reply to Message

I think you way understimate the compiler here.... namely dead code elimination.

Mirek

Subject: Re: Clang vs. GCC
Posted by Didier on Sun, 07 Feb 2010 13:47:50 GMT
View Forum Message <> Reply to Message

Yes you are wright.

In this particular case the compiler will probably eliminate the loop.

But if the loop is more complex, this kind of optimization is very handy.

Subject: Re: Clang vs. GCC
Posted by dolik.rce on Sun, 07 Feb 2010 14:35:56 GMT
View Forum Message <> Reply to Message

Wow, nice discussion. Thanks everybody for your thoughts.

The loop gets executed in debug mode. I haven't tried in optimal, but I believe it is optimized.

Template specialization is probably a way to make this work in Clang. But as I said before, that is too high price.

Fixing Clang would be better solution, should I file a bug on their site? Or probably someone who knows C++ better than me should report it

I hope that Clang will be usable soon. Just on the side: What is needed to get support for new compiler? Just a build method in ide/Builders? And is there some documentation on how buildscripts work? (I mean method Script).

Honza

Subject: Re: Clang vs. GCC
Posted by Sgifan on Wed, 24 Feb 2010 07:53:14 GMT
View Forum Message <> Reply to Message

I read today that clang+LLVM is completely self hosting from now up.

Maybe the bug you show is still present though.

If one day clang is usable with u++ i would be curious to learn how it improves or not the compilation speed.

Thanks for the test you performed
