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()'
