Author: Nikhil Joshi, IBM, nikhilnjoshi@in.ibm.com
Document number: N3348=12-0038
Date: 2012-01-11
Project: Programming Language C++, Evolution Working Group
Reply-to: Nikhil Joshi, IBM, nikhilnjoshi@in.ibm.com
Revision: 1

Scoping of operator new

1 The Problem

Third-party libraries, especially any libraries that handle huge memory, overload the “new” operator.

Many a times these libraries overload the default new rather than overloading it only for each class contained in the library. Overloading for each class is deemed tedious, clumsy and even unnecessary, as this adds to the code maintenance wows of the developers of these libraries.

While using these libraries in a (large) project which itself handles huge memory, and therefore has a memory manager developed, might have the global new operator overloaded for the same reason as the library.

This creates problem, confusion in usage of the operator as currently C++ doesn’t allow scoping of the operator new.

Example

In library.h we have

```
void * operator new(size_t iBytes)
{
      //Do some internal thing
      refcount++;

      return operator new (iBytes); /*Use of ::new or default implementation
of new is intended here but will it be used?*/
}


class some_class
{
private:
      char *astring;
      Init();
}
```

```
some_class::Init()
{
	…
	astring = new char[56]; //should use new from the library
	…
}

In myCode.cpp we have
#include <mymem.h> //Has another “new” implementation
#include <library.h>

int main(int argc, char *argv[])
{
	some_class *sc;

	sc = new some_class; /*which "new" will be used is not evident looking
at the code, cannot clarify usage as scope::new syntax is not allowed*/
/* also this can lead to debugging nightmare in case of leak/crash */

}
```

2 The Proposal

Allow overloaded new to be scoped.

Example

Above code can be written as

In library.h

```
void * operator new(size_t iBytes)
{
	//Do some internal thing
	refcount++;

	return std::operator new (iBytes); /*we are sure what is getting used*/
}


class some_class
{
private:
	char *astring;
	Init();
}

some_class::Init()
```

```
{
	…
	astring = new char[56]; //should use new from the library
	…
}

In myCode.cpp we have
#include <mymem.h> //Has another “new” implementation
#include <library.h>

int main(int argc, char *argv[])
{
	some_class *sc;

	sc = new library_namepsace::some_class; /*assuming the library has
namespace, else current syntax will still work */

}
```

3 Interactions and Implementability

3.1 Interactions

Using this mechanism

- Each user of overloaded new can encase their version in their namespace
- Code in the namespace uses the new in their namespace
- Can use the default new using syntax ::new, and
- Use new from another namespace as *namespace_name*::new
- Existing code and behavior doesn’t change

For above reason this addition should interact well with existing C++ features.

3.2 Implementability

This is possible to implement using existing C++ framework as

- Operators are defined as special functions with restriction that number, type of parameters cannot be changed – this remains as it is in this case as well
- Since operators are functions and functions are already scoped in C++, operator new can be scoped without changing basic design
- With the default implementation of new scoped at global level (as in ::new)

4 Conclusion

Adding scoping of operator new will enable implementers of classes to limit scope of their version of new, and allow users of these classes to use either their version of new, implementers of the classes or the default supplied by compiler vendor.

In general, it will give choice of which implementation of new to use, and will be nice addition to C++.

5 Acknowledgements

I thank Renji Panicker, Arun Kumar for the discussions and resulting suggestions and Michael Wong for his suggestions and help in putting this proposal together.