CoCoALib - Bug #591

Problem with template instantiation and order of include directives

16 Jul 2014 14:05 - John Abbott

Status:	In Progress	**Start date:**	16 Jul 2014
Priority:	Low	**Due date:**	
Assignee:		**% Done:**	50%
Category:	Portability	**Estimated time:**	5.00 hours
Target version:	CoCoALib-1.0	**Spent time:**	5.20 hours

Description

Some versions of g++ (v4.3.2, v4.4.7, v4.6.3) and intel C++ compiler gave errors when compiling ex-UtilsVector1.C if, in the file degree.H, the #include directive for UtilsVector.H was placed before the other #include directives. The error produced indicated that some prototypes for cmp were not visible (in ptic for two @MachineInt@s).

We have no idea why, but since several compilers complain we wonder whether it isn't a strange limitation of C++.

Logging the problem here. The solution is just to move the #include directive in degree.H to after the other two.

Related issues:

Related to CoCoALib - Bug #264: Compilation problem with "degree.H" (inline f...	**Closed**	**12 Oct 2012**

History

#1 - 16 Jul 2014 14:07 - John Abbott

- % Done changed from 0 to 10

JAA failed to find anything helpful on the internet.

I'm just hoping that the solution of moving the #include will be sufficient for the foreseeable future.

#2 - 16 Jul 2014 14:14 - John Abbott

Clang 3.0 on my computer gives no error.

#3 - 09 Jan 2017 16:56 - John Abbott

- Description updated

The problem persists with g++ 5.3.1. It must be C++ thing, some weird restriction about calling "global" fns from inside template code.

#4 - 10 Jan 2017 11:12 - John Abbott

- Status changed from New to In Progress

It really is a C++ trap for the unwary... grrr!

The following code fails to compile because the last line (iter(vs);) needs the second defn of func, but that is not visible at the point where the template fn was defined. I'm at a loss for words -- why does C++ have this "feature"???

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

void func(int n)
{
  cout << "int ";
```

```
}

template <typename T>
void iter(const std::vector<T>& v)
{
  const int n = v.size();
  for (int i=0; i < n; ++i)
    func(v[i]);
  cout << endl;
}

void func(const std::string& str)
{
  cout << "str ";
}

int main()
{
  vector<int> vi; vi.push_back(1);
  vector<string> vs; vs.push_back("abc");
  iter(vi);
  iter(vs);
}
```

#5 - 10 Jan 2017 11:20 - John Abbott

There remains the question of how to correctly organize the header files in CoCoALib so that this "wonderful feature" of C++ does not cause too much grief.

#6 - 11 Jan 2017 15:02 - John Abbott

Mario and I looked in Stroustrup's C++ book (v.4), and the magic phrase appears to be **point-of-instantiation binding** (sec. 26.3.3).

The matter is discussed on the following thread:
http://stackoverflow.com/questions/30514337/point-of-instantiation-and-name-binding
In summary, the point-of-instantiation binding apparently **does not work for built-in C++ types**; instead the **point-of-definition binding** is used.
Grrr!

What I still do not know is how best to organize the few templates in CoCoALib header files to avoid future pain from these arcane C++ rules. Ideas are welcome!

#7 - 12 Mar 2020 14:39 - John Abbott

- Target version changed from CoCoALib-1.0 to CoCoALib-0.99850

- % Done changed from 10 to 50

What should we do with this issue? The correct "solution" is for us to learn the foibles of C++.

It does highlight an important, awkward point of C++. We do not use templates that much, but I suppose it will sooner or later cause us grief again.

In practice... should we just close/reject this issue?

#8 - 09 Feb 2024 10:18 - John Abbott

- Target version changed from CoCoALib-0.99850 to CoCoALib-1.0