On the same machine with ICC 13 they run correctly. With ICC 14.0.2 6 (9 on a different machine) of the analysisdata tests fail (even with debug build type). But running the failing tests individually causes them to pass.

Errors start with:

```
[ RUN      ] AverageModuleTest.HandlesMultipointData
.../src/testutils/refdata.cpp:860: Failure
Value of: strValue
Actual: "3"
Expected: refStrValue
Which is: "2"
```

```
Google Test trace:
../src/testutils/refdata.cpp:853: Checking '/InputData/Frame0/[2]/Count'
../src/testutils/refdata.cpp:717: Failure
Failed
Mismatch while checking reference data item '/InputData/Frame0/[2]/FirstColumn'
Expected: it is absent.
Actual: it is present.
```

```
../src/testutils/refdata.cpp:921: Failure
Value of: value
Actual: 1.1000000238418579
Expected: refValue
Which is: 1
Difference: 0.1 (838861 single-prec. ULPs)
Tolerance: abs. 4.76837e-07, 4 ULPs
```

```
Google Test trace:
../src/testutils/refdata.cpp:902: Checking '/InputData/Frame0/[2]/[1]/Value'
```

valgrind doesn't show any errors.
On the one machine I get (same for release-5-0 and master):

```
[ FAILED ] 6 tests, listed below:
[ FAILED ] AverageModuleTest.HandlesMultipointData
[ FAILED ] SimpleHistogramModuleTest.ComputesCorrectly
[ FAILED ] SimpleHistogramModuleTest.ComputesCorrectlyWithAll
[ FAILED ] LifetimeModuleTest.BasicTest
[ FAILED ] LifetimeModuleTest.CumulativeTest
[ FAILED ] LifetimeModuleTest.HandlesMultipleDataSets
```

On the other

```
[ FAILED ] SimpleHistogramModuleTest.ComputesCorrectly
[ FAILED ] SimpleHistogramModuleTest.ComputesCorrectlyWithAll
[ FAILED ] AverageModuleTest.BasicTest
[ FAILED ] AverageModuleTest.HandlesMultipleDataSets
[ FAILED ] AverageModuleTest.HandlesDataSetAveraging
[ FAILED ] AverageModuleTest.CanCustomizeXAxis
[ FAILED ] AverageModuleTest.CanCustomizeNonUniformXAxis
[ FAILED ] FrameAverageModuleTest.BasicTest
[ FAILED ] FrameAverageModuleTest.HandlesMultipleDataSets
```

The reason different number and different tests fail, is that the tests are executed in a different order on the other machine. And there seems to be no flag to guarantee the same order.

For the first machine AverageModuleTest.HandlesMultipointData with just AnalysisDataCommonTest/2.CallsModuleCorrectly fails. AverageModuleTest.HandlesMultiplicityData without any of the AnalysisDataCommonTest/2.* passes. But the other 5 tests still fail with AnalysisDataCommonTest/2* disabled. If I disable AnalysisDataCommonTest/* then suddenly 9 tests fail:

```
[ FAILED ] 9 tests, listed below:
[ FAILED ] AverageModuleTest.BasicTest
[ FAILED ] AverageModuleTest.CanCustomizeXAxis
[ FAILED ] AverageModuleTest.CanCustomizeNonUniformXAxis
[ FAILED ] FrameAverageModuleTest.BasicTest
[ FAILED ] FrameAverageModuleTest.ComputesCorrectly
[ FAILED ] SimpleHistogramModuleTest.ComputesCorrectlyWithAll
[ FAILED ] LifetimeModuleTest.BasicTest
[ FAILED ] LifetimeModuleTest.CumulativeTest
[ FAILED ] LifetimeModuleTest.HandlesMultipleDataSets
```

#3 - 07/10/2014 05:29 PM - Teemu Murtola

Could you add tracing to presentAllData() in datatest.cpp, printing all the values. What gets printed for the failing tests, compared to what the code sets in the static objects?

#4 - 07/10/2014 05:35 PM - Roland Schulz

Dont understand what you mean? Do you want me to add printf to presentAllData?

#5 - 07/10/2014 05:47 PM - Teemu Murtola

Yes.

#6 - 07/11/2014 10:48 PM - Roland Schulz

The problem is that ICC gets confused with the multiple different SimpleInputData/MultipointInputData classes. If a previous test used a different one, the subsequent one simply uses the same one even if it is from the wrong compilation unit. I can't see anything wrong with it because these classes are the anonymous namespace and thus it should be OK to have multiple ones with the same name. If I rename them all to be unique then all tests pass.

#7 - 07/12/2014 12:14 AM - Roland Schulz

I was able to reproduce the issue in a small program. Do you see anything which could be a reason this is not valid? Otherwise I'll post an ICC bug report.

m.cpp

```
void f1();
void f2();
int main()
```
f1.cpp

#include <stdio.h>

namespace {
    class A {
    public:
        static int get() {
            static A a;
            return a.data;
        }
        A() : data(1) {};
    private:
        int data;
    };
}

void f1() {
    printf("%d\n", A::get());
}

f2.cpp

#include <stdio.h>

namespace {
    class A {
    public:
        static int get() {
            static A a;
            return a.data;
        }
        A() : data(2) {};
    private:
        int data;
    };
}

void f2() {
    printf("%d\n", A::get());
}

#8 - 07/12/2014 04:52 AM - Teemu Murtola
That's one of the issues I was suspecting. I don't see anything wrong with your test program. Possibly you could also check the object files to see how the symbols in the unnamed namespaces are getting mangled (the problem is probably there, leading the linker to link the calls incorrectly). Or then it has to do with the static variables in the get() methods and their initialization.

#9 - 07/12/2014 07:33 AM - Roland Schulz
The symbols show that the compiler only creates one static variable. For normal functions it is fine but not for (static) functions inside classes.

A slightly simpler example. Of course it should give "0 0" but gives "0 1" with ICC 14.

f1.cpp

namespace {
    class A
    ```
    ```
I think we should put a workaround in the code given that the problem is in the current ICC and thus will be around for a while. Also we plan to add a Jenkins build-config with ICC 14 and so we need some solution for it.

Which workaround do you prefer:

1. Given the classes unique names
2. Given the namespace names (e.g. gmx_test_analysisdata, gmx_test_arraydata)
3. Given the namespace names but only for ICC14 (with an ifdef)
4. Something else?

Does renaming only the function or the variable solve the problem? We should try to keep the hacks as local as possible.

How do we prevent other occurrences of the problem? Not all such cases are necessarily caught by tests, and all the skelns solutions require knowing all code locations that are affected (except if there’s a compiler flag that fixes the issue).

The problem is that the static variable is made a weak symbol. Thus when the 2 units are linked together one is overwritten by the other. I looked for compiler-flags or attributes to disable weak linkage but couldn’t find any. That might have been a nicer workaround.

Yes changing the variable name works too.
Yeah we cannot not know no other code is affected. We would need to trust our tests. Not sure what is worse: 1) not supporting ICC 14 2) risking other problems undiscovered by the tests.

#13 - 07/12/2014 08:52 PM - Teemu Murtola
It's probably correct that the static is a weak symbol (at least in some cases, the static must be weak for it to work correctly), but they should have different names in the different compilation units because of the unnamed namespace.

As a code-level workaround, I'd suggest adding an icc 14 -specific ifdef that names the variables uniquely. That's very local, and doesn't affect any other code.

If we are concerned enough about the possible occurrence of the issue, we could probably use libclang or the gmxtree.py script (with some extensions) to flag all classes with identical names in different compilation units. But that's quite a lot of work, so I'm not sure whether it pays off.

#14 - 07/12/2014 11:18 PM - Roland Schulz
You are right the weak symbol isn't the problem. ICC13 uses also weak symbols and it works (g++ doesn't use weak symbols so I assumed that was the problem). The odd thing is that the symbol with ICC 13 and 14 in the .a file is:

```
0000000000000000 V (anonymous namespace)::A::get()::a
```

and still in after linking together (I get the same no matter which version of ICC I use for linking) the .o files generated by 13 gives:

```
000000000000600b44 V (anonymous namespace)::A::get()::a
000000000000600b48 V (anonymous namespace)::A::get()::a
```

whereas the .o files generated by 14 gives:

```
000000000000600b44 V (anonymous namespace)::A::get()::a
```

I don't know how the linking works, so maybe that is actually not that odd - but it looks odd to me.

#15 - 07/12/2014 11:21 PM - Roland Schulz
Never mind. The mangled name is different for 13 whereas the mangled name is the same for 14.

#16 - 07/13/2014 02:37 AM - Gerrit Code Review Bot
Gerrit received a related patchset '1' for Issue #1558.
Uploader: Roland Schulz (roland@rschulz.eu)
Change-Id: Ie861224bcc61df2f26025d1dd106bcab827308bb
Gerrit URL: https://gerrit.gromacs.org/3801

#17 - 07/13/2014 05:42 AM - Roland Schulz
- Status changed from New to Fix uploaded

#18 - 07/15/2014 05:48 AM - Teemu Murtola
- Category set to testing
- Assignee set to Roland Schulz
- Target version set to 5.0.1

#19 - 08/10/2014 08:20 PM - Roland Schulz
- Status changed from Fix uploaded to Closed

#20 - 08/16/2014 01:34 AM - Erik Lindahl
- Subject changed from analysisdata test fail with ICC 14 to analysisdata test fail on several compilers
- Status changed from Closed to Accepted

Reopening, since this occurs both on Pathscale and maybe Fujitsu/PGI too.

Only checking for a specific Intel version through a predefined macro appears to be a very fragile solution for a bug that occurs with multiple compilers (and possible other future ones).

Could we create a test that makes it possible to detect this during cmake configuration?
I think it will not be worth the effort to try to detect this in CMake, since the detection code will be very tricky. The actual incorrect behavior can be observed only at runtime, but CMake cannot run test binaries in cross-compilation scenarios. So the only way to do this would be to compile a test binary from multiple source files, and run nm on the result to verify that the correct symbols get created. And that verification part dives deep into the actual implementation of the compilers/linkers, so it may anyways require a separate approach for each compiler...

Rather, we could just apply the workaround always, and rather spend the effort in writing a script that detects all occurrences of this bug in the codebase to avoid introducing more. The pattern used in all these cases is quite convenient for complicated test data that gets generated during the test runs.

Right - I'm not worried about the test, but I realized that if/when we introduce similar constructs in the main code, this could potentially result in major silent data corruption. I'll upload a patch and see if Roland has any different opinions.

PS: Any idea for how to create a script that would detect all occurrences in the codebase?

I would put it in the script that is run by make doc-check, as it already has a more or less compete listing of code constructs available (the xml parsing script potentially needs to be extended, though, and it may not have all the necessary information to avoid all false positives). An alternative would be something based on libclang, but that may need some additional supporting setup.

- Target version changed from 5.0.1 to 5.0.2
- Status changed from Accepted to Closed