CoCoALib - Design #1572

Use noexcept

29 Jan 2021 20:18 - John Abbott

Status:	Resolved	**Start date:**	29 Jan 2021
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	70%
Category:	Improving	**Estimated time:**	0.00 hour
Target version:	CoCoALib-0.99880	**Spent time:**	5.60 hours

Description

Scott Meyers recommends using **noexcept** where it fits naturally.

Check to see where we can use it.

Related issues:

Related to CoCoALib - Design #1225: Move to C++14 (skipping C++11)	**In Progress**	**06 Sep 2018**
Related to CoCoALib - Design #1787: Iterator design: compatible with C++STL? ...	**In Progress**	**07 Mar 2024**

History

#1 - 29 Jan 2021 20:18 - John Abbott

- Related to Design #1225: Move to C++14 (skipping C++11) added

#2 - 29 Jan 2021 20:19 - John Abbott

Maybe not many functions are applicable *e.g.*

- the fn must not call a fn which is not noexcept (with args which could cause an exception to be thrown),
- the fn must not alloc heap memory (even indirectly), *e.g.* I/O (of CoCoA objects) could throw an exception
- the fn must not do arg checking which can throw an exception (via CoCoA_THROW_ERROR, see next entry)
- **SPECIAL CASE** a fn which could throw only via CoCoA_ASSERT), may be declared noexcept

Here is a list of the files which have already been checked: (update when new files are done)

- **bool3.H**
- **BigInt.H**
- **BigIntOps.H** and .C
- **MachineInt**
- **combinatorics**
- **convert** (some are not noexcept because of poor impl)
- **degree** (many are not because of arg checks)
- **DivMask**
- **error**
- **exception**
- **VerificationLevel**
- **NumTheory-XYZ** (all files done)
- **SignalWatcher**
- **SmallFp** (hardly any since they are inline)
- **time**
- **ULong2Long**
- **utils**
- **VectorOps**
- **end-of-list**

These files are (probably) not applicable:

- interrupt
- **utils-gmp**
- essentially all the rest
- **end-of-list**

#3 - 04 Feb 2021 14:49 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 50

I am not sure whether a function which has a local static variable can be noexcept.
I suppose, so long as the local static object does not require any heap storage, it can be noexcept.
One problem is that it would be quite hard to test this... (*i.e.* I have no idea how to do it).

A quick search on internet produced nothing clearly helpful.
Advice is welcome.

#4 - 06 Feb 2021 22:00 - John Abbott

- Status changed from In Progress to Resolved

- Assignee set to John Abbott

- % Done changed from 50 to 70

I have checked all files which I think might benefit from **noexcept**, and have changed most functions which can be changed.

- I have decided that it probably makes no sense to declare inline functions (which call no other functions) noexcept, since surely the compiler can see that no exceptions can emanate from the code
- I have **not been consistent** about skipping inline functions (since I had already changed some before thinking that it was probably pointless)
- I am also uncertain about functions which contain local static variables (so have no declared them as noexcept)
- strictly I cannot declare noexcept any function containing **CoCoA_ASSERT**... this is inconvenient!

Not really sure where to obtain advice about the above.

#5 - 08 Feb 2021 15:00 - John Abbott

After discussion with Anna:

- **(A)** a fn which could throw (but only via CoCoA_ASSERT) may be declared noexcept
- **(B)** it is probably still useful to declare inline fns as noexcept, because noexcept is part of the public interface (and gives info to the caller without having to search to see whether the defn is inline).

What happens if a noexcept fn throws an exception via CoCoA_ASSERT?
According to the C++ doc this will trigger a call to std::terminate, which is reasonable after an assertion has failed.
The usual debugging approach of putting a breakpoint in CoCoA::JustBeforeThrowing will still work because no exception has yet been thrown, so terminate won't be called until later.

One caveat: if you use CoCoA-5 which has been compiled with CoCoA_DEBUG set then an assertion failure will cause termination (whereas without noexcept CoCoA-5 should capture the exception and survive). Anyway, CoCoA-5 with debugging active is normally only for testing/debugging.

IDEA PUT ON HOLD
It is possible to declare a fn **noexcept(CoCoA_DEBUGGING_IS_INACTIVE)**, with a fairly obvious meaning.
I have decided not to implement this (for the moment, anyway) because it complicates the source code without bringing any real benefit.

#6 - 05 Mar 2023 18:52 - John Abbott

I have just modified configure so that it defines the CPP macro **CoCoA_DEBUG_MODE**.
This new macro is either **true** or **false**. It could be used inside a **noexcept** declaration.

#7 - 07 Mar 2023 20:36 - John Abbott

Point 1
As in the previous comment: for fns which use CoCoA_ASSERT we can now write something like

```
long MyFunc(...)  noexcept(!CoCoA_DEBUG_MODE) {...}
```

I must go through all the source again to see where this could be added... sigh!

Point 2
When a function is called with a "bad input argument" it can throw an exception, or it can return an "obviously impossible result".
An advantage of the "obviously impossible" return value is that the fn could then be declared noexcept... (maybe, if nothing else inside can throw).

Example what about IsPrime? If its parameter were **unsigned** then we could make it noexcept by returning false for 0 and 1.
Mmmm, what to do?

#8 - 07 Mar 2023 20:38 - John Abbott

- Target version changed from CoCoALib-0.99850 to CoCoALib-0.99880

Postponing to have more time to discuss & think; also there is not enough time to make the checks implied by the last 2 comments.

#9 - 22 Apr 2024 21:22 - John Abbott

Another question is about operator* for iterators: if an error is thrown when "beyond the end" then it cannot be noexcept.
Does this matter?

#10 - 22 Apr 2024 21:22 - John Abbott

- Related to Design #1787: Iterator design: compatible with C++STL? Advancing beyond end? added

#11 - 22 Apr 2024 22:39 - Nico Mexis

According to the C++ standard, a function is either not-throwing or potentially-throwing. The noexcept specifier may only be used on a function that is not-throwing.
After a short search, I also found this question on StackOverflow, which basically asked about the same matter: https://stackoverflow.com/q/33203875
So, I think no, the noexcept specifier cannot be used on the operator* of the iterators.

#12 - 23 Apr 2024 21:57 - John Abbott

What I meant to say in comment 9 above is: if we choose a design for iterators where operator* never throws then we could mark it as noexcept.

Morally operator* is "noexcept" with correct use, but the fn specification also has to cater for incorrect use. Well, I suppose the same applies to many functions which perform arg checking...