Testing, Reproducibility and Random Numbers

General

Software needs testing .. and many tests can be automized with unittests - or integration tests.
The program/test should be reproducible to allow (later) debugging/fixing.

Crash early!

Especially when the program or it's results must be reliable, **crashing early** might be advised!
In this case, additional checks should be enforced and executed. Note, that the C `assert()` is removed in non-debug compiled binaries.

Besides executing explicitly programmed checks, there are also other possibilities:

- setup signal handler, e.g. segmentation fault (SIGSEGV)
- setup floating point traps, see Numeric / Math / Linear Algebra
- compile with automatic checks
  - see https://gcc.gnu.org/onlinedocs/libstdc++/manual/using_macros.html for
    - `_GLIBCXX_DEBUG`
    - `_GLIBCXX_ASSERTIONS`
    - `_GLIBCXX_SANITIZE_VECTOR`
  - see https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html for
    - sanitizers like the prominent AddressSanitizer 'ASan'
    - stack-protector
    - and others

A welcomed side effect of crashing early:
the stack trace and the visible variable contents might reveal the initial cause of the problem
- at least with a much higher probability than after having continued and covered the problem.

Core-Dumps

A core-dump / minidump mechanism might be of interest.
For tracking very rare or specific errors, this mechanism might also get delivered to customers.

Pseudo Random Number Generators (PRNG)

For tests, a huge amount of '*random*' numbers can be generated, from one *seed*-number.
There's no need to have a cryptographically safe PRNG.

Test with defined *seed*-numbers or simply log/save the used seed number.

Algorithms / Libraries

Properties

- size of the *seed* or *state*
  - small size is of interest to use one PRNG per thread
- quick initialization from seed
- speed, for producing next number
- period size - until the sequence repeats

Linear congruential generator (LCG)

LCG is perhaps the simplest (and fastest) algorithm with a small seed; see https://en.wikipedia.org/wiki/Linear_congruential_generator

Fast skipping might be of interest; see https://www.nayuki.io/page/fast-skipping-in-a-linear-congruential-generator

Xorshift

Also a very fast PRNG. Here some links

- https://en.wikipedia.org/wiki/Xorshift
- xoshiro / xoroshiro generators
- http://pracrand.sourceforge.net/
- https://sourceforge.net/projects/gjrand/
- http://simul.iro.umontreal.ca/testu01/tu01.html

Mersenne Twister

This PRNG got hyped in the C++ community for it's ridiculously huge period size - but has a big seed size and it's initialization isn't the fastest. IMHO, in most applications, the huge period size isn't necessary.

Anyway, here a link to a SIMD-oriented implementation: http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/SFMT/index.html

Other Links

- PCG - comparison of Mersenne Twister, Arc4, ..
  - https://www.pcg-random.org/
- 64 bit version of Mersenne Twister
  - http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/emt64.html

        - https://www.reddit.com/r/programming/comments/2momvr/pcg_a_family_of_better_random_number_generators/
        - https://github.com/imneme/pcg-cpp
- https://en.cppreference.com/w/cpp/numeric/random/uniform_int_distribution
- CppCon 2016: Cheinan Marks “I Just Wanted a Random Integer!
    - https://www.youtube.com/watch?v=4_QO1nm7uJs
- CppCon 2016: Walter E. Brown “What C++ Programmers Need to Know about Header <random>”
    - https://www.youtube.com/watch?v=6DPkyvkMkk8
- CppCon 2022: Roth Michaels “Fast, High-Quality Pseudo-Random Numbers for Non-Cryptographers”
    - https://www.youtube.com/watch?v=I5UY3yb0128
    - https://github.com/CppCon/CppCon2022/blob/main/Presentations/Fast-High-Quality-Pseudo-Random-Numbers-CPPCon2022-Roth-Michaels.pdf
- Vectorized (SIMD) generation
    - https://lemire.me/blog/2018/07/23/are-vectorized-random-number-generators-actually-useful/
    - https://forum.juce.com/t/fast-simd-based-random-number-generator-whether-and-how-to-vectorize-random-nextint/44430
- Github
    - https://github.com/imneme/pcg-cpp
    - https://github.com/DEShawResearch/random123
    - https://github.com/Reputeless/Xoshiro-cpp