Enforce storage stability (slides)

Jens Gustedt

Enforce storage stability

ISO/IEC JTC 1/SC 22/WG14 N2756

Jens Gustedt

INRIA – Camus

ICube – ICPS
Université de Strasbourg

https://modernc.gforge.inria.fr/

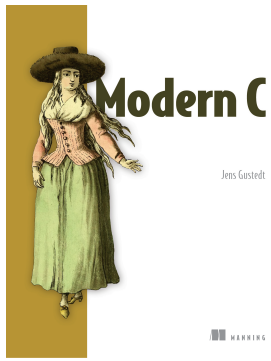

Table of Contents

our failure

C has no commonly understood strategy for initialization

- users often don't understand the corner cases
  - component-wise initialization by assignment is common
  - `memcmp` is the only tool in the C language or library to compare `struct`, but does it work with padding?
  - `atomic_compare_exchange_strong` is based on a combination of `memcpy` and `memcmp`
- implementations have different strategies for incomplete initialization
  - lazy (just use what's there)
  - freeze (produce a value and stick to it)
  - wobbly (unreliably produce a value)
  - crash
  - not even consistent for the same platform, e.g depending on the size of the storage instance

Let's do our homework

Regardless what strategy we decide (or not)

- we should provide reasonable definitions
  - for valid use cases of uninitialized memory (if any)
  - ensure that all C features are consistent
    - `memcmp`
    - `atomic_compare_exchange_strong`
  - provide clear guidelines for implementations that handle the common cases

Reasonable expectations

A store operation conveys an intent.

A use of an object after a store operation is intentional and should not be undefined per se.

Expect an object to be usable

- whenever an lvalue is assigned to, even partially
- whenever a byte is written into storage

For a usable object expect

- multiple evaluations lead to the same value

Systematic pitfalls

Completely uninitialized objects

- access may crash
- evaluation may unbehave

Large storage instances

- may be implemented page-wise
- uninitialized pages may unbehave

Table of Contents

Proposed design principles

observation stability

Every observable change to the abstract state shall be either induced by a store operation, by an IO operation or an external events.

modification locality

A valid store operation shall have no observable effect to storage outside the storage instance that is operand to the operation.

storage stability

Consecutive valid calls to `memcpy` and `memcmp` without intermediate store shall be consistent.

Table of Contents

initialized objects and storage instances

Value initialization (semantic initialization)

An object is value-initialized if any of the following holds:

- *It stems from an object definition with an explicit or implicit initializer.*
- *It is a function parameter or a value capture.*
- *It (or an object that contains it) has been the target of an assignment operation.*
- *It has an aggregate type and all its elements or members have been value-initialized.*
- *It is not an array type and a pointer to it has been argument to a call of one of the library functions that store into their pointed-to parameter.*

initialized objects and storage instances

Byte initialization (storage initialization)

It is byte-initialized if it is not atomic and if any of the following holds:

- *Its storage instance is allocated with the `calloc` library function.*
- *It has scalar, structure or union type and at least one representation byte has a stored value.*
- *It has structure or union type and at least one of its members has been value-initialized.*
- *It is a member of a byte-initialized structure or union.*
- *It is the representation byte of a value-initialized or byte-initialized object.*

initialized objects and storage instances

Initialization

An object is initialized if it is value-initialized or byte-initialized, otherwise it is uninitialized.

Notes

- A structure can be initialized by writing one byte into its representation.
- An array with more than one element cannot be initialized by writing one byte into its representation.

Rationale

- Structures have always been regarded as a whole.
  - They have a compile time fixed size.
- Arrays have no value and size information is futile.
- Representations of padding bits and bytes shall be stable.

initialized objects and storage instances

Stable storage instance

A storage instance . . . is stable . . . if at least one of the following conditions holds . . .

1. As a whole it represents an object that has been initialized.
2. At least one representation byte has been initialized and the macro `__STDC_STORAGE_GRANULARITY__`
   - is undefined or expands to zero, or,
   - is strictly positive and the size of the storage instance is smaller or equal.

Table of Contents

Normative change

Stable storage instances represent stable values . . .

As long as a stable storage instance is not the target of any further write operation, all read operations of a given lvalue that is represented by the storage instance result in the same value . . .

. . . and values are consistent

. . . and reads to different lvalues that have overlapping representations by the storage instance result in consistent values.