Introduce the term storage instance

Jens Gustedt

ISO/IEC JTC 1/SC 22/WG14 **N 2328**

December 9, 2018 **v 1**

Introduce the term storage instance

Modification request for C2x

Jens Gustedt
INRIA and ICube, Université de Strasbourg, France

There is a lack of terminology to describe the entity that is reserved and released by either an allocation (`malloc`/`free`) or by the definition of a variable or compound literal.

1. INTRODUCTION

The current revision of the C standard has no precise words to describe the maximal area of storage that is either obtained from

— allocating through `malloc`/`realloc`/`aligned_alloc` (allocated storage duration)
— instantiations of objects through the encouter of definitions (all other storage durations).

Already the term *storage duration* suggest that the "something" that is created through such an event would be "storage", but there are no precise words for it. In the contrary these beast are called very differently in different places. Some citations from the C standard:

— ... a *new instance of the object* is created each time ..
— ... *Allocated objects* have no declared type. ...
— ... that would not make the structure *larger than the object* being accessed ...
— The value of a pointer that refers to *space* deallocated by a call to the `free` or `realloc` function ...
— ... functions return a null pointer or a pointer to *an allocated object* ...
— The `longjmp` that returns control back to the point of the `setjmp`-invocation might cause *memory associated* with a variable length array object to be squandered.

The terms "space", "storage", "memory", and (maximal)"object" describing basically all the same thing.
Especially the use of the term "object" is unfortunate and produces a lot of confusion. There is a footnote

> *When referenced, an object can be interpreted as having a particular type.*

So it seems implied that all objects have a type. Also objects can have subobjects, e.g the members of a structure object are themselves objects.

2. FIX TERMINOLOGY

We should better make clearer distinctions between terms:

— A data storage facility that is allocated (or instantiated) and that by itself bares no type, is a different concept than the object that is represented by it.
— An object in the abstract state machine is a different concept than the storage (memory, address, space ...) that is used to represent it. An object should always have type, storage (memory, address, space ...) has not. Some objects have no known address (register variables).
— A memory location (address) that is used to identify an object or storage space is something else than the object or storage space itself. This is explicitly mentioned by the standard: over time, the same address can be reused by several "object instances" (automatic storage duration), "space" (allocated storage duration), and will most likely also occur for thread local objects.

We propose to add the term *storage instance*, as being a "*maximal region of data storage*" in the execution environment that is created when either an allocation is encountered or an object instance is created.
The choice for the term itself (storage instance) stems from the fact that it seemed the easiest to integrate to the closely related concepts of *storage duration* and *storage class*. Also, this ensure a consistent terminology: storage should be the entity that is target of a *store* operation. The *instance* part of the term is important to emphasize that this is an entity that has temporal limits within the execution.

3. SUGGESTED CHANGES

This proposal is only intended to clarify the existing model and not to add any new features. For clarification, realitively few text additions and modifications are needed. They can all be found in the appendix that consists of the relevant pages of diff-mark to C17. *Beware*, that these pages are *not contiguous*.

3.1. Text additions

We propose two text additions, that introduce the term and put it into context:

now 3.19:. A definition of the term with two notes that clarify where "storage instances" come from, their relative placement and accessibility by different threads.
6.2.6.1, p1:. This puts storage instances into their context (object representations) and states their basic properties, in particular that most of them can be viewed as 'unsigned char' array. A footnote clarify the absence of any induced positioning between any storage instances.

3.2. Text modifications

With these additions there are two types of text modifications remaining, namely some that really only are replacements of terms (space → storage instance, e.g.) and others that are completely reformulated. For the latter we have:

6.2.4:. Here the paragraphs 1 and 2 are swapped, and the now paragraph 2 is a bit sharpened.
6.2.6.1, p3:. Object representations can now be introduced a bit more clearly.
6.5, p18:. Clarify the extend of a flexible array member.
7.22.3:. Mostly just a consequent use of the new terminology. Clarification that `realloc` does a byte-wise copy of the initial part. (And thus implicitly preserves effective type.)

Appendix: pages with diffmarks of the proposed changes

The following page numbers are from the particular snapshot and may vary once the changes are integrated.