# N2788: **Memory layout of union members**

Javier A. Múgica - javier@aerotri.es

July 14th, 2021

## Purpose

To make clear that the different members of a union share their first bytes in memory.

## Problematic

Union types are introduced **6.2.5. Types:** "A union type describes an overlapping nonempty set of member objects, each of which has an optionally specified name and possibly distinct type." Soon after, in **6.2.6 Representation of types**, it is said:

6 When a value is stored in an object of structure or union type, including in a member object, the bytes of the object representation that correspond to any padding bytes take unspecified values.57) The value of a structure or union object is never a trap representation, even though the value of a member of the structure or union object may be a trap representation.

7 When a value is stored in a member of an object of union type, the bytes of the object representation that do not correspond to that member but do correspond to other members take unspecified values.

But it is not specified the way the members overlap, not even that they must actually overlap.

Proceeding with the sections of the standard, we find at the description of the '.' and '->' operators when applied to structures and unions, in **6.5.2.3**:

3 A postfix expression followed by the **.** operator and an identifier designates a member of a structure or union object. The value is that of the named member,105)

105)If the member used to read the contents of a union object is not the same as the member last used to store a value in the object, the appropriate part of the object representation of the value is reinterpreted as an object representation in the new type as described in 6.2.6 (a process sometimes called "type punning"). This might be a trap representation.

6 One special guarantee is made in order to simplify the use of unions: if a union contains several structures that share a common initial sequence (see below), and if the union object currently contains one of these structures, it is permitted to inspect the common initial part of any of them anywhere that a declaration of the completed type of the union is visible. Two structures share a common initial sequence if corresponding members have compatible types (and, for bit-fields, the same widths) for a sequence of one or more initial members.

This implies in no way that the structures' initial sequence must share the memory. Further, footnote 105 is wrong, since in 6.2.6 it is described what happens to bytes that do not correspond to other members, not to those that do "correspond" to other members and it is not specified what "the appropriate part of the object representation" is, likely because the inital-segment-overlap layout is assumed.

Later on, in **6.5.8 Relational operators:**

"All pointers to members of the same union object compare equal."

Since a pointer to any data *could* also store the pointer to the including union, if any, and even the union layout, this still does not imply that they must share the memory. More generally, a pointer is an abstract entity that could store an arbitrary amount of information. Therefore, any requirement on pointers to union members, even imposing equality of pointers which are copied to other pointer types (say, a void* pointer) and later on cast to the original member type, or any other combination, does not imply the sharing of the initial bytes by a union's members.

Next, in **6.7.2.1 Struct and union specifiers:**

18 The size of a union is sufficient to contain the largest of its members. The value of at most one of the members can be stored in a union object at any time. A pointer to a union object, suitably converted, points to each of its members (or if a member is a bit-field, then to the unit in which it resides), and vice versa.

To this, the previous comment applies.

And that's all, if I didn't miss another relevant information for the layout of unions.

---

One may object that my interpretation of pointers' possibilities is too loose. But if my interpretation is wrong, this can only be shown to be the case by referring to the relevant points of the standard dealing with pointers. So, in the end, one would derive that "union members have their initial bytes shared" as one finds the proof of a mathematical theorem.

Further, it seems strange that the description of structures' and unions' layout appears under "**Declarations > Type specifiers.** This is not the case for any other type, other than bit-fields. I would not expect it there. Also, that the behaviour of pointers to union members appears not when talking about pointers or the representation of types, but at **"Relational operators"** and as a footnote in the section **"Struct and union specifiers".** This paper does not address these other issues, but as consequence of the proposed change footnote 105 is fixed, as well as its being just a footnote.

## Proposed wording

Replace paragraph 7 in **6.2.6 Representation of types:**

~~When a value is stored in a member of an object of union type, the bytes of the object representation that do not correspond to that member but do correspond to other members take unspecified values.~~

by

If two members A and B of an object of union type have sizes *m* and *n* respectively, with $m \leq n$, the first *m* bytes of the object representation of B share the same memory space as the *m* bytes of the object representation of A, in the same order, and are said to *overlap*. For this reason, when a value is stored in a member of an object of union type, the bytes of the object representation of any other member that overlap with bytes of the object representation of the stored member take the same values. The bytes that do not overlap take unspecified values.