Designate the address of a struct/union's first member as the address of the struct/union.

Document: n3728

Author: Ryan Karl

Date: 2025-10-19

Changes: Introduce a normative guarantee that, within any structure object, the address of the structure and the address of its first member are equal.

Analysis:

The C standard guarantees pointer representation uniformity in section 6.2.5 and discusses pointers in 6.3.2.3 but does not explicitly state that a pointer to a struct and a pointer to its first member must compare equal. This omission creates ambiguity: most implementations assume zero offset for the first member, but the text does not require it. Formalizing this rule removes any doubt and codifies an expected practice.

Recommendation:

Augment the standard with a clear discussion of this expected behavior: when a structure or union object has a first member, the pointer to the structure may be converted to a pointer to the first member type and back without change; the two pointers compare equal.

Suggested Rewording:

…
6.2.5 Types
…
30 A pointer to void shall have the same representation and alignment requirements as a pointer to a character type. Similarly, pointers to qualified or unqualified versions of compatible types shall have the same representation and alignment requirements. All pointers to structure types shall have the same representation and alignment requirements as each other. All pointers to union types shall have the same representation and alignment requirements as each other. Pointers to other types may not have the same representation or alignment requirements.

…
6.3.3.3 Pointers
1 A pointer to void may be converted to or from a pointer to any object type. A pointer to any object type may be converted to a pointer to void and back again; the result shall compare equal to the original pointer.
2 For any qualifier q, a pointer to a non-q-qualified type may be converted to a pointer to the q-qualified version of the type; the values stored in the original and converted pointers shall compare equal.
3 An integer constant expression with the value 0, such an expression cast to type void *, or the predefined constant nullptr is called a null pointer constant. If a null pointer constant or a value of the type nullptr_t (which is necessarily the value nullptr) is converted to a pointer type, the resulting pointer, called a null pointer, is guaranteed to compare unequal to a pointer to any object or function.

4 Conversion of a null pointer to another pointer type yields a null pointer of that type. Any two null pointers shall compare equal.
5 An integer may be converted to any pointer type. Except as previously specified, the result is implementation-defined, may not be correctly aligned, may not point to an entity of the referenced type, and can produce an indeterminate representation when stored into an object.
6 Any pointer type may be converted to an integer type. Except as previously specified, the result is implementation-defined. If the result cannot be represented in the integer type, the behavior is undefined. The result is not required to be in the range of values of any integer type.
7 A pointer to an object type may be converted to a pointer to a different object type. If the resulting pointer is not correctly aligned for the referenced type, the behavior is undefined. Otherwise, when converted back again, the result shall compare equal to the original pointer. When a pointer to an object is converted to a pointer to a character type, the result points to the lowest addressed byte of the object. Successive increments of the result, up to the size of the object, yield pointers to the remaining bytes of the object.
8 A pointer to a function of one type may be converted to a pointer to a function of another type and back again; the result shall compare equal to the original pointer. If a converted pointer is used to call a function whose type is not compatible with the referenced type, the behavior is undefined.
9 Within a structure object that declares at least one member, the address of the structure object and the address of its first member shall compare equal. Converting a pointer to the structure type to a pointer to the first member type and back yields the original pointer value.

…

Acknowledgments: I thank the UB Study group for their helpful comments.