Making Offsetof a Core Language Feature

Document: n3958
Author: Martin Uecker
Date: 2026-09-06

Charter Principles

- Codify existing practice to address evident deficiencies
- Avoid ambiguities
- Ease library independence

Rationale

The goal of this paper is to remove undefined behavior in offsetof and to standardize existing practice. Originally any invalid member-designator was undefined behavior, but this was not clear leading to DR 496. The response to issue 0496 already clarified that the member-designator is not restricted to a single identifier which was the intent since C89 (item 10 in the May 20, 1988 list of changes between the second and third public review drafts of C89 is "The offsetof macro has been generalized to allow more than just a simple identifier as its second argument.").[1] In N2350 the case of declaring new types was made undefined behavior. Before publication of C23 the scope of this undefined behavior was reduced by declaring types was made well-defined (DE-137). There is still explicit undefned behavior when there is a comma in the type-name and when the member-designator references a bit-fields. During previous discussions it was pointed out that offsetof should be a core language feature to address this. As many compilers already have a built-in to better support offsetof, this standardizes existing practice.

Another situation where there is undefined behavior occurs if the indices in array subscription operators are not integer constant expressions, but this is commonly supported by compilers as a conforming language extension. Hence, it is proposed to allow designators that include array subscriptions that contain expressions that are not integer-constant expression. In this case, the result of offsetof is then also not an integer constant expression. There may then still be undefined behavior related to out-of-bands array subscription as this seems difficult to avoid at this time.

Example (existing support for variable indices):

https://godbolt.org/z/Y9oMaxGPT

```
#include <stddef.h>

#define MAX(a, b) ((a) > (b) ? (a) : (b))
#define fam_sizeof(T, i) MAX(sizeof(T), offsetof(T, buf[i]))

struct foo { int N; char buf[]; };

size_t foo_sizeof(int n)
{
      return fam_sizeof(struct foo, n);
}
```

1 Joseph Myers, personal communication.

Example (existing compiler builtin):

https://godbolt.org/z/nq6e1erPE

```
int g(int i)
{
	return __builtin_offsetof(struct foo { int x, y; int buf[10]; }, buf[i]);
}
```

References

https://www.open-std.org/jtc1/sc22/wg14/issues/c11c17/issue0496.html
https://www.open-std.org/jtc1/sc22/wg14/www/docs/n2350.htm

Acknowledgment: Joseph Myers and Jens Gustedt for comments on an early draft.

Wording (relative to N3886 + N3909 “discarded”)

6.4.2 Keywords
Syntax
1 keyword: one of
… offsetof ...

Semantics

2 The previously listed tokens (case sensitive) are reserved (in translation phases 7 and 8) for use as keywords except in an attribute token, and shall not be used otherwise.
3 Table 6.1 provides alternate spellings for certain keywords. These can be used wherever the keyword can.[46)]

Table 6.1 — Keywords and their spellings

The spelling of these keywords, their alternate forms, and of false, ~~**and**~~ true, **and offsetof** inside expressions that are subject to the # and ## preprocessing operators is unspecified.[47)]

6.5.4 Unary operators
6.5.4.1 General

Syntax
1 unary-expression:
postfix-expression
++ unary-expression
-- unary-expression
unary-operator cast-expression
_Countof unary-expression
_Countof (type-name)
sizeof unary-expression
sizeof (type-name)
alignof (type-name)
offsetof (type-name , member-designator)
static-assertion

member-designator:
identifier designator-list$_{opt}$

designator-list:
designator
designator-list designator

designator:
[expression]
. identifier

6.5.4.5 The sizeof, _Countof, ~~and~~ alignof, and offsetof operators

Constraints

2 For the offsetof operator, the type name and member designator shall be such that the expression

```
( type-name ){ } . member-designator
```

fulfills the constraints of the member array subscripting and . operators. The member-designator shall not designate a bit-field.

Semantics

7 The `offsetof` operator yields the offset in bytes from the beginning of any object of type `type-name` to the subobject designated by the member-designator. The `offsetof` expression discards the `type-name` operand. If the expression

```
& ( static type-name ){ } . member-designator
```

is an address-constant, the offset expression is an integer constant expression and all size expressions possibly contained in the `member-designator` are discarded. Otherwise, the size expressions are evaluated and there is undefined behavior if any of the array subscriptions is out of range as described in 6.5.3.2. If there is a flexible array member (cf. 6.7.3.2), it is assumed that the object has SIZE_MAX for determination of an out of range condition.

~~6~~8 The value of the result of all operators is implementation-defined, and its type (an unsigned integer type) is size_t, defined in <stddef.h> (and other headers).

11 EXAMPLE 5 The offsetof operator can be used with a designator-list including array subscription with a non-constant index.

```
size_t buf_offsetof(int i)
{
      return offsetof(struct { size_t N; char buf[]; }, buf[i]);
}
```

6.7.11 Initialization

designation:
    designator-list =

~~**designator-list:**~~
    ~~**designator**~~
    ~~**designator-list designator**~~

~~**designator:**~~
    ~~**[expression]**~~
    ~~**. identifier**~~

10 If a designator has the form

[~~**constant**~~ expression]

then the current object (defined below) shall have array type and the expression shall be an integer constant expression. If the array is of unknown size, any nonnegative ~~**value**~~ **integer constant expression** is valid.

~~**7.21 Common definitions <stddef.h>**~~

~~**offsetof(type, member-designator)**~~

~~**which expands to an integer constant expression that has type size_t, the value of which is the offset in bytes, to the subobject (designated by member-designator), from the beginning of any object of type type. The type and member designator shall be such that given static type t; then the expression &(t. member-designator) evaluates to an address constant. If the specified type name contains a comma not between matching parentheses. or id the specified member is a bit-field, the behavior is undefined..**~~

B.21 Common definitions <stddef.h>

~~**offsetof(type, member-designator)**~~

J.2 Undefined behavior

(XX) An array subscript is out of range when evaluating the offsetof operator.

~~**(120) The type parameter of an offsetof macro contains a comma not between matching parentheses (7.22).**~~

~~**(123) The member-designator parameter of an offsetof macro is an invalid right operand of the . operator for the type parameter, or designates a bit-field (7.22).**~~