Author: Javier A. Múgica

Date: 2026 - jan - 14

Array constant expressions

Background

If an expression is an array constant, selection of an element from it via an ICE index does not yield a constant expression:

```
constexpr int p[3]= {2,3,5,7,11};
float y[p[3]];  //p[3] is not a constant expression; y not a fixed size array
```

This is in contrast to selecction of a member from a structure constant:

```
struct A{int n,ω,φ;};
constexpr struct A s={10,2,4};
float y[s.φ];  //s.φ is a constant expression; y is a fixed size array
```

The reason is that the operation of array subscripting was defined until recently through conversion of the array to pointer and a subsequent pointer indirection.

The paper **N3360** proposed changing this situation, defining array subscripting directly as selecting an element from the array. In line with this, it proposed allowing array subscripting as a further derivation on named and compound literal constants. Although the proposal was welcomed, the committee preferred to keep this array-subscripting part of the same outside from it. The reduced version was approved as in **N3517** between the Graz and Brno 2025 meetings. The time has come, therefore, to consider this change to array subscripting.

The wording chosen

The wording proposed in **N3360** was:

6 A compound literal with storage-class specifier **`constexpr`** is a *compound literal constant*, as is a postfix expression that applies the `.` member access operator to a compound literal constant of structure or union type or the `[]` array subscripting operator to a compound literal constant of array type with an integer constant expression less than the length of the array as subscript, even recursively. A compound literal constant is a constant expression with the type and value of the unnamed object.

7 ... is a named constant, as is a postfix expression that applies the . member access operator to a named constant of structure or union type or the `[]` array subscripting operator to a named constant of array type with an integer constant expression less than the length of the array as subscript, even recursively. ...

The first impression upon reading this is that the constraint on the subscript is missing "not negative and". The justification for this in that proposal was that this constraint is present elsewhere for any indexing of an array of known length by an integer constant expression. This is not apparent in the subclause on constant expressions. Further, we prefer that an expression with a negative index not be considered a constant expression, rather than being a constant expression that breaks a constraint.

For this reason we have changed that wording to *where the subscript is an integer constant expression within the bounds of the array*. We believe there is no room for ambiguity here: an index past the last elements is not within bounds, as the very expression "past the last element" implies.

N3360 also modified the second paragraph for address constant expressions from

12 The array-subscript `[]` and member-access `->` operator, the address `&` and indirection `*` unary operators, and pointer casts can be used in the creation of an address constant, but the value of an object shall not be accessed by use of these operators.

to

13 Any constant expression can be used as operand in the creation of an address constant. Additionally, a non-constant expression formed with the array-subscript `[]` and member-access `->` operator, the address `&` and indirection `*` unary operators, and pointer casts can be used in the creation of an address constant, but then the value of an object shall not be accessed by use of these operators.

The original wording allows expression like

```
&str.a          &arr[1]
```

and the one here below at the left, but precludes the other two:

```
&ptr[str.a]     &ptr[arr[1]]     &ptr[E->a]
```

even if `E` always points (by the way the expression `E` is constructed) to the same constant object. The second and third ones are precluded but the first one is not because of the inclusion of `[]` and `->`, but the omission of `.`, in "The array-subscript `[]` and member-access `->` operator, the address `&` and indirection `*` unary operator". Thus, the effect of that paragraph is not to enlarge the class of address constants, but to restrict it. The operations listed there other than `[]`: `->`, `&` and `*` are pointer related. We can understand therefore the presence of `[]` alongside: it used to be a pointer operation too. Therefore, the solution is simpler than the wording presented in **N3360:** just remove the array-subscript `[]` from that list.

The consequence is that the middle expression above, `&ptr[arr[1]]`, becomes allowed, in agreement with `arr[1]` having become a constant expression.

Proposed wording

Blue text is to be added; grey text to be removed.

6.6.1 Constant expressions, general

5 A compound literal with storage-class specifier **`constexpr`** is a *compound literal constant*, as is a postfix expression that applies the `.` member access operator to a compound literal constant of structure or union type or the `[]` array subscripting operator to a compound literal constant of array type where the subscript is an integer constant expression within the bounds of the array, even recursively. A compound literal constant is a constant expression with the type and value of the unnamed object.

6 An identifier that is:

— an integer constant expression,

— a predefined constant, or

— declared with storage-class specifier **`constexpr`** and has an object type,

is a *named constant*, as is a postfix expression that applies the `.` member access operator to a named constant of structure or union type or the `[]` array subscripting operator to a named constant of array type where the subscript is an integer constant expression within the bounds of the array, even recursively. [...]

10 The array-subscript `[]` and member-access `->` operator, the address `&` and indirection `*` unary operators, and pointer casts can be used in the creation of an address constant, but the value of an object shall not be accessed by use of these operators.[120)]

Alternative

If the term "within the bounds of the array" is disliked it may be replaced (two instances) by

> that is less than the array length and not negative

We have already expressed our opinion on the unambiguity of the first one.