# n2529 - New pointer-proof keyword to determine array length

**Proposal for the upcoming C2X standard**
**Submitter: Xavier Del Campo Romero**
**Submission date: 2020-06-04**

**Problem**

Extracting the number of elements in an array is a typical operation on C programs, as relying on magic constants or macros is very error-prone and will cause undefined behavior if the size of the array is somehow modified.

Tipically, most developers use the following operation to determine the number of elements inside an array, usually via a user-defined macro:

```
sizeof (x) / sizeof *(x)
```

However, there is a well-known issue that occurs when a pointer is (most times accidentally) used instead of an array. Despite the similarities between pointers and arrays, a pointer will much likely return an unexpected value when this construct is used, creating a silent error that can be only detected at run-time. For example:

```
#include <stddef.h>

 #define ARRAY_SIZE(x) sizeof (x) / sizeof *(x)

 static void foo(int *const arr) {
     for (size_t i = 0 ; i < ARRAY_SIZE(arr); i++) {
         /* sizeof *int / sizeof int = 1, not 6. */
         printf("arr[%zu] = %d\n", i, arr[i]);
     }
 }

 int main(const int argc, const char *argv[]) {
     int arr[] = {1, 2, 3, 4, 5, 6};

     foo(arr);
     return 0;
 }
```

By using C11 features and GNU extension *typeof*, the *ARRAY_SIZE* macro can be improved so it returns a compile-time error when a pointer is given instead of an array:

```
#define ARRAY_SIZE(a)  \
    _Generic(&(a), \
        typeof (*a)**: (void)0, \
        typeof (*a)*const *: (void)0, \
        default: sizeof (a) / sizeof ((a)[0]))
```

If this macro was used on the example above, the following compile-time error would trigger:

```
$ gcc lengthof.c -std=gnu11
 lengthof.c: In function 'foo':
 lengthof.c:7:30: error: void value not ignored as it ought to be
          typeof (*a)*const *: (void)0,                \
                               ^~~~~~~
 lengthof.c:11:29: note: in expansion of macro 'ARRAY_SIZE'
      for (size_t i = 0 ; i < ARRAY_SIZE(arr); i++) {
                              ^~~~~~~~~~
```

However, error description is vague and confusing to the developer, and the GNU extension typedef is not part of the standard, as of C11.

**Proposal**

Since there is no portable, pointer-proof way to determine the number of elements of an array, this document suggests adding a new keyword to the C standard that aims not to introduce any breaking changes to existing code.

According to the standard, symbol names preceded by a leading underscore and a capital letter or another underscore are reserved for the implementation. Therefore, the *_Lengthof* operator is suggested, aiming to provide the same functionality as the *ARRAY_SIZE* macro above, while providing better diagnostic messages when a pointer is accidentally given.

**Examples**

The example above has been modified by introducing this new *_Lengthof* operator:

```
#include <stddef.h>
#include <stdio.h>

 int main(const int argc, const char *argv[]) {
     int arr[] = {1, 2, 3, 4, 5, 6};

     for (size_t i = 0 ; i < _Lengthof arr; i++) {
         printf("arr[%zu] = %d\n", i, arr[i]);
     }

     return 0;
 }
```

Therefore, the following modified example must produce a diagnostic error because a pointer is given:

```
#include <stddef.h>
 #include <stdio.h>

 static void foo(int *const arr) {
     for (size_t i = 0 ; i < _Lengthof arr; i++) { /* Diagnostic message here. */
         printf("arr[%zu] = %d\n", i, arr[i]);
     }
 }

 int main(const int argc, const char *argv[]) {
```

```
    int arr[] = {1, 2, 3, 4, 5, 6};

    foo(arr);
    return 0;
 }
```

**Usage**

Similarly to the *sizeof* operator, the *_Lengthof* operator returns a value of type *size_t* that is resolved at compile-time, unless using *_Lengthof* on variable-length arrays, where complexity is O(1), as in *sizeof*. Except from this latter case, *_Lengthof* returns an integer constant that can be used with other constructs such as *_Static_assert*. For example:

```
int main(const int argc, const char *argv[]) {
    int arr[] = {1, 2, 3, 4, 5, 6};
    _Static_assert (_Lengthof arr == 6);
    return 0;
 }
```

**Reason behind this proposal**

This new keyword introduces a portable solution that provides better safety than the *sizeof*-based one, which surely has introduced along the years silent bugs into many applications and caused confusion and frustration among developers. The final objective of this proposal is therefore to increase safety on applications written in C without extra memory footprint.