API05-C. Use conformant array parameters

Traditionally, C arrays are declared with an index that is either a fixed constant or empty. An array with a fixed constant index indicates to the compiler how much space to reserve for the array. An array declaration with an empty index is an incomplete type and indicates that the variable references a pointer to an array of indeterminate size.

The term *conformant array parameter* comes from Pascal; it refers to a function argument that is an array whose size is specified in the function declaration. Since C99, C has supported conformant array parameters by permitting array parameter declarations to use extended syntax. Subclause 6.7.6.2, paragraph 1, of C11 [ISO/IEC 9899:2011] summarizes the array index syntax extensions:

> *The [and] may delimit an expression or *. If they delimit an expression (which specifies the size of an array), the expression shall have an integer type. If the expression is a constant expression, it shall have a value greater than zero.*

Consequently, an array declaration that serves as a function argument may have an index that is a variable or an expression. The array argument is demoted to a pointer and is consequently not a variable length array (VLA). Conformant array parameters can be used by developers to indicate the expected bounds of the array. This information may be used by compilers, or it may be ignored. However, such declarations are useful to developers because they serve to document relationships between array sizes and pointers. This information can also be used by static analysis tools to diagnose potential defects.

```
int f(size_t n, int a[n]);  /* Documents a relationship between n and a */
```

Standard Examples

Subclause 6.7.6.3 of the C Standard [ISO/IEC 9899:2011] has several examples of conformant array parameters. Example 4 (paragraph 20) illustrates a variably modified parameter:

```
void addscalar(int n, int m, double a[n][n*m+300], double x);

int main(void) {
  double b[4][308];
  addscalar(4, 2, b, 2.17);
  return 0;
}

void addscalar(int n, int m, double a[n][n*m+300], double x) {
  for (int i = 0; i < n; i++)
  for (int j = 0, k = n*m+300; j < k; j++)
    /* a is a pointer to a VLA with n*m+300 elements */
    a[i][j] += x;
}
```

Example 4 illustrates a set of compatible function prototype declarators

```
double maximum(int n, int m, double a[n][m]);
double maximum(int n, int m, double a[*][*]);
double maximum(int n, int m, double a[ ][*]);
double maximum(int n, int m, double a[ ][m]);
```

Noncompliant Code Example

This code example provides a function that wraps a call to the standard `memset()` function and has a similar set of arguments. However, although this function clearly intends that p point to an array of at least n chars, this invariant is not explicitly documented.

```
void my_memset(char* p, size_t n, char v) {
  memset( p, v, n);
}
```

Noncompliant Code Example

This noncompliant code example attempts to document the relationship between the pointer and the size using conformant array parameters. However, the variable n is used as the index of the array declaration before n is itself declared. Consequently, this code example is not standards-compliant and will usually fail to compile.

```
void my_memset(char p[n], size_t n, char v) {
  memset( p, v, n);
}
```

Compliant Solution

This compliant solution declares the `size_t` variable n before using it in the subsequent array declaration. Consequently, this code complies with the standard and successfully documents the relationship between the array parameter and the size parameter.

```
void my_memset(size_t n, char p[n], char v) {
  memset(p, v, n);
}
```

Exceptions

API05-C-EX0: The extended array syntax is not supported by MSVC. Consequently, C programs that must support Windows need not use conformant array parameters. One option for portable code that must support MSVC is to use macros:

```
#include <stddef.h>

#if defined (_MSC_VER)
  #define N(x)
#else
  #define N(x)  (x)
#endif

int f(size_t n, int a[N(n)]);
```

Risk Assessment

Failing to specify conformant array dimensions increases the likelihood that another developer will invoke the function with out-of-range integers, which could cause an out-of-bounds memory read or write.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
API05-C	High	Probable	Medium	**P12**	**L1**

Bibliography

[ISO/IEC 9899:2011]	Subclause 6.7.6.2, "Array Declarators" Subclause 6.7.6.3, "Function Declarators (Including Prototypes)"