Name

n3327 − VLA is a misnomer (rebuttal to n3187)

Category

Terminology

Author

Alejandro Colomar Andres; maintainer of the Linux man−pages project.

Cc

Martin Uecker
Jens Gustedt
"Ballman, Aaron"
Joseph Myers
Chris Bazley
Jakub Åukasiewicz
Alex Celeste
JeanHeyd Meneide
Ville Voutilainen

History

n3327 v1; 2024-09-01.

Synopsis

"length" in "variable length array" actually refers to the size in bytes. For consistency and avoiding ambiguity, we should rename that term as "variable size array".

Problem description

An array has two properties that are related, and can be confused with one another:

- number of elements
- size in bytes

The terms above are the technical way to refer to those properties. However, the standard often uses colloquial terms, and often results in using them in a way that conflicts with those two technical terms.

This has ultimately resulted in standardizing the term "variable length array", which probably derivated from colloquial use, where such arrays were mostly unidimensional, with an element of a fundamental type. Actual arrays can have elements of aggregate types, which themselves can be arrays, and using non-standard extensions, one can even declare an array of a struct type which itself contains a VLA. The term VLA englobes all such arrays, and not only those that have a variable number of elements.

The standard definition of VLA is in N3301::6.7.7.3p4:

```
If the size is an integer constant expression
// size above refers to "number of elements"
and the element type has a known constant size,
the array type is not a variable length array type;
otherwise,
the array type is a variable length array type.
```

Which one can translate to the following code:

```
#define is_vla(a)  (is_array(a) && !__builtin_constant_p(sizeof(a)))
```

Ironically, it is the variability of the size that determines if an array is a "variable length array".

This inconsistent terminology can cause confusion.

Proposal description

Rename "variable length array" to "variable size array". This proposal does not attempt to modify other uses of size or length, as that would require an agreement on which term to use for the number of elements of an array, which is a different (albeit related) question, and is more controversial at the moment. I plan to do that in a separate paper after this one is accepted. This will keep this proposal small and simple.

There are a very few cases where "variable length array" is misused in the standard to actually refer to an array whose number of elements is variable. This proposal does not attempt to fix those misuses, and will also rename those uses to "variable size array". It is once we fix the problematic VLA term that we will have a chance to come up with a new term for an array whose number of elements is variable. Or maybe since those uses are few, explicit wording can be used without a new specific term. In any case, this prososal does not attempt to do that, and does just one thing.

Future directions

The very few cases where "variable length array" was being used to refer to an array with a variable number of elements should be fixed.

The uses of "size" to refer to the number of elements of an array should be replaced by a different term. "size" should exclusively be used to refer to the result of the *sizeof* operator.

The term used to refer to the number of elements of an array should be uniformized into a single term (to be decided which).

Proposed wording

6.2.4 Storage durations of objects

p6

```
 For such an object that does not have
-a variable length array type,
+a variable size array type,
```

p7

```
 For such an object that does have
-a variable length array type,
+a variable size array type,
```

6.2.5 Types

p28

```
 A type has known constant size if it is complete and
-is not a variable length array type.
-is not a variable size array type.
```

6.2.7 Compatible type and composite type

p3

```
 Otherwise,
-if one type is a variable length array
+if one type is a variable size array
 whose size is specified by an expression that is not evaluated,
 the behavior is undefined.
```

```
 Otherwise,
-if one type is a variable length array
+if one type is a variable size array
 whose size is specified,
 the composite type is
-a variable length array of that size.
+a variable size array of that size.
```

```
 Otherwise,
-if one type is a variable length array
+if one type is a variable size array
 of unspecified size,
 the composite type is
-a variable length array of unspecified size.
+a variable size array of unspecified size.
```

6.5.3.6 Compound literals

p2

```
 The type name shall specify a complete object type or an array of unknown size,
-but not a variable length array type.
+but not a variable size array type.
```

6.5.4.5 The sizeof and alignof operators

p2

```
-If the type of the operand is a variable length array type,
+If the type of the operand is a variable size array type,
 the operand is evaluated;
```

EXAMPLE 3 (p8)

```
 In this example,
-the size of a variable length array
+the size of a variable size array
 is computed and returned from a function:
```

```
-    char b[n+3]; // variable length array
+    char b[n+3]; // variable size array
```

6.5.7 Additive operators

EXAMPLE (p12)

```
 Pointer arithmetic is well defined with
-pointers to variable length array types.
+pointers to variable size array types.
```

6.6 Constant expressions

Footnote 115)

```
 An integer constant expression is required in contexts such as
 the size of a bit-field member of a structure,
 the value of an enumeration constant,
-and the size of a non-variable length array.
-and the size of a non-variable size array.
```

Footnote 118)

```
 For example, in the declaration
-int arr_or_vla[(int)+1.0];,
+int arr_or_vsa[(int)+1.0];,
 while possible to be computed by some implementations
 as an array with a size of one,
 it is implementation-defined whether this results in
-a variable length array declaration
+a variable size array declaration
 or a declaration of an array of
 known constant size
 of automatic storage duration.
```

6.7.2 Storage-class specifiers

EXAMPLE 4 (p20)

```
-int array[K]; // not a VLA
+int array[K]; // not a VSA
```

6.7.3.6 Typeof specifiers

EXAMPLE 5 (p10)

```
-Variable length arrays
+Variable size arrays
 with typeof operators
 performs the operation at execution time rather than translation time.
```

```
-size_t vla_size (int n) {
+size_t vsa_size (int n) {
-    typedef char vla_type[n + 3];
+    typedef char vsa_type[n + 3];
-    vla_type b; // variable length array
+    vsa_type b; // variable size array
     return sizeof(
         typeof_unqual(b)
     ); // execution-time sizeof, translation-time typeof operation
 }

 int main () {
-    return (int)vla_size(10); // vla_size returns 13
+    return (int)vsa_size(10); // vsa_size returns 13
 }
```

6.7.7.1 General

p3

```
 there is a declarator specifying
-a variable length array type,
+a variable size array type,
```

6.7.7.3 Array declarators

p2

```
 If an identifier is declared to be an object
 with static or thread storage duration,
-it shall not have a variable length array type.
+it shall not have a variable size array type.
```

p4

```
 If the size is not present,
 the array type is an incomplete type.
 If the size is * instead of being an expression,
 the array type is
-a variable length array type
+a variable size array type
 of unspecified size,
 which can only be used
 as part of the nested sequence of declarators or abstract declarators for
 a parameter declaration,
 not including anything inside an array size expression
 in one of those declarators; 159)
 such arrays are nonetheless complete types.
 If the size is an integer constant expression
 and the element type has a known constant size,
-the array type is not a variable length array type;
+the array type is not a variable size array type;
-otherwise, the array type is a variable length array type.
+otherwise, the array type is a variable size array type.
```

```
-(Variable length arrays with automatic storage duration
+(Variable size arrays with automatic storage duration
 are a conditional feature that implementations may support;
 see 6.10.10.4.)
```

p5

```
 The size of each instance of
-a variable length array type
+a variable size array type
 does not change during its lifetime.
```

EXAMPLE 4 (p10)

```
 All valid declarations of variably modified (VM) types are
 either at block scope or function prototype scope.
 Array objects declared
 with the thread_local, static, or extern storage-class specifier
 cannot have
-a variable length array (VLA) type.
+a variable size array (VSA) type.
 However,
 an object declared with the static storage-class specifier
-can have a VM type (that is, a pointer to a VLA type).
+can have a VM type (that is, a pointer to a VSA type).
 Finally,
 only ordinary identifiers
 can be declared with a VM type
 and identifiers with VM type cannot, therefore,
 be members of structures or unions.

     extern int n;
-    int A[n]; // invalid: file scope VLA
+    int A[n]; // invalid: file scope VSA
     extern int (*p2)[n]; // invalid: file scope VM
     int B[100]; // valid: file scope but not VM

-     void fvla(int m, int C[m][m]); // valid: VLA with prototype scope
+     void fvsa(int m, int C[m][m]); // valid: VSA with prototype scope

-     void fvla(int m, int C[m][m]) // valid: adjusted to auto pointer to VLA
+     void fvsa(int m, int C[m][m]) // valid: adjusted to auto pointer to VSA
     {
-         typedef int VLA[m][m]; // valid: block scope typedef VLA
+         typedef int VSA[m][m]; // valid: block scope typedef VSA

         struct tag {
             int (*y)[n]; // invalid: y not ordinary identifier
             int z[n]; // invalid: z not ordinary identifier
         };
-         int D[m]; // valid: auto VLA
+         int D[m]; // valid: auto VSA
-         static int E[m]; // invalid: static block scope VLA
+         static int E[m]; // invalid: static block scope VSA
-         extern int F[m]; // invalid: F has linkage and is VLA
+         extern int F[m]; // invalid: F has linkage and is VSA
-         int (*s)[m]; // valid: auto pointer to VLA
+         int (*s)[m]; // valid: auto pointer to VSA
-         extern int (*r)[m]; // invalid: r has linkage and points to VLA
+         extern int (*r)[m]; // invalid: r has linkage and points to VSA
-         static int (*q)[m] = &B; // valid: q is a static block pointer to VLA
+         static int (*q)[m] = &B; // valid: q is a static block pointer to VSA
     }
```

6.7.7.4 Function declarators

p11

```
 If the function declarator is not part of a definition of that function,
 parameters can have incomplete type
 and can use the [*] notation in their sequences of declarator specifiers
-to specify variable length array types.
+to specify variable size array types.
```

EXAMPLE 4 (p19)

```
-    // a is a pointer to a VLA with n*m+300 elements
+    // a is a pointer to a VSA with n*m+300 elements
```

6.7.8 Type names

EXAMPLE (p3)

```
-(e) pointer to a variable length array of an unspecified number of int s,
+(e) pointer to a variable size array of an unspecified number of int s,
```

6.7.9 Type definitions

p3

```
 Any array size expressions associated with
-variable length array declarators
+variable size array declarators
 and typeof operators
 are evaluated each time the declaration of the typedef name is reached
 in the order of execution.
```

EXAMPLE 5 (p8)

```
 If a typedef name denotes
-a variable length array type,
+a variable size array type,
 the length of the array is fixed at the time the typedef name is defined,
 not each time it is used:
```

6.7.11 Initialization

p4

```
 An entity
-of variable length array type
+of variable size array type
 shall not be initialized except by an empty initializer.
```

6.8.7.1 General

EXAMPLE 2 (p4)

```
 A goto statement which jumps past
 any declarations of objects with variably modified types
 is not conforming.
 A jump within the scope, however, is valid.

-    goto lab3; // invalid: going INTO scope of VLA.
+    goto lab3; // invalid: going INTO scope of VSA.
```

```
            {
                double a[n];
                a[j] = 4.4;
            lab3:
                a[j] = 3.3;
-               goto lab4; // valid: going WITHIN scope of VLA.
+               goto lab4; // valid: going WITHIN scope of VSA.
                a[j] = 5.5;
            lab4:
                a[j] = 6.6;
            }
-           goto lab4; // invalid: going INTO scope of VLA.
+           goto lab4; // invalid: going INTO scope of VSA.
```

6.9.2 Function definitions

p6

-Variable length array types +Variable size array types
 of unspecified size
 shall not be used as part of
 a parameter declaration in a function definition.

6.10.10.4 Conditional feature macros

p1

```
-__STDC_NO_VLA__
+__STDC_NO_VSA__
     The integer literal 1,
     intended to indicate that
     the implementation does not support
-    variable length arrays
+    variable size arrays
     with automatic storage duration.
     Parameters declared with
-    variable length array types
+    variable size array types
     are adjusted and then
     define objects of automatic storage duration with pointer types.
     Thus, support for such declarations is mandatory.
+
+__STDC_NO_VLA__
+    Synonym of __STDC_NO_VLA__ for hysterical raisins.
```

7.13.3.1 The longjmp function

EXAMPLE (p5)

```
 The longjmp function that
 returns control back to the point of the setjmp invocation
 can cause memory associated with
-a variable length array object
+a variable size array object
 to be squandered.
```

J.2 Undefined behavior

p1

```
 (15)
 A program requires
 the formation of a composite type from
-a variable length array type
+a variable size array type
 whose size is specified by
 an expression that is not evaluated (6.2.7).
```

J.6.2 Rule based identifiers

p2

```
 __STDC_NO_VLA__
+__STDC_NO_VSA__
 __STDC_UTF_16__
```

M.3 Fifth Edition

p1

mandated support for variably modified types -(but not variable length arrays themselves); +(but not variable size arrays themselves);

M.6 Second Edition

p1

```
-variable length arrays
+variable size arrays (then called variable length arrays)
```

Index

__STDC_NO_VLA__

```
 __STDC_NO_VLA__ macro, 191
+__STDC_NO_VSA__ macro, 191
 __STDC_UTF_16__ macro, 190
```

macro

```
 __STDC_NO_VLA__, 191
+__STDC_NO_VSA__, 191
 __STDC_UTF_16__, 190
```

variable length array

```
-variable length array,
+variable size array,
 129, 130, 191
```