Initializer Repetition Counts

David Keaton (dmk@dmk.com)

22 December, 1995

1. Introduction

1.1 Purpose

This document specifies the form and interpretation of a pure extension to the language portion of the C standard to provide important additional flexibility to initializers.

1.2 Scope

This document, although extending the C standard, still falls within the scope of that standard, and thus follows all rules and guidelines of that standard except where explicitly noted herein.

1.3 References

1. ISO/IEC 9899:1990, *Programming Languages — C*.
2. WG14/N494 X3J11/95-095, Prosser & Keaton. C9X Addition, *Designated Initializers*, 8 December, 1995.

All references to the ISO C standard will be presented as subclause numbers. For example, §6.4 references constant expressions.

1.4 Rationale

Initializer repetition counts provide a mechanism for initializing multiple elements of an array with the same value, a practice common in numerical programming. They add useful functionality that already exists in Fortran so that programmers migrating to C need not suffer the loss of a program-text-saving notational feature.

Initializer repetition counts integrate easily into the C grammar and do not impose any additional run-time overhead on a user's program. They also combine well with designated initializers (see [2] and [3]). A form of repetition counts is known to have been implemented in at least one C compiler, from Whitesmith's.

Note that this proposal only applies the feature to arrays. In theory it could be extended to any aggregate or union type. The general feeling is that this would cause more programming errors than it solves, but the issue is open for debate and the proposal could easily be changed to accommodate all aggregate and union types.

2. Language

2.1 Designated Initializers

The syntax for initializer repetition counts was originally chosen so that it would not depend on the existence of designated initializers [2]. However, they do combine to form an even more convenient notation. This is discussed further below.

Since designated initializers have been accepted into the C language, more possibilities for the syntax of initializer repetition counts can be considered. For example, the designator, repetition-count-expression, and stride-expression could all be combined within one set of square brackets. If this is done, it might be decided that subscript ranges would be a better idea than repetition counts.

Negative repetition counts are not proposed here because any functionality they would add could just as easily be obtained by combining designated initializers and initializer repetition counts. Requiring nonnegative repetition counts also simplifies their specification.

2.2 Initializer Repetition Counts

The syntax, constraints, and semantics for initializers in §6.5.7 are augmented by the following:

Syntax

initializer-list:
    designation$_{opt}$ *repetition-count*$_{opt}$ *initializer*
    initializer-list , *designation*$_{opt}$ *repetition-count*$_{opt}$ *initializer*

repetition-count:
    `* [` *repetition-count-expression stride*$_{opt}$ **`] =`**

repetition-count-expression:
    constant-expression

stride:
    `:` *stride-expression*

stride-expression:
    constant-expression

Constraints

No initializer shall attempt to provide a value for an object not contained within the entity being initialized.[1]

An object initialized with a repetition-count shall have array type and the repetition-count-expression shall be an integral constant expression that shall evaluate to a nonnegative value. If the array is of unknown size, any nonnegative repetition count value is valid.

A stride-expression shall be an integral constant expression that shall evaluate to a nonzero positive value.

Semantics

The *span* of subobjects specified by a repetition-count is a set of consecutive subobjects. If a designation is present, the span begins at the designated subobject, otherwise it begins with the *current object*.[2]

A repetition-count-expression defines the length of a span. In the absence of a stride, the following initializer initializes all the subobjects in the span. The initializer itself is evaluated exactly once.[3]

A stride-expression with a value *s* indicates that the following initializer initializes only the first subobject in the span and every s^{th} subobject thereafter within the span; the others are skipped.

If an array of unknown size is initialized, its size is determined by the largest indexed element that is explicitly initialized.[4]

1. This exact wording was actually already added to the C9X draft by designated initializers [2]. It replaces the former first constraint of ‘‘There shall be no more initializers in an initializer list than there are objects to be initialized.’’
2. The current object is defined in the C9X draft as specified by designated initializers [2].
3. There is room for discussion here. The intent is that compilers that extend initializers to include nonconstant expressions should generate the side effects exactly once.
4. This encompasses both the former ‘‘size determined by the number of initializers’’ rule and the designated initializers [2] rule that ‘‘size is determined by the largest indexed element with an explicit initializer.’’

Examples

The following sets the entire array **`costs`** to initially contain large values.

```
double costs[1000] = { *[1000] = HUGE_VAL };
```

The following is an example of a way that designated initializers [2] might be combined with initializer repetition counts to achieve a one-line initialization of the interior of an array. The designator comes before the **`*`** and the repetition count comes after it.

```
int interior_mask[100][100] = { [1]*[98] = { [1]*[98] = 1 } };
```

Repetition counts combined with designated initializers could be used to initialize an array with a particular value, and then override certain locations.[5]

```
char primes[] = {
      * [100]              = TRUE,
      [0]   * [2]          = FALSE,
      [2*2] * [100-2*2 : 2] = FALSE,
      [3*3] * [100-3*3 : 3] = FALSE,
      [5*5] * [100-5*5 : 5] = FALSE,
      [7*7] * [100-7*7 : 7] = FALSE
};
```

Initializers with repetition counts behave just as if the initializer had been listed the number of times specified by the repetition-count-expression (except that the initializer is evaluated only once). This is illustrated in the case of partially elided braces as follows.

```
int array[10][10] = { *[100] = 42 };
```

5. This improved version of the primes example was provided by Hallvard B. Furuseth, h.b.furuseth@usit.uio.no.