Designated Initializers

David Prosser (dfp@novell.com)
David Keaton (dmk@dmk.com)

8 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*.

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

1.4 Rationale

Designated initializers provide a mechanism for initializing sparse arrays, 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.

This feature also allows initialization of sparse structures, common in systems programming, and allows initialization of unions via any member, regardless of whether or not it is the first member.

Designated initializers integrate easily into the C grammar and do not impose any additional run-time overhead on a user's program. Their initial C implementation appeared in a compiler by Ken Thompson at AT&T Bell Laboratories.

2. Language

2.1 Designated Initializers

The syntax for initializers in §6.5.7 is changed to the following, and the constraints and semantics are augmented by the following:[1]

Syntax

initializer:
  assignment-expression
  { *initializer-list* **}**
  { *initializer-list* **,** **}**

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

1. The **=** punctuator that ends the designation is unnecessary, strictly speaking. The redundancy can help an implementation recover from syntactic errors, and also simplifies any future extensions in this area (such as some form of repetition).

designation:
designator-list **=**

designator-list:
designator
designator-list designator

designator:
[*constant-expression* **]**
. *identifier*

Constraints

No initializer shall attempt to provide a value for an object not contained within the entity being initialized.[2]

If a designator has the form

[*constant-expression* **]**

then the current object (defined below) shall have array type and the expression shall be an integral constant expression. If the array is of unknown size, any nonnegative value is valid.

If a designator has the form

. *identifier*

then the current object (defined below) shall have structure or union type and the identifier shall be a member of that type.

Semantics

Each brace-enclosed initializer list has an associated *current object*. When no designations are present, subobjects of the current object are initialized in order according to the type of the current object: array elements in increasing subscript order, structure members in declaration order, and the first member of a union.[3] In contrast, a designation causes the following initializer to begin initialization of the subobject described by the designator. Initialization then continues forward in order, beginning with the next subobject after that described by the designator.[4]

Each designator list begins its description with the current object associated with the closest-surrounding brace pair. Each item in the designator list (in order) specifies a particular member of its current object and changes the current object for the next designator (if any) to be that member.[5] The current object that results at the end of the designator list is the subobject to be initialized by the following initializer.

The initialization shall occur in initializer list order, each initializer provided for a particular subobject overriding any previously listed initializer for the same subobject; all subobjects that are not initialized explicitly shall be initialized implicitly the same as objects that have static storage duration.

2. This replaces the former first constraint of "There shall be no more initializers in an initializer list than there are objects to be initialized."

3. If the initializer list for a subaggregate or contained union does not begin with a left brace, its subobjects are initialized as usual, but the subaggregate or contained union does not become the current object: current objects are associated only with brace-enclosed initializer lists.

4. After a union member is initialized, the next object is not the next member of the union; instead, it is the next subobject of an object containing the union.

5. Thus, a designator can only specify a strict subobject of the aggregate or union that is associated with the surrounding brace pair. Note, too, that each separate designator list is independent.

If an array of unknown size is initialized, its size is determined by the largest indexed element with an explicit initializer.[6]

Examples

Arrays can be initialized to correspond to the elements of an enumeration by using designators:

```
enum { Mem_One, Mem_Two, /*...*/ };
const char *nm[] = {
      [Mem_Two] = "Mem Two",
      [Mem_One] = "Mem One",
      /*...*/
};
```

Structure members can be initialized to nonzero values without depending on their order:

```
div_t answer = { .quot = 2, .rem = -1 };
```

Designators can be used to provide explicit initialization when unadorned initializer lists might be misunderstood:

```
struct { int a[3], b; } w[] = { [0].a = {1}, [1].a[0] = 2 };
```

Space can be "allocated" from both ends of an array by using a single designator:

```
int a[MAX] = {
      1, 3, 5, 7, 9, [MAX-5] = 8, 6, 4, 2, 0
};
```

In the above, if **MAX** is greater than ten, there will be some zero-valued elements in the middle; if it is less than ten, some of the values provided by the first five initializers will be overridden by the second five.

Finally, any member of a union can be initialized:

```
union { /*...*/ } u = { .any_member = 42 };
```

6. This encompasses the former "size determined by the number of initializers" rule.