JTC1/SC22/WG14 - N3121
Title: Forward Declaration of Parameters v2 (updates N2780)
Author: Martin Uecker
Date: 2023-04-22
Prior Art: GCC

Introduction

It is not possible to use a later parameter in a size specifier, which makes it impossible to annotate existing APIs where the length of a buffer is passed before the pointer. Parameter forward declarations as implemented in GCC as an extensions solve this problem [1]. Originally proposed in N2780 for C23, the feature did not receive sufficient consensus. With the removal of K&R function definitions, this also became a problem for function definitions and was brought up again as open issue. This paper addresses issues pointed out on the reflector [2].

Example:

```
void foo(size_t len; char buf[static len], size_t len);
```

Alternative:

It was proposed to allow referring to later arguments in size expressions as a more elegant solution. While this initially seemed appealing, it turned out that this would be more complicated to specify and implement: It requires more complicated changes to existing parsers, has backwards compatibility issues, and there are problems related to mutual dependencies between parameters [3]. In C, identifiers generally require a declaration before use (the only exception are labels). Thus, this solution would require the invention of new rules.

Examples of other forward declarations in C:

```
extern int counter;   // object with external linkage
void foo(void);       // function prototype
struct foo;           // tag
```

Example (backwards compatibility issue):

```
int a;
int foo(char buf[static a], size_t a); // meaning would change
```

Syntax

GCC supports comma and semicolon to separate multiple forward declarations. Here we propose to allow only the semicolon, because then it is directly visible whether a declaration is a forward parameter declaration or a parameter declaration. The syntax is robust against typos, because confusing a semicolon with a comma would either cause an invalid re-declaration of the same parameter name or a forward declaration for a parameter that does not exist. The syntax is not used in C++. Function declarations using run-time size expressions in argument types are not supported in C++. Thus, the use of such new extension is possible only in function declarations which are already not compatible with C++. It is possible to hide both run-time size expressions and forward declaration behind a macro.

General Issues

The GCC extension is obscure and rarely used. Nevertheless, it is enabled by default and does not seem to cause problems. Apart from the removal of K&R function definitions, another reason it becomes more useful recently is that compilers recently started to use size expressions for improved compile-time and run-time bounds checking and this already sparked increased interest in adding such annotations to existing APIs. Because of this, the extension was also requested by users [4].

References:

[1] https://gcc.gnu.org/onlinedocs/gcc/Variable-Length.html

[2] [SC22WG14.20184] Parameter forward declarations (N2780)

[3] Ritchie DM. Variable-size arrays in C. The Journal of C Language Translation 1990;2:81-86.

[4] https://github.com/llvm/llvm-project/issues/47617

(proposed wording on next page)

Proposed Wording

6.7 Declarations

Constraints

3 If an identifier has no linkage, there shall be no more than one declaration of the identifier (in a declarator or type specifier) with the same scope and in the same name space, except that:
— a typedef name may be redefined to denote the same type as it currently does, provided that type is not a variably modified type;
— tags may be redeclared as specified in 6.7.2.3.
--- parameters declared in a parameter forward declaration are redeclared in the parameter list as specified in 6.7.6.3.

6.7.6 Declarators

1 Syntax

```
parameter-type-list:
      parameter-forward-declaration ; parameter-type-list
      parameter-list
      parameter-list , …
parameter-forward-declaration:
      attribute-specifier-sequence_opt declaration-specifiers declarator
parameter-list:
      parameter-declaration
      parameter-list , parameter-declaration
parameter-declaration:
      attribute-specifier-sequence_opt declaration-specifiers declarator
      attribute-specifier-sequence_opt declaration-specifiers abstract-declarator_opt
```

6.2.2 Linkages of identifiers

2 In the set of translation units and libraries that constitutes an entire program, each declaration of a particular identifier with external linkage denotes the same object or function. Within one translation unit, each declaration of an identifier with internal linkage denotes the same object or function. **With the exception of parameter forward declarations and their respective parameter declarations,** each declaration of an identifier with no linkage denotes a unique entity.

6.2.7 Compatible type and composite type

4 For an identifier with internal or external linkage declared in a scope in which a prior declaration of that identifier is visible 60), if the prior declaration specifies internal or external linkage, the type of the identifier at the later declaration becomes the composite type. **The type of a parameter with a parameter forward declaration becomes the composite type at the parameter declaration.**

6.7.6.3 Function declarators

Constraints

4 An identifier declared in a parameter forward declaration shall also be declared in the parameter list.

Semantics

5 A parameter type list specifies the types of, and may declare identifiers for, the parameters of the function. **Parameter forward declarations may provide forward declarations of the identifiers of the parameters (for use in size expressions).**

12 The storage class specifier in the declaration specifiers for a parameter declaration, if present, is ignored unless the declared parameter is one of the members of the parameter type list for a function definition. The optional attribute specifier sequence in a parameter declaration **and in a parameter forward declaration** appertains to the parameter.

6.9.1 Function definitions

10 On entry to the function, the size expressions **of parameter declarations and forward parameter declarations of** ~~**of each**~~ variably modified ~~**parameter**~~ **type** are evaluated and the value of each argument expression is converted to the type of the corresponding parameter as if by assignment. (Array expressions and function designators as arguments were converted to pointers before the call.