JTC1/SC22/WG14 - N3140
Title: Parameter Forward Declarations
Author: Martin Uecker, Jens Gustedt
Date: 2023-06-22

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 specifier-qualifier-list identifier attribute-specifier-sequence_opt
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

2 The only storage-class specifier that shall occur in a parameter declaration is register.

4 The identifier in a parameter forward declaration shall declare a parameter that is also declared by the declarator of a parameter declaration in the parameter list. The types specified in the parameter forward declaration and the corresponding parameter declaration shall be compatible.

Semantics

5 A parameter list **in the 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.XXX) It is implementation-defined whether forward declarations with types other than integer types or with other syntactic forms for the declaration are accepted. If other types or syntactic forms are accepted, the behavior is implementation-defined.**

XXX) The type declared by the parameter declaration then becomes the composite type according to 6.2.7p4.

8 If the **parameter type** list terminates with an ellipsis **...**

10 If, in a parameter declaration, an identifier can be treated either as a typedef name or as a parameter name, it shall be taken as a typedef name.

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 each of the optional attribute specifier sequences in a parameter forward declaration** appertain~~s~~ to the **declared** parameter.

21 EXAMPLE 6 The following prototypes have valid parameter forward declarations:

```
void transpose(double[*][*], int, int);
void transpose(int n; int m; double x[n][m], int n, int m) ;

void transpose(int m; int n; double x[n][m], int n, int m)
{
      // ...
}

int foo(size_t a; char buf[static a], size_t a);
```

Here, the three declarations of transpose have compatible type; in particular, the presence and order of the parameter forward declarations has no impact on the resulting prototype.

The following prototypes have invalid parameter forward declarations:

```
void f(int x; const int x);      // incompatible types
void g(int x;  int y);           // missing parameter declaration for ‘x’
void h(int x; int x; int x);     // invalid redeclaration, according to 6.7p3
void i(int x; int x, int x);     // invalid redeclaration, according to 6.7p3
```

The following prototypes with parameter forward declarations may be supported by an extension. If so, the behavior is implementation-defined:

```
void b(double n; double n);      // type other than an integer type
void c(int (*a)[3]; char buf[sizeof *a], int (*a)[3]);
       // extended syntactic form of the declaration and type other than an integer type
```