The New C Standard (Excerpted material)

An Economic and Cultural Commentary

Derek M. Jones
derek@knosof.co.uk

6.7.5.1 Pointer declarators

Semantics

derived-declarator-type-list

If, in the declaration "`T D1`", `D1` has the form 1560

```
* type-qualifier-list(opt) D
```

and the type specified for *ident* in the declaration "`T D`" is "*derived-declarator-type-list T*", then the type specified for *ident* is "*derived-declarator-type-list type-qualifier-list* pointer to *T*".

Commentary

The term *derived-declarator-type-list* is very rarely used outside of the C Standard, even during Committee discussions.

C++

The C++ Standard uses the term *cv-qualifier-seq* instead of *type-qualifier-list*.

For each type qualifier in the list, *ident* is a so-qualified pointer. 1561

Commentary

The C++ Standard says it better:

8.3.1p1 *The cv-qualifiers apply to the pointer and not to the object pointed to.*

So qualifiers to the right of the * token qualify the pointer type and qualifiers to the left of the * token qualify the pointed-to type.

Example

The following declares three objects to have the same type.

```
typedef int * p_i;

p_i   const g_1;
const p_i   g_2;
const int  *g_3;
```

pointer types to be compatible

For two pointer types to be compatible, both shall be identically qualified and both shall be pointers to compatible types. 1562

Commentary

compatible type if

There is no requirement that the declarators of the two pointer types, being checked for compatibility, be built from the same sequence of tokens. For instance:

```
typedef int * int_ptr;

int *   p1;
int_ptr p2; /* The type of p1 is compatible with that of p2. */
```

C++

The C++ Standard does not define the term *compatible type*, although in the case of qualified pointer types the term *similar* is defined (4.4p4). When two pointer types need to interact the C++ Standard usually specifies that the qualification conversions (clause 4) are applied and then requires that the types be the same. These C++ issues are discussed in the sentences in which they occur.

1563 EXAMPLE The following pair of declarations demonstrates the difference between a “variable pointer to a constant value” and a “constant pointer to a variable value”.

```
const int *ptr_to_constant;
int *const constant_ptr;
```

The contents of any object pointed to by **ptr_to_constant** shall not be modified through that pointer, but **ptr_to_constant** itself may be changed to point to another object. Similarly, the contents of the **int** pointed to by **constant_ptr** may be modified, but **constant_ptr** itself shall always point to the same location.

The declaration of the constant pointer **constant_ptr** may be clarified by including a definition for the type “pointer to **int**”.

```
typedef int *int_ptr;
const int_ptr constant_ptr;
```

declares **constant_ptr** as an object that has type “const-qualified pointer to **int**”.

Commentary

A typename can be qualified with a type qualifier (e.g., `const int_ptr`), but the underlying type cannot be changed by adding a type specifier (e.g., `unsigned int_ptr`)

type specifier
syntax

References