Completeness of types

Clive D.W. Feather
Email: clive@davros.org
Last changed 2010-03-12

Introduction

This is a proposal for a number of changes to the C standard to address a problem with the description of complete and incomplete types.

All text is based on N1401.

The issue

The Standard says, in 6.2.5#1:

> Types are partitioned into *object types* (types that fully describe objects), *function types* (types that describe functions), and *incomplete types* (types that describe objects but lack information needed to determine their sizes).

The word "partition" means a division into mutually exclusive subsets that completely cover the original set. In other words, this sentence is saying that each type is exactly one of an object type, a function type, or an incomplete type.

But a little thought will show that this is nonsense. Consider the code:

```
struct s
{
    struct s *next;     // Point 1
    int i;
};

struct s item;          // Point 2
```

At point 1 the type `struct s` is an incomplete type. At point 2 it is a complete type. Yet both of them are the same type.

The answer, of course, is that completeness *is a scoped property of a type*. That is, it varies from place to place within a translation unit, without altering the type itself. This is clear from much of the text of the Standard but, unfortunately, not from the definition of the term.

The proposal

The rest of this paper proposes detailed changes to the text of the Standard to address this issue. Apart from the first item, they are in textual order.

6.2.5#1: change from:

Types are partitioned into *object types* (types that fully describe objects), *function types* (types that describe functions), and *incomplete types* (types that describe objects but lack information needed to determine their sizes).

to:

Types are partitioned into *object types* (types that describe objects) and *function types* (types that describe functions). At various points within a translation unit an object type may be *incomplete* (there is insufficient information to determine the size of objects of that type) or *complete* (there is enough information).[FN]

[FN] A type may be incomplete or complete throughout a translation unit, or may change states at different places.

5.2.4.1#1, third bullet:

Change “structure, union, or incomplete type” to “structure, or union type”.

6.2.5#14: insert between the two sentences:

The basic types are complete object types.

6.2.5#19:

Change “it is an incomplete type” to “it is an incomplete object type”.

6.2.5#20:

Change “object, function, and incomplete types” to “object and function types”.

6.2.5#20 first bullet:

Delete footnote 40 and the reference to it, and after the first sentence insert:

The element type must be complete whenever the array type is specified.

6.2.5#20 last bullet

Change “function type, an object type, or an incomplete type” to “function or object type”.

Append:

A pointer type is a complete object type.

6.2.7#1:

Change “If both are complete types,” to “If both are completed anywhere within their respective translation units,”.[1]

6.3.2.1#1:

Change “an object or incomplete type other than **`void`**” to “an object type other than **`void`**”.

6.3.2.3#1:

Change “incomplete or object type” to “object type” in two places.

6.3.2.3#7:

Change “incomplete or object type” to “object type” in two places.

6.5.2.1#1:

Change “pointer to object type” to “pointer to complete object type”.

6.5.2.2#1:

Change “an object type” to “a complete object type”.

6.5.2.2#4:

Change “any object type” to “any complete object type”.

6.5.2.3#5:

Change “complete” to “completed”.[2]

6.5.2.5#1:

Change “an object type” to “a complete object type”.

6.5.6#2:

Change “an object type” to “a complete object type”.

6.5.6#3:

Change “compatible object types” to “compatible complete object types”.

---

[1] This addresses DR314 question 1.
[2] This is not essential but I believe it is slightly clearer.

6.5.8#2:

Delete the third bullet and the “or” at the end of the second bullet.

6.5.8#5:

Change “object or incomplete types” to “object types”.

6.5.9#2:

Change “object or incomplete type” to “object type”.

6.5.9#5:

Change “object or incomplete type” to “object type”.

6.5.15#3 last bullet:

Change “object or incomplete type” to “object type”.

6.5.16.1#3 fourth bullet:

Change “object or incomplete type” to “object type”.

6.5.16.2#1:

Change “an object type” to “a complete object type”.

6.6#7 last bullet:

Change “an object type” to “a complete object type”.

6.7.2.1#2: change:

A structure or union shall not contain a member with incomplete or function type

to:

A struct-declaration-list shall not declare a member with incomplete or function type

6.7.2.1#7: change:

The type is incomplete until after the `}` that terminates the list.

to:

The type is incomplete until immediately after the `}` that terminates the list, and complete thereafter.[3]

[3] Or delete this sentence entirely, since 6.7.2.3#4, as amended says the same thing.

6.7.2.1#8:

Change “any object type” to “any complete object type”.

6.7.2.2#4: change:

The enumerated type is incomplete until after the `}` that terminates the list of enumerator declarations.

to:

The enumerated type is incomplete until immediately after the `}` that terminates the list of enumerator declarations, and complete thereafter.[4]

6.7.2.3#4: change:

The type is incomplete[118)] until the closing brace of the list defining the content, and complete thereafter.

to:

Irrespective of whether there is a tag or what other declarations of the type are in the same translation unit, the type is incomplete[118)] until the closing brace of the list defining the content, and complete thereafter.

6.7.3#2:

Change “object or incomplete type” to “object type”, or change the entire paragraph to:

The only types that shall be restrict-qualified are pointer types whose referenced type is an object type.[5]

6.7.5#7:

Change “structure, union, or incomplete type” to “structure, or union type”.

6.7.8#3:

Change “an object type” to “a complete object type”.

---

[4] Ditto.

[5] A pointer to a function returning int is (indirectly) a pointer type derived from an object type, but was not intended to be included here.

6.7.8#22: change:

At the end of its initializer list, the array no longer has incomplete type.

to:

The array type is completed at the end of its initializer list.

6.9.1#3:

Change “an object type” to “a complete object type”.

6.9.1#7:

Change “an object type” to “a complete object type”.

7.15#3:

Change “an object type” to “a complete object type”.

7.20.1#2:

Change “an object type” to “a complete object type” in relation to **`fpos_t`** but *not* in relation to **`FILE`**.[6]

7.24.1#3:

Change “an object type” to “a complete object type” in all five places.

7.27.1#2:

Change “an object type” to “a complete object type”.

J.2#1:

Change “an object type” to “a complete object type” in two places (second 6.6 bullet and second 6.9.1 bullet).

Lacunae

This is not related to this proposal, but I noticed it while reading the text and drafting it: in 6.7.2.3#6, I suspect that “**`enum`** *identifier*” should be “**`enum`** $identifier_{opt}$” (twice).

[6] Because of 7.20.3#6, there is no particular need to be able to copy FILE objects or otherwise manipulate them other than through the provided interfaces, all of which use pointers. Therefore it does not matter if the type is incomplete.