

## Re: mutually referential (Pg 140 K&R2)

---

*Source:* [http://coding.derkeiler.com/Archive/C\\_CPP/comp.lang.c/2005-04/msg02020.html](http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2005-04/msg02020.html)

---

- *From:* Chris Torek <nospam@xxxxxxxx>
  - *Date:* 23 Apr 2005 19:50:13 GMT
- 

>Chris Torek wrote:

```
>> struct s { long l; };
>> void f(void) {
>> struct s; /* this "vacuous" declaration clears the way */
>> struct s { char c; } var;
>> /* this is a new, different type also named s */
>> ...
>> }
```

In article <d4e4hq\$ifh\$1@xxxxxxxxxxxxxxxx>

Jeremy Yallop <jeremy@xxxxxxxxxxxxxxxxxxxxxxxx> wrote:

>In this example the vacuous declaration doesn't make any difference;  
>the inner "full" declaration is valid and declares a type distinct  
>from the outer declaration regardless. The vacuous declaration is  
>needed when you want a name in an inner scope to match a tag defined  
>in the (same) inner scope instead of a tag from the outer scope.

Ah, yes, I believe you are correct.

[snippage]

```
>> In general, incomplete structure types are only usefully completed
>> in the same scope (it is possible to complete them in an inner
>> scope, but this has relatively little practical application).
```

>I don't think there is a way to complete incomplete structure types in  
>an inner scope.

This does seem to be the case (i.e., given an incomplete "struct X" at some outer scope, any attempt to complete it at any nested scope will instead define a new type of the same name).

To generalize: the vacuous (empty/forward) declarations of struct tags is often not required, but never actually harmful. It is required if you are overriding an outer-scope name for mutually referential structures limited to some inner scope, and it is often required if you want to have prototype arguments using a type that will not be defined (i.e., will be left incomplete):

## Re: mutually referential (Pg 140 K&R2)

```
/* interface.h */  
  
struct bob; /* required */  
  
void raise_bob(struct bob *);  
void lower_bob(struct bob *);  
void move_bob_sideways(struct bob *, int);
```

but:

```
/* other_interface.h */  
  
/* struct jane; -- allowed but not required */  
  
struct jane *new_jane(void);  
void move_jane_sideways(struct jane *, int);
```

Here the first mention of the type "jane" occurs outside a prototype, at the outermost scope, so it springs into being at that point and the declaration of the function `move_jane_sideways()` refers to the existing, incomplete type.

One can either learn all these rules, or just write out the vacuous declarations. :-)

If you like typedefs, or are commanded :-)) to use them, I recommend writing out all the typedef sequences first:

```
typedef struct bob BOB;  
typedef struct plumb PLUMB;  
  
/* optional: now complete it */  
struct bob {  
    BOB *list_of_bobs;  
    PLUMB *parent;  
    ... fill this in ...  
};  
  
void move_bob_sideways(BOB *, int);
```

Note that by listing all the types first, and giving them all their aliases — you can do this with the "value meal combo deal" construct above, or by writing it all out:

```
struct bob;  
typedef struct bob BOB;  
  
struct plumb;  
typedef struct plumb PLUMB;
```

— by giving them their aliases *\*before\** you define them, you

Re: mutually referential (Pg 140 K&R2)

are able to use the aliases inside the definition.

—  
In-Real-Life: Chris Torek, Wind River Systems  
Salt Lake City, UT, USA (40°39.22'N, 111°50.29'W) +1 801 277 2603  
email: forget about it <http://web.torek.net/torek/index.html>  
Reading email is like searching for food in the garbage, thanks to spammers.

---

• **References:**

- ◆ **mutually referential (Pg 140 K&R2)**  
    ◇ From: G Patel
- ◆ **Re: mutually referential (Pg 140 K&R2)**  
    ◇ From: Chris Torek
- ◆ **Re: mutually referential (Pg 140 K&R2)**  
    ◇ From: Jeremy Yallop
  
- Prev by Date: **Re: How do I check if a file is empty in C??**
- Next by Date: **Re: How is strlen implemented?**
- Previous by thread: **Re: mutually referential (Pg 140 K&R2)**
- Next by thread: **Re: mutually referential (Pg 140 K&R2)**
- Index(es):
  - ◆ **Date**
  - ◆ **Thread**