WG14/N296

X3J11/93-042

••

From netcom.com!segfault!rfg Thu Oct 28 02:54:04 1993 remote from uunet
Received: by plauger.UUCP (UUL1.3#20134)
from uunet with UUCP; Thu, 28 Oct 93 07:36:10 EST
Received: from netcomsv.netcom.com (via uucp4.netcom.com) by relay1.UU.NET with SMTP
(5.61/UUNET-internet-primary) id AA17693; Thu, 28 Oct 93 02:54:04 -0400
Received: from segfault.UUCP by netcomsv.netcom.com with UUCP (4.1/SMI-4.1)
id AA03280; Wed, 27 Oct 93 23:53:50 PDT
From: uunet!netcom.com!segfault!rfg
Received: from segfault (localhost) by segfault. (5.0/SMI-4.1)
id AA28280; Wed, 27 Oct 93 23:51:31 PDT
To: plauger!pjp (P.J. Plauger)
Subject: ANSI/ISO C Defect report #rfg1
Reply-To: uunet!netcom.com!segfault!rfg
Date: Wed, 27 Oct 1993 23:51:30 -0700
Message-Id: <28279.751791090@segfault>
Content-Length: 1341

(Please submit the following defect report for me. Thanks.)

ANSI/ISO C Defect report #rfg1:
---

There appears to be an inconsistancy between the constraints on "passing" values versus "returning" values.

The constraints for function calls clearly indicate that a diagnostic is required if any given actual argument is passed (to a prototyped function) into a corresponding formal parameter whose type is not assignment compatatable with respect to the type of the passed value.

In the case of values returned by a return statement however, there seems to be no such compatability constrain imposed upon the expression given in the return statement and the corresponding (declared) function return type.

A new constraint should be added to the standard like:

> If present, the expression given in a return statement shall have a type such that its value may be assigned to an object with the unqualified version of the return type of the containing function.

(This exactly mirrors the existing constraint on parameter matching imposed upon calls to prototyped functions.)

-- Ronald F. Guilmette ----------
------ domain address: rfg@netcom.com ----------
------ uucp address: ...!uunet!netcom.com!rfg ----------

••

••

From netcom.com!segfault!rfg Thu Oct 28 03:41:53 1993 remote from uunet
Received: by plauger.UUCP (UUL1.3#20134)
from uunet with UUCP; Thu, 28 Oct 93 07:36:11 EST
Received: from netcomsv.netcom.com (via uucp5.netcom.com) by relay1.UU.NET with SMTP
(5.61/UUNET-internet-primary) id AA02038; Thu, 28 Oct 93 03:41:53 -0400
Received: from segfault.UUCP by netcomsv.netcom.com with UUCP (4.1/SMI-4.1)
id AA03573; Thu, 28 Oct 93 00:42:26 PDT
From: uunet!netcom.com!segfault!rfg
Received: from segfault (localhost) by segfault. (5.0/SMI-4.1)
id AA28373; Thu, 28 Oct 93 00:39:07 PDT
To: plauger!pjp (P.J. Plauger)
Subject: ANSI/ISO C Defect report #rfg2
Reply-To: uunet!netcom.com!segfault!rfg
Date: Thu, 28 Oct 1993 00:39:07 -0700
Message-Id: <28372.751793947@segfault>
Content-Length: 1426

(Please submit the following defect report for me. Thanks.)

ANSI/ISO C Defect report #rfg2:
---

There is an ambiguity with respect to the constraints which (or may not) apply to initializations.

Section 3.5.7 of the ("classic") ANSI C standard says:

> "...the same type constraints and conversions as for simple assignment apply."

Note however that this rule itself appears within a semantics section, thus leading some implementors to feel that no diagnostics are required in cases where an attempt is made to provide an initializer for a give scalar and where the type of the initializer is NOT assignment compatible with the type of the scalar object being initialized.

This ambiguity should be removed by adding an explicit constraint to the section covering initializations, such as:

> Each scalar initializer expression given in an initializer shall have a type such that its value may be assigned to an object with the unqualified version of the corresponding scaral object to be initialized by the given scalar initializer expression.

(This roughly mirrors the existing constraint on parameter matching imposed upon calls to prototyped functions.)

-- Ronald F. Guilmette ----------

------ domain address: rfg@netcom.com ----------
------ uucp address: ...!uunet!netcom.com!rfg ----------
••
••
From netcom.com!segfault!rfg Thu Oct 28 04:32:39 1993 remote from uunet
Received: by plauger.UUCP (UUL1.3#20134)
from uunet with UUCP; Thu, 28 Oct 93 07:36:12 EST
Received: from netcomsv.netcom.com (via uucp6.netcom.com) by relay1.UU.NET with SMTP
(5.61/UUNET-internet-primary) id AA15059; Thu, 28 Oct 93 04:32:39 -0400
Received: from segfault.UUCP by netcomsv.netcom.com with UUCP (4.1/SMI-4.1)
id AA19113; Thu, 28 Oct 93 01:33:18 PDT
From: uunet!netcom.com!segfault!rfg
Received: from segfault (localhost) by segfault. (5.0/SMI-4.1)
id AA28432; Thu, 28 Oct 93 01:30:11 PDT
To: plauger!pjp (P.J. Plauger)
Subject: ANSI/ISO C Defect report #rfg3
Reply-To: uunet!netcom.com!segfault!rfg
Date: Thu, 28 Oct 1993 01:30:10 -0700
Message-Id: <28431.751797010@segfault>
Content-Length: 1230

(Please submit the following defect report for me. Thanks.)

ANSI/ISO C Defect report #rfg3:
---

Section 3.5.4.2 of the ("classic") ANSI C standard fails to contain any constraint which would prohibit the element type of an array type from being either a function type or an incomplete type.

I believe that such a constraint is clearly needed.

This problem could be solved by adding an explicit constraint to the section covering array declarators, such as:

> The element type specified in an array type declaration shall be an object type.

Note that some similar sort of constraint should also appear elsewhere, e.g. in the section describing casts, so as to insure that there will be absolutely NO valid way of specifying any array type where the element type is a non-object type. This additional constraint is needed in order to assure diagnostics for such things as:

```
typedef void (FUNC_TYPE) ();

void foobar ()
{
    ... (FUNC_TYPE (*)(10)) expression;
}
```

-- Ronald F. Guilmette ----------
------ domain address: rfg@netcom.com ----------
------ uucp address: ...!uunet!netcom.com!rfg ----------
••
••
From netcom.com!segfault!rfg Thu Oct 28 13:21:41 1993 remote from uunet
Received: by plauger.UUCP (UUL1.3#20134)
from uunet with UUCP; Thu, 28 Oct 93 18:41:57 EST
Received: from netcomsv.netcom.com (via uucp6.netcom.com) by relay1.UU.NET with SMTP
(5.61/UUNET-internet-primary) id AC28161; Thu, 28 Oct 93 13:21:41 -0400
Received: from segfault.UUCP by netcomsv.netcom.com with UUCP (4.1/SMI-4.1)
id AA24325; Thu, 28 Oct 93 10:22:12 PDT
From: uunet!netcom.com!segfault!rfg
Received: from segfault (localhost) by segfault. (5.0/SMI-4.1)
id AA28786; Thu, 28 Oct 93 10:18:32 PDT
To: plauger!pjp (P.J. Plauger)
Subject: ANSI/ISO C Defect report #rfg4
Reply-To: uunet!netcom.com!segfault!rfg
Date: Thu, 28 Oct 1993 10:18:32 -0700
Message-Id: <28785.751828712@segfault>
Content-Length: 1352

(Please submit the following defect report for me. Thanks.)

ANSI/ISO C Defect report #rfg4:
---

Section 4.1.5 of the ("classic") ANSI C standard fails to contain any constraint which would prohibit the type argument given in an invocation of the offsetof() macro from being an incomplete type.

This situation can arise in examples such as the following:

```
#include <stddef.h>

struct S
{
        int member1;
        int member2(1+offsetof(struct S,member1));
};
```

I believe that a constraint prohibiting the type argument to offsetof() from being an incomplete type is clearly needed.

This problem could be solved by adding an explicit constraint to ("classic") section 4.1.5, such as:

The type argument given in an invocation of the offsetof() macro

shall be the name of a complete struct type or a complete union type.

(Note that this way of expressing the constraint also makes it completely clear that diagnostics are required for cases where the type given in the invocation is, for instance, a function type, an array type, an enum type, a pointer type, or a built-in arithmentic type.)

-- Ronald F. Guilmette ----------
------ domain address: rfg@netcom.com ----------
------ uucp address: ...!uunet!netcom.com!rfg ----------
••
••
From netcom.com!segfault!rfg Fri Oct 29 03:10:09 1993 remote from uunet
Received: by plauger.UUCP (UUL1.3#20134)
from uunet with UUCP; Fri, 29 Oct 93 07:33:49 EST
Received: from netcomsv.netcom.com (via uucp4.netcom.com) by relay1.UU.NET with SMTP
(5.61/UUNET-internet-primary) id AA14442; Fri, 29 Oct 93 03:10:09 -0400
Received: from segfault.UUCP by netcomsv.netcom.com with UUCP (4.1/SMI-4.1)
id AA04927; Fri, 29 Oct 93 00:09:54 PDT
From: uunet!netcom.com!segfault!rfg
Received: from segfault (localhost) by segfault. (5.0/SMI-4.1)
id AA29873; Fri, 29 Oct 93 00:07:21 PDT
To: plauger!pjp (P.J. Plauger)
Subject: ANSI/ISO C Defect report #rfg5
Reply-To: uunet!netcom.com!segfault!rfg
Date: Fri, 29 Oct 1993 00:07:20 -0700
Message-Id: <29872.751878440@segfault>
Content-Length: 1635

(Please submit the following defect report for me. Thanks.)

ANSI/ISO C Defect report #rfg5:
---

Section 3.3.3.4 of the ("classic") ANSI C standard provides the following constraint:

> "The sizeof operator shall not be applied to an expression that has function type or an incomplete type..."

The logical implication of this constraint is that neither function types nor incomplete types have "sizes" per se... at least not as far as the standard is concerned.

I have noted however that neither ("classic") section 3.3.2.4 (Posfix increment and decrement operators) nor ("classic") section 3.3.3.1 (Prefix increment and decrement operators) contain any constraints which would prohibit the incrementing or decrementing of pointers to function types or pointers to incomplete types.

I believe that this logical inconsistancy needs to be addressed (and rectified) in the standard. It seems that the most appropriate way to do this is to add the following additional constraint to 3.3.2.4:

> The operand of the posfix increment or decrement operator shall not have a type which is a pointer to incomplete type or a pointer to function type.

Likewise, the following new constraint should be added to section 3.3.3.1:

> The operand of the prefix increment or decrement operator shall not have a type which is a pointer to incomplete type or a pointer to function type.

-- Ronald F. Guilmette ----------
------ domain address: rfg@netcom.com ----------
------ uucp address: ...!uunet!netcom.com!rfg ----------
••
••
From netcom.com!segfault!rfg Tue Nov 2 04:57:29 1993 remote from uunet
Received: by plauger.UUCP (UUL1.3#20134)
from uunet with UUCP; Tue, 2 Nov 93 07:40:34 EST
Received: from netcomsv.netcom.com (via uucp5.netcom.com) by relay1.UU.NET with SMTP
(5.61/UUNET-internet-primary) id AA16525; Tue, 2 Nov 93 04:57:29 -0500
Received: from segfault.UUCP by netcomsv.netcom.com with UUCP (4.1/SMI-4.1)
id AA10822; Tue, 2 Nov 93 01:58:12 PST
From: uunet!netcom.com!segfault!rfg
Received: from segfault (localhost) by segfault. (5.0/SMI-4.1)
id AA09086; Tue, 2 Nov 93 01:54:59 PST
To: plauger!pjp (P.J. Plauger)
Subject: ANSI/ISO C Defect report #rfg6
Reply-To: uunet!netcom.com!segfault!rfg
Date: Tue, 02 Nov 1993 01:54:58 -0800
Message-Id: <9085.752234098@segfault>
Content-Length: 3330

(Please submit the following defect report for me. Thanks.)

ANSI/ISO C Defect report #rfg6:
---

Section 3.2.1.5 of the ("classic") ANSI C standard explicitly allows an implementation to evaluate a floating-point expression using some type which has MORE precision that the apparent type of the expression itself:

> "The values of floating operands and the results of floating expressions may be represented in greater precision and range than that required by the type."

A footnote on this rule also says explicitly that:

> "Cast and assignment operators still must perform their specified conversions, as described in 3.2.1.3 and 3.2.1.4."

As noted in the first of these two quotes (above) some compilers (most notably for x86 and mx680x0 target systems) may perform floating-point expression evaluation using a type which has more precision and/or range that that of the "apparent type" of the expression being evaluated.

The clear implication of the above rules is that compilers must sometimes generate code to implement narrowing of floating-point expression results, when (a) those results were genereated using a format with more precision and/or range than the "apparent type" of the expression would seem to call for, and where (b) the expression result is the operand of a cast or is used as an operand of an "assignment operator".

My question is simply this: For the purposes of the above rules, does the term "assignment operator" mean exactly (and only) those operators listed in ("classic") section 3.3.16, or should implementors and users expect that other operations described within the standard as being similar to "assignment" will also producing floating-point narrowing effects (under the right conditions)?

Specifically, may (or must) implicit floating-point narrowing occur as a result of parameter passing if the actual argument expression is evaluated in a format which is wider than its "apparent type"? May (or must) implicit floating-point narrowing occur as a result of a return statement if the return statement contains a floating-point expression which is evaluated in some format which is wider than its "apparent type"?

Here are two examples illustrating these two questions. Imagine that these examples will be compiled for a type of target system which is capable of performing floating-point addition ONLY on floating-point operands which are represented in the same FP format normally used to hold type `long double' operands in C:

```
        ================================================================
=====
        extern void callee ();  /* non-prototyped */

        double a, b;

        void caller ()
        {
          callee(a+b);  /* evaluated in long double format then narrowed? */
        }
        ================================================================
=====
```

```
        ================================================================
=====
        double a, b;

        double returner ()
        {
          return a+b;  /* evaluated in long double format then narrowed? */
        }
        ================================================================
=====
```

-- Ronald F. Guilmette, Sunnyvale, California ----------
------ domain address: rfg@netcom.com ----------
------ uucp address: ...!uunet!netcom.com!rfg ----------
••
••

```
From netcom.com!segfault!rfg  Tue Nov  2 05:41:55 1993 remote from uunet
Received: by plauger.UUCP (UUL1.3#20134)
        from uunet with UUCP; Tue, 2 Nov 93 07:40:36 EST
Received: from netcomsv.netcom.com (via uucp6.netcom.com) by relay1.UU.NET with
SMTP
        (5.61/UUNET-internet-primary) id AA26034; Tue, 2 Nov 93 05:41:55 -0500
Received: from segfault.UUCP by netcomsv.netcom.com with UUCP (4.1/SMI-4.1)
        id AA09581; Tue, 2 Nov 93 02:42:37 PST
From: uunet!netcom.com!segfault!rfg
Received: from segfault (localhost) by segfault. (5.0/SMI-4.1)
        id AA09212; Tue, 2 Nov 93 02:39:12 PST
To: plauger!pjp (P.J. Plauger)
Subject: ANSI/ISO C Defect report #rfg7
Reply-To: uunet!netcom.com!segfault!rfg
Date: Tue, 02 Nov 1993 02:39:10 -0800
Message-Id: <9211.752236750@segfault>
Content-Length: 1033
```

(Please submit the following defect report for me.  Thanks.)

ANSI/ISO C Defect report #rfg7:
---

In section 3.6.6.4 (The Return Statement) of the ("classic") ANSI C standardxi
it says:

> "If the expression has a type different from that of the function
> in which it appears, it is converted as if it were assigned to an
> object of that type."

This is nonsensical.  The type of the containing function is a function
type... and that's different from an object type.

I believe that should be changed to read:

> "If the expression has a type different from that of the return type of the function in which it appears, it is converted as if it were assigned to an object having the same type as the return type of the containing function."

-- Ronald F. Guilmette, Sunnyvale, California ----------
------ domain address: rfg@netcom.com ----------
------ uucp address: ...!uunet!netcom.com!rfg ----------
••
••

From netcom.com!segfault!rfg Tue Nov 2 05:42:05 1993 remote from uunet
Received: by plauger.UUCP (UUL1.3#20134)
from uunet with UUCP; Tue, 2 Nov 93 07:40:37 EST
Received: from netcomsv.netcom.com (via uucp6.netcom.com) by relay1.UU.NET with SMTP
(5.61/UUNET-internet-primary) id AA26125; Tue, 2 Nov 93 05:42:05 -0500
Received: from segfault.UUCP by netcomsv.netcom.com with UUCP (4.1/SMI-4.1)
id AA09592; Tue, 2 Nov 93 02:42:46 PST
From: uunet!netcom.com!segfault!rfg
Received: from segfault (localhost) by segfault. (5.0/SMI-4.1)
id AA09220; Tue, 2 Nov 93 02:40:20 PST
To: plauger!pjp (P.J. Plauger)
Subject: ANSI/ISO C Defect report #rfg8
Reply-To: uunet!netcom.com!segfault!rfg
Date: Tue, 02 Nov 1993 02:40:19 -0800
Message-Id: <9219.752236819@segfault>
Content-Length: 2030

(Please submit the following defect report for me. Thanks.)

ANSI/ISO C Defect report #rfg8:
---

Section 3.3.2.2 (Function Calls) of the ("classic") ANSI C standard says:

> "If the expression that denotes the called function has a type which includes a prototype, the arguments are implicitly converted, as if by assignment, to the types of the corresponding parameters."

The problem with this statement is the phrase "as if by assignment". The above rule fails to yield an unambiguous meaning in cases where an assignment of the actual to the formal would be prohibited by other rules of the language, as in:

```
void callee (const int formal);
int actual;
void caller () { callee(actual); }
```

(Here, the name of the formal parameter `formal' may be initialized but not assigned to... because it is a non-modifiable lvalue.)

A similar problem exists within section 3.6.6.4 (The Return Statement) of the ("classic") ANSI C standard. It says:

> "If the expression has a type different from that of the function in which it appears, it is converted as if it were assigned to an object of that type."

This statement leaves the validity of the following code open to question:

```
const int returner () { return 99; }
```

Last but not least, section 3.5.7 (Initialization) of the ("classic") ANSI C standards says:

> "The initializer for a scalar shall be a single expression, optionally enclosed in braces. The initial value of the object is that of the expression; the same type constraints and conversions as for simple assignment apply."

This statement leaves the validity of the following code open to question:

```
const int i = 99;
```

(Note that *assignment* to the data object `i' is not normally permitted, as its name does not represent a modifiable lvalue.)

-- Ronald F. Guilmette, Sunnyvale, California ----------
------ domain address: rfg@netcom.com ----------
------ uucp address: ...!uunet!netcom.com!rfg ----------

••
••

From netcom.com!segfault!rfg Wed Nov 3 03:12:29 1993 remote from uunet
Received: by plauger.UUCP (UUL1.3#20134)
from uunet with UUCP; Wed, 3 Nov 93 07:17:45 EST
Received: from netcomsv.netcom.com (via uucp5.netcom.com) by relay2.UU.NET with SMTP
(5.61/UUNET-internet-primary) id AA28960; Wed, 3 Nov 93 03:12:29 -0500
Received: from segfault.UUCP by netcomsv.netcom.com with UUCP (4.1/SMI-4.1)
id AA25110; Wed, 3 Nov 93 00:10:42 PST
From: uunet!netcom.com!segfault!rfg
Received: from segfault (localhost) by segfault. (5.0/SMI-4.1)
id AA10749; Wed, 3 Nov 93 00:05:13 PST
To: plauger!pjp (P.J. Plauger)
Subject: ANSI/ISO C Defect report #rfg9
Reply-To: uunet!netcom.com!segfault!rfg
Date: Wed, 03 Nov 1993 00:05:12 -0800
Message-Id: <10748.752313912@segfault>
Content-Length: 1438

(Please submit the following defect report for me. Thanks.)

ANSI/ISO C Defect report #rfg9:
---

Section 3.5 of the ("classic") ANSI C standard (constraints subsection) says:

> "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 for tags as specified in 3.5.2.3."

Section 3.5.2.3 of the ("classic") ANSI C standard (semantics subsection) says:

> "Subsequent declarations {of a tag} in the same scope shall omit the bracketed list."

Given that one of the above two rules appears in a constraints subsection, while the other appears in a semantics subsection, it is ambiguous whether or not diagnostics are strictly required in the following cases (in which more than one defining declaration of each tag appears within a single scope):

```
void example ()
{
        struct S { int member; };
        struct S { int member; };  /* diagnostic required? */

        union U { int member; };
        union U { int member; };   /* diagnostic required? */

        enum E { member };
        enum E { member };         /* diagnostic required? */
}
```

-- Ronald F. Guilmette, Sunnyvale, California ----------
------ domain address: rfg@netcom.com ----------
------ uucp address: ...!uunet!netcom.com!rfg ----------

••
••

From netcom.com!segfault!rfg Wed Nov 3 19:44:11 1993 remote from uunet
Received: by plauger.UUCP (UUL1.3#20134)
from uunet with UUCP; Wed, 3 Nov 93 20:34:20 EST
Received: from netcomsv.netcom.com (via uucp4.netcom.com) by relay1.UU.NET with SMTP
(5.61/UUNET-internet-primary) id AA20803; Wed, 3 Nov 93 19:44:11 -0500
Received: from segfault.UUCP by netcomsv.netcom.com with UUCP (4.1/SMI-4.1)
id AA13368; Wed, 3 Nov 93 16:43:53 PST
From: uunet!netcom.com!segfault!rfg
Received: from segfault (localhost) by segfault. (5.0/SMI-4.1)
id AA13077; Wed, 3 Nov 93 16:07:44 PST

To: plauger!pjp (P.J. Plauger)
Subject: ANSI/ISO C Defect report #rfg10
Reply-To: uunet!netcom.com!segfault!rfg
Date: Wed, 03 Nov 1993 16:07:43 -0800
Message-Id: <13076.752371663@segfault>
Content-Length: 1503

(Please submit the following defect report for me. Thanks.)

ANSI/ISO C Defect report #rfg10:
---

According to section 3.5 of the ("classic") ANSI C standard:

> "If an identifier for an object is declared with no linkage, the type for the object shall be complete by the end of its declarator, or by the end of its init-declarator if it has an initializer."

Note that this rule appears in a semantics section, so it would seem that comformant implementations are permitted but not strictly required to produce diagnostics for violations of this rule.

Anyway, my interpretation of the above rule is that conformant implementations are permitted (and even encouraged it would seem) to issue diagnostics for code such as the following, in which formal parameters for functions (which, by definition, have no linkage) are declared to have incomplete types:

```
typedef int AT();

void example1 (int arg());      // diagnostic permitted/encouraged?
void example2 (AT arg);                 // diagnostic permitted/encouraged?
```

I believe that section 3.5 needs to be reworded so as to clarify that code such as that shown above is perfectly valid ANSI/ISO C code, and that conformant implementations should not reject such code out of hand.

-- Ronald F. Guilmette, Sunnyvale, California ----------
------ domain address: rfg@netcom.com ----------
------ uucp address: ...!uunet!netcom.com!rfg ----------
••
••
From netcom.com!segfault!rfg Wed Nov 3 19:59:49 1993 remote from uunet
Received: by plauger.UUCP (UUL1.3#20134)
from uunet with UUCP; Wed, 3 Nov 93 20:34:24 EST
Received: from netcomsv.netcom.com (via uucp5.netcom.com) by relay1.UU.NET with SMTP
(5.61/UUNET-Internet-primary) id AA27007; Wed, 3 Nov 93 19:59:49 -0500
Received: from segfault.UUCP by netcomsv.netcom.com with UUCP (4.1/SMI-4.1)
id AA25080; Wed, 3 Nov 93 17:00:31 PST

From: uunet!netcom.com!segfault!rfg
Received: from segfault (localhost) by segfault. (5.0/SMI-4.1)
id AA13427; Wed, 3 Nov 93 16:56:53 PST
To: plauger!pjp (P.J. Plauger)
Subject: ANSI/ISO C Defect report #rfg11
Reply-To: uunet!netcom.com!segfault!rfg
Date: Wed, 03 Nov 1993 16:56:52 -0800
Message-Id: <13426.752374612@segfault>
Content-Length: 1358

(Please submit the following defect report for me. Thanks.)

ANSI/ISO C Defect report #rfg11:
---

According to section 3.5 of the ("classic") ANSI C standard:

> "If an identifier for an object is declared with no linkage, the type for the object shall be complete by the end of its declarator, or by the end of its init-declarator if it has an initializer."

It would appear that the above rule effectly renders the following code "not strictly conforming" (because this code violates the above rule):

```
typedef struct incomplete_S ST;
typedef union  incomplete_U UT;

void example1 (ST arg);          // diagnostic permitted/encouraged?
void example2 (UT arg);          // diagnostic permitted/encouraged?
```

I have noted however that many/most/all "conforming" implementations do in fact accept code such as that shown above (without producing any diagnostics).

Is it the intention of X3J11 that code such as that shown above should be considered to be "strictly conforming"? If so, then some change to the wording now present in section 3.5 is in order (to allow for such cases).

-- Ronald F. Guilmette, Sunnyvale, California ----------
------ domain address: rfg@netcom.com ----------
------ uucp address: ...!uunet!netcom.com!rfg ----------
••
••
From netcom.com!segfault!rfg Thu Nov 4 00:44:37 1993 remote from uunet
Received: by plauger.UUCP (UUL1.3#20134)
from uunet with UUCP; Thu, 4 Nov 93 05:42:32 EST
Received: from netcomsv.netcom.com (via uucp4.netcom.com) by relay1.UU.NET with SMTP

(5.61/UUNET-internet-primary) id AA10392; Thu, 4 Nov 93 00:44:37 -0500
Received: from segfault.UUCP by netcomsv.netcom.com with UUCP (4.1/SMI-4.1)
id AA26505; Wed, 3 Nov 93 21:44:17 PST
From: uunet!netcom.com!segfault!rfg
Received: from segfault (localhost) by segfault. (5.0/SMI-4.1)
id AA13876; Wed, 3 Nov 93 21:25:12 PST
To: plauger!pjp (P.J. Plauger)
Subject: ANSI/ISO C Defect report #rfg12
Reply-To: uunet!netcom.com!segfault!rfg
Date: Wed, 03 Nov 1993 21:25:11 -0800
Message-Id: <13875.752390711@segfault>
Content-Length: 1890

(Please submit the following defect report for me. Thanks.)

ANSI/ISO C Defect report #rfg12:
---

Section 3.5 of the ("classic") ANSI C standard says (in its constraints section):

> "All declarations in the same scope that refer to the same object or function shall specify compatible types."

However in section 3.1.2.6 we have the following rule:

> "All declarations that refer to the same object or function shall have compatible type; otherwise the behavior is undefined."

There is a conflict between the meaning of these two rules. The former rule indicates declaring something in two or more incompatible ways (in a given scope) *must* cause a diagnostic, while the latter rule indicates that doing the exact same thign may result in undefined behavior (i.e. possibly silent acceptance of the code by the implementation).

Furthermore, the use of the term "refer to" in both of these rules seems both unnecessary and potentially confusing. Why not just talk instead about declarations "declaring" things, rather than "referring to" those things?

To eliminate the first problem I would suggest that the rules quoted above from section 3.1.2.6 should be clarified as follows:

> "If any pair of declarations of the same object or function which appear in different scopes declare the object or function in question to have two different incompatible types, the behavior is undefined."

(Actually the rule regarding declaration compatability which now appears in 3.1.2.6 seems entirely misplaced anyway. Shouldn't it just be taken

out of 3.1.2.6 and moved to the section on declarations, i.e. 3.5?)

-- Ronald F. Guilmette, Sunnyvale, California ----------
------ domain address: rfg@netcom.com ----------
------ uucp address: ...!uunet!netcom.com!rfg ----------
••