Discussion of C11 DR452
N1888

Blaine Garst
October 28, 2014
revised Nov 5 with N#

In DR452 (derived from N1762) Shao Miller asks several questions regarding effective types of non-lvalue expressions. The following issues are derived from DR452 after some small corrections in the examples.

Issue 1: Given

```
union u2 {
    int x;
    long y;
    char ca[2];
  };

int func2(void) {
    union u2 o2 = { .ca = "a" };
```

what is the result of the expression `(0,o2).ca == o2.ca` ?

Based on

6.5.17 Comma Operator p 2 ...the right operand is evaluated; the result has its type and value

(which is not an lvalue), and

> 6.2.4 p 8 A non-lvalue expression with structure or union type, where the structure or union contains a member with array type (including, recursively, members of all contained structures and unions) refers to an object with automatic storage duration and *temporary lifetime*. 36)

seems to require that the answer be false. This defeats the obvious optimization.

Issue 2: The effective type rule from 6.5.p6

The effective type of an object for an access to its stored value is the declared type of the object, if any.87) If a value is stored into an object having no declared type through an lvalue having a type that is not a character type, then the type of the lvalue becomes the effective type of the object for that access and for subsequent accesses that do not modify the stored value. If a value is copied into an object having no declared type using memcpy or memmove, or is copied as an array of character type, then the effective type of the modified object for that access and for subsequent accesses that do not modify the value is the effective type of the object from which the value is copied, if it has one. For all other accesses to an object having no declared type, the effective type of the object is simply the type of the lvalue used for the access.

does not seem to apply to an object with temporary lifetime resulting from a comma expression. As such, it does not seem to have an effective type. As such, type punning is seemingly allowed:

```
long func3() {
    union u2 o3 = { .x=42 };

    return (0, o3).y;
}
```

Another consequence is with regard to aliasing

```
struct s1 {
    int x;
    long y;
};

void func4(long * lp, struct s1 *s1p) {
     int c;

     for (c = 0; c < (0, *s1p).x; ++c) {
           --*lp;
     ]
}
```

and in another translation unit

```
struct s1 o1 = { 20 };
func4(&o1.y, &o1);
```

the lack of the effective type for `(0, *stp)` would disallow the optimization of computing `(0, *s1p).x` only once.

This second issue is resolved by defining an effective type of `(0, o1)`.