Name

n3294 − The *[[restrict()]]* function attribute as a replacement of the *restrict* qualifier

Category

Feature and deprecation.

Author

Alejandro Colomar Andres; maintainer of the Linux man−pages project.

Cc

GNU C library
GNU Compiler Collection
Linux man-pages
Paul Eggert
Xi Ruoyao
Jakub Jelinek
Martin Uecker
LIU Hao
Jonathan Wakely
Richard Earnshaw
Sam James
Emanuele Torre
Ben Boeckel
"Eissfeldt, Heiko"
David Malcolm

Description

restrict qualifier

The *restrict* qualifier is not useful for diagnostics. Being defined in terms of accesses, the API is not enough for a caller to know what the function will do with the objects it receives.

That is, a caller cannot know if the following call is correct:

```
void f(const int *restrict a, int **restrict b);

f(a, &a);
```

Having no way to determine if a call will result in Undefined Behavior makes it a dangerous qualifier.

The reader might notice that this prototype and call is very similar to the prototype of *strtol*(3), and the use reminds of a relatively common use of that function.

Diagnostics

A good replacement of the *restrict* qualifier should allow to specify in the API of the following function that it doesn't accept pointers that alias.

```
void
replace(const T *restrict new, T **restrict ls, size_t pos)
{
      memcpy(ls[pos], new, sizeof(T));
}
```

This proposal suggests the following:

```
[[restrict(1)]] [[restrict(2)]]
void
replace(const T *restrict new, T **restrict ls, size_t pos);

replace(arr[3], arr, 2);  // UB; can be diagnosed
```

Qualifiers

It is also unfortunate that *restrict* is a qualifier, since it doesn't follow the rules that apply to all other qualifiers. While it is discarded easily, its semantics make it as if it couldn't be discarded.

Function attribute

The purpose of restrict is to

- Allow functions to optimize based on the knowledge that certain objects are not accessed by any other object in the same scope; usually a function boundary, which is the most opaque boundary, and where this information is not otherwise available.
- Diagnose calls that would result in Undefined Behavior under this memory model.

Qualifiers don't seem to be good for carrying this information, but function attributes are precisely for adding information that cannot be expressed by just using the type system.

An attribute would need to be more strict than the *restrict* qualifier to allow diagnosing non-trivial cases, such as the call shown above.

A caller only knows what the callee receives, not what it does with it. Thus, for diagnostics to work, the semantics of a function attribute should be specified in terms of what a function is allowed to receive.

[[restrict]]

The *[[restrict]]* function attribute specifies that the pointer to which it applies is the only reference to the array object to which it points (except that a pointer to one past the last element may overlap another object).

If the number of elements is specified with array notation or a compiler-specific attribute, the array object to be considered is a subobject of the original array object, which is limited by the number of elementsspecified in the function prototype.

For the following prototype:

```
[[restrict(1)]] [[restrict(2)]]
void add_inplace(size_t n, int a[n], const int b[n]);
```

In the following calls, the caller is able to determine with certainty if the behavior is defined or undefined:

```
char  a[100] = ...;
char  b[50] = ...;

add_inplace(50, a, a + 50);  // Ok
add_inplace(50, a, b);       // Ok
add_inplace(50, a, a);       // UB
```

In the first of the three calls, the parameters don't alias inside the function, since the subobjects of 50 elements do not overlap each other, even though they are one single array object to the outer function.

Optimizations

This function attribute allows similar optimizations than those allowed by the *restrict* qualifier.

strtol(3)

In some cases, such as the *strtol*(3) function, the prototype will be different, since this attribute is stricter than *restrict*, and can't be applied to the same parameters. For example, the prototype for *strtol*(3) would be

```
[[restrict(2)]]
long
strtol(const char *str, char **endp, int base);
```

This could affect optimizations, since now it's not clear to the implementation that *str* is not modified by any other reference. Compiler-specific attributes can help with that. For example, the *[[gnu::access()]]* attribute can be used in this function to give more information:

```
[[restrict(2)]]
[[gnu::access(read_only, 1)]]
```

```
[[gnu::access(write_only, 2)]]
long
strtol(const char *str, char **endp, int base);
```

The fact that *endp* is write-only lets the callee deduce that **endp* cannot be used to write to the string (since the callee is not allowed to inspect **endp*).

Another concern is that a global variable such as *errno* might alias the string. This is already a concern in several ISO C calls, such as *rename*(2). But in the case of *strtol*(3), it would be a regression. There are ways to overcome that, such as designing helper functions in a way that the attribute can be applied to add extra information.

It is important that diagnostics are easy to determine, to avoid false negatives and false positives, so that code is easily safe. Optimizations, while important, need not be as easy to apply as diagnostics. If an implementation wants to be optimal, it will do the extra work for being fast.

Multiple aliasing pointers

In some cases, it might be useful to allow specifying that some pointers may alias each other, but not others.

Strings

Another way to determine that *str* cannot be aliased by any other object such as *errno* would be to use an attribute that marks *str* as a string. An object of type *int* shouldn't be allowed to represent a string, so regardless of character types being allowed to alias any other type, an attribute such as *[[gnu::null_terminated_string_arg()]]* might be used to determine that the global *errno* does not alias the string.

Deprecation

The *restrict* qualifier would be deprecated by this attribute, similar to how the *noreturn* function specifier was superseded by the *[[noreturn]]* function attribute.

Backwards compatibility

Removing the *restrict* qualifier from function prototypes does not cause problems in most functions. Only functions with *restrict* applied to a pointee would have incompatible definitions. The only standard functions where this would happen are:

tmpfile_s()
fopen_s()
freopen_s()

Those functions are not widely adopted, so the problem would likely be minimal.

Proposal

6.7.13.x The restrict function attribute

Constraints

The *restrict* attribute shall be applied to a function.

A 1-based index can be specified in an attribute argument clause, to associate the attribute with the corresponding parameter of the function, which must be of a pointer type.

(Optional.) Several indices can be specified, separated by commas.

The attribute can be applied several times to the same function, to mark several parameters with the attribute.

(Optional.) The argument attribute clause may be omitted, which is equivalent to specifying the attribute once for each parameter that is a pointer.

Semantics

If a function is defined with the *restrict* attribute, the corresponding parameter shall be the only reference to the array object that it points to. If the function receives another reference to the same array object, the behavior is undefined. If the function accesses the array object through an lvalue that is not derived from that pointer, the behavior is undefined.

(Optional.) If more than one parameters are specified in the same attribute argument clause, then all of those pointers are allowed to point to the same array object.

If the number of elements is specified with array notation (or a compiler-specific attribute), the array object to be considered for aliasing is a sub-object of the original array object, limited by the number of elements specifiedr [1].

[1] For the following prototype:

```
[[restrict(1)]] [[restrict(2)]]
void f(size_t n, int a[n], const int b[n]);
```

In the the following calls, the caller is able to determine if the behavior is defined or undefined:

```
char a[100] = /*...*/;
char b[50] = /*...*/;

f(50, a, a + 50);  // Ok
f(50, a, b);       // UB; a diagnostic is recommended
f(50, a, a + 2);   // UB; a diagnostic is recommended
```

History

Revisions of this paper:

0.1 Original draft for removing *restrict* from the first parameter of *strtol*(3).

0.2 Incorporate feedback from glibc and gcc mailing lists.

0.3 Re-purpose, to deprecate *restrict* and propose *[[restrict()]]* instead.

See also

The original discussion about *restrict* and *strtol*(3).