Slay Some Earthly Demons I

Document: N3244
Author: Martin Uecker
Date: 2024-05-01

Reviewing the list of operations which have undefined behavior it is suggested to revisit some of them and consider defining behavior for some of them. There are other such operations where defining behavior should be considered, this is a first list of cases which appear to be low-hanging fruits.

Annex J

(19) A non-array lvalue with an incomplete type is used in a context that requires the value of the designated object (6.3.2.1).

Example

https://godbolt.org/z/3bona6TdG

```
struct f *p;

void g(void)
{
   *p;
}
```

Analysis

Unused extension point. It is suggested to modify the definition of lvalue to require a complete type (or an array type which can remain incomplete because it is subject to array-pointer decay rather than lvalue conversion).

Proposed Change

6.3.2.1

1 An lvalue is an expression ~~(~~with an **array type or a complete** object type ~~**other than void)**~~ that potentially designates an object; ...

2 … ~~**If the lvalue has an incomplete type and does not have array type, the behavior is undefined**~~**.** …

6.5.1

3 An identifier primary expression designating an object **with an array type or a complete type** is an lvalue.

Annex J

(21) An lvalue having array type is converted to a pointer to the initial element of the array, and the array object has register storage class (6.3.2.1).

Example

https://godbolt.org/z/s9YMeGnan

Analysis

Pointer-decay of arrays declared with register storage class should be made implementation defined. (In the future, it would be useful to allow array subscription without implying that the address is taken.)

Proposed Change

6.3.2.1

3 Except when it is the operand of the sizeof operator, or typeof operators, or the unary & operator, or is a string literal used to initialize an array, an expression that has type "array of type" is converted to an expression with type "pointer to type" that points to the initial element of the array object and is not an lvalue. If the array object has register storage class, the behavior is ~~**undefined**~~ **implementation-defined**.

(Editorial note: Remove from Annex J.2 and add to J.3)

Additional change: Clarify semantics of "address taken"

3 … If the lvalue **(that does not have array type)** designates an object of automatic storage duration that ~~**could have been declared with the register storage class (**~~never had its address taken~~**)**~~ , and that object is uninitialized (not declared with an initializer and no assignment to it has been performed prior to use), the behavior is undefined. **An address of an object is taken by application of the address operator to an lvalue designating the object, if the object is an element of an array that was designated by an expression that is converted to a pointer as described below, or by being a member of a structure or union object whose address is taken.**

Annex J

(40) A pointer is converted to other than an integer or pointer type (6.5.5).

Example

https://godbolt.org/z/Gf155Tjfa

```
struct f { struct f *x; } *p;
```

```
void g(void)
{
   (struct f)p;
}
```

Analysis

Unused extension point.

Proposed Change

6.5.5

Constraints

4 A pointer type shall ~~**not**~~ be converted **only** to ~~**any floating**~~ **an integer or pointer** type….

Annex J

(56) An identifier for an object is declared with no linkage and the type of the object is incomplete after its declarator, or after its init-declarator if it has an initializer (6.7).

Example

https://godbolt.org/z/WrqeM3588

```
struct f;

void i()
{
   struct f y;
   int i[];
}

void g(struct f x);
```

Analysis

Unused extension point.

Proposed Change

6.7.1

Move to Constraints

5~~**8**~~ If an identifier for an object **that does not have function prototype scope** 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. **XXX)** ~~**In the case of function parameters, it is the adjusted type (see 6.7.7.4) that is required to be complete.**~~

XXX) In function definitions (but not in function declarations that are not part of a function definition) the adjusted type of a parameter needs to be complete (see 6.7.7.4p3 and 6.7.7.4p11).

Annex J

(57) A function is declared at block scope with an explicit storage-class specifier other than extern (6.7.2).

Example

https://godbolt.org/z/sP3rxbWYG

```
void i()
{
   static void f();
}
```

Analysis

Should either be a constraint violation or an explicitly implementation-defined extension. Note that GCC allows auto for forward declarations of nested functions.

Proposed Change

Move to Constraints

Option 1 (constraint violation):

4 ~~**13**~~ The declaration of an identifier for a function that has block scope shall have no explicit storage-class specifier other than extern.

Option 2 (implementation-defined):

4 ~~**13**~~ **For a** ~~**the**~~ declaration of an identifier for a function that has block scope ~~**shall have no**~~ **and where the declaration has an** explicit storage-class specifier other than extern**, the behavior is implementation-defined.**

Annex J

(60) When the complete type is needed, an incomplete structure or union type is not completed in the same scope by another declaration of the tag that defines the content (6.7.3.4).

Analysis

Does not seem to exist anymore as possible behavior.

Proposed Change

Remove from Annex J.2

Annex J

(63) The specification of a function type includes any type qualifiers (6.7.4).

Example

https://godbolt.org/z/Md6r8PE8K

```
typedef int f();

const f g;
```

Analysis

Should be a constraint violation to detect mistakes and / or so it could potentially be used in the future (e.g. wording for N3089 could modify this again).

Proposed Change

6.7.4

Constraints

4 A function type shall not have any qualifiers.

Semantics

10 If the specification of an array type includes any type qualifiers, both the array and the element type are so-qualified. ~~**If the specification of a function type includes any type qualifiers, the behavior is undefined.**~~152)

Additional change: Remove an incorrect use of „shall“ implying undefined behavior.

11 ~~**For**~~ two qualified types **are** ~~**to be**~~ compatible, ~~**both shall have the**~~ **if they are** identically qualified version**s** of a compatible type; the order of type qualifiers within a list of specifiers or

qualifiers does not affect the specified type.

Annex J

(67) A function with external linkage is declared with an inline function specifier, but is not also defined in the same translation unit (6.7.5).

Example

https://godbolt.org/z/YT8vTKME1

```
extern inline void f();
```

Analysis

Compilers should detect this as this indicates a mistake.

Proposed Change

6.7.5

Constraints

4 If a function declaration declaring an identifier with external linkage has an inline function specifier, then the function shall also be defined in the same translation unit.

Semantics

7 Any function with internal linkage can be an inline function. For a function with external linkage, the following restrictions apply: ~~**If a function is declared with an inline function specifier, then it shall also be defined in the same translation unit.**~~ If all …

Annex J

(69) The definition of an object has an alignment specifier and another declaration of that object has a different alignment specifier (6.7.6).

Example

https://godbolt.org/z/qGbcTx5zb

```
_Alignas(32) int x;
int x;

extern _Alignas(32) int y;
extern int y;
```

```
int z;
_Alignas(32) int z;
```

Analysis
Compiler can detect this in the same TU.

Proposed Change

Constraints

6 **If the definition of an object has an alignment specifier, any other declaration of that object in the same translation unit shall either specify equivalent alignment or have no alignment specifier. If the definition of an object does not have an alignment specifier, any other declaration of that object in the same translation unit shall also have no alignment specifier.**

Semantics

8 ~~**If the definition of an object has an alignment specifier, any other declaration of that object shall either specify equivalent alignment or have no alignment specifier. If the definition of an object does not have an alignment specifier, any other declaration of that object shall also have no alignment specifier.**~~ If declarations of an object in different translation units have different alignment specifiers, the behavior is undefined.

Alternative wording for the remaining part in the Semantics Section:

8 ~~**If the definition of an object has an alignment specifier, any other declaration of that object shall either specify equivalent alignment or have no alignment specifier. If the definition of an object does not have an alignment specifier, any other declaration of that object shall also have no alignment specifier. If declarations of an object in different translation units have different alignment specifiers, the behavior is undefined.**~~ **If the definition of an object has an alignment specifier, no declaration of the object in a different translation unit shall specify stricter alignment.**

Annex J

(75) A storage-class specifier or type qualifier modifies the keyword void as a function parameter type list (6.7.7.4).

Analysis

This is now implementation-defined.

Proposed Change

Remove from Annex J.2.

Annex J

(77) A declaration for which a type is inferred contains a pointer, array, or function declarators (6.7.10).

Analysis

This is now implementation-defined.

Proposed Change

Remove from Annex J.2 and add new wording to J.3 (see 6.7.1p12)

Annex J

(78) A declaration for which a type is inferred contains no or more than one declarators (6.7.10)

Analysis

This is now implementation-defined.

Proposed Change

Remove from Annex J.2 and add new wording to J.3 (see 6.7.1p12)

Annex J

(79) The value of an unnamed member of a structure or union is used (6.7.11).

Analysis

Does not seem to exist anymore as possible behavior.

Proposed Change

Remove from Annex J.2

Annex J

(80) The initializer for a scalar is neither a single expression, nor an empty initializer, nor a single expression enclosed in braces (6.7.11).

(81) The initializer for a structure or union object is neither an initializer list nor a single expression that has compatible structure or union type (6.7.11).

(82) The initializer for an aggregate or union, other than an array initialized by a string literal, is not a brace-enclosed list of initializers for its elements or members (6.7.11).

Example

Proposed Change

Constraints

3 The type of the entity to be initialized shall be an array of unknown size or a complete object type.

4 The initializer for a scalar shall be a single expression, optionally enclosed in braces, or it shall be an empty initializer.

5 The initializer for an object that has structure or union type, shall be either a single expression that has compatible type or a brace-enclosed list of initializers for the elements or named members.

6 The initializer for an array shall be either a string literal, optionally enclosed in braces, or a brace-enclosed list of initializers for the elements or named members.
An array initialized by character string literal or UTF-8 string literal shall have a character type as element type, and an array initialized with a wide string literal shall have element type compatible with a qualified or unqualified wchar_t, char16_t, or char32_t.
~~4 The type of the entity to be initialized shall be an array of unknown size or a complete object type.~~ An entity of variable length array type shall not be initialized except by an empty initializer. An array of unknown size shall not be initialized by an empty initializer.

Semantics

12 **~~The initializer for a scalar shall be a single expression, optionally enclosed in braces, or it shall be an empty initializer.~~** If the initializer **for a scalar** is not the empty initializer, the initial value of the object is that of the expression (after conversion); the same type constraints and conversions as for simple assignment apply, taking the type of the scalar to be the unqualified version of its declared type

14 **~~The initializer for a structure or union object shall be either an initializer list as described below, or a single expression that has compatible structure or union type. In the latter case,~~ If the initializer for a struct or union object is a single expression,** the initial value of the object, including unnamed members, is that of the expression.179)

15 **For** an array of character type **~~may be~~** initialized by a character string literal or UTF-8 string literal, **~~optionally enclosed in braces.~~** successive bytes of the string literal (including the terminating null character if there is room or if the array is of unknown size) initialize the elements of the array.

16 **For** an array **~~with element type compatible with a qualified or unqualified wchar_t, char16_t, or char32_t may be~~** initialized by a wide string literal**, ~~with the corresponding encoding prefix (L, u, or U, respectively), optionally enclosed in braces.~~** Successive wide characters of the wide string literal (including the terminating null wide character if there is room or

if the array is of unknown size) initialize the elements of the array.

~~17 Otherwise, the initializer for an object that has aggregate or union type shall be a brace-enclosed list of initializers for the elements or named members.~~

Annex J

(87) An identifier for an object with internal linkage and an incomplete type is declared with a tentative definition (6.9.3).

Analysis

Unused extension point.

Example

https://godbolt.org/z/1Ejfajfff

```
struct foo;
static struct foo x;
```

Proposed Change

Move to Constraints Section:

1~~3~~ If the declaration of an identifier for an object is a tentative definition and has internal linkage, the declared type shall not be an incomplete type

---

Annex J

(58) A structure or union is defined without any named members (including those specified indirectly via anonymous structures and unions) (6.7.3.2).

https://godbolt.org/z/Gf6Ycnvce

Analysis

This may lead to zero-sized objects (which are widely supported but would need much more consideration). It is suggested to make this implementation-defined.

Proposed Change

6.7.3.2

10 The member declaration list is a sequence of declarations for the members of the structure or union. If the member declaration list does not contain any named members, either directly or via an anonymous structure or anonymous union, the behavior is **~~undefined~~ implementation-defined.**