Type System

struct symbol is used to represent symbols & types but most parts pertaining to the types are in the field 'ctype'. For the purpose of this document, things can be simplified into:

```
struct symbol {
        enum type type; // SYM_...
        struct ctype {
                struct symbol *base_type;
                unsigned long modifiers;
                unsigned long alignment;
                struct context_list *contexts;
                struct indent *as;
        };
};
```

  - type
  - size_bits
  - rank
  - variadic
  - string
  - designated_init
  - forced_arg
  - accessed
  - transparent_union

- `base_type` is used for the associated base type.
- `modifiers` is a bit mask for type specifiers (MOD_UNSIGNED, ...), type qualifiers (MOD_CONST, MOD_VOLATILE), storage classes (MOD_STATIC, MOD_EXTERN, ...), as well for various attributes. It's also used internally to keep track of some states (MOD_ACCESS or MOD_ADDRESSABLE).
- `alignment` is used for the alignment, in bytes.
- `contexts` is used to store the informations associated with the attribute `context()`.
- `as` is used to hold the identifier of the attribute `address_space()`.

Kind of types

SYM_BASETYPE

Used by integer, floating-point, void, 'type', 'incomplete' & bad types.

  - .ctype.base_type points to `int_ctype`, the generic/abstract integer type

- .ctype.modifiers has MOD_UNSIGNED/SIGNED/EXPLICITLY_SIGNED set accordingly.

For floating-point types:

- .ctype.base_type points to `fp_ctype`, the generic/abstract float type
- .ctype.modifiers is zero.

For the other base types:

- .ctype.base_type is NULL
- .ctype.modifiers is zero.

SYM_NODE

It's used to make variants of existing types. For example, it's used as a top node for all declarations which can then have their own modifiers, address_space, contexts or alignment as well as the declaration's identifier.

Usage:

- .ctype.base_type points to the unmodified type (which must not be a SYM_NODE itself)
- .ctype.modifiers, .as, .alignment, .contexts will contains the 'variation' (MOD_CONST, the attributes, ...).

SYM_PTR

For pointers:

- .ctype.base_type points to the pointee type
- .ctype.modifiers & .as are about the pointee too!

SYM_FN

For functions:

- .ctype.base_type points to the return type
- .ctype.modifiers & .as should be about the function itself but some return type's modifiers creep here (for example, in int foo(void), MOD_SIGNED will be set for the function).

SYM_ARRAY

For arrays:

- .ctype.base_type points to the underlying type
- .ctype.modifiers & .as are a copy of the parent type (and unused)?
- for literal strings, the modifier also contains MOD_STATIC
- sym->array_size is *expression* for the array size.

SYM_STRUCT

For structs:

- .ctype.base_type is NULL
- .ctype.modifiers & .as are not used?
- .ident is the name tag.

SYM_UNION

Same as for structs.

SYM_ENUM

For enums:

- .ctype.base_type points to the underlying type (integer)
- .ctype.modifiers contains the enum signedness
- .ident is the name tag.

SYM_BITFIELD

For bitfields:

- .ctype.base_type points to the underlying type (integer)
- .ctype.modifiers & .as are a copy of the parent type (and unused)?
- .bit_size is the size of the bitfield.

SYM_RESTRICT

Used for bitwise types (aka 'restricted' types):

- .ctype.base_type points to the underlying type (integer)
- .ctype.modifiers & .as are like for SYM_NODE and the modifiers are inherited from the base type with MOD_SPECIFIER removed
- .ident is the typedef name (if any).

SYM_FOULED

Used for bitwise types when the negation op (~) is used and the bit_size is smaller than an `int`. There is a 1-to-1 mapping between a fouled type and its parent bitwise type.

Usage:

- .ctype.base_type points to the parent type
- .ctype.modifiers & .as are the same as for the parent type
- .bit_size is bits_in_int.

SYM_TYPEOF

Should not be present after evaluation:

- .initializer points to the expression representing the type
- .ctype is not used.

Typeofs with a type as argument are directly evaluated during parsing.

SYM_LABEL

Used for labels only.

SYM_KEYWORD

Used for parsing only.

SYM_BAD

Should not be used.

SYM_UNINTIALIZED

Should not be used.