Numeric:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x + y | x – y | x \* y | x / y | x % y | x \*\* y |
| x ~ y | x++ | ++x | x-- | --x |  |

Compare:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x == y | x === y | x != y | x !== y | x < y | x > y |
| x <=> y | ?x |  |  |  |  |

Ternary:

|  |  |
| --- | --- |
| (x) => y ; z | w (x) <= y ; z |

Access, dereference, execute:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x.y | x->y | \*x | x.\*y | x->\*y | x(y) |
| x[y] | x<y> | y“x”z |  |  |  |

Logical:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| !x | x && y | x || y | x ^^ y |  |  |

Bitwise:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x << y | x >> y | !!x | x & y | x | y | x ^ y |
| x !& y |  |  |  |  |  |

Assignment:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x = y | x =+ y | x =– y | x =\* y | x =/ y | x =% y |
| x =^ y | x =~ y | x =! y | x =<< y | x =>> y | x =!! y |
| x =& y | x =| y | x =^ y | x =!& y | x ??= y |  |

You can override operators behaviour or define your own following the syntax below, however, ternary, dereference, access and execute operators cannot be overridden. Only binary custom operators are supported.

*override operator++(void) -> any {*

*<? pre-increment operator redefinition here ?>*

*}*

*operator &==:> (any var) -> any {*

*<? Custom operator definition here ?>*

*}*

*operator feet (any var) -> any {*

*<? Custom operator definition here ?>*

*}*

ASCII characters only are allowed for custom operators. Unlike symbolic operators operators that consist of letters must always be separated by whitespaces. The operator name cannot contain a mixture of letters and symbols, you either use one or another. Maximum length for a custom operator is 8 symbols.

**Operator priority:**

From left to right. Use parentheses to override the order (they still follow the rule though)

x = 2 + 2 \* 2 \*\* 2 + (2 \*\* 5) ~ 2 *<? 10 ?>*

*//нужно продумать детальней сей вопрос*

Consisting of letters custom operators must not start with a digit nor contain whitespaces or tab characters. There’s no such a thing as identifier that contains a whitespace or a tab character.

Data types:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Numeric |  |  |  |  |  |
| *dec* | *hex* | *bin* | *enum* |  |  |
| Structures |  |  |  |  |  |
| *enum* | *chain* | *union* | *vector* |  |  |
| Other |  |  |  |  |  |
| *void* | *any* | *string* | *fun* | *tuple* |  |

Type qualifiers:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *const* | *restrict* | *ptr* | *variadic* | *export* | *persist* | *virtual* |
| *override* | *final* | *consteval* | *constinit* | *public* | *private* | *protected* |
| *unsigned* | *real* | *extended* | *thread* | *atomic* | *default* | *unsafe* |

Hidden namespaces:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *inner* | *outer* |  |  |  |  |

Comments: *<? Your comment here ?>*

Identifiers must not:

* Start with a digit;
* Have any of the operator-related characters within them;
* Contain whitespaces or tab characters.

You can define a type alias using *def* keyword:

def dec = int

Type aliases can be overridden at any time:

int = void

Alternatively, you can use macros.