Miranda Operators and other Lexical Notation

`||`	Start of comment line
`%`	Start of directive line
`+`	Add operator
`-`	Subtract/negate operator
`*`	Multiply operator
`/`	Division operator
`div`	Integer division operator
`mod`	Modulus/remainder operator
`^`	Raise-to-the-power operator
`~`	Logical NOT operator
`&`	Conditional AND operator
`\/`	Conditional OR operator
`=`	Assignment operator Equal-to operator
`~=`	Not-equal-to operator
`<`	Less-than operator
`>`	Greater-than operator
`<=`	Less-than-or-equal-to operator
`>=`	Greater-than-or-equal-to operator
`,`	Guard/Case specifier in conditional expressions
`if`	Specifier to evaluate conditional expressions
`otherwise`	Specifier for default guards/cases
`|`	Separator in list comprehension
`<-`	List comprehension generator
`;`	Filter specifier in list comprehension
`.`	Function composition operator
`where`	Clause to specify local definitions
`:`	Append-head operator ("cons")
`++`	List concatenation operator
`--`	List subtraction operator
`#`	List length/size operator
`!`	List indexing operator
`..`	Range operator
`::`	Function type declarator Type query operator in interpreter ("has type")
`::=`	Algebraic data type declarator
`==`	Type synonym declarator
`abstype`	Abstract data type declarator
`with`	Abstract data type signatures declarator
`type`	Empty data type
`()`	Empty value in IO () type
`show`	Represents a value as a string
`readvals`	Reads values from a file
`$+`	Reads values from keyboard input
`[ and ]`	List constructors; "," as separator
`( and )`	Tuple constructors; "," as separator Infix-to-Prefix constructors
`$`	Prefix-to-Infix constructor
`' and '`	Literal char constructors
`" and "`	String constructors