Contents

Comparisons and Ordering

```
(==)   :  {a}   (Cmp a) => a -> a -> Bit
(!=)   :  {a}   (Cmp a) => a -> a -> Bit
(===)  :  {a,b} (Cmp b) => (a -> b) -> (a -> b) -> a -> Bit
(!==)  :  {a,b} (Cmp b) => (a -> b) -> (a -> b) -> a -> Bit

(<)    :  {a} (Cmp a) => a -> a -> Bit
(>)    :  {a} (Cmp a) => a -> a -> Bit
(<=)   :  {a} (Cmp a) => a -> a -> Bit
(>=)   :  {a} (Cmp a) => a -> a -> Bit

min    :  {a} (Cmp a) => a -> a -> a
max    :  {a} (Cmp a) => a -> a -> a

instance Cmp Bit
// No instance for functions.
instance (Cmp a, fin n) => Cmp [n]a
instance (Cmp a, Cmp b) => Cmp (a, b)
instance (Cmp a, Cmp b) => Cmp { x : a, y : b }
instance                   Cmp Integer
instance (fin n, n>=1)  => Cmp (Z n)
instance                   Cmp Rational
```

Signed Comparisons

```
(<$) : {a} (SignedCmp a) => a -> a -> Bit
(>$) : {a} (SignedCmp a) => a -> a -> Bit
(<=$) : {a} (SignedCmp a) => a -> a -> Bit
(>=$) : {a} (SignedCmp a) => a -> a -> Bit

// No instance for Bit
// No instance for functions.
instance (fin n, n >= 1)            => SignedCmp [n]
instance (SignedCmp a, fin n)       => SignedCmp [n]a
        // (for [n]a, where a is other than Bit)
instance (SignedCmp a, SignedCmp b) => SignedCmp (a, b)
instance (SignedCmp a, SignedCmp b) => SignedCmp { x : a, y : b }
```

Zero

```
zero       : {a} (Zero a) => a
```

Every base and structured type in Cryptol is a member of class `Zero`.

Arithmetic

```
(+)     : {a} (Ring a) => a -> a -> a
(-)     : {a} (Ring a) => a -> a -> a
(*)     : {a} (Ring a) => a -> a -> a
negate  : {a} (Ring a) => a -> a

(/)     : {a} (Integral a) => a -> a -> a
(%)     : {a} (Integral a) => a -> a -> a

(/.)    : {a} (Field a) => a -> a -> a
recip   : {a} (Field a) => a -> a

floor   : {a} (Round a) => a -> Integer
ceiling : {a} (Round a) => a -> Integer
trunc   : {a} (Round a) => a -> Integer
round   : {a} (Round a) => a -> Integer

(^^)    : {a, e} (Ring a, Integral e) => a -> e -> a

(/$)    : {n} (fin n, n >= 1) => [n] -> [n] -> [n]
(%$)    : {n} (fin n, n >= 1) => [n] -> [n] -> [n]
lg2     : {n} (fin n) => [n] -> [n]
```

The prefix notation `- x` is syntactic sugar for `negate x`.

```
// No instance for `Bit`.
instance (fin n)          => Ring ([n]Bit)
instance (Ring a)         => Ring ([n]a)
instance (Ring b)         => Ring (a -> b)
instance (Ring a, Ring b) => Ring (a, b)
instance (Ring a, Ring b) => Ring { x : a, y : b }
instance                      Ring Integer
instance (fin n, n>=1)    => Ring (Z n)
instance                      Ring Rational
```

Note that because there is no instance for `Ring Bit` the top two instances do not actually overlap.

```
instance                  Integral Integer
instance (fin n)       => Integral ([n]Bit)

instance Field Rational

instance Round Rational
```

Boolean

```
False      : Bit
True       : Bit

(&&)       : {a} (Logic a) => a -> a -> a
(||)       : {a} (Logic a) => a -> a -> a
(^)        : {a} (Logic a) => a -> a -> a
complement : {a} (Logic a) => a -> a

(==>)      : Bit -> Bit -> Bit
(/\)       : Bit -> Bit -> Bit
(\/)       : Bit -> Bit -> Bit

instance                        Logic Bit
instance (Logic a)          => Logic ([n]a)
instance (Logic b)          => Logic (a -> b)
instance (Logic a, Logic b) => Logic (a, b)
instance (Logic a, Logic b) => Logic { x : a, y : b }
// No instance for `Logic Integer`.
// No instance for `Logic (Z n)`.
// No instance for `Logic Rational`.
```

Sequences

```
join      : {parts,each,a} (fin each) => [parts][each]a -> [parts * each]a
split     : {parts,each,a} (fin each) => [parts * each]a -> [parts][each]a

(#)       : {front,back,a} (fin front) => [front]a -> [back]a -> [front + back]a
splitAt   : {front,back,a} (fin front) => [from + back] a -> ([front] a, [back] a)
```

```
reverse   : {n,a} (fin n) => [n]a -> [n]a
transpose : {n,m,a} [n][m]a -> [m][n]a

(@)         : {n,a,ix}   (Integral ix) =>  [n]a -> ix    -> a
(@@)        : {n,k,ix,a} (Integral ix) =>  [n]a -> [k]ix -> [k]a
(!)         : {n,a,ix}   (fin n, Integral ix) => [n]a -> ix    -> a
(!!)        : {n,k,ix,a} (fin n, Integral ix) => [n]a -> [k]ix -> [k]a
update      : {n,a,ix}   (Integral ix)        => [n]a -> ix -> a -> [n]a
updateEnd   : {n,a,ix}   (fin n, Integral ix) => [n]a -> ix -> a -> [n]a
updates     : {n,k,ix,a} (Integral ix, fin k) => [n]a -> [k]ix -> [k]a -> [n]a
updatesEnd  : {n,k,ix,d} (fin n, Integral ix, fin k) => [n]a -> [k]ix -> [k]a -> [n]a

take        : {front,back,elem} (fin front) => [front + back]elem -> [front]elem
drop        : {front,back,elem} (fin front) => [front + back]elem -> [back]elem
head        : {a, b} [1 + a]b -> b
tail        : {a, b} [1 + a]b -> [a]b
last        : {a, b} [1 + a]b -> b

groupBy     : {each,parts,elem} (fin each) => [parts * each]elem -> [parts][each]elem
```

Function `groupBy` is the same as `split` but with its type arguments in a different order.

Shift And Rotate

```
(<<)   : {n,ix,a} (Integral ix, Zero a) => [n]a -> ix -> [n]a
(>>)   : {n,ix,a} (Integral ix, Zero a) => [n]a -> ix -> [n]a
(<<<)  : {n,ix,a} (fin n, Integral ix)  => [n]a -> ix -> [n]a
(>>>)  : {n,ix,a} (fin n, Integral ix)  => [n]a -> ix -> [n]a

// Arithmetic shift only for bitvectors
(>>$)  : {n,ix} (fin n, n >= 1, Integral ix) => [n] -> ix -> [n]
```

Random Values

```
random    : {a} => [256] -> a
```

Debugging

```
 undefined   : {a} a
 error       : {n a} [n][8] -> a
 trace       : {n, a, b} (fin n) => [n][8] -> a -> b -> b
 traceVal    : {n, a} (fin n) => [n][8] -> a -> a
```