

## Statements

**Declaration:**

```
var i: int;
var i: int := 5, j: real;
var i: int :- Find();
```

**Assignment:**

```
i := 5;
i :| i > 0;
f :- Find(5);
i, j, k := k, j, i;
i, j, k := m();
i := *;
```

**Method Call:**

```
m(5,6,7);
i, j, k := p(5,6,7);
i, j, k :- p(5,6,7);
```

**Conditional:**

```
if ... { ... } else ...
if x: int | P(x) { ... } else ...
if case ... => ... case ... => ...
match s case p => ... case q => ...
```

**Loop:**

```
while ... { ... }
while case ... => ... case ... => ...
for i: int := ... to ... { ... }
for i: int := ... downto ... { ... }
break;
continue;
```

**Labeled:**

```
label L: ...
```

**Forall:**

```
forall i | 0 <= i < j { ... }
forall e <- s { ... }
```

**Others:**

```
{ ... }
return ;
return ..., ...;
yield ;
yield ..., ...;
assert ... ;
assume ... ;
expect ..., msg ;
print ..., ..., ...;
reveal ..., ..., ... ;
modify ..., ..., ... ;
calc { ... ; == ... ; == ... ; }
```

## Expressions

**Logical Operators:**

```
<==> ==> <== && || !
```

**Comparison operators:**

```
== != < <= > >= !! in !in
```

**Infix and Unary operators:**

```
+ - * / % | & ^ ! << >>
```

**Conditional:**

```
if ... then ... else ...
match ... case ... => ... case ... => ...
```

**Tests and Conversions:**

```
e is Type
e as Type
```

**Lambda expression:**

```
i => i*i
(i, j) => i+j
(i: int) requires ... => ...
(i: int, r: real) => ...
```

**Allocations:**

```
new MyClass
new MyClass(4,5,6)
new MyClass.Init(5,6,7)
new int[10]
new int[][5,6,7,8,9]
new int[5](i => i*i)
new int[10,10]((i,j)=>i+j)
```

**Collections:**

```
[ e1, e2, e3 ]
seq(n, i requires 0<=i<n => f(i))
{ e1, e2, e3 }
iset{ e1, e2, e3 }
set x: nat | x < 10 :: x*x
multiset{ e1, e2, e3 }
multiset(s)
map[ 1:= 'a', 2 := 'b' ]
map x: int | 0<=x<10 :: -x := x
m.Keys m.Values m.Items
```

**Two-state:**

```
old(o)          old@L(o)
allocated(o)    allocated@L(o)
unchanged(o)    unchanged@L(o)
fresh(o)        fresh@L(o)
```

**Primaries:**

```
this null true false
5 0.0 0xABCD 'c' "asd" @"asd"
( e )
| e |
e.f
e.fn(3,4,5)
e.fn(3,4,option:=5)
```

## Declarations & Specifications

## More expressions

```

module { ... }

const c: int := 6

var f: T      in types only

method m(i: int)
  returns (r: real)
  requires ...
  ensures ...
  modifies ...
  decreases ...
  { ... }

function f(i: int): int
  requires ...
  ensures ...
  reads ...
  decreases ...
  { expr }

class A<T> extends I { ... }

trait I<T,U> extends J, K { ... }

datatype D = A(val: int) | B | C
  { ... }

type T
type Tuple3 = (int, real, nat)
type T = x: int | x > 0
newtype T = x: int | x > 0

while ...
  invariant ...
  modifies ...
  decreases ...
  { ... }

for i: int ... to ...
  invariant ...
  modifies ...
  decreases ...
  { ... }

```

```

Arrays & sequences:
  a[6]
  a[j..k] a[j..] a[..k] a[..]
  s[ 2 : 2 : 2 : ]

Updates:
  d.(f := x)
  s.[ 2 := 6, 3 := 7]
  mp.[ 2 := "Two", 3 := "Three"]

Quantifiers, Let expressions:
  forall x: int :: x > 0
  exists x: int :: x > 0
  var k := j*j; k*k
  var k :| k > 0; k + 1
  var k :- f(); k + 1
  var R(x,y) := T(); x+y

Statements in expressions:
  assert P(x); x > 0
  assume P(x); x > 0
  expect P(x); x > 0
  reveal ... ; x > 0
  calc { ... } x > 0
  L(x); f(x)      (lemma call)

```

## Types

```

int bool real nat char string
bv16 array<int> array3<int>
ORDINAL

set iset multiset seq map imap

Function types:
int->int int-->int int~>int

(int, real, nat)    tuple type

newtype
datatype
class
trait
iterator

```