Expression Precedence and Associativity

<table>
<thead>
<tr>
<th>Operators</th>
<th>Uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>- 0 []</td>
<td>member access, call and index</td>
</tr>
<tr>
<td>new (right)</td>
<td>constructor call</td>
</tr>
<tr>
<td>:</td>
<td>cast</td>
</tr>
<tr>
<td>** (right)</td>
<td>exponentiation</td>
</tr>
<tr>
<td>reduce scan dmapped</td>
<td>reduction, scan, apply domain map</td>
</tr>
<tr>
<td>! ~ (right)</td>
<td>logical and bitwise negation</td>
</tr>
<tr>
<td>* / %</td>
<td>multiplication, division, modulus</td>
</tr>
<tr>
<td>unary + -</td>
<td>positive identity, negation</td>
</tr>
<tr>
<td>&lt;&lt; &gt;&gt;</td>
<td>shift left, shift right</td>
</tr>
<tr>
<td>&amp;</td>
<td>bitwise/logical and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>+ -</td>
<td>addition, subtraction</td>
</tr>
<tr>
<td>...</td>
<td>range construction</td>
</tr>
<tr>
<td>&lt;= &lt; &gt;</td>
<td>ordered comparison</td>
</tr>
<tr>
<td>== !=</td>
<td>equality comparison</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>short-circuiting logical and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>in</td>
<td>loop expression</td>
</tr>
<tr>
<td>by # align</td>
<td>range stride, count, alignment</td>
</tr>
<tr>
<td>if forall [for]</td>
<td>conditional expression, parallel iterator expression, serial iterator expression</td>
</tr>
<tr>
<td>,</td>
<td>comma separated expression</td>
</tr>
</tbody>
</table>

*Left-associative except where indicated

Statements

```chapel
if cond then stmt1(); else stmt2();
if cond { stmt1(); } else { stmt2(); }
select expr {
  when equiv1 do stmt1();
  when equiv2 { stmt2(); }
  otherwise stmt3();
}
while condition do ...;
while condition { ... } do { ... } while condition;
do { ... } while condition;
for index in aggregate do ...;
for index in aggregate { ... } label outer for ... break; or break outer;
continue; or continue outer;
```

Procedures

```chapel
proc bar(r: real, i: imag): complex {
  return r + i;
}
proc foo(i) return i**2 + i + 1;
```

Formal Argument Intents

<table>
<thead>
<tr>
<th>Intent</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>copied in</td>
</tr>
<tr>
<td>out</td>
<td>copied out</td>
</tr>
<tr>
<td>inout</td>
<td>copied in and out</td>
</tr>
<tr>
<td>ref</td>
<td>passed by reference</td>
</tr>
<tr>
<td>const</td>
<td>passed by value or reference, but with local modifications disabled</td>
</tr>
<tr>
<td>const in</td>
<td>copied in with local modifications disabled</td>
</tr>
<tr>
<td>const ref</td>
<td>passed by reference with local modifications disabled</td>
</tr>
<tr>
<td>blank</td>
<td>like ref for arrays, syncs, singles, atomics; otherwise like const</td>
</tr>
</tbody>
</table>

Named Formal Arguments

```chapel
proc foo(arg1: int, arg2: real) { ... }
foo(arg2=3.14, arg1=2);
```

Default Values for Formal Arguments

```chapel
proc foo(arg1: int, arg2: real = 3.14);
foo(2);
```
Ranges

for (i,s) in zip(1..n, squares(n)) do …

Domains and Arrays

for D: domain(1) = (1..n); domain (index set)
for A: [D] real; array
for D: domain(int); associative domain
var Set: domain(D); sparse domain

Domain Maps

for B = new dmap{
    return 3.14159*r*r2;
}.method definition
for D2: domain(1) dmapped B.
distributed domain

Data Parallelism

for all i in D do A[i] = 1.0; domain iteration
for all a in A do a = 1.0; array iteration
var c = new Circle(r=2.0);
dynamic dispatch

Reductions and Scans

var sum = + reduce A;
var pre = + scan A;
var ml = minloc reduce (A, A.domain);

Iterators

for i in 1..n do …; iterate over iterator

Enumerated Types

enum day {sun,mon,tue,wed,thu,fri,sat};
var today: day = day.fri;

Tuples

var pair: (string, real); heterogeneous tuple
var coord: 2*int;

var: ("one", 2.0);
(coord(2) = pair;

tuple indexing

Unions

union U {
    var i: int;
    var r: real;
}.union definition

Classes

class Circle {
    class construction
    proc Circle.area() {
        return 3.14159*r*r2;
    }method definition
    var p: Point; assignment
    new p(1.0, 1.0);
}.class definition

record

var x, y: real;
var p: Point;
var r: real;

Extern Declarations

extern C_function(x: int);
extern C_variable:

Task Parallelism

begin task();
cobegin { task1(); task2(); }
coforall i in aggregate do task(i);
sync { begin task1(); begin task2(); }
serial condition do stmt();

Atomic Example

var count: atomic int;
if count.fetchAdd(1)==n-1 then
    done = true; n'th task to arrive

Synchronization Examples

var data$: sync int;
data$ = produce1();
consume(data$);
var go$: single real;
go$=set();
use1(go$);
use2(go$);

Locality

Built-in Constants:

numLocales: int; set via -nl
LocaleSpace = {0..numLocales-1};
Locales: [LocaleSpace] locale;

var c: Circle;
on Locales[i] {
    migrate task to new locale
    writeln(here.id);
    c = new Circle();
}allocate class on locale

writeIn(c.locale);
query locale of class instance
on c do ( … )
data-driven task migration

More Information

www: http://chapel.cray.com/
contact: chapel_info@cray.com
bugs: chapel-bugs@lists.sourceforge.net
discussion: chapel-users@lists.sourceforge.net