**Quick Start: one-line “hello, world”**

1. Create the file hello.chpl:
   ```
   writeln("hello, world ");
   ```
2. Compile and run it:
   ```
   $ chpl hello.chpl
   $ ./hello
   hello, world
   ```

**Comments**

// single-line comment

/* multi-line
   comment */

**Primitive Types**

<table>
<thead>
<tr>
<th>Type</th>
<th>Default size</th>
<th>Other sizes</th>
<th>Default init</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>impl. dep.</td>
<td>8, 16, 32, 64</td>
<td>false</td>
</tr>
<tr>
<td>int</td>
<td>64</td>
<td>8, 16, 32</td>
<td>0</td>
</tr>
<tr>
<td>uint</td>
<td>64</td>
<td>8, 16, 32</td>
<td>0</td>
</tr>
<tr>
<td>real</td>
<td>64</td>
<td>8, 16, 32</td>
<td>0.0</td>
</tr>
<tr>
<td>imag</td>
<td>64</td>
<td>32</td>
<td>0.01</td>
</tr>
<tr>
<td>complex</td>
<td>128</td>
<td>64</td>
<td>0.0+0.0i</td>
</tr>
<tr>
<td>string</td>
<td>n/a</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Variables, Constants and Configuration**

```chapel
default n: int = 100; // set n to 100 if there is no assignment
default m: int = 4; // set m to 4 if there is no assignment
```  

```chapel
var x: real = 3.14; // variable of type real set to 3.14
var z = -2.0i; // variable of type imag set to -2.0i
const epsilon: real = 0.01; // runtime constant
param debug: bool = false; // compile-time constant
config const n: int = 100; // set n to 100 if there is no assignment
config param d: int = 4; // set d to 4 if there is no assignment
```

**Modules**

```chapel
module M1 { var x = 10; } // module definition
module M2 { use M1; // module use
  proc main() { ...x... } // main definition
}
```

**Expression Precedence and Associativity**

<table>
<thead>
<tr>
<th>Operators</th>
<th>Uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>. () []</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</td>
<td>reduction, scan, apply domain map</td>
</tr>
<tr>
<td>dmapped</td>
<td></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 +- (right)</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>bitwise/logical xor</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>+ - ..</td>
<td>addition, subtraction</td>
</tr>
<tr>
<td>&lt;= &gt;= &lt; &gt;</td>
<td>ordered comparison</td>
</tr>
<tr>
<td>== !=</td>
<td>equality comparison</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>by # align</td>
<td>range stride, count, alignment</td>
</tr>
<tr>
<td>in</td>
<td>loop expression</td>
</tr>
<tr>
<td>if 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*

**Casts and coercions**

```chapel
var i: 2.0:int; // explicit conversion real to int
var x: real = 2; // implicit conversion int to real
```

**Conditional and Loop Expressions**

```chapel
var half = if i%2 then i/2+1 else i/2;
```

**Assignments**

**Simple Assignment:**

```

**Compound Assignments:**

```

**Swap Assignment:**

```

**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;
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>copy-initialized 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, 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);
```
Records

record Point {
  var x, y: real;
}
var p: Point;
writeln(sqrt(p.x**2+p.y**2));
p = new Point(1.0, 1.0);

classes

class Circle {
  var r: real;
}
var c = new owned Circle(r=2.0); initialization
proc Circle.area() method definition
  return 3.14159*r*r;
writeln(c.area());

Unions

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

Tuples

var pair: (string, real); heterogeneous tuple
var coord: 2*int;
coord = ("one", 2,0);
var (s, r) = pair; destructuring
coord(2) = 1; tuple indexing, 1-based

Enumerated Types

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

Ranges

var every: range = 0..n; range definition
var evens = every by 2; strided range
var R = evens # 5; counted range
var odds = evens align 1; aligned range

Domains and Arrays

var emptyRectDom:domain(1); domain (index set)
const D = (1..n); domain literal
var A: [D] real; array
var Set: domain(int); associative domain
Set += 3; add index to domain
var SD: sparse subdomain(D); sparse domain

Domain Maps

use BlockDist;
const D = (1..n) dmapped
  Block(boundingBox=(1..n)); block distribution
var A: [D] real; distributed array

Data Parallelism

forall i in D do A[i] = 1.0; domain iteration
[i in D] A[i] = 1.0; "
forall a in A do a = 1.0; array iteration
[a in A] a = 1.0; "
A = 1.0; "
array assignment

Reductions and Scans

Pre-defined: + * & ^& || min max
minloc maxloc

var sum = + reduce A; 1 2 3 => 6
var presc = + scan A; 1 2 3 => 13 6
var ml = minloc reduce (A, A.domain);

Iterators

iter squares(n: int) {
  for i in 1..n do
    yield i**2;
}
for s in squares(n) do ...; iterate over iterator

Zipper Iteration

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

Extern Declarations

extern proc C_function(x: int);
extern var C_variable: real;
extern { /* C code here */ }

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\textsuperscript{th} task to arrive

Synchronization Examples

var data$: sync int;
data$ = produce1(); consume(data$);
data$ = produce2(); consume(data$);
var go$: single real;
go$ = set(); use1(go$); use2(go$);

Locality

Built-in Constants
config const numLocales: int; $./prog -nl 4
const LocaleSpace = {0..numLocales-1};
const Locales: [LocaleSpace] locale;

Example
var c: owned Circle;
on Locales[i] { migrate task to new locale
  writeln(here); print the current locale
  c = new owned Circle(); allocate class on locale
} writeln(c.locale); query locale of class instance
on c do { ... } data-driven task migration

More Information
www: https://chapel-lang.org
contact: chapel_info@cray.com
bugs: http://chapel-lang.org/bugs.html
discussion: https://gitter.im/chapel-lang/chapel
chapel-users@lists.sourceforge.net