Quick Start: one-line “hello, world”

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

Comments
// single-line comment
/* multi-line
   comment */can be nested*/ */

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>32</td>
<td>0.0</td>
</tr>
<tr>
<td>imag</td>
<td>64</td>
<td>32</td>
<td>0.0i</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

var x: real = 3.14;  // variable of type real set to 3.14
var isSet: bool;   // variable of type bool set to false
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; $./a.out -n=4
config param d: int = 4; $ chpl -sd=3 x.chpl

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 + -</td>
<td>positive identity, negation</td>
</tr>
<tr>
<td>&lt;&lt; &gt;&gt;</td>
<td>shift left, 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>...</td>
<td>range construction</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>short-circuiting logical or</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, for</td>
<td>conditional expression, parallel iterator expression, serial iterator expression</td>
</tr>
<tr>
<td>,</td>
<td>comma separated expression</td>
</tr>
<tr>
<td>#</td>
<td></td>
</tr>
</tbody>
</table>

*Left associative except where indicated

Casts and coercions

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

Conditional and Loop Expressions

var half = if i%2 then i/2+1 else i/2;
writeln(for i in 1..n do i**2);

Assignments

Simple Assignment: =
Compound Assignments: += -= *= /= **= &= |= ^= &=& ||= <<= >>=
Swap Assignment: <=>

Statements

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

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, 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

proc foo(arg1: int, arg2: real) ( ... )
foo(arg2=3.14, arg1=2);

Default Values for Formal Arguments

proc foo(arg1: int, arg2: real = 3.14);
foo(2);
Chapel Quick Reference

Records

```plaintext
record Point {
  var x, y: real;
}
var p: Point;
var c: new Circle(r=2.0);
writeln(sqr(p.x**2+p.y**2));
p = new Point(1.0, 1.0);
```

Classes

```plaintext
class Circle {
  var r: real;
}
proc Circle.area() return 3.14159*r**2;
writeln(c.area());
var c = new Circle(r=2.0);
proc Oval.area() return 3.14159*r**2;
delete c;
c = new Oval(r=1.0, r2=2.0);
```

Unions

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

Tuples

```plaintext
var pair: (string, real);  
var coord: 2*int;
pair = ("one", 2.0);
var (s, r) = pair;
coord(2) = 1;
```

Enumerated Types

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

Ranges

```plaintext
var every: range = 0..n;  
var evens = every by 2;  
var R = evens # 5;  
var odds = evens align 1;  
```

Domains and Arrays

```plaintext
var emptyRectDom: domain(1);  
const D = (1..n);  
var A: [D] real;  
var Set: domain(int);  
Set += 3;
var SD: sparse subdomain(D);  
```

Domain Maps

```plaintext
use BlockDist;
const D = (1..n) dmapped
  Block(boundingBox=(1..n));
var A: [D] real;  
```

Data Parallelism

```plaintext
forall i in D do A[i] = 1.0;  
forall a in A do a = 1.0;  
A = 1.0;
```

Reductions and Scans

```plaintext
Pre-defined:
  + * & | ^ && || 
  min max minloc maxloc
var sum = + reduce A;
var pre = + scan A;
var ml = minloc reduce (A, A.domain);
```

Iterators

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

Zipper Iteration

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

Extern Declarations

```plaintext
extern C_function(x: int);
extern C_variable: real;
extern {* c code here */}
```

Task Parallelism

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

Atomic Example

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

Synchronization Examples

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

Locality

```plaintext
Built-in Constants
config const numLocales: int;  
const LocaleSpace = {0..numLocales-1};
const Locales: [LocaleSpace] locale;
```

Example

```plaintext
var c: Circle;
on Locales[i] {
  writeln(here);
  c = new Circle();
  writeln(c);  
  migrate task to new locale
}
```

More Information

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