Chapel Quick Reference

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 */ 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>0.01</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>&quot;</td>
</tr>
</tbody>
</table>

Variables, Constants and Configuration

var x: real = 3.14; variable of type real set to 3.14
var x: bool = false; 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; $:\text{prog}-n=4
config param d: int = 4; $chpl-sd=3 \times \text{chpl}$

Modules

module M1 { var x = 10; } module definition
module M2 {
  use M1; module use
  proc main(){ writeln(x); } main function
}

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>bitwise/logical or</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>short-circuiting logical and</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</td>
<td>conditional expression, parallel iterator expression, serial iterator expression</td>
</tr>
<tr>
<td>for</td>
<td>expression list</td>
</tr>
</tbody>
</table>

*Left-associative except where indicated

Casts and coercions

var i: 2.0:int; explicit conversion real to int
var x: 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

Intent | Semantics
-------|---------------
in | copy-initialized in
out | copied out
inout | copied in and out
ref | passed by reference
const | passed by value or reference, with local modifications disabled
const in | copied in, with local modifications disabled
const ref | passed by reference, with local modifications disabled
blank | like ref for arrays, syncs, singles, atomics; otherwise like const

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);
Records

record Point {
    var x, y: real;
}
var p: Point;
var r: real;

class Circle {
    var p: Point;
    var r: real;
}
override proc Circle.area() {
    return 3.14159*r*r2;
}
proc Circle.area() {
    return 3.14159*r*r2;
}

class Oval: Circle {
    var r2: real;
    override proc Oval.area() {
        return 3.14159*r*r2;
    }
    c = new Oval(r=1, r2=2);
    writeln(c.area());
}
var nc: owned Circle? = nil;

Unions

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

Tuples

var pair: (string, real);
var coord: 2*int;
pair = ("one", 2.0);
var (a, r) = pair;
coord(0) = 1;

Enumerated Types

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

Ranges

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

Domains and Arrays

var rectangular:domain(1); 1-d domain (index set)
const D = (1..n); domain literal
var A: [D] real;
array of real numbers
var Set += 3;
set
var r: real;

Data Parallelism

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

Synchronization Examples

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

Locality

config const numLocales: int = 4;
const LocaleSpace = {0..numLocales-1};
var c: owned Circle?;
on Locales[i] {
    migrate task to new locale
    writeln(here);
    c = new Circle();
}

Example

var c: owned Circle?;
on c do {
    writeln(c.locale);
    query locale of class instance
    data-driven task migration
}

More Information

www: https://chapel-lang.org
user resources: https://chapel-lang.org/users.html