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></td>
<td>8, 16, 32</td>
<td>0</td>
</tr>
<tr>
<td>uint</td>
<td></td>
<td>8, 16, 32, 64</td>
<td>0</td>
</tr>
<tr>
<td>real</td>
<td></td>
<td>32</td>
<td>0.0</td>
</tr>
<tr>
<td>imag</td>
<td></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>&quot;</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; $ ./chpl -n=4
config param d: int = 4; $ chpl -sd=3 x.chpl

Modules

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

Expression Precedence and Associativity*

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

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;
write(new 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 expr1 do stmt1();
when expr2 { stmt2(); }
otherwise stmt3(); }
while condition do ...
while condition { ...
do { } while condition;
for index in aggregate do ...
for index in aggregate { ...
try { } catch error { } ...
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>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

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**

```chapel
class Circle {    // class definition
  var p: Point;    // declaring fields
  var r: real;    // declaring fields
}
```

**Classes**

```chapel
class Circle {    // class definition
  var p: Point;    // declaring fields
  var r: real;    // declaring fields
}
```

**Enumerated Types**

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

**Ranges**

```chapel
var every: range = 0..<n;    // range definition
var evens = every by 2;    // stripped range
var R = evens # 5;    // counted range
var odds = evens align 1;    // aligned range
var open = 0..<n;    // open range
```

**Domains and Arrays**

```chapel
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: domain(int);    // associative domain
Set += 3;    // add index to domain
var SD: sparse subdomain(D);    // sparse domain
```

**Domain Maps**

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

**Data Parallelism**

```chapel
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 do a = 1.0;    // array assignment
```

**Reductions and Scans**

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

```chapel
var sum = + reduce A;    // 123 => 6
var pre = + scan A;    // 123 => 136
var ml = minloc reduce (A, A.domain);    //
```

**Iterators**

```chapel
iter squares(n: int) {    // serial iterator
  for i in 1..n do
    yield i**2;    // generate a value
}
for s in squares(n) do ...;    // loop over iterator
```

**Zipper Iteration**

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

**Extern Declarations**

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

**Task Parallelism**

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

**Atomic Example**

```chapel
var count: atomic int;
if count.fetchAdd(1)==n-1 then
  done = true;    // n\textsuperscript{th} task to arrive
```

**Synchronization Examples**

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

**Locality**

```chapel
config const numLocales: int;    // $./prog-ul
const LocaleSpace = {0..numLocales-1};
const Locales: [LocaleSpace] locale;
```

**Example**

```chapel
var c: owned Circle?;
on Locales[i] {    // migrate task to new locale
  writeln(here);    // print the current locale
  c = new Circle();    // allocate class on locale
}
```

**More Information**

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