Chapel Language Specification 0.6.0
Draft Edition for Internal Release

Cray Inc
411 First Ave S, Suite 600
Seattle, WA 98104

12.3.1 Explicit Naming 62

1 Scope

Chapel is a new parallel programming language that is under d

2 Not

formal
  formal

3 Organization

This specification is organized as follows:

- Section 1, Scope, describes the scope of this specification.
- Section 2, Notation, introduces the notation that is used th

5.1.2 Locality Aware Programming

Locality-aware programming, in the style of HPF and ZPL, provides without requiring a fragmentation of control structure. Th

5.2.4 Statements and Expressions

Egamples ofp

Statement	**Example**
For Loop	

Expression	**Example**

braces	**use**
( )	

7 Types

7.1.6 The String Type

Strings are a primitive type designated by the symbol `string`. Their length is unbounded.

7.1.7 Primitive Type Literals

Bool literals are designated by the following syntax:

bool-literal :

Note that real literals require that a digit follow the decim

9 Conversions

9.1.3 Implicit Class Conversions

An expression of class tyoe

A *call-expression* is resolved to a particular function according to the algorithm for function resolution described in §13.8.

A *named-*

```
def *(a: real(128), b: real(128)): real(128)

def *(a: imag(32), b: imag(32)): real(32)
def *(a: imag(64), b: imag(64)): real(64)
def *(a: imag(128), b: imag(128)): real(128)

def *(a: complex(64), b:
    def /(a: complex
```

10.9.8 Exponentiation Operators

For each of these definitions that return a value, the result is computed by applying the AND operation to the bios of the operands.

It is a compiile-time error to apply the bitwise and opeandtor t

```
def <<(a: int(32), b): int(32)
```

10.13 Relational Operators

Values of one primitive or enumerated type can be assigned to another primitive or enumerated type if an

11.8 The For Loop

11.8.3 Parameter For Loops

Parameter for loops are unrolled by the compile so that the i ndex variable is a paaeteraherhan a variable. The syntax for a paaeter for loop statement is gi ven by:

```
param-for-staemen :
  for param identifier in
```

arity	**operators**
unary	`+ - ! ~`
binary	`+ - * / % ** && || ! == <= >= < > << >> & | ^ #`

The arity and precedence of the operator must be maintained when it is overloaded. Operator resolution follows the same algorithm as function resolution.

13.8 Function Resolution

Given a function call, the function that the call resolves to is determined according to the following algorithm:

-

- If

14.4.1 Class Method Declarations

Methods are functions that are bound to a class. Tœy are decl ared with the following syontaxo:

-eclaration -statement :-ef *type-binding fh nctio n*
method-

16 Unions

This section is forthcoming.

17.4.4TR373 9 0 0341 3(.)-lcparing.4T34.737(H)-.6 244(o)-2.94496(m)1.89551(o)-2.94496(g)-2.94496(l)0.965521(n)1.931m [louule

18.5 Iteration over Sequences

18.7.2 Sequence Indexing by Tuples

If `s` is a sequence and `t` is a tuple of integers of size **`k`**, then the expression `s(t)` indexes into the sequence `s` **`k`** times using the integers in the tuple. In this case, `s` must be a sequence whose rank is at least as great as the size of `t`. If `s` has rank less than the size of the tuple, then the result is a sequence.

If the integers in tuple `t` en t60 Tf 9.6 0 Td[(.)-304.069(I)-4.2603(f)-4.2603]TJ /R374575-242.549(s)3.84.9602 0 Td [(s)-2.24962-242

mut6008.347(i)0.9(d)-5.8912Td[(.)-304.069(I)-4.2603(f)-4.1-5.8887(e)-25992 0 Td [(m)0.965521(i)0.99(t6008.347

18.10.1 Sequences in Select Statements

When a sequence expression is used as a top-level expression in the condition of a select statement, there are two interpretations. If the condition in the when expression is itself a sequence, the equality operator is used to compare the sequences and then an implicit && reduction is applied to produce a single bool alue. If the condition in the when expression is a scalar, the equality opeerat521(o)-5.88993(n)-234.75.88993(n)-234.753(i)0.9732626(s)3.56067(c)-

18.12.3 The

19.1.2 Index Types

If the zipper iteration is illegal, then the assignment is illegal. Notice that the assignment is implemented with

values of their element type. In this case, each element in the array is
mplicitly

)-4.261531(h)-5.8887(o)ih25135]T516(s)-237.075(o-4.26153(e)-1.66454(m)-4.92563(a)-88.1257(e)-254.626(a7.776454(i)0.96307u(g)-5.8887(n)-5.8911

Efiample. The efipression `[1..5, 1..5]` defines a

19.6 Opaque Domains and Arrays

This section is forthcoming.

19.6.1 Opaque Domain and Array Types

This section is forthcoming.

19.6.2 Opaque Domain Index Types

This section is forthcoming.

19.6.3 Adding Indices to Opaque Domains

This section is forthcoming.

19.6.4 Removing Indices from Opaque Domains

This section is forthcoming.

19.7 Enumerated Domains and Arrays

This section is forthcoming.

19.7.1 Enumerated Domain and Array Types

This section is forthcoming.

19.7.2 Enumerated Domain Index Types

This section is forthcoming.

19.8 Association of Arrays to Domains

This section is forthcoming.

20 Iterators

An iterator is a function that conceptually returns a sequence of values rather than simply a single value.

21 Generics

Chapel supports generic functions and types that are parameterizable over both types and parameters. The generic functions and types look similar to non-generic functions and types already discussed.

21.1 Generic Functions

A function is generic if any of the following conditions hold:

- Some form1.66516(n)6l 1.66516(n)6rgument is specified with an intent of type or param.
- Some form1.66516(n)6l 1.66516(n)6rgument has no specified type and no default valu e.
- Some form1.66516(n)6l 1.66516(n)6rgument is specified with a queried type.
- The type of some form1.66516(n)6l 1.66516(n)6rgument is a generic type, e.g., seq
-

21.3.3 Fields without Types

21.4 fihere Expressions

22.1.2 Forall Expressions

```
var
```

23 Locality and Distribution

```
        forall d in D {
```

25 Input and Output

This section is forthcoming.

, 5488&=546=t4&19r

Index

&, 46
&&, 48
&&=, 54
&=, 54
(/, 89
*, 43
* tuples, 87
**, 46
<=, 49
**=, 54
<<, 47
*=, 54
<<=, 54, 13, 8, 42, 13, 8 =
+, 42, 51 , 48
+ (unary), 41
+=, 54
-, 42
- (unary), 41
-=, 54
/, 44
/=, 54
<, 49
<<, 47
<<=, 54
<=, 49
=, 54
==, 50
<, 49