formal
  formal tag identifier formal

- Section 25, Input and Output, describes support for input an

Acknowledgments

Control of Locality A second principle in Chapel is to allow the user to optionally and incrementally specify where data and computation shoulde placedn the ph

```
    select (x.rank) {
      when 1 do return norm(x, norm2);
      when 2 do return norm(x, normFrob);
      otherwise compilerError("Norms not implemented for arrr5 rrnks > 2D");
    }
  }
}

module TestNorm {
  use Norm;

  def testNorm(arr: []) {
    // test all possible norms of arr
    var testType = if (arr.rrnk == 1) then "vector" else "mrtrix"=9453(=)-22(9453(=)-22(9 -9.47969 Td [54)-5.8
                                                            ritel(arr;
                                                            ritel([illegible]
```

In this simple example, the variable `stack1`

When `Stack` is instantiated, a type is specified for the type alias, `itemType`. The `top` field is a pointer to the top of the stack, which is a `MyNode` object of

6 Lexical Structure

This section describes the lexical components of Chapel programs.

Types

8.5 Configuration Variables

If the keyword `config` precedes the keyword `var`, `const`, or `param`, the variable, constant, or parame-
to2(72(p)-5.8911.96264 Tf 26.8801 0 Tdgn)-5.89115(g)-5.89188996m

9 Conversions

A conversion allows an expression of one type to be converted into another type. Conversions can be either implicit or explicit.

Implicit conversions c88993(r)-4.260358.819(o)-5.88993(c)-1.66516(c)-1.66393(u)-5.88993(r)-245.153(d)-5.88993(u)-5.89115(r)-4.25

9.2.3 Explicit Class Conversions

An expression of class type `C` can be explicitly converted to another class type `D` provided that `C`

10 Expressions

This section defines expressions in Chapel. Forall expressions are described in

A *call-expression* is resolved to a particular function according to the algori

10.5 Casts

A cast is specified with the following syntax:

cast-expression :
  expression : *type*

The expression is converted to the specified type. Except for the casts listed below, casts are rest53

10.9.3 Addition Operators

The addition operators are predefined as follows:

```
def +(a: int(32), b: int(32)): int(32)
def +(a: int(64), b: int(64)): int(64)
def +(a: uint(32), b: (64)
def +(a: uint(32), b: (64[(i)-2.25145(n)-2.25145(t)-2.25145]TJ /R350 7.97011 Tf 14.4 0 Td [(()-2.25145(6)-2.25145(
def +(a: uint(32(), b:int
def r77(w)uint700861(s)3.5692(e)-2.249
def r77(w)uint700861(s)3.5692(e)-2.249
def r77(w(64)700861(s)3.5692(e)-2.249l
def +m861(s)3.5692(e)-2.249
def +m861(s)3.5692(e)-2.24992(t8-2.24992(t)-2.24992]TJ /R350 7.97011 Tf 19.080
def c2(e)-2.249
def +m861(s)3.5692(e)-2.24992(t8-2.24992(t)-2.2
def r77(w)uint700861(s)3.5692(e)-2.2
```

```
def -(a: uint(64), b: int(64))
def -(a: int(64), b: uint(64))

def -(a: real
```

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

```
def /(a: complex(256), b: complex
```

For each of these definitions that return a value, the result is computed by applying the logical and operation

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

10.12.3 The Logical Or Operator

The logical or operator is predefined over bool type. It retur

Expressions

The expression that follows the keyword `select`, the select expression, is compared with the list of expres-

Call the expressions following the keyword `select`

12 Modules

12.4 Nested Modules

13 Functions

This section defines functions. Methods and iterators are functions and most of this section applies to them as well. They are defined separately in §20 and §14.4.

13.1 Function Definitions

13.4.1 Named Arguments

<table>
<tr><th>arity</th><th>operators</th></tr>
<tr><td>unary</td><td><code>+ - ! ~</code></td></tr>
<tr><td>binary</td><td><code>+ - * / % ** && || ! == <= >= < > << >> & | ^ #</code></td></tr>
</table>

- If 2

Example. The code

```
    def mywriteln(x: int ...?k) {
      for param i in 1..k do
  writeln(x(i));
}
```

14 Classes

Classes are an abstraction of a data structure where the stor

```
class Actor {
  var name: string;
  var age: uint;
}
```

defines a new class type called `Actor` that has two fields: the string field `name` and1.9310449(t)0.965521(h)-5.89115(e)-

```
class C {
  var x: int;
  def =x(value: int) {
    if value < 0 then
      halt("x assigned negative value");
    x = value;
  }
}
```

a setter is defined for field `x`

16 Unions

Unions have the semantics of records, however, only one field in the union can contain data at any particular point in the program's execution. Unions are safe so that an access to a field that does not contain data is a me error. When a union is constructed, it is in an unset s

17 Tuples

A tuple is an ordered set of components that allows for the specification of a light-weight record with anony-

17.5.1 Declaring Homogeneous Tuples

18 Sequences

A sequence is an ordered set of elements of the same type.

18.5 Iteration over Sequences

18.12.3 The *spread* Function

```
def spread(s: seq, length: int, dim: int = 1)
```

The spread function takes a sequence of rank and returns a new sequence of rank $+ 1$. When `dim` is equal

18.13 Arithmetic Sequences

Arithmetic sequences contain an ordered set of values of integral type that can be specified with a low bound , a high bound , and a stride s. If the stride is negative, the values contained by the arithmetic sequence are

19.1.2 Index Types

19.1.6 Domain Promotion of Scalar Functions

Domain promotion of a scalar function is defined over the doma

19.2.8 Array Initialhecification

sparse-

20 Iterators

21 Generics

Chapel supports generic functions and types that are parame

21.3.3 Fields without Types

```
def
```

22.7.4 Synchronization Variables of Record and Class Types

A variable of record or class type can be a single or sync variable. The semantics c8(o)-5.8887(f)-221.087(s)3.56067(i)0.965521(n)-5.89
are applied only to the variable and not to acce

nization sema

23 Locality and Distribution

23.1.3 Querying the Locale of a Variable

Every variable `v` is associated with some locale which can be querabed using the following syntax:

```
locale-access:
  expression . locale
```

When the

23.2.2 On and Iterators

When a loop iterates over a sequence specified by an iterator, on-statements inside the iterator control where the corresponding loop body is executed.

Example. An iterator over a distributed tree might include an iterator over the nodes as defined in the following code:

```
class Tree {
  var left, right: Tree;
  iterator nodes {
    on this yield this;
    if left then
      forall t in left.nodes do
        yield t;
    if right then
      forall t in right.nodes do
        yield t;
  }
}
```

Given this code and a binary tree of type `Tree` stored in variable `tree`, then we can use the nodes iterator to iterate over the tree with the following code:

23.3.2 Distributed Arra

24 Reductions and Scans

Chapel provides a set of built-in reductions and scans with parallel semantics, a mechanism for defining more reductions and scans with efficient implementations, and sy

25 Input and Output

25.2 Standard files

Returns the rounded integral value of the argument determined by the current rouding direction.

```
def rint(x: real): real
```

Returns the rounded integral value of the argument determined by the current rouding direction.

```
def round(x: real): real
```

Returns the rounded integral value of the argument. Cases halfway between twonvudfira4.7995521(fi)1.9288 [(R)4.52

```
int(x   real):      rea1f
```

Returns (.)e529448(t)0.965521(w)11.8993(s)-237.375 (h)-5.889931(u)-5.8538(d)-222.707(b)-5.8887(y)-246.162

```
real):
```

26.3 Random

The module `Random` supports the generation of pseudo-random values and streams of values. The current interface is minimal and should be expected to grow and evolve over time.

class `RandomStream`

> Implements a pseudo-random stream of values. Our current implementation generates the values using a linear congruential generator. In future versions of this

Index