



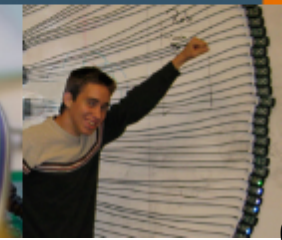
Growing the Fortress Programming Language by Example

Sukyoung Ryu

Member of Technical Staff, Project Fortress
Sun Microsystems, Inc.



**2008
Sun Labs
Open House**



Growing a Language

- Languages have gotten much bigger.
- You can't build one all at once.
- Therefore it must grow over time.
- What happens if you design it to grow?
- How does the need to grow affect the design?
- Need to grow a user community, too.

Growing a Language

“So I think the sole way to win
is to plan for growth
with help from users.”

Guy L. Steele Jr.

keynote talk, OOPSLA 1998;

Higher-Order and Symbolic Computation 12, 221-236 (1999)

What Primitive Data Types to Include?

- Integers and floating-point (what sizes? bignums?)
- Complex numbers, rational numbers, intervals
- Arrays, vectors, and matrices
- Rational intervals, complex intervals
- Complex vectors and matrices
- What about physical units (meters, kilograms)?

“I might say ‘yes’ to *each* one of these,
but it is clear that I *must* say ‘no’ to *all of them!*”

Growing a Language: Fortress

“Whenever possible,
implement a proposed language feature
in a library
rather than building it into the compiler.”

Types Defined by Libraries

- Booleans, integers, rationals, characters, and strings
- Matrices and multidimensional arrays
- Vectors, lists, sets, and maps
- File supports, heaps, and skip lists

Example: Primitive Types

```
object Boolean
  extends { SequentialGenerator[()], StandardTotalOrder[Boolean] }
end

trait Any end

trait String extends { Any } end

trait Char extends { Any } end

trait Number extends { Any } end

trait Integral extends { Number } end

trait Z32 extends { Integral } end

trait Z64 extends { Integral } end

trait R64 extends { Number } end

trait IntLiteral extends { Z32, Z64, R64 } end

trait FloatLiteral extends { R64 } end
```

Example: Arrays

```
trait Array[E, I] extends { ReadableArray[E, I], MutableIndexed[E, I] }  
  abstract opr [r: Range[I]] : Array[E, I]  
  abstract opr [_: OpenRange[Any]] : Array[E, I]  
  abstract ivmap[R](f: (I, E) → R): Array[R, I]  
  abstract map[R](f: E → R): Array[R, I]  
  abstract shift(newOrigin: I): Array[E, I]  
  abstract init(i: I, v: E): ()  
  abstract fill(f: I → E): Array[E, I]  
  abstract fill(v: E): Array[E, I]  
  abstract assign(f: I → E): Array[E, I]  
  abstract copy(): Array[E, I]  
  abstract replica[U](): Array[U, I]  
  (* Freeze array (return mutable copy). *)  
  abstract freeze(): ImmutableArray[E, I]  
end
```

Example: Lists

```
trait List[E] extends { Equality[E], ZeroIndexed[E] }
    excludes { Number, HasRank }
    getter left(): Maybe[E]
    getter right(): Maybe[E]
    getter extractLeft(): Maybe[(E, List[E])]
    getter extractRight(): Maybe[(List[E], E)]
    append(f: List[E]): List[E]
    addLeft(e: E): List[E]
    addRight(e: E): List[E]
    take(n: Z32): List[E]
    drop(n: Z32): List[E]
    split(n: Z32): (List[E], List[E])
    split(): (List[E], List[E])
    reverse(): List[E]
    zip[F](other: List[F]): Generator[(E, F)]
    filter(p: E → Boolean): List[E]
    toString(): String
    concatMap[G](f: E → List[G]): List[G]
end
```

Example: Skip Lists

```
trait SkipList[[Key, Val, nat pInverse]]
  comprises { ... }

  toString() : String

  (* The number of values stored. *)
  size() :  $\mathbb{Z}32$ 

  (* Given a key, try to return a value that matches that key. *)
  search(k : Key) : Maybe[[Val]]

  (* Add a (key, value) pair. *)
  add(k : Key, v : Val) : SkipList[[Key, Val, pInverse]]

  (* Remove a (key, value) pair. *)
  remove(k : Key) : SkipList[[Key, Val, pInverse]]

  (* Merge two Skip Lists. *)
  merge(other : SkipList[[Key, Val, pInverse]]) : SkipList[[Key, Val, pInverse]]
end
```

Operators Defined by Libraries

- Equality and ordering operators
- Arithmetic operators
- Juxtaposition operators
- Reduction operators
- Range operators

Example: Operators (I)

opr $=(a : \text{Any}, b : \text{Any}) : \text{Boolean}$

opr $\neq(a : \text{Any}, b : \text{Any}) : \text{Boolean}$

opr $-(a : \mathbb{Z}32) : \mathbb{Z}32$

opr $+(a : \mathbb{Z}32, b : \mathbb{Z}32) : \mathbb{Z}32$

opr $-(a : \mathbb{Z}32, b : \mathbb{Z}32) : \mathbb{Z}32$

opr $\cdot(a : \mathbb{Z}32, b : \mathbb{Z}32) : \mathbb{Z}32$

opr $\div(a : \mathbb{Z}32, b : \mathbb{Z}32) : \mathbb{Z}32$

opr REM($a : \mathbb{Z}32, b : \mathbb{Z}32$) : $\mathbb{Z}32$

opr MOD($a : \mathbb{Z}32, b : \mathbb{Z}32$) : $\mathbb{Z}32$

opr GCD($a : \mathbb{Z}32, b : \mathbb{Z}32$) : $\mathbb{Z}32$

opr LCM($a : \mathbb{Z}32, b : \mathbb{Z}32$) : $\mathbb{Z}32$

opr juxtaposition

$(a : \mathbb{Z}32, b : \mathbb{Z}32) : \mathbb{Z}32$

opr juxtaposition $[[T \text{ extends Number}, \text{nat } n, \text{nat } m, \text{nat } p]]$

$(me : \text{Vector}[[T, n]], other : \text{Vector}[[T, n]]) : T$

opr juxtaposition $[[T \text{ extends Number}, \text{nat } n, \text{nat } m, \text{nat } p]]$

$(me : \text{Matrix}[[T, n, m]], other : \text{Matrix}[[T, m, p]]) : \text{Matrix}[[T, n, p]]$

Example: Operators (II)

opr \sum [[T]](g : (Reduction[[Number], $T \rightarrow$ Number) \rightarrow Number): Number

opr \prod [[T]](g : (Reduction[[Number], $T \rightarrow$ Number) \rightarrow Number): Number

opr # [[I extends Integral]](lo : I, ex : I): Range[[I]]

opr # (lo : IntLiteral, ex : IntLiteral): Range[[Z32]]

opr # [[I extends Integral, J extends Integral]]

(lo : (I, J), ex : (I, J)): Range[[I, J]]

opr # [[I extends Integral, J extends Integral, K extends Integral]]

(lo : (I, J, K), ex : (I, J, K)): Range[[I, J, K]]

opr (x : T) # [[T]] : LowerRange[[T]]

opr (x : T): [[T]] : LowerRange[[T]]

opr # [[T]](x : T) : ExtentRange[[T]]

opr : [[T]](x : T) : UpperRange[[T]]

opr # (): OpenRange[[Any]]

opr : (): OpenRange[[Any]]

Constructs Defined by Libraries

- Generators and reductions
- Ranges
- Aggregate expressions
- Comprehensions

Example: Constructs

```

trait Reduction[[ R ]]
  abstract getter toString(): String
  abstract empty(): R
  abstract join(a: R, b: R): R
end

```

```

trait Range[[T]]
  extends StandardPartialOrder[[Range[[T]]]]
  comprises { RangeWithLower[[T]], RangeWithUpper[[T]],
              RangeWithExtent[[T]], PartialRange[[T]] }
  excludes { Number }
  opr =(self, _: Range[[T]]): Boolean
end

```

```

opr { [[E]] es: E ... } : Set[[E]]

```

```

opr BIG{ [[T, U]] g: (Reduction[[SomeList]], T → SomeList) → SomeList } : Set[[U]]

```

Example: Generators

```
for  $i \leftarrow 1 \# 1024$  do  
    doSomething(i)  
end
```

- Gets desugared into a library call with
 - > a generator clause: $1 \# 1024$
 - > a loop body: *doSomething(i)*
- The library is responsible for recursively subdividing the loop iterations and generating tuple tasks.
- This means that if you change the generator to exploit new hardware, you can do it at the Fortress language level in a library.

Utilities Defined by Libraries

- Casting and instanceof testing
- Identity and function composition
- Assertions
- Print functions
- Floating-point constants
- Quick sort

Example: Functions and Constants

```
cast[[T extends Any]](x : Any) : T  
instanceOf[[T extends Any]](x : Any) : Boolean  
  
identity[[T extends Any]](x : T) : T  
opr ◦[[A, B, C]](f : B → C, g : A → B) : A → C  
  
assert(flag : Boolean) : ()  
assert(flag : Boolean, failMsg : String) : ()  
assert(x : Any, y : Any, failMsg : Any ... ) : ()  
  
printTaskTrace() : ()  
print(a : String) : ()  
println(a : String) : ()  
printThreadInfo(a : String) : ()  
printThreadInfo(a : Number) : ()  
  
π : FloatLiteral  
e : FloatLiteral  
infinity : ℝ64  
  
quicksort[[T]](lt : (T, T) → Boolean, arr : Array[[T, ℤ32]], left : ℤ32, right : ℤ32) : ()
```

Example: Demo Program

Tennis Ranking System in Fortress

Tennis Club

- More than 50 members
- Weekly meetings on Saturdays
- Small meetings on Mondays and Wednesdays
- Singles and doubles games
- Winter leagues and summer leagues

Tennis Ranking System

- Calculate a new ranking result for every 2 weeks.
- For each player, calculate:
 - > the ranking
 - > the number of games won/lost in the season so far
 - > the number of games won/lost in this term
 - > the points earned in the season so far
 - > the points earned in this term
- MVP: who earned the most points
- MEP: who played the most games
- MIP: who rose the most ranks

Tennis Ranking System Rules (I)

- 1. Every player begins with 1000 points.
- 2. Existing members who have not played any games yet are not included in the ranking system.
- 3. New members can join the system any time.
- 4. A game is either a singles or a doubles game.
- 5. If the ranking difference between the two teams is bigger than or equal to 10, the game is considered invalid; it is not considered when the points are calculated.

Tennis Ranking System Rules (II)

- 6. For each singles game:
 - > winner: $10(\text{winner's ranking} - \text{loser's ranking}) + 100$
 - > loser: $5(\text{loser's ranking} - \text{winner's ranking}) - 50$
- 7. For each doubles game:
 - > winners:
 $5(\text{sum of winners' rankings} - \text{sum of losers' rankings}) + 100$
 - > losers:
diff = sum of winners' rankings - sum of losers' rankings
if diff > 0 OR even(diff)
then $-5 (\text{diff DIV } 2) - 50$
else $-5 (\text{diff DIV } 2) + 1 - 50$

Tennis Ranking System Rules (III)

- 8. If any player has never played any game before this term, then ranking differences are ignored.
- 9. If the same teams play more than once with the same combination in a term, only the first game is considered valid.
- 10. If a player does not play any game in a term, the player gets a 50 point penalty.

Tennis Ranking System in Fortress

```
(validDiff, notPlayed) = validGame(winner, loser, diff)
(*a winner gets "10 diff + 100" points and
  a loser loses "5 diff + 50" points*)
if validDiff  $\wedge$  notPlayed
then won(winner, loser, 10 diff + 100)
     lost(loser, winner, -5 diff - 50)
else if  $\neg$ (validDiff)
     then outFileWrite(fout, "Invalid game: " winnerName " vs "
                        loserName " has " |diff| " ranking "
                        "differences. \n")
     else outFileWrite(fout, "Invalid game: " winnerName " vs "
                        loserName " has played before. \n")
end
end
```

Result (with Korean Characters)

Invalid game: 유은진/송은희 vs 오채린/양소림 has 26 ranking differences.

Invalid game: 박승찬 vs 정호일 has 11 ranking differences.

Invalid game: 박승찬/송은희 vs 지현수/양소림 has played before.

demos/tennis050307 ~ demos/tennis051707

순위	선수	승패	경기수	포인트
1	박승찬	27 / 6 (14 / 1)	33 (15)	3340 (915)
2	기철수	45 / 37 (5 / 3)	82 (8)	2985 (200)
3	오왕열	38 / 21 (4 / 1)	59 (5)	2965 (355)
...				
25	남현석	7 / 13 (1 / 1)	20 (2)	1050 (75)
26	배진우	6 / 13 (1 / 4)	19 (5)	970 (70)
	이수원	11 / 17 (1 / 2)	28 (3)	970 (-50)
28	이민정	3 / 0 (1 / 0)	3 (1)	945 (140)

MVP: 박승찬 (915 points)

MEP: 지현수 박승찬 (15 games)

MIP: 이용수 (ranking 24 => 16)

Extensive Uses of Libraries

```
import Set.{...}
import Map.{...}
import List.{...}
import QuickSort.{...}
import File.{...}

digits = {"" "0", "1", "2", "3", "4", "5", "6", "7", "8", "9"}

database : Map[[String, Player]] := mapping[[String, Player]]()

mvpPlayers : List[[String]] := emptyList[[String]]()

quicksort[[ $\mathbb{Z}32$ , String]](lt, toSortResult, 0, toSortResult.size() - 1)

fin : FileReadStream := FileReadStream previous
fout : BufferedWriter = outFileOpen today
initialize(fin)
```

Changes in Language Features

- Declaration of multiple variables:
 - > Do not commingle mutable and immutable variables in a single variable declaration:

$(\text{var } x, y): \mathbb{Z}32 = (5, 6)$

- > Move the `var` modifier out of the tuple notation:

`var (x, y): $\mathbb{Z}32 \dots = (5, 6)$`

- Generalization of if expression:

`if generators(* at most one iteration *)`

`then body1`

`else body2`

`end`

Bug Fixes in the Fortress Interpreter

- Tuple binding: $(x, y) = f(z)$
- Ignored binding: `_ = println` “Ignore this value.”
- Settable field parameters of objects
- Set implementation
- On-demand import statements: `import Map {...}`

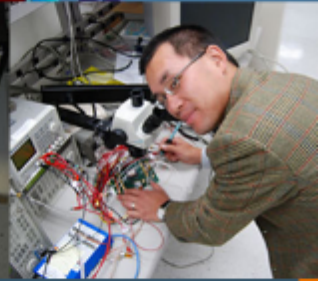
More Features Implemented

- Trait and object declarations in APIs
- Parameterized Set, Map, and List implementations
- Quicksort
- File read/write operations using generators
- String library functions: *length* and *substring*
- Exception handling
- Shortcut operators: \wedge :
- Aggregate expressions and comprehensions

Summary

- Fortress has designed to grow.
- Most primitive types are defined in libraries.
- Many language constructs are defined in libraries.
- Fortress has grown with help from the community.

`http://projectfortress.sun.com`



Sukyoung Ryu
sukyoung.ryu@sun.com



**2008
Sun Labs
Open House**

