

Fortress Boot Camp: Language Overview by Example

Sukyoung Ryu

Programming Language Research Group
May 21, 2008

Copyright © 2008 Sun Microsystems, Inc. (“Sun”). All rights are reserved by Sun except as expressly stated as follows. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted, provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers, or to redistribute to lists, requires prior specific written permission of Sun.

Hello, World!^a

```
api Executable
  run(args : String...): ()
end

component hello
  export Executable
  run(args : String...) = println "Hello, World!"
end
```

- `fortress run HelloWorld.fss`
- `fortress run hello.fss`
- `fortress hello.fss`
- `PFC/Library/FortressLibrary.fsi`

^a components, APIs, exports, top-level functions, function calls, string literals, `Executable`

Imports and Exports^a

```
api FortressWorld  
  message: String  
end
```

```
component hello  
  import FortressWorld.message  
  export Executable  
  
  run(args : String ...) = println message  
end
```

^a top-level variables, variable references, imports

Imports and Exports^a

```
api FortressWorld
  message: String
end
```

```
component FortressWorld
  message = "Welcome All!"
end
```

```
component hello
  import FortressWorld.message
  export Executable
  run(args : String ...) = println message
end
```

^a top-level variables, variable references, imports

Imports and Exports^a

```

api FortressWorld
  message: String
end

component FortressWorld
  export FortressWorld
  message = "Welcome All!"
end

```

```

component hello
  import FortressWorld.message
  export Executable

  run(args : String ...) = println message
end

```

^a top-level variables, variable references, imports

More Import Statements^a

```
component hello  
  import FortressWorld.message  
  export Executable  
  
  run(args : String ...) = println message  
end
```

- `import FortressWorld.message`
- `import FortressWorld.{...}`
- `import FortressWorld`
- `import api FortressWorld`

^a`imports`

Common Types^a

```

component commonTypes
  export Executable
  run(args : String ...) = do
    printType(x : Any) = println(x “: ” (typecase x of
      () ⇒ “()”
      ℤ32 ⇒ “ZZ32”
      else ⇒ “Any”
    end))

    printType5.7; printType true; printType 'c'
  end
end

```

^a local functions, literals (() , numbers, chars, strings), do , typecase , fortify

Common Types^a

```

component commonTypes
  export Executable
  run(args : String ...) = do
    printType(x : Any) = println(x “: ” (typecase x of
      () ⇒ “()”
      ℤ32 ⇒ “ZZ32”
      ℝ64 ⇒ “RR64”
      Boolean ⇒ “Boolean”
      Char ⇒ “Char”
      String ⇒ “String”
      else ⇒ “Any”
    end))
    printType5.7; printType true; printType 'c'
  end
end

```

- [PFC/ProjectFortress/LibraryBuiltin/FortressBuiltin.fsi](#)

^a local functions, literals ((), numbers, chars, strings), do, typecase, fortify

Overloaded Function^a

```
component overloadedFunction
  export Executable
  typeName(_: ()) = “()”
  typeName(_: Z32) = “ZZ32”
  typeName(_: Any) = “Any”
  run(args : String ...) = do
    printType(x) = println(x “: ” (typeName x))
    printType5.7; printType true; printType 'c'
  end
end
```

- <http://projectfortress.sun.com/Projects/Community/report/1>

^aoverloading, overloaded local functions

Overloaded Function^a

```
component overloadedFunction
  export Executable
  typeName(_: ()) = “()”
  typeName(_: Z32) = “ZZ32”
  typeName(_: R64) = “RR64”
  typeName(_: Boolean) = “Boolean”
  typeName(_: Char) = “Char”
  typeName(_: String) = “String”
  typeName(_: Any) = “Any”
  run(args : String ...) = do
    printType(x) = println(x “: ” (typeName x))
    printType5.7; printType true; printType 'c'
  end
end
```

^aoverloading, overloaded local functions

Juxtaposition^a

```
component juxtExample
  import Constants.{ $\pi$ }
  export Executable

  run(args : String...) = do
    var x =  $\pi/4$ 
    y := 4.0
    n = 2
  end
end
```

^alocal variables, operator applications, *if*, *fortify*

Juxtaposition^a

```
component juxtExample
  import Constants.{ $\pi$ }
  export Executable

  run(args : String...) = do
    var x :  $\mathbb{R}64$  =  $\pi/4$ 
    y :  $\mathbb{R}64$  := 4.0
    n = 2
  end
end
```

^alocal variables, operator applications, *if*, *fortify*

Juxtaposition^a

```

component juxtExample
  import Constants.{  $\pi$  }
  export Executable

  run(args : String ...) = do
    var x :  $\mathbb{R}64$  =  $\pi/4$ 
    y :  $\mathbb{R}64$  := 4.0
    n = 2
    u = n(n + 1) sin 3n x log log y
    w = (n(n + 1))(sin(3n x))(log(log y))
    if u  $\neq$  w then println("FAIL: " u "  $\neq$  parenthesized equivalent " w)
      else println("OK")
    end
  end
end
end

```

^alocal variables, operator applications, if, fortify

Juxtaposition^a

```
component juxtExample
  import Constants.{  $\pi$  }
  export Executable

  run(args : String ...) = do
    var x :  $\mathbb{R}64$  =  $\pi/4$ 
    y :  $\mathbb{R}64$  := 4.0
    n = 2
    u =  $n(n + 1) \sin 3n x \log \log y$ 
    w =  $(n(n + 1))(\sin(3n x))(\log(\log y))$ 
    assert(u, w, "FAIL: ", u, " != parenthesized equivalent ", w)
  end
end
```

- [PFC/Library/FortressLibrary.fsi](#)
- [PFC/ProjectFortress/tests/](#)

^alocal variables, operator applications, `if`, `fortify`

Case and Extremum Expressions^a

```
component caseExtremum
  export Executable
  mile = 1.6
  km = 1
  run(args : String...) = do
    case km of
      0 # 2 ⇒ println "OK"
      else ⇒ throw FailCalled("FAIL.")
    end
  end
end
end
```

^a case, extremum expressions, throw, ranges, fortify

Case and Extremum Expressions^a

```
component caseExtremum
  export Executable
  mile = 1.6
  km = 1
  run(args : String...) = do
    case km of
      0 # 2 ⇒ println "OK"
      else ⇒ throw Failed("FAIL.")
    end

    case most > of
      1 mile ⇒ println "Miles are larger."
      1 km ⇒ throw Failed("Miles are larger.")
    end
  end
end
```

^a case, extremum expressions, throw, ranges, fortify

Label and Exit^a

```

component labelExit
  export Executable
  run(args : String ...) = do
    index: Z32 := 0
    println(label found
            while index < |args| do
              if args[index] = "key" then exit found with index end
              index += 1
            end
            -1
          end found)
  end
end

```

- fortress labelExit.fss
- fortress labelExit.fss label exit test finding key value

^a label, exit, while, assignments, varargs parameters, fortify

Arrays^a

```

component arrayExample
  export Executable
  run(args : String ...) = do
    a = [ 10 11
          12 13 ]
  end
end

```

- [PFC/Library/FortressLibrary.fsi](#)

^a arrays

Arrays^a

```

component arrayExample
  export Executable
  run(args : String ...) = do
    a : Z32[2, 2] = [ 10 11
                    12 13]
  end
end

```

^aarrays

Arrays^a

```

component arrayExample
  export Executable
  run(args : String ...) = do
    a : Z32[2, 2] = [ 10 11
                    12 13 ]
    b : Array2[[Z32, 2, 0, 2, 0]] = [ 10 11; 12 13 ]
    println(a00 " " b01)
    println(b10 " " a11)
  end
end

```

^aarrays

Arrays^a

```

component arrayExample
  export Executable
  run(args : String ...) = do
    a : Z32[2, 2] = [ 10 11
                    12 13 ]
    b : Array2[[Z32, 2, 0, 2, 0]] = [ 10 11; 12 13 ]
    println(a00 " " b01)
    println(b10 " " a11)

    z : String[3] = array1[[String, 3]]()
    z.fill("cat"); z1 := "dog"
    println(z.get(0)); println(z.rank())
  end
end

```

^a arrays

Sets^a

```

component setExample
  export Executable
  run(args : String ...) = do
    s = { e | i ← 0 # 5, e ← {2i, 2i + 1} }
    t = { e | i ← 0 : 5, e ← {2i, 2i + 1} }
    println s
    println t
  end
end

```

^asets, comprehensions, ranges

Sets^a

```

component setExample
  import Set.{...}
  export Executable
  run(args : String ...) = do
    s = { e | i ← 0#5, e ← {2i, 2i + 1} }
    t = { e | i ← 0:5, e ← {2i, 2i + 1} }
    println s
    println t
  end
end

```

- [PFC/Library/Set.fsi](#)

^asets, comprehensions, ranges

Maps^a

```

component mapExample
  import Map.{...}
  export Executable
  run(args : String ...) = do
    map = { "David" ↦ 3, "Elias" ↦ 4, "Peter" ↦ 1 }
    for (k, v) ← map do println(k " maps to " v) end
  end
end

```

- [PFC/Library/Map.fsi](#)

^amaps, for, try

Maps^a

```

component mapExample
  import Map.{...}
  export Executable
  run(args : String ...) = do
    map = { "David" ↦ 3, "Elias" ↦ 4, "Peter" ↦ 1 }
    for (k, v) ← map do println(k " maps to " v) end
    try r = map["Mary"]
      throw FailCalled("Found Mary |-> " r)
    catch x
      NotFound ⇒ println("Mary is not found.")
      Any ⇒ throw FailCalled("Unexpected exception")
    end
  end
end
end

```

^amaps, for, try

Lists^a

```

component listExample
  import List.{...}
  export Executable
  run(args : String ...) = do
    list' = ⟨7, 5, 3⟩
    list'' = ⟨8, 6, 4⟩
    l = list'.append(list'')
  end
end

```

- [PFC/Library/List.fsi](#)

^alists, function expressions, reductions

Lists^a

```
component listExample
  import List.{...}
  export Executable
  fiveOrMore = fn x ⇒ x > 4
  run(args : String ...) = do
    list' = ⟨7, 5, 3⟩
    list'' = ⟨8, 6, 4⟩
    l = list'.append(list'')
    assert(l.filter(fiveOrMore), ⟨7, 5, 8, 6⟩, “filter”)
  end
end
```

^alists, function expressions, reductions

Lists^a

```
component listExample
  import List.{...}
  export Executable
  fiveOrMore = fn x ⇒ x > 4
  run(args : String ...) = do
    list' = ⟨7, 5, 3⟩
    list'' = ⟨8, 6, 4⟩
    l = list'.append(list'')
    assert(l.filter(fiveOrMore), ⟨7, 5, 8, 6⟩, “filter”)
    assert(BIG ∧ [e ← l] fiveOrMore(e), false, “all five or more”)
  end
end
```

^alists, function expressions, reductions

Type Annotations^a

```
component typeAnnotation
  import List.{...}
  export Executable
  run(args : String ...) = do
    a: List[[R64]] = ⟨ 1.0, 2.0, 3.0, 4.0 ⟩
  end
end
```

^a as, asif

Type Annotations^a

```
component typeAnnotation
  import List.{...}
  export Executable
  run(args : String ...) = do
    a: List[[R64]] = ⟨ 1.0 asif R64, 2.0, 3.0, 4.0 ⟩
  end
end
```

^a as, asif

Type Annotations^a

```
component typeAnnotation
  import List.{...}
  export Executable
  run(args : String ...) = do
    a: List[[R64]] = ⟨ 1.0 asif R64, 2.0, 3.0, 4.0 ⟩
    assert(a, ⟨1, 2, 3, 4⟩, “ lists ought to compare at RR64”)
  end
end
```

^a as, asif