**Evaluating Postfix and its Correctness**

[sealed](http://scala-lang.org) [abstract](http://scala-lang.org) [class](http://scala-lang.org) Expr

[case](http://scala-lang.org) [class](http://scala-lang.org) [Var](http://scala-lang.org)(varID: String) [extends](http://scala-lang.org) Expr

[case](http://scala-lang.org) [class](http://scala-lang.org) IntLiteral(value: Int) [extends](http://scala-lang.org) Expr

[case](http://scala-lang.org) [class](http://scala-lang.org) Plus(lhs: Expr, rhs: Expr) [extends](http://scala-lang.org) Expr

[case](http://scala-lang.org) [class](http://scala-lang.org) Times(lhs: Expr, rhs: Expr) [extends](http://scala-lang.org) Expr

[sealed](http://scala-lang.org) [abstract](http://scala-lang.org) [class](http://scala-lang.org) Token

[case](http://scala-lang.org) [class](http://scala-lang.org) ID(str : String) [extends](http://scala-lang.org) Token

[case](http://scala-lang.org) [class](http://scala-lang.org) Const(c : Int) [extends](http://scala-lang.org) Token

[case](http://scala-lang.org) [class](http://scala-lang.org) Add [extends](http://scala-lang.org) Token

[case](http://scala-lang.org) [class](http://scala-lang.org) Mul [extends](http://scala-lang.org) Token

[object](http://scala-lang.org) Print {

[def](http://scala-lang.org) postfix(e : Expr) : List[Token] = e [match](http://scala-lang.org) {

[case](http://scala-lang.org) [Var](http://scala-lang.org)(id) => List(ID(id))

[case](http://scala-lang.org) IntLiteral(c) => List(Const(c))

[case](http://scala-lang.org) Plus(lhs,rhs) => postfix(lhs) ::: postfix(rhs) ::: List(Add())

[case](http://scala-lang.org) Times(lhs,rhs) => postfix(lhs) ::: postfix(rhs) ::: List(Mul())

}

}

[object](http://scala-lang.org) Eval {

// recursive interpretation of expression

[def](http://scala-lang.org) infix(env : Map[String,Int], expr : Expr) : Int = expr [match](http://scala-lang.org) {

[case](http://scala-lang.org) [Var](http://scala-lang.org)(id) => env(id)

[case](http://scala-lang.org) IntLiteral(v) => v

[case](http://scala-lang.org) Plus(e1,e2) => infix(env,e1) + infix(env,e2)

[case](http://scala-lang.org) Times(e1,e2) => infix(env,e1) \* infix(env,e2)

}

// non-recursive evaluation of postfix expression

[def](http://scala-lang.org) postfix(env : Map[String,Int],

pexpr : Array[Token]) : Int = {

[var](http://scala-lang.org) stack : Array[Int] = [new](http://scala-lang.org) Array[Int](512)

[var](http://scala-lang.org) top : Int = 0

[var](http://scala-lang.org) pos : Int = 0

[while](http://scala-lang.org) (pos < pexpr.length) {

pexpr(pos) [match](http://scala-lang.org) {

[case](http://scala-lang.org) ID(v) =>

top = top + 1

stack(top) = env(v)

[case](http://scala-lang.org) Const(c) =>

top = top + 1

stack(top) = c

[case](http://scala-lang.org) Add() =>

stack(top - 1) = stack(top - 1) + stack(top)

top = top - 1

[case](http://scala-lang.org) Mul() =>

stack(top - 1) = stack(top - 1) \* stack(top)

top = top - 1

}

pos = pos + 1

}

[return](http://scala-lang.org) stack(top)

}

}

[object](http://scala-lang.org) Test {

[def](http://scala-lang.org) main(args : Array[String]) = {

[val](http://scala-lang.org) expr = Plus([Var](http://scala-lang.org)("x"),Times([Var](http://scala-lang.org)("y"),[Var](http://scala-lang.org)("z")))

[val](http://scala-lang.org) env = Map("x"->3, "y"->4, "z"->5)

println("env = " + env)

[val](http://scala-lang.org) resInterpreted = Eval.infix(env,expr)

println("resInterpreted = " + resInterpreted)

[val](http://scala-lang.org) pexpr : Array[Token] = Print.postfix(expr).toArray

println("postfix: " + (pexpr.toList))

[val](http://scala-lang.org) resCompiled = Eval.postfix(env,pexpr)

println("resCompiled = " + resCompiled)

[val](http://scala-lang.org) theoremHolds =

(Eval.postfix(env,Print.postfix(expr).toArray) ==

Eval.infix(env,expr))

println("theoremHolds (for this case) = " + theoremHolds)

}

}