```
abstract class Tipo
case class Inteiro() extends Tipo
case class Boleano() extends Tipo
case class Unidade() extends Tipo
case class Funcao(t1: Tipo, t2: Tipo) extends Tipo
case class Refer(t: Tipo) extends Tipo


abstract class Expr
case class N (n:Int) extends Expr
case class B (b:Boolean) extends Expr
case class Sum (e1: Expr, e2: Expr) extends Expr
case class Prod (e1: Expr, e2: Expr) extends Expr
case class Dif (e1: Expr, e2: Expr) extends Expr
case class Eq (e1: Expr, e2: Expr) extends Expr
case class If (e1: Expr, e2: Expr, e3: Expr) extends Expr
case class Asg (e1: Expr, e2: Expr) extends Expr
case class Deref (e: Expr) extends Expr
case class Ref (e:Expr) extends Expr
case class Skip() extends Expr
case class Seq (e1: Expr, e2: Expr) extends Expr
case class W (e1: Expr, e2: Expr) extends Expr
case class Fn (s:String, t: Tipo, e: Expr) extends Expr
case class App (e1: Expr, e2: Expr) extends Expr
case class X (s:String) extends Expr
case class Let (s:String, t: Tipo, e1: Expr, e2: Expr) extends Expr
case class LetRec (f: Tipo, e1: Expr, e2: Expr) extends Expr


class L3Interpreter {

// Verificador de Tipos


  def typecheck(e:Expr, gamma: List[(String,Tipo)]) : Option[Tipo] =
     e match {
        case N (_) => Some(Inteiro())
        case B (_) => Some(Boleano())
        case Sum (e1, e2) =>
           (typecheck(e1,gamma), typecheck(e2,gamma)) match {
             case (Some(Inteiro()), Some(Inteiro())) => Some(Inteiro())
             case _ => None
           }
 /*     case Prod (e1, e2) =>
        case Dif (e1, e2) =>
        case Eq (e1, e2) =>
        case If (e1, e2, e3) =>
        case Asg (e1, e2) =>
        case Deref (e) =>
        case Ref (e:Expr) =>
        case Skip() =>
        case Seq (e1, e2) =>
        case W (e1, e2) =>
        case Fn (s:String, t: Tipo, e) =>
        case App (e1, e2) =>
        case X (s:String) =>
        case Let (s:String, t: Tipo, e1, e2) =>
        case LetRec (f: Tipo, e1, e2) =>
     }
*/

// Avaliacao

  def isvalue(e:Expr) : Boolean = e match {
    case N(_) => true
    case X(_) => true
    case B(_) => true
    case Fn(_,_,_) => true
    case Skip() => true
    case _ => false
  }

  type Endereco = String

  type Memoria = List[(Endereco,Int)]

  def step(e: Expr, sigma: Memoria): Option[(Expr, Memoria)] =  e match {
    case N(_) => None
    case B(_) => None
    case Sum (e1, e2) => (e1,e2) match{
        case (N(n1),N(n2)) => Some ((N(n1 + n2), sigma))
        case (e1, e2) => if (isvalue(e1)) {
                          step(e2,sigma) match {
                            case Some((e2lin, sigmalin)) =>
                                          Some((Sum(e1,e2lin), sigmalin))
                            case None => None
                          }
                        } else {
                          step(e1, sigma) match {
                            case Some((e1lin, sigmalin)) =>
                                          Some((Sum(e1lin, e2), sigmalin))
                            case None => None
                          }
```

```
                       }
        }
//  case Prod (e1, e2) => ...
//  case Dif (e1, e2) => ...
//  case Eq (e1, e2) => ...
    case If(B(true), e2, e3)  => Some(e2, sigma)
    case If(B(false), e2, e3) => Some(e3, sigma)
//  case If(e1, e3, e3) => ....
//  .....
  }

  def eval(e: Expr, sigma:Memoria): Option[(Expr, Memoria)] =
   step(e,sigma) match {
      case None => Some((e,sigma))
      case Some((elin, sigmalin)) => eval(elin, sigmalin)
   }
}




object L3 {
  def main (args: Array[String]) {

// Expressao e memoria para teste

    val ex:Expr = Sum(Sum(N(5),N(10)), Sum(N(10),N(100)))
    val sigma: List[(String,Int)] = List(("l1",5), ("l2",7))
    val gamma: List[(String,Tipo)] = List(("x",Inteiro()), ("y", Inteiro()))

    val interpretador = new L3Interpreter()

    val tipo = interpretador.typecheck(ex,gamma)

    val res = interpretador.eval(ex,sigma)

    println()
    println("Expressao L3:  " + ex)
    println()
    println("Tipo: " + tipo)
    res match {
     case Some((exp_final, sigma_final)) =>
           println("Resultado da avaliacao de (5 + 10) + (10 + 100): " + exp_final)
           println(sigma_final)
    }
}
}
```