```
      structure Exp: EXPRESSION
      structure Val: VALUE
      exception Unimplemented
      val evaluate: Exp.Expression -> Val.Value
   end

                  (* type checking *)
signature TYPE =
   sig
      type Type

(*constructors and decstructors*)
      exception Type
      val mkTypeInt: unit -> Type
          and unTypeInt: Type -> unit

      val mkTypeBool: unit -> Type
          and unTypeBool: Type -> unit

      val prType: Type->string
   end



signature TYPECHECKER =
   sig
      structure Exp: EXPRESSION
      structure Type: TYPE
      exception NotImplemented of string
      exception TypeError of Exp.Expression * string
      val typecheck: Exp.Expression -> Type.Type
   end;

                  (* the interpreter*)

functor Interpreter
   (structure Ty: TYPE
    structure Value : VALUE
    structure Parser: PARSER
    structure TyCh: TYPECHECKER
    structure Evaluator:EVALUATOR
      sharing Parser.E = TyCh.Exp = Evaluator.Exp
          and TyCh.Type = Ty
```

```
          and Evaluator.Val = Value
   ): INTERPRETER=

struct
  val eval= ref true    (* toggle for evaluation *)
  and tc  = ref true    (* toggle for type checking *)
  fun interpret(str)=
    let val abstsyn= Parser.parse str
        val typestr= if !tc then
                     Ty.prType(TyCh.typecheck abstsyn)
                     else "(disabled)"
        val valuestr= if !eval then
                  Value.printValue(Evaluator.evaluate abstsyn)
                      else "(disabled)"

    in  valuestr ^ " : " ^ typestr
    end
    handle Evaluator.Unimplemented =>
               "Evaluator not fully implemented"
         | TyCh.NotImplemented msg =>
               "Typechecker not fully implemented " ^ msg
         | Value.Value   => "Run-time error"
         | Parser.Syntax msg => "Syntax Error: " ^ msg
         | Parser.Lexical msg=> "Lexical Error: " ^ msg
         | TyCh.TypeError(_,msg)=> "Type Error: " ^ msg
end;

                    (* the evaluator *)

functor Evaluator
  (structure Expression: EXPRESSION
   structure Value: VALUE):EVALUATOR=

   struct
      structure Exp= Expression
      structure Val= Value
      exception Unimplemented

      local
         open Expression Value
         fun evaluate exp =
            case exp
              of BOOLexpr b => mkValueBool b
```

```
              | NUMBERexpr i => mkValueNumber i
              | SUMexpr(e1, e2) =>
                  let val e1' = evaluate e1
                      val e2' = evaluate e2
                  in
                     mkValueNumber(unValueNumber e1' +
                                   unValueNumber e2')
                  end

              | DIFFexpr(e1, e2) =>
                  let val e1' = evaluate e1
                      val e2' = evaluate e2
                  in
                     mkValueNumber(unValueNumber e1' -
                                   unValueNumber e2')
                  end

              | PRODexpr(e1, e2) =>
                  let val e1' = evaluate e1
                      val e2' = evaluate e2
                  in
                     mkValueNumber(unValueNumber e1' *
                                   unValueNumber e2')
                  end

              | EQexpr _ => raise Unimplemented
              | CONDexpr _ => raise Unimplemented
              | CONSexpr _ => raise Unimplemented
              | LISTexpr _ => raise Unimplemented
              | DECLexpr _ => raise Unimplemented
              | RECDECLexpr _ => raise Unimplemented
              | IDENTexpr _ => raise Unimplemented
              | LAMBDAexpr _ => raise Unimplemented
              | APPLexpr _ => raise Unimplemented

      in
         val evaluate = evaluate
      end
   end;

                      (* the typechecker *)

functor TypeChecker
```

```
  (structure Ex: EXPRESSION
   structure Ty: TYPE)=
struct
  structure Exp = Ex
  structure Type = Ty
  exception NotImplemented of string
  exception TypeError of Ex.Expression * string

  fun tc (exp: Ex.Expression): Ty.Type =
    case exp of
      Ex.BOOLexpr b => raise NotImplemented
                         "(boolean constants)"
    | Ex.NUMBERexpr _ => Ty.mkTypeInt()
    | Ex.SUMexpr(e1,e2)  => checkIntBin(e1,e2)
    | Ex.DIFFexpr _ => raise NotImplemented "(minus)"
    | Ex.PRODexpr _ => raise NotImplemented "(product)"
    | Ex.LISTexpr _ => raise NotImplemented "(lists)"
    | Ex.CONSexpr _ => raise NotImplemented "(lists)"
    | Ex.EQexpr _ => raise NotImplemented "(equality)"
    | Ex.CONDexpr _ => raise NotImplemented "(conditional)"
    | Ex.DECLexpr _ => raise NotImplemented "(declaration)"
    | Ex.RECDECLexpr _ => raise NotImplemented "(rec decl)"
    | Ex.IDENTexpr _   => raise NotImplemented "(identifier)"
    | Ex.LAMBDAexpr _  => raise NotImplemented "(function)"
    | Ex.APPLexpr _ => raise NotImplemented "(application)"

  and checkIntBin(e1,e2) =
    let val t1 = tc e1
        val _  = Ty.unTypeInt t1
                 handle Ty.Type=>
                 raise TypeError(e1,"expected int")
        val t2 = tc e2
        val _  = Ty.unTypeInt t2
                 handle Ty.Type=>
                 raise TypeError(e2,"expected int")
     in Ty.mkTypeInt()
    end;

  val typecheck = tc

end; (*TypeChecker*)
```

```
                    (* the basics -- nullary functors *)

functor Type():TYPE =
struct
  datatype Type = INT
                | BOOL

  exception Type

  fun mkTypeInt() = INT
  and unTypeInt(INT)=()
    | unTypeInt(_)= raise Type

  fun mkTypeBool() = BOOL
  and unTypeBool(BOOL)=()
    | unTypeBool(_)= raise Type

  fun prType INT = "int"
  |   prType BOOL= "bool"
end;



functor Expression(): EXPRESSION =
   struct
      type 'a pair = 'a * 'a

      datatype Expression =
         SUMexpr of Expression pair   |
         DIFFexpr of Expression pair   |
         PRODexpr of Expression pair   |
         BOOLexpr of bool   |
         EQexpr of Expression pair   |
         CONDexpr of Expression * Expression * Expression   |
         CONSexpr of Expression pair   |
         LISTexpr of Expression list   |
         DECLexpr of string * Expression * Expression   |
         RECDECLexpr of string * Expression * Expression   |
         IDENTexpr of string   |
         LAMBDAexpr of string * Expression   |
         APPLexpr of Expression * Expression   |
         NUMBERexpr of int
```