

# SML Datatype Example

## CptS 355

10th October 2005

```
type unaryOp = int -> int
type binaryOp = (int*int) -> int
datatype ExpTree = Leaf of int
                  | Unary of (unaryOp*ExpTree)
                  (* Unary illustrates use of a tuple in a datatype *)
                  | Binary of {left: ExpTree, f: binaryOp, right: ExpTree}
                  (* Binary illustrates use of a record in a datatype *)

(* ExpTree is example of using datatype to define a *discriminated union*
 * type. Similar to union in C or variant record in pascal but any value of
 * the type is automatically tagged with the branch to which it belongs.
 * This tag can be used in function defs and case expressions to choose the
 * code to use for any particular value
 *)

fun eval0 t = (case t of
               (Leaf i) => i
               | (Unary (f, t)) => f (eval0 t)
               (* The Unary case shows use of pattern matching on a tuple *)
               | (Binary n) => (#f n) (eval0 (#left n), eval0 (#right n))
               (* The Binary case shows use of field selectors on a record *)
               )

(* eval0 shows the use of a case expression to deconstruct an ExpTree *)

fun eval1 (Leaf i) = i
  | eval1 (Unary (f, t)) = f (eval1 t)
  | eval1 (Binary n) = (#f n) (eval1 (#left n), eval1 (#right n))
(* eval1 shows use of a multi-branch fun declaration instead of a case expression *)

fun eval2 (Leaf i) = i
  | eval2 (Unary n) = (#1 n) (eval2 (#2 n))
  (* Here the Unary case shows use of field selectors on a tuple *)
  | eval2 (Binary {left, f, right}) = f (eval2 left, eval2 right)
  (* ... and Binary shows use of pattern matching on a tuple *)

val one = Leaf 1
val two = Leaf 2
val three = Leaf 3
val four = Leaf 4
val fourPlusThree = Binary {left=four, f=(op +), right=three}
val negTwo = Unary (op ~, two)
```

```
val twoTimesThree = Binary {left=two, f=(op * ), right=three}
val fourPlusThreePlustwoTimesThreeTimesnegTwo = Binary{left=Binary{
    left=fourPlusThree,
    f=(op +),
    right=twoTimesThree},
    f=(op * ),
    right=negTwo}
```