Chapter 2: Types, Values, and Effects

1. Computation in Ml consists of evaluation of expressions. Each expression has three important characteristics
    a. It may or may not have a type
    b. It may or may not have a value
    c. It may or may not engender an effect
2. Type Checking
    a. A name for the type
    b. The values of the type
    c. The operations that may be performed on values of the type
3. Examples of Arithmetic expressions
    a. 3, 3 + 4, 4 div 3, 4 mod 3
4. Types
    a. Real
        i. Value: 3, 3.13, 0.1E6, ~1
        ii. Operations: +, -, *, /, <, …
    b. Char
        i. Values: #"a", #"b"
        ii. Operations: ord, chr, =, <, …
    c. String
        i. Values: "abc", "1234"
        ii. Operations: ^, size, =, <, …
    d. Bool
        i. Values: true, false
        ii. Operations: if exp then exp else exp
5. Overloading
    a. Adding floating point and integer values yield error (3 + 3.14)
        i. To fix the error, have to real(3) + 3.14, which converts integer 3 to floating point
    b. Use div for integer, use / for floating point division
    c. In if-else statement, if one clause is evaluated, the other is simply discarded without further consideration
        i. Therefore, if 1 < 2 else 0 else (1 div 0) does not yield error despite the 1 div 0 error since 1 < 2 is always true and (1 div 0) is discarded
    d. If-else can also be written as
        i. if not exp then exp1 else exp2
6. Type errors
    a. Misusing operator
        i. #"1" + 1 (adding char and int)

ii. #”2” ^ 2 (concatenating char and string)

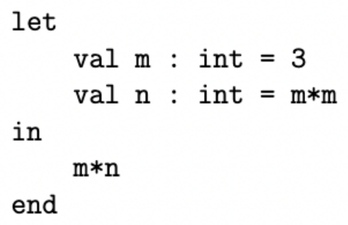
iii. 3.13 + 2 (adding integer and float)

Chapter 3: Declarations

1. Basic Bindings
   a. Type bindings
      i. type float = real
      ii. type count = int and average = real
      iii. Introduces one or more new type constructors simultaneously in the sense that the definitions of the type constructors may not involve any of the type constructors being defined
      iv. Therefore, type float = real and average = float is nonsensical since the type constructors float and average are introduced simultaneously, and hence cannot refer to one another
      v. This means that…
         1. type var$_1$ = typ$_1$ and … and var$_n$ = typ$_n$
   b. Value bindings
      i. val m : int = 3+2
      ii. val pi : real = 3.14 and e : real = 2.17
      iii. Value binding specifies both tye type and value of a variable
      iv. The purpose of binding is to make a variable available for use within its scope
2. Compound Declarations
   a. Bindings may be combined to form declarations
   b. We may write declaration
      i. val m : int = 3 + 2
      ii. val n: int = m * m (binds m to 5 and n to 25)
      iii. Binding is not assignment (binding of variable never changes; once bound to a value, it is always bound to that value within the scope of the binding)
      iv. Shadow of a binding by introducing a second binding for variable within the scope of the first binding
      v. val n: real = 2.17
         val n: real = 25
3. Limiting Scope
   a. Scope of a variable or type constructor may be delimited by using let expressions and local declarations
   b. Let:
      let dec in exp end
   c. The scope of the declaration dec is limited to the expression exp. The binding introduced by dec are discarded upon completion of evaluation of exp
   d. Local:
      local dec in dec’ end

e. The scope of the bindings in dec is limited to the declaration dec’. After processing dec’, the bindings in dec may be discarded

f. Example 1

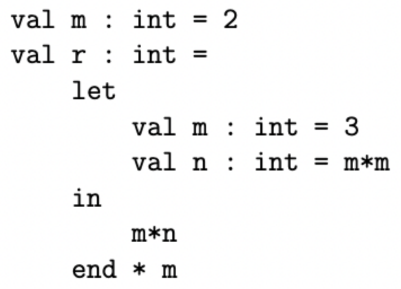
   i. 

```
let
    val m : int = 3
    val n : int = m*m
in
    m*n
end
```

   ii. This expression has type int and value 27

   iii. Bindings for m and n are local to the expression m * n, and are not accessible from outside the expression

g. Example 2

   i. 

```
val m : int = 2
val r : int =
    let
        val m : int = 3
        val n : int = m*m
    in
        m*n
    end * m
```

   ii. Evaluates to 54

   iii. The binding of m is temporarily overridden during evaluation of the let expression, then restored upon completion of this evaluation

Chapter 4: Functions

1. Functions and Application
    a. The values of function type consist of primitive functions, such as addition, square root, and function expressions (also called lambda expressions) of the form fn var: typ => exp
    b. Example
        i. fn x : real => Math.sqrt (Math. sqrt x)
        ii. (fn x : real => Math.sqrt (Math.sqrt x)) (16.0), → calculates the fourth root of 16.0
    c. Giving name
        i. val fourthroot: real -> real = fn x : real => Math.sqrt (Math.sqrt x)
    d. Function writing
        i. fun fourthroot (x : real) : real = Math. sqrt (Math.sqrt x)
    e. Function applications in ML are evaluated according to the call-by-value rule: arguments to a function are evaluated before function is called
    f. For example, when calling fourthroot (2.0 + 2.0),
        i. Evaluate fourthroot to the function value fn x: real => Math.sqrt(Math.sqrt x))
        ii. Evaluate the argument 2.0 + 2.0 to its value 4.0
        iii. Bind x to the value 4.0
        iv. Evaluate Math.sqrt (Math.sqrt (Math.sqrt x)) to approximately 1.414
            1. Evaluate Math.sqrt to function value (primitive square root function)
            2. Evaluate the argument expression Math.sqrt x to its value, apprxomiately 2.0
                a. Evaluate Math.sqrt to a function value (the primitive square root function)
                b. Evaluate x to its value 4.0
                c. Compute square root of 4.0, yielding 2.0
            3. Compute square root of 2.0, yielding 1.414
        v. Drop the binding for the variable x
2. Binding and Scope, Revisited
    a. fn var: typ => exp
       Binds the variable var within the body exp of the function
    b. Unlike val bindings, function expressions bind variable without giving it a specific value (value is determined when function is applied, temporarily, for the duration of evaluation of its body)