

CS 3721: Programming Languages Lab

Lab #07: Programming in ML

Due date: March 11, 3:30pm. At the beginning of the next recitation.

Goals of this lab:

- Be able to interact with ML and interpret ML's response ; Be able to write simple expressions of numbers, booleans and tuples ; Be able to list several of ML's basic primitive types and write expressions belonging to those types ; Be able to read/write expressions constructing tuples ; Be able to write *val* declarations and expressions using pattern matching on tuples ; Be able to declare functions using *fun*

1. *Primitive types.* For ML expressions, user interaction with the ML compiler has the form:

```
-<expression>;  
val it = <value> : <type>
```

First line is the user input, and the second one is output from the ML compiler and runtime system. *it* is a special identifier bound to the value of the last expression entered. Therefore, "*it* = <value> : <type>" means that the value of the expression is < *value* > and its type is < *type* >. The keyword *val* stands for value. Now, you've learned how to interpret the output from the ML compiler. Followings are some expressions. Please, write down their values and types (fill in the blanks).

```
- 3487 + (45 - 691); ==> val _____ = _____ : _____  
- "cs utsa";          ==> val _____ = _____ : _____  
- 55 <> 4;            ==> val _____ = _____ : _____  
- (14.3 - 3.7) + 5.8; ==> val _____ = _____ : _____  
- it + 3.6;          ==> val _____ = _____ : _____
```

2. *Operations provided to support int values.* "+, -, *, div" are binary infix operators on integers. Find out the result of $\frac{2+3*6-1}{3}$. Write down your ML expression, and the output. (In ML, 1-tuples are used for grouping subexpressions like parentheses used in C/JAVA.)

3. *Operations provided to support real values.* "+, -, *, /" are binary infix operators on reals. Find out the result of $\frac{2.4+3.2*6.21-1.1}{3.4}$. Write down your ML expression, and the output.

4. *Operations provided to support bool values.* "andalso, orelse, not" are operators on bools. Find out the result of the expression, "*true* \wedge 2 = 2) \vee 5 <> 5 \vee \neg (3 = 4)". Write down your ML expression, and the output. ("<>" is the *not equal* operator.)

5. *Operation provided to support string values.* String concatenation operator, “^”, is provided for string values. For example,

```
- "Baris" ^ " " ^ "Tas"; ==> val it = "Baris Tas" : string
```

Please, write down an expression that concatenates your full name (First, last, and middle names - if you have a middle name-), and the output.

6. *Erroneous operations.* As we have seen, the arithmetic operators “+, -, *” may be applied to either integers or real numbers. Note that when + has two integer arguments the result is an integer, and when + has two real arguments the result is a real number. However, it is a type error to combine integer and real arguments. Followings are erroneous expressions. For each expression, what causes the error? (*Hint:* Error messages tell a lot!)

```
- 5 - 3.2; ==> reason:
- not 5 = 5; ==> reason:
- 6 andalso true; ==> reason:
- 5 / 2; ==> reason:
- 7.3 div 5.2; ==> reason:
- 4 ^ 4; ==> reason:
- if true then 3 else false; ==> reason:
```

7. *Tuples.* A tuple may be a pair, triple, quadruple, and so on. They may be formed of any types of values. Followings are some expressions containing tuples. Please, write down their values and types as you did in question 1.

```
- (4, 5); ==>
- (3.2, 4.3, 3.1); ==>
- (1, 2.2, 3, 3.3); ==>
- (4+3, 2.1+3.1, 2=2, 2<>2 orelse false, "utsa");
  ==>
```

The operator “#” is provided to pull apart the components in a tuple. For example, #1 selects the first component, #2 selects the second component, and so on.

An example:

```
- #2(1, 2, 3); ==> val it = 2 : int (* 2nd component is accessed.*)
```

Now, write an expression in order to access the fourth component of the following tuple:

```
(4+3, 2.1+3.1, 2=2, 2<>2 orelse false, "utsa").
```

Also, write down the output.

8. *Value declarations.* A pattern is an expression containing variables such as x, y, z, ... and constants combined by certain forms such as tupling. And the general form of value declaration associates a value with a pattern. This form is:

```
val <pattern> = <expression>;
```

Examples:

```
- val x = 5; ==> val x = 5 : int
- val r = 5.0 : real; (*explicitly, saying the type of 5.0 is real*)
- val t = (1, 2);
val t = (1,2) : int * int
- val (x, y) = t; (*pattern matching*)
val x = 1 : int
val y = 2 : int
```

Now, write a *val* declaration that assigns $\frac{2.1 * 3.31 - 2.7 + 1.1}{3.4 + 2.0}$ to *x*, your name to *y* (as a string), and the integer 5 to *z*. Also, write down the output. (*Hint*: Since this is only one statement, make use of pattern matching.)

9. *Function declarations.* A function declaration uses tuple pattern as its argument, and it has the form:

```
fun f(<pattern>) = <expression>;
```

An example:

```
- fun f(x) = 5 * x;
val f = fn : int -> int
```

The output says that *f* is a function (where *fn* comes from) taking an *int* pattern as its argument, and returns an *int* value. “->”, in the output, separates the type of the parameter from the type of the result of a function. You can also rewrite the above function as:

```
- fun f x = 5 * x;
val f = fn : int -> int
```

Another example:

```
- fun f(x,y) = x * y;
val f = fn : int * int -> int
```

You may think that the above function takes two parameters *x*, and *y*. Indeed, it takes just one argument, the tuple (*x*, *y*). Pattern matching takes the tuple apart, binds *x* to *x* and *y* to *y*.

What are the types of the following functions (type of the component(s) in the tuple pattern - the argument -, and the return value)?

```
- fun f() = 3 + 5; ==> _____
- fun f(x, y, z) = (x / y, z ^ z); ==> _____
- fun f(a, b, c, d) = a + b - c div (d * a); ==> _____
- fun f(x, y, z, w) = (x = (x + y), z andalso w); ==> _____
```

Now, build a function named *double* that takes the tuple, (*x*, *y*, *z*) as its argument and returns another triple tuple doubling the components *x*, *y*, and *z*. After that, declare values *a*, *b*, and *c* which are the doubles of the arguments *x*, *y*, and *z* doing pattern matching on the result of the function that you built. Test your function and *val* declaration on integers (Pass 1, 2, 3 into your *double* function for *x*, *y*, and *z* respectively.).