

CS 3721: Programming Languages Lab

Lab #09 Solutions: Imperative Programming in ML

Due date: April 1, 3:30pm. At the beginning of the next recitation.

Goals of this lab:

- Be able to read/write short imperative programs in ML using *ref* cells, if, while do.
1. *Reference Cells.* A *reference cell* is the assignable region of memory in ML. A reference is itself a value; if x has type τ then a reference to x is written *ref* x and has type τ *ref*. Operations on reference cells are:

```
ref v => creates a reference cell containig value v
!r    => dereferencing: returns the value contained in reference cell r
r := v => places value v in reference cell r
```

Reference cells correspond to the variables of C, Pascal and similar languages where as the value declarations that we have seen so far correspond to the constants of languages like C. Here are some examples:

```
- val r = ref 5;
val r = ref 5 : int ref
```

The identifier r is bound to a new reference cell with contents 5. The output says that the value of r is this reference cell having type *int ref*.

```
- r := !r + 1;
val it = () : unit
```

The value in the reference cell, r , is incremented.

```
- !r;
val it = 6 : int
```

!r returns the contents of r , which is the integer 6.

- (a) Create three real reference cells x , y and z which have the initial values 3.5, 4.2 and 5.1, respectively. Then, assign the mathematical expression, $\frac{x+y*x-y}{z}$ to the variable x (in this case, the reference cell x). Finally, return the contents of x . (Use *let*.)

Solution:

```
let
  val x = ref 3.5;
  val y = ref 4.2;
  val z = ref 5.1
in
  x := (!x + !y * !x - !y) / !z;
  !x
end;

=> val it = 2.74509803922 : real
```

- (b) Define a reference cell *nameList* which initially has empty list as its value. Then, add your first and last name to this list using the cons operator “::”. The final value of the reference cell should be [“firstName”, “lastName”]. Return the value of the reference cell.

Solution:

```
val nameList = ref ([]:(string list));
nameList := "baris" :: "tas" :: !nameList;
!nameList;
```

2. *Loops*. For iteration, ML has a *while* command, and its syntax is:

```
while  $E_1$  do  $E_2$ 
```

If more than one expression is needed in the *while* body, the following syntax is used:

```
while  $E_1$  do ( $E_2$ ;  $E_3$ ; ...;  $E_n$ )
```

Example:

```
val i = ref 0;
while !i < 10 do i := !i + 1;
```

Now, write both recursive and imperative versions of the function *sum* that computes the expression, $\sum_{i=1}^n i$. The function takes one argument which is *n* in the summation. (Use *let*, *while*, and *reference cells* in the imperative version.)

Test your functions with the following:

```
- sum(15);
val it = 120 : int
```

- (a) Purely functional version:

Solution:

```
fun sum(n) = if n = 1 then 1 else n + sum(n-1);
```

- (b) Imperative version:

Solution:

```
fun sum(n) =
  let
    val i = ref 1;
    val total = ref 0;
  in
    while !i <= n do (
      total := !total + !i;
      i := !i + 1
    );
    !total
  end;
```

3. The following is the pseudocode for the GCD (Greatest Common Divisor) algorithm:

```
gcd(m, n)
begin
  while m != n do
    if m > n
      m = m - n
    else
      n = n - m
    end
  end
  return m
end
```

Write this algorithm in ML, and test your function with the following:

```
- gcd(30, 24);
val it = 6 : int
```

Solution:

```
fun gcd(m, n) =
  let
    val x = ref m;
    val y = ref n;
  in
    while !x <> !y do
      if !x > !y then x := !x - !y else y := !y - !x;
    !x
  end;
```