CAS CS 320
Lec 3

Recursion and Datatype in SML

1. Functional Programming
    a. Names (FP) vs variables (Imperative)
        i. The binding is formed between the name and the value (it is never changed → cannot update, it is permanent)
2. Recursion
    a. General recursion
    b. Tail recursion (loops)
3. Local names
4. Declarations
    a. def foo(): …
    b. val pi = 3.14
    c. val area_of_circle = fn radius => pi * radius * radius
    d. fun area_of_circle(radius) = pi * radius *radius
5. Recursion example
    a. fun f91(x) =
        if x > 100 then x - 100
        else f91(f91(x+11))
    b. In python,
    def f91(x):
        if x > 100:
            return x - 100
        else
            return f91(f91(x+11))
    c. The outside recursion f91(f91(x+11)) is called a tail recursive call (call follows the return keyword immediately)
    d. The inside recursion is not a tail recursive call because it is not the last recursion
6. Tail recursive call and tail recursive function
    a. fun foo(x) = foo(x+1)
    b. Tail recursive function → if all the recursive calls are tail calls, it is tail recursive function
    c. Tail recursive optimization → if you call foo(0), Overflow Exception
    d. def foo(x):
        return foo(x+1)

7. Fibonacci
    a. fun fibo(n) =
        if n < 2 then n
        else fibo(n-2) + fibo(n-1)
    b. The + is the tail call since it is the last thing you do before you return. Therefore, fibo(n-2) and fibo(n-1) are not tail recursive call
8. Datatype
    a. datatype abc = A | B | C (three constructors)
    b. datatype intopt = NONE (indicates that the value given by user is invalid) | SOME of int (input by the user is valid and is convertible to integer)
    c. fun parseInt (rep: string): int =
    (* if a string that is inconvertible to integer is given, raise exception *)
    OR
    fun parseInt (rep: string): intopt → monadic style
9. Integer division
    a. fun intdiv(x: int, y: int): intopt =
        if y <> 0 then SOME(x div y) else NONE
    OR
    fun intdiv(x: int, y: int): int =
        if y <>0 then (x div y) else raise DivZero
10. Recursive datatype
    a. datatype intseq = ISnil | IScons of int * intseq (every constructor has either 0 or 1 argument)
    b. val xs = ISnil
    val ys = IScons(1, xs) → constructing sequence, has one element
    1 → head of sequence
    xs → tail of sequence
    val zs = IScons(2, ys) → has two elements
    Create sequence with arbitrary length
11. Polymorphic datatype
    a. datatype ‘a list = nil (construct empty list) | cons of ‘a * (‘a list)
    Type variable uses “ ‘ ” before a variable
    b. val xs = []
    val ys = 1 :: nil
    OR
    val ys = 1 :: [] (* sequence of length 1 containing one element (which is 1) *)
    c. :: is an operator that is the same as “cons” that needs to be written between two arguments
    val zs = 1 :: 2 :: []

12. Programming (datatype)

a. datatype weekday = Monday | Tuesday | Wednesday | Thursday | Friday

b.
```
fun print_weekday(wday: weekday): unit =
    case wday of
        Monday => print “Monday” | (* pattern matching clause *)
        Tuesday => print “Tuesday” |
        …
```

c.
```
fun intdiv(x: int, y: int): int option =
    if y <> 0 then SOME(x div y) else NONE
```

d.
```
fun use_intdiv(x: int, y: int): unit =
    let
        val opt = intdiv(x,y)
    in
        case opt of
            NONE => print “The divider y is zero!” |
            SOME res => print “The result equals ” ^ Int.toString(res)
             ^ “\n”)
            (* res is the argument of SOME *)
    end
```

e.
```
fun pow_int_int (x: int, y: int): int = (*tail recursive version of power)
    let
        fun loop(y: int, res: int): int =
            if y > 0 then loop (y-1, x * res) else res
    in
        loop(y,1)
    end
```

f.
```
fun list_extend(xs: ‘a list, x0: ‘a): ‘a list =
    (
    case xs of
        nil => [x0] | x1 :: xs => x1 :: (list_extend(xs,x0))
    )
    (* x1 is the head of the list, and xs is the tail of the list, containing all the
    elements after the first entry of the list *)
```

g.
```
fun list_is_nil(xs: ‘a list): bool =
    (
    case xs of nil => true | _ :: _ => false
    )
```

h.
```
fun list_is_cons(xs: ‘a list): bool =
    (
    case xs of nil => false | _ :: _ => true
```

```
)
```

i.
```
fun list_unnil(xs: ‘a list): unit =
    (
    case xs of nil => () | _ => raise ConsMatch320
    )
```

j.
```
fun list_uncons(xs: ‘a list): ‘a * ‘a list =
    (
    case xs of
        x1 :: xs => (x1, xs) |
        _ => raise ConsMatch320
    )
```

13.