Printing values

Print(x): string → unit

print ("cat") => (): unit

print ("cat \n dog") => ()

Sequential composition

print ("tony")

print ("killed")

print ("john")

Statement lists

$(E_1 ; E_2 ; \dots ; E_n) \Rightarrow val(E_n)$

value is value of E_n

(print ("1"); print ("2"); "done") => "done"

1 2

only returns the value of last element in a list

myprint ([1, 2, 3, 4, 5])

1

2

3

4

5

=> $1^2 + 2^2 + 3^2 + 4^2 + 5^2$ => 55: int

```
myPrint : int list => int
fun myPrint [] = 0
  | myPrint (N::L) = (print (int.toString (N));
                      print ("\n");
                      N + N + myPrint(L));
```

myPrint [1, 2, 3] 1::[2, 3]

=> (print ("1"); print ("\n"); 1 + [+ myPrint ([2, 3))

print ("1")

print ("\n")

=> 1 + myPrint 2::[3]

=> 1 + (print("2"); print("\n"); 2 + 2 + myprint [3])

=> print ("2")

print ("\n")

=> 1 + 4 + myprint [3] 3:: []

=> 1 + 4 + (print ("3"); print ("\n"); 3 + 3 + myPrint[])

- print ("3")

- print ("\n")

=> 1 + 4 + 9 + ~~myprint[]~~ 0

=> 14

References & assignment

```
ref(7): int ref
ref [1, 2, 3]: (int list) ref
```

int list ref

```
val x = ref 0;
(x := 1) => ()
(x := 2) => ()
```

```
val L = [ref1, ref2, ref 3]
changeLast (7, L) => ()
L => [ref 1, ref 2, ref 7]
```

```
val L = [1, 2, 3];
L => [1, 2, 3]
newlast (7, L) => [1, 2, 7]
L => [1, 2, 3]
newlast (23, L) => [1, 2, 23]
L => [1, 2, 3]
```

```
fun newlist (Y, [X]) = [Y]
    | newlast (Y, (X::L)) = X :: newlast (Y, L);
```

```
fun changeLast (Y, [X]) = (X := Y)
    | changeLast (Y, (X::L)) = changeLast (Y, L);
```

Dereferencing

```
val L = [ref1, ref 2, ref3];
hd L => ref 1
```

! (hd L) ⟹ 1

! (ref 7) ⟹ 7

sum [ref 1, ref 2, ref 3]

⟹ 1 + 2 + 3 = 6

sum : int ref list → int

```
fun sum [] = 0
  | sum (R::L) = !R + (sum L)
```

int · real · real

node (T1, T2)

leaf (ref N)

```
datatype Btree = leaf of int ref
               | node of Btree * Btree
node(leaf(ref 1), node (leaf (ref 2),
                        leaf (ref 3)))
```

sum of above ⟹ 1 + 2 + 3 = 6

```
fun sum (leaf R) = !R
  | sum (Node (T1, T2)) = sum (Node(T1), Node(T2))
```

sum : Btree → int

```
fun change (N, leaf R) = (R = N)
  | change (N, node (T1, T2))
```

```
                = change (N, T1);
```

change : int * Btree => unit

Iteration

(while $expr_1$ do $expr_2$;)

```
let val N = ref0 in
    while !N < 100
    do (print (!N);
        print ("\n");
        N := !N + 1)
end
```

iteration

```
let val N = exp1
    in exp2
end
```

```
fun printN (-1) = ()
  | printN (N) = (print (N);
                  print ("\n");
                  printN (N-1))
```

recursion