Basic ML

1 Using ML

To launch ML you can use:

- command line "poly"
- if "name" is the file name use on command line "poly < name"

1.1 Types of variables

1. Integer:
   - positive integer → 0, 1, 20
   - negative integer → ∼20 (not - for negative integer)
   - Hexadecimals → 0x124
2. Real: must have decimal point (or E and e letters)
   - positive real → 0, 1, 20
   - negative real →∼20
3. Boolean:
   - true and false (only lower-case)
     e.g.
     > true;
     val it = true: bool
     > 1 = 3;
     val it = false: bool
4. String:
   - use double quotes, e.g. "name"
   - Special characters:
     - \n: newline
     - \t: tab
     - \\: backslash
     - \": double-quote
5. Characters: like string but need to add # at the start
   - single char #"a"

1.2 Arithmetic operators

- $+$ *and* $-$
- $*$ (multiplication)
- / (used for division of reals), div (division of integers, rounding down)
- mod → reminder of integer division
- Precedence rule;
- **e.g.**
  $> 3.0 - 4.5 + 6.7;$
  val it = 5.2: real
  $> 43\ div\ (8\ mod\ 3) * 5;$
  val it = 105: int
  $> 3 + 4.0 \rightarrow$ error: int * real
  Correct: $> 3.0 + 4.0$

1.3 String operators

- symbol ^ to concatenate strings ("string" ^ "string")
- **e.g.**
  > "house" ^ "cat" → "housecat"

1.4 Comparison

- =, <, >, <=, >=, <> (<> means not-equal)
- used to compare integers, reals, characters and strings, but equality and non-equality comparisons of **reals is not allowed**
- **e.g.**
  $> 1 < 2; \rightarrow$ true
  $> 1.0 < 2.0; \rightarrow$ true
  $> 3.0 = 2.0;$ or $> 3.0 <> 2.0; \rightarrow$ error
  $>$ "*abc*" <= "*ab*"; → false
  $>$ "*abc* <= "*ac*"; → true
  $> \#$"*Z*" $<$ #"*a*"; → true

1.5 Logical operations

- andalso: $1 < 2$ *andalso* $3 > 4$; → false
- orelse: $1 < 2$ *orelse* $3 > 4$; → true
- not
  $>$ *not* $1 < 2$; → error
  Correct $>$ *not* $(1 < 2)$; → false

1.6 If-then-else

- **e.g.**
  $>$ *if* $1 < 2$ *then* $3 + 4$ *else* $5 + 6$; → res: 7
- Note:
  - **else is not** optional: *if* $1 < 2$ *then* $3 + 4$; → error
  - **then** and **else** must have same type:
    - $>$ *if* $1 < 2$ *then* $3 + 4$ *else* $5.0 + 6.0$; → error
      Correct $>$ *if* $1 < 2$ *then* $3 + 4$ *else* $5 + 6$;

1.7 Conversion

- **integer to real** → real(num)
  e.g. real(4) → output: 4.0
- **real to integer**:
  - **floor**: round down
    e.g. floor 3.5 → output: 3
    floor ∼3.5 → output: ∼4
  - **ceil**: round up
    e.g. ceil 3.5 → output: 4
    ceil ∼3.5 → output: ∼3
  - **round**: nearest int
    e.g. round 3.5 → output: 4
  - **trunc**: truncate
    e.g. trunc 3.5 → output: 3
- **char to int** → ord <char>
  e.g. ord #"a"; → output: 97
  ord #"a" - ord #"A"; → output: 32
- **int to char**(ASCII) → chr <int>
  e.g. chr 97; → output: #"a"
- **char to string** → str <char>
  e.g. str #"a" → output: "a"

1.8 Variables

- syntax:

  val <name> = <value>
  val <name>:<type> = <value>

  - val a = 5; → int
  - val b = 5.0 → real
  - val char = "a" → char
  - val string = "word" → string

1.9 Tuple

- can contain any types of variables
- **e.g.**
  > (1, 2); or > (1.0, 2);
  > val box = (4, 5.0, "*string*"); → int * real * string
- to print elements of a tuple stored in a variable (#<elem> <var name>):
  > #1 box → output: 4
  > #2 box → output: 5.0
  > #3 box → output: "string"
- complex: val t = ((1,1),(true,1.0),(1.3,"asd")) →((int * int) * (bool * real) * (real * string))

1.10 Lists

- Syntax: [1, 2, 3] → list of int
- all the elements of a list must be of the **same type**
  val L = [2, 3, 4]
- **hd(list name)**: return the first element of a list
  hd(L) → return 2
- **tl(list name)**: return all the elements after the head
  tl(L) → return [3, 4]
- concatenation of lists using **@**:
  > [1, 2]@[3, 4]; → return [1, 2, 3, 4]
  Note: both lists must be the same type, [1, 2] @ ["a", "b"] → error
- add an element to a list using **::**
  > 8::L; → return [8, 2, 3, 4]
  Note: **nil** (**null integer list**)
- string to list: **explode("string")**
  e.g. > explode("abc"); → output: it = [#"a", #"b", #"c"]
- list to string: **implode([<elements>])**
  e.g. > implode([#"1", #"2", #"3"]); → output: it = "123"

1.11 Functions

- Syntax:

  fun <name><param> = <expression>
  fun <name><param>:<type> = <expression>

- **e.g.**

  fun to_upper(c) = chr(ord(c) -32);
  > to_upper(#"b") → it = #"B"

  fun max3 (a:real, b,c) = //maximum of 3 reals
  # if a>b then if a>c then a else c
  # else
  # if b>c then b else c;
  > max3 (5.0,4.0,7.0); → it = 7.0

  fun third l = hd(tl(tl l)); //return the third element of a list
  > third [2,3,4] → it = 4

  fun thirdchar s = third (explode s); //return the third element of a string
  > thirdchar "abcd" → it = "c"

  fun cycle l = tl(l) @ [hd(l)]; //cycle of 1 element
  > cycle [1,2,3,4] \ it = [2,3,4,1]

  fun i_cycle(i,L) = //cycle of i elements
  # if i=0 then L
  # else i_cycle (i-1, cycle(L));
  > i_cycle(1,[2,2,3,4]); → it = [3,4,1,2]

  fun q(a,b,c) = (min3(a,b,c), max3(a,b,c)); //return max and min of 3 numbers
  > q(1,2,3) \ it = (1,3)

  N.B. you can use function to create other functions, like in the previous example

1.11.1 Recursive function

- **e.g.**

```
fun fact n =                                    //factorial
# if n=1 then 1
# else n * fact(n-1);
> fact 5; →  it = 120

fun duplicate L =                               //duplicate the elements of a list
# if L = nil then nil
# else [hd L] @ [hd L] @ duplicate (tl L);
> duplicate [1,4,2]; →  it = [1,1, 4,4, 2,2]
```