```
 1 (* Expressions *)
 2 datatype exp
 3   = Ident of string
 4   | Number of int
 5   | ArithExp of aExp
 6   | BooleanExp of bExp
 7 and aExp
 8   = Plus of exp * exp
 9   | Minus of exp * exp
10   | Multiply of exp * exp
11 and bExp
12   = LargerThan of exp * exp
13   | LessThan of exp * exp
14   | Equal of exp * exp
15
16 (* Create buffer for storing variables *)
17 type variable = string * exp
18 type buffer = variable list
19 (* Cons the new variable to the start of the buffer *)
20 fun buf_set (b: buffer, (k, v): variable) =
21   (k, v)::List.filter (fn (k1, v1) => not (k = k1)) b
22 (* Search buffer for variable with the correct key *)
23 fun buf_get ([]: buffer, key: string) = NONE
24   | buf_get ((k, v)::b: buffer, key: string) =
25     if k = key then SOME(v) else buf_get(b, key)
26
27 (* Error in case variable is not found in buffer *)
28 exception BufferError of string
29 val buffErr : string = "Variable not found!"
30
31 (* Directions *)
32 datatype direction = North | South | East | West
33
34 (* Statements *)
35 datatype stmt
36   = Stop
37   | Move of direction * exp
38   | Assign of string * exp
39   | While of bExp * stmt list
40
41 (* Variable declaration *)
42 type varDecl = string * exp
43
44 (* Position (x, y) *)
45 type position = int * int
46 type start = position
47
48 (* Robot, grid and program *)
49 type robot = varDecl list * start * stmt list
50 type grid = int * int
51 type program = grid * robot
52
53 (* state is for use in evalStmt *)
54 type state = grid * position * buffer
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
```

```
(* Evaluate an expression *)
fun evalExp (b: buffer, Ident x) =
    let val num = buf_get(b, x)
    in if isSome num then evalExp(b, valOf(num)) else raise BufferError buffErr end
  | evalExp (b: buffer, Number x)     = x
  | evalExp (b: buffer, ArithExp x)   = evalArithExp (b, x)
  | evalExp (b: buffer, BooleanExp x) = if evalBoolExp (b, x) then 1 else 0
and evalArithExp (b: buffer, Plus (exp1, exp2)) = evalExp(b, exp1) + evalExp(b, exp2)
  | evalArithExp (b: buffer, Minus (exp1, exp2)) = evalExp(b, exp1) - evalExp(b, exp2)
  | evalArithExp (b: buffer, Multiply (exp1, exp2)) = evalExp(b, exp1) * evalExp(b, exp2)
and evalBoolExp (b: buffer, LargerThan (exp1, exp2)) = evalExp(b, exp1) > evalExp(b, exp2)
  | evalBoolExp (b: buffer, LessThan (exp1, exp2)) = evalExp(b, exp1) < evalExp(b, exp2)
  | evalBoolExp (b: buffer, Equal (exp1, exp2)) = evalExp(b, exp1) = evalExp(b, exp2)

(* Exception if position is out of bounds *)
exception OutOfBoundsException of position

(* Check if position is within the grid *)
fun checkBounds((x, y): grid, (posX, posY): position) =
  if posX >= 0 andalso posY >= 0 andalso posX <= x andalso posY <= y
    then (posX, posY): position else raise OutOfBoundsException (posX, posY)

(* Evaluate a Statement *)
fun move (g: grid, (x0, y0): position, n: int, d: direction) = case d of
    North => checkBounds(g, (x0, y0 + n))
  | South => checkBounds(g, (x0, y0 - n))
  | East => checkBounds(g, (x0 + n, y0))
  | West => checkBounds(g, (x0 - n, y0))

(* Evaluate a statement *)
fun evalStmt (s: state, Stop) = s
  | evalStmt (s: state, While x) =
      evalWhile(s, While x)
  | evalStmt ((board, pos, buf): state, Move (dir, num)) =
      (board, move(board, pos, evalExp(buf, num), dir), buf)
  | evalStmt ((board, pos, buf): state, Assign (k, v)) =
      (board, pos, buf_set(buf, (k, Number (evalExp(buf, v)))))
and evalWhile ((board, pos, buf): state, While (cond, statements)) =
  if evalExp(buf, BooleanExp cond) = 0 then (board, pos, buf)
  else evalWhile(evalStmtList((board, pos, buf), statements), While (cond, statements))
and evalStmtList (s: state, []: stmt list) = s
  | evalStmtList (s: state, (Stop::xs): stmt list) = s
  | evalStmtList (s: state, (x::xs): stmt list) =
      evalStmtList(evalStmt(s, x), xs)

(* Get the position of a state *)
fun get_pos((board, pos, buf): state) : position = pos

(* Interpret the program *)
fun interpret ((g: grid, (decls, p: position, stmtlst): robot): program) =
  let val startState : state = (g, p, decls)
  in get_pos(evalStmtList(startState, stmtlst)) end
  handle OutOfBoundsException pos =>
    (print "Robot fell of grid!\n"; (pos))
```

```
140 (* program - Tests *)
141 val test1 : program =
142   ((64, 64),
143    ([],
144     (23, 30),
145     [Move (West, Number 15),
146      Move (South, Number 15),
147      Move (East, ArithExp
148                    (Plus
149                      (Number 2, Number 3))),
150      Move (North, ArithExp
151                    (Plus
152                      (Number 10, Number 27))),
153      Stop]))
154
155 val test2 : program =
156   ((64, 64),
157    ([("i", Number 5),
158      ("j", Number 3)],
159     (23, 6),
160     [Move (North, ArithExp
161                    (Multiply
162                      (Number 3, Ident "i"))),
163      Move (East, Number 15),
164      Move (South, Number 4),
165      Move (West, ArithExp
166                   (Plus
167                     (ArithExp
168                       (Plus
169                        (ArithExp
170                         (Multiply (Number 2, Ident "i")),
171                         ArithExp
172                         (Multiply (Number 3, Ident "j")))),
173                      Number 5))),
174      Stop]))
175
176 val test3 : program =
177   ((64, 64),
178    ([("i", Number 5),
179      ("j", Number 3)],
180     (23, 6),
181     [Move (North, ArithExp
182                    (Multiply
183                      (Number 3, Ident "i"))),
184      Move (West, Number 15),
185      Move (East, Number 4),
186      While (LargerThan
187                (Ident "j", Number 0),
188              [Move (South, Ident "j"),
189               Assign ("j", ArithExp
190                            (Minus
191                              (Ident "j", Number 1)))]),
192      Stop]))
193
194 val test4 : program =
195   ((64, 64),
196    ([("j", Number 3)],
197     (1, 1),
198     [While (LargerThan
199             (Ident "j", Number 0),
200              [Move (North, Ident "j")]),
201      Stop]))
```