```
(*
    Progetto di Linguaggi di Programmazione - A.A. 2017/18
*)

(*
    Implementiamo gli storage come dizionari, le cui chiavi corrispondono
    a interi (locazioni di memoria)
*)
fun make_storage k v def = fn x => if x = k then v else def;
fun add_to_storage s l v = fn x => if x = l then v else s x;

(* creazione storage *)
val storage = make_storage 1 1 0;
(* aggiunta chiave-valore *)
val storage = add_to_storage storage 2 2;

(* storage dei borrow *)
(* 0 : not borrowed | -1 : mutably borrowed | n>0 : numero di borrow immutabili *)
(*
    Abbiamo preferito usare una rappresentazione numerica per gli stati di borrow
    semplicemente per facilita' di implementazione
*)
val borrows = make_storage 0 0 0;

exception BorrowException;
fun is_borrowed borrow_storage addr = (borrow_storage addr) <> 0;
fun is_mutably_borrowed borrow_storage addr = (borrow_storage addr) < 0;
fun is_immutably_borrowed borrow_storage addr = (borrow_storage addr) > 0;

(* Effettua il borrow mutabile della variabile in una data locaazione *)
fun mut_borrow address borrow_storage =
    if
        (* Si puo' effettuare il borrow mutabile solo per variabili non borrowed *)
        (not (is_borrowed borrow_storage address))
    then
        add_to_storage borrow_storage address ~1
    else
        raise BorrowException;
(* Effettua il borrow immutabile della variabile in una data locazione *)
fun imm_borrow address borrow_storage =
    if
        (not (is_mutably_borrowed borrow_storage address))
    then
        add_to_storage borrow_storage address ((borrow_storage address) + 1)
    else
        raise BorrowException;

datatype Int = ConstInt of int | MutInt of int;
datatype Type = TypeI | TypeMR | TypeCR;
datatype Var = Var of int * Type;
datatype MutRef = MutRef of Var;
datatype ConstRef = ConstRef of Var;
datatype Val = IVal of Int | MRVal of MutRef | CRVal of ConstRef;

fun evalInt(ConstInt(i)) = i
    | evalInt(MutInt(i)) = i;
fun evalVarLoc(Var(l,t)) = l;
fun evalVarVal(Var(l,t), s) = s l;
fun evalVarType(Var(l,t)) = t;
fun evalMutRef(MutRef(v)) = evalVarLoc(v);
fun evalConstRef(ConstRef(v)) = evalVarLoc(v);
fun evalMutDeref(MutRef(v), s) = evalVarVal(v, s);
fun evalConstDeref(ConstRef(v), s) = evalVarVal(v, s);
fun evalVal(IVal(i)) = evalInt(i)
    | evalVal(MRVal(r)) = evalMutRef(r)
```

```
66      | evalVal(CRVal(r)) = evalConstRef(r);
67
68  datatype ExprV =
69      ExprVar of Var
70      | ExprVal of Val
71      | ExprMutDeref of MutRef
72      | ExprConstDeref of ConstRef;
73
74  fun evalExprV (ExprVar v) s = evalVarVal(v,s)
75      | evalExprV (ExprVal v) s = evalVal(v)
76      | evalExprV (ExprMutDeref r) s = evalMutDeref(r,s)
77      | evalExprV (ExprConstDeref r) s = evalConstDeref(r,s);
78
79  (* Espressioni *)
80  datatype Expr = ExprEV of ExprV | ExprMutRef of MutRef | ExprConstRef of ConstRef;
81
82  fun evalExpr (ExprEV(e), s) = evalExprV(e) s
83      | evalExpr (ExprMutRef(r), s) = evalMutRef(r)
84      | evalExpr (ExprConstRef(r), s) = evalConstRef(r);
85
86  (* Istruzioni *)
87  datatype Instr = AssignVar of Var * Expr | AssignRef of Var * Expr;
88
89  (* Il match non esaustivo previene automaticamente assegnamenti non consentiti *)
90  fun   evalInstr(AssignVar(Var(l,TypeI), ExprEV(e)), s, b) =
91          if
92              (not (is_borrowed b l))
93          then
94              (add_to_storage s l (evalExprV e s), b)
95          else
96              raise BorrowException
97      | evalInstr(AssignVar(Var(l,TypeMR), ExprMutRef(e)), s, b) =
98          if
99              (not (is_borrowed b l))
100         then
101             (
102             if
103                 (b (evalMutRef e)) < 0
104             then
105                 (* Annulliamo il borrow mutabile per la variabile correntemente puntata
                        prima di fare il borrow della successiva *)
106                 (add_to_storage s l (evalMutRef e), (mut_borrow (evalMutRef e) (
                        add_to_storage b (s l) 0)))
107             else
108                 (add_to_storage s l (evalMutRef e), (mut_borrow (evalMutRef e) b))
109             )
110         else
111             raise BorrowException
112     | evalInstr(AssignVar(Var(l,TypeCR), ExprConstRef(e)), s, b) =
113         if
114             (not (is_borrowed b l))
115         then
116             (
117             if
118                 (b (evalConstRef e)) > 0
119             then
120                 (* Diminuiamo di 1 il numero di borrow immutabili effettuati sulla variabile
                        precedentemente in borrow *)
121                 (add_to_storage s l (evalConstRef e), (imm_borrow (evalConstRef e) (
                        add_to_storage b (s l) ((b (s l)) + ~1))))
122             else
123                 (add_to_storage s l (evalConstRef e), (imm_borrow (evalConstRef e) b))
124             )
125         else
126             raise BorrowException
```

```
127     | evalInstr(AssignRef(Var(l,TypeMR), ExprEV(e)), s, b) = (add_to_storage s (s l) (
            evalExprV e s), b)
128     | evalInstr(AssignRef(Var(l,TypeMR), ExprMutRef(e)), s, b) = (add_to_storage s (s l) (
            evalMutRef e), (mut_borrow (evalMutRef e) b))
129     | evalInstr(AssignRef(Var(l,TypeMR), ExprConstRef(e)), s, b) = (add_to_storage s (s l) (
            evalConstRef e), (imm_borrow (evalConstRef e) b));
130
131 datatype Program =
132     MakeProgram of Instr
133     | Concat of Program * Instr;
134
135 fun evalProgram(MakeProgram(i),s,b) = evalInstr(i,s,b)
136     | evalProgram(Concat(p,i),s,b) = let val res = evalProgram(p,s,b) in evalInstr(i, #1 res
            , #2 res) end;
137
138 (* Esempio 1 : Assegnamenti *)
139 val v = Var(10, TypeI);
140 val storage = add_to_storage storage (evalVarLoc v) 100; (* dichiara v nella locazione 10,
        inizializziamo tale locazione a 100 *)
141
142 (* Assegnamo un nuovo valore a v *)
143 val i = AssignVar(v, ExprEV(ExprVal(IVal(ConstInt(200)))));
144 val p = MakeProgram(i);
145 val res = evalProgram(p,storage,borrows);
146 val res_storage = #1 res;
147 res_storage (evalVarLoc v); (* Otteniamo 200, come ci aspettavamo *)
148
149 (* Esempio 2 : Assegnamenti attraverso riferimenti *)
150 val v = Var(10, TypeI);
151 val storage = add_to_storage storage (evalVarLoc v) 100;
152
153 val r = Var(20, TypeMR);
154 val i1 = AssignVar(r, ExprMutRef(MutRef(v)));
155 val i2 = AssignRef(r, ExprEV(ExprVal(IVal(ConstInt(200)))));
156 val p = Concat(MakeProgram(i1), i2);
157
158 val res_storage = #1 (evalProgram(p, storage, borrows));
159 res_storage (evalVarLoc v); (* 200 *)
160
161 (* Esempio 3 : Eccezione borrow *)
162 val v = Var(10, TypeI);
163 val storage = add_to_storage storage (evalVarLoc v) 100;
164
165 (* Due borrow mutabili di v *)
166 val r = Var(20, TypeMR);
167 val i1 = AssignVar(r, ExprMutRef(MutRef(v)));
168 val r2 = Var(30, TypeMR);
169 val i2 = AssignVar(r, ExprMutRef(MutRef(v)));
170
171 val p = MakeProgram(i1);
172 val sb = evalProgram(p, storage, borrows); (* OK! *)
173 val p2 = Concat(MakeProgram(i1), i2);
174 val sb2 = evalProgram(p2, storage, borrows); (* BorrowException! Abbiamo eseguito due borrow
        mutabili sulla stessa variabile *)
175
176 (* Esempio 4 : Borrow mutabili e immutabili *)
177 val v = Var(10, TypeI);
178 val storage = add_to_storage storage (evalVarLoc v) 100;
179
180 val r1 = Var(20, TypeCR);
181 val r2 = Var(21, TypeCR);
182 val r3 = Var(22, TypeMR);
183 val i1 = AssignVar(r1, ExprConstRef(ConstRef(v)));
184 val i2 = AssignVar(r2, ExprConstRef(ConstRef(v)));
185 val i3 = AssignVar(r3, ExprMutRef(MutRef(v)));
```

```
val p1 = MakeProgram(i1);
val p2 = Concat(p1, i2);
val p3 = Concat(p2, i3);

evalProgram(p1, storage, borrows); (* OK! *)
evalProgram(p2, storage, borrows); (* OK! - Posso eseguire pi  borrow immutabili *)
evalProgram(p3, storage, borrows); (* BorrowException! Borrow mutabile dopo borrow
    immutabili *)

(* Esempio 5 :
    let mut x = 5;
    let mut y = 6;
    let mut a = &mut x;
    a = &mut y;
    *a = 7;
    println!("{}", x); // 5
    println!("{}", y); // 7
*)

(* Dichiarazione di variabili *)
val x = Var(1, TypeI);
val y = Var(2, TypeI);
val a = Var(3, TypeMR);

(* Istruzioni *)
val i1 = AssignVar(x, ExprEV(ExprVal(IVal(ConstInt(5)))));
val i2 = AssignVar(y, ExprEV(ExprVal(IVal(ConstInt(6)))));
val i3 = AssignVar(a, ExprMutRef(MutRef(x)));
val i4 = AssignVar(a, ExprMutRef(MutRef(y)));
val i5 = AssignRef(a, ExprEV(ExprVal(IVal(ConstInt(7)))));

val p = Concat(Concat(Concat(Concat(MakeProgram(i1), i2), i3), i4), i5);

(* Esecuzione del programma *)
val storage = (make_storage 0 0 0); (* Inizializza lo storage a zero *)
val borrows = (make_storage 0 0 0) (* Nessuna variabile presa in borrow *)
val res = evalProgram(p, storage, borrows);

(* Verifica dei risultati *)
val res_storage = #1 res;
res_storage (evalVarLoc x);
res_storage (evalVarLoc y);

(* Esempio 5.2 - Esempio 5, ma riscritto in un'unica espressione *)

val res = evalProgram(Concat(Concat(Concat(Concat(MakeProgram(AssignVar(Var(1, TypeI),
    ExprEV(ExprVal(IVal(ConstInt(5))))), AssignVar(Var(2, TypeI), ExprEV(ExprVal(IVal(
    ConstInt(6))))), AssignVar(Var(3, TypeMR), ExprMutRef(MutRef(Var(1, TypeI))))),
    AssignVar(Var(3, TypeMR), ExprMutRef(MutRef(Var(2, TypeI))))), AssignRef(Var(3, TypeMR),
     ExprEV(ExprVal(IVal(ConstInt(7))))), (make_storage 0 0 0), (make_storage 0 0 0));
```