```
%%%% FILE: gv1.pro
%%%% TYPE: Prolog source
%%%% Line: very simple global variable ADT
%%%% Date: 9/23/17

%% Essential functionality

declare(Var, Val) :-
      retract(binding(Var,_)),
      assert(binding(Var, Val)).
declare(Var, Val) :-
      assert(binding(Var, Val)).

bind(Variable, Value) :-
      retract(binding(Variable, _)),
      assert(binding(Variable, Value)).

valueOf(Variable, Value) :-
      binding(Variable, Value).

undeclare(Var) :-
      retract(binding(Var, _)).

%%binding display functionality

bindings :-
      binding(Variable, Value),
      write(Variable), write(' -> '), write(Value), nl, fail.
bindings.

%%Arithmetic operator functionality

inc(Variable) :-
      retract(binding(Variable, Value)),
      NewValue is Value + 1,
      assert(binding(Variable, NewValue)).

dec(Variable) :-
      retract(binding(Variable, Value)),
      NewValue is Value - 1,
      assert(binding(Variable, NewValue)).

add(Variable1, Variable2, Operation) :-
      retract(binding(Operation, _)),
      binding(Variable1, Value1),
      binding(Variable2, Value2),
      NewValue is Value1 + Value2,
      assert(binding(Operation, NewValue)).
add(Variable1, Variable2, Operation) :-
      binding(Variable1, Value1),
      binding(Variable2, Value2),
      NewValue is Value1 + Value2,
      declare(Operation, NewValue).
```

```
sub(Variable1, Variable2, Operation) :-
      retract(binding(Operation, _)),
      binding(Variable1, Value1),
      binding(Variable2, Value2),
      NewValue is Value1 - Value2,
      assert(binding(Operation, NewValue)).
sub(Variable1, Variable2, Operation) :-
      binding(Variable1, Value1),
      binding(Variable2, Value2),
      NewValue is Value1 - Value2,
      declare(Operation, NewValue).

mul(Variable1, Variable2, Operation) :-
      retract(binding(Operation, _)),
      binding(Variable1, Value1),
      binding(Variable2, Value2),
      NewValue is Value1 * Value2,
      assert(binding(Operation, NewValue)).
mul(Variable1, Variable2, Operation) :-
      binding(Variable1, Value1),
      binding(Variable2, Value2),
      NewValue is Value1 * Value2,
      declare(Operation, NewValue).

div(Variable1, Variable2, Operation) :-
      retract(binding(Operation, _)),
      binding(Variable1, Value1),
      binding(Variable2, Value2),
      NewValue is Value1 / Value2,
      assert(binding(Operation, NewValue)).
div(Variable1, Variable2, Operation) :-
      binding(Variable1, Value1),
      binding(Variable2, Value2),
      NewValue is Value1 / Value2,
      declare(Operation, NewValue).

pow(Variable1, Variable2, Operation) :-
      retract(binding(Operation, _)),
      binding(Variable1, Value1),
      binding(Variable2, Value2),
      NewValue is Value1 ** Value2,
      assert(binding(Operation, NewValue)).
pow(Variable1, Variable2, Operation) :-
      binding(Variable1, Value1),
      binding(Variable2, Value2),
      NewValue is Value1 ** Value2,
      declare(Operation, NewValue).
```