```
/* Heng Low Wee U096901R
Problem Set 2, Problem 1

Write a Prolog predicate definition that counts the number of occurrences of
an operator inside an expression. For instance, the query:
?- count(a+b*c-(2+3*4)/(5*(2+a)+(b+c)^f((d-e)*(x-y))), *, C).
would count the number of occurrences of operator * in the expression
given as the first argument. The return value should be bound to the
variable passed as the third argument. For the query given above, the
answer should be C=4.

*/

count(Expr, Op, C) :-
    (Expr =.. [Operator, LeftOperand, RightOperand]
        ->  (count(LeftOperand, Op, C1),
            count(RightOperand, Op, C2),
            (Operator = Op
                -> C is C1+C2+1
                ; C is C1+C2
                ),
            !)
        ;   (Expr =.. [Operator, LeftOperand],
            count(LeftOperand, Op,C1),
            (Operator = Op
                -> C is C1+1
                ; C is C1
                ),
            !)
    ).

count(Expr, _, 0) :- atomic(Expr) ; number(Expr).




/* Heng Low Wee U096901R
Problem Set 2, Problem 2

Consider the language of arithmetic expressions with
the operators +, -, *, and /, where the operands are either
numeric constants, or Prolog atoms denoting mathematical variables.
Thus, the Prolog term:
```

```
(x+2)*(a-x)

stands for the mathematical function f(x)=(x+2)(a-x),
where a would be symbolic constant. Write a Prolog program that
computes the derivative of such an expression. For instance, the
query:

?- derive((x+2)*(a-x),x,D).

should derive the expression given as the first argument with
respect to the variable given as the second argument. The result
should be placed in the variable given as the third argument.
For the query above, the answer should be

C = (1+0)*(a-x)+(x+2)*(0-1)

Do not perform any arithmetic simplification in your solution
to this problem. Simplification is the topic of the next problem.

*/

derive(Expr, Wrt, Result) :-
    Expr =.. [Operator, Left, Right],
    derive(Left, Wrt, R1),
    derive(Right, Wrt, R2),
    d(Operator, Left, Right, R1, R2, Result), !.

% Base cases
derive(Expr, _, 0) :- number(Expr), !.
derive(Expr*Wrt, Wrt, Expr) :- number(Expr), !.
derive(Wrt*Expr, Wrt, Expr) :- number(Expr), !.
derive(Expr, Wrt, 0) :- Expr \= Wrt, !.
derive(Expr, Wrt, 1) :- Expr = Wrt, !.

d(+, _, _, DL, DR, Result1) :- Result1 = DL + DR, !.
d(-, _, _, DL, DR, Result1) :- Result1 = DL - DR, !.
d(*, L, R, DL, DR, Result1) :- Result1 = DL * R + L * DR, !.
d(/, L, R, DL, DR, Result1) :- Result1 = (DL * R - L * DR)/(R*R), !.




/* Heng Low Wee, U096901R
Problem Set 2, Problem 3
```

```
Write a Prolog program that performs arithmetic expression simplification.
Your program should at least eliminate multiplications by 0 and 1, and
additions with 0. Ideally, it should also convert any subexpression
containing only constants into the value of that expression, and convert
multiplications with -1 into the negative of the multiplicand.
For instance, the expression

(1+0)*(a-x)+(x+2)*(0-1)

could be simplified into:

a-x-(x+2)

Further simplifications may include distribution of high-precedence
operators, and grouping of terms that contain the same variable.
The expression above could be further simplified into:

a-2*x-2


READ HERE:
Sample Run...

simplify((1+0)*(a-x)+(x+2)*(0-1), C).

*/

simplify(X, Result) :- number(X), Result is X, !.
simplify(X, Result) :- atom(X), Result = X.

simplify(Expr, Result) :-
    (Expr =.. [Operator, Left, Right]
        ->  (simplify(Left, R1),
            simplify(Right, R2),
            d1(Operator, R1, R2, Result))
        ;   (Expr =.. [Operator, Right],
            simplify(Right, R3),
            d2(Operator, R3, Result))
    ),
    !.

% Additions with numbers
d1(+, DL, DR, Result1) :- number(DL), number(DR), Result1 is DL+DR, !.
% Subtraction with numbers
```

```
d1(-, DL, DR, Result1) :- number(DL), number(DR), Result1 is DL-DR, !.
% Multiplication with numbers
d1(*, DL, DR, Result1) :- number(DL), number(DR), Result1 is DL*DR, !.
% Division with numbers
d1(/, DL, DR, Result1) :- number(DL), number(DR), Result1 is DL/DR, !.

% Multiplication/Division by 0
d1(*, 0, _, Result1) :- Result1 is 0, !.
d1(*, _, 0, Result1) :- Result1 is 0, !.
d1(/, 0, _, Result1) :- Result1 is 0, !.
d1(/, _, 0, Result1) :- Result1 = infinity, !.

% Multiplication/Division by 1
d1(*, 1, DR, Result1) :- Result1 = DR, !.
d1(*, DL, 1, Result1) :- Result1 = DL, !.
d1(/, 1, DR, Result1) :- (number(DR) -> Result1 is 1/DR ; Result1 = 1/DR), !.
d1(/, DL, 1, Result1) :- Result1 = DL, !.

% Multiplication/Division by -1
d1(*, -1, DR, Result1) :- Result1 = -DR, !.
d1(*, DL, -1, Result1) :- Result1 = -DL, !.
d1(/, -1, DR, Result1) :- (number(DR) -> Result1 is -1/DR; Result1 = -1/DR),
!.
d1(/, DL, -1, Result1) :- Result1 = -DL, !.

% Additions with 0
d1(+, 0, DR, Result1) :- Result1 = DR, !.
d1(+, DL, 0, Result1) :- Result1 = DL, !.
% Subtraction with 0
d1(-, 0, DR, Result1) :- Result1 = -DR, !.
d1(-, DL, 0, Result1) :- Result1 = DL, !.

% Additions with variables
d1(+, DL, DR, Result1) :- DR =.. [-, Tail], Result1 = DL-Tail, !.
d1(+, DL, DR, Result1) :- DR =.. [_, _, _], Result1 = DL+DR, !.
d1(+, DL, DR, Result1) :- DL=DR, Result1 = 2*DL, !.
d1(+, DL, DR, Result1) :- Result1 = DL+DR, !.

% Subtraction with variables
d1(-, DL, DR, Result1) :- DR =.. [-, Tail], Result1 = DL+Tail, !.
d1(-, DL, DR, Result1) :- DR =.. [_, _, _], Result1 = DL-DR, !.
d1(-, DL, DR, Result1) :- DL=DR, Result1 is 0, !.
d1(-, DL, DR, Result1) :- Result1 = DL-DR, !.

% Multiplication with variables
```

```
d1(*, DL, DR, Result1) :- ((number(DL),atom(DR)) -> Result1 = DL*DR ; Result1
= DR*DL), !.

% Division with variables
d1(/, DL, DR, Result2) :- atom(DL), number(DR), Result2 = Result1 + DL,
Result1 is 1/DR, !.
d1(/, DL, DR, Result1) :- (DL=DR -> Result1 is 1 ; Result1 = DL/DR), !.

d2(-, Term, Result1) :- number(Term), Result1 is -Term, !.
d2(-, Term, Result1) :- Result1 = -Term, !.
```