Logic programming assignment II (PROP fall 2012)

Your task is to implement in Prolog an interpreter for a simple, imperative programming language without type declarations, which we may call C-- and that should allow the following type of statements:

Assignment and simple arithmetic expressions
An assignment in C-- is written on the form:

<Variable> `:=` <Expression>

where <Variable> is a variable denoted by lower-case letters, and <Expression> is a simple arithmetic expression, consisting of either a single numeric value (we make no distinction between integers and reals), a single variable or a sequence <value or variable> <operator> <value or variable> …, where <operator> is either `plus` or `minus`. Expressions are evaluated from left to right. Variables need not be declared, and they are assumed to contain numeric values only. A variable that has not been assigned a value is assumed to be zero (0).

Some examples:
`a := 5` (the variable ‘a’ is assigned the value 5).

`a := b + 2` (the variable ‘a’ is assigned the value of the variable ‘b’ + 2, if ‘b’ has not been assigned a value prior to this, ‘a’ will be assigned the value 2)

`a := 3 - 2 + 1` (the expression is evaluated from left to right and the result, i.e., 2 in this case, is assigned to ‘a’)

Input/output
`write a` (the value of the variable ‘a’ is written to standard output)
`write a + 2` (the expression is evaluated and the result is written to standard output)
`read b` (the user is requested to input a value from standard input, e.g., by displaying a prompt such as ‘b:’, and the value is assigned to the variable ‘b’)

Control structure
Several statements can be grouped into a single statement using ‘begin’ and ‘end’:

```
begin
      read a
      b := a + 1
      write b
end
```

The included statements are executed in sequence.

The same statement can be executed repeatedly by the following loop construct:

`while` <Condition> <Statement>

where <Condition> consists of a variable, an operator, and an arithmetic expression. The allowed operators are: `<` (less than), `<=` (less than or equal to), `>` (greater than), `>=` (greater than or equal to) and `=` (equal). The expression should follow the above syntax. The statement is executed as long as the condition evaluates to true.

For example, if variable ‘a’ is set to 8, the statement in the following while construct will be executed twice:

```
while a < 10
begin
      a := a + 1
      write a
end
```

A program written in C-- has to be on the following form:

```
begin
      <Statement 1>
      <Statement 2>
      …
      <Statement n>
end
```

where each statement follows the above syntax.

The main Prolog predicate for executing a program written in C-- should be called run/1, where the argument contains a list of C-- statements. For example, the query

```
?- run([
    begin,
    a,:=,10,
    while,a,>,5,
    begin,
      write,a,
      a,:=,a,-,1,
    end,
    end
    ]).
```

should give the following output:

```
10
9
8
7
6
yes
```

Hint: you may structure the program in the following way:

```
run(Program):-
        parse(ParseTree,Program,[]),
        execute(ParseTree,[],_Variables).
```

The predicate parse/3 syntactically analyzes the code (Program) in the form of a list and produces a parse tree (ParseTree), preferably using a definite clause grammar (see lecture slides), while execute/3 executes the code and keeps track of variable assignments (the second argument is a list with variable assignments prior to the execution, and the third argument is a list with variable assignments after the execution).

If you are aiming for a higher grade, you may extend the above, for example by reading the program from file instead of from a list, by introducing additional control constructs, e.g., if-then-else and for-loops, more data types, e.g., text strings, type declarations (if someone likes that), error messages (during parsing and execution), more advanced arithmetics, e.g., allowing parentheses, procedural declarations, etc.

Good luck!