Assignment 3 - Syntax Based Semantics

Total Points = 50

Instructions:

1. You must use SWI Prolog to implement this problem.
2. You must submit this assignment on Canvas by clicking the title link for the assignment. Make sure your programs compile (without any compiler errors). You will not receive credit if your program for a problem does not compile. If you are unable to complete any of the programs, submit the parts that work (with no compiler errors) for partial credit.
3. Your grade will be based on meeting the requirements, functionality (does the program do what it is supposed to do), readability (is the code nicely indented and formatted), and understandability (are the literals meaningful and is the code well documented with appropriate comments). You must incorporate all the good programming practices and styles.

Deliverables:

1. You should **submit one .pl file** named **assignSemantics.pl.** This file should contain your code for the problem in the assignment for the grader to run. This file will only contain the code and not the sample runs.
2. You should also submit **one PDF file** named **assignSemantics.pdf**. This PDF file should contain code to the solution of the problems (in **text format only, no snapshot** will be accepted) and your program's sample runs with output (in **text format only, no snapshot** will be accepted). Make sure to paste your code correctly (with **correct indentation**). If you have any text for the grader to read before grading, you can include it at the top of this file as comments or in canvas notes.

Q1. Program the semantics of the language whose definition is given below (the language is like the language given for parsing using DCG, except for two productions added for Expressions).

P in Program
K in Block
D in Declaration
C in Command
E in Arithmetic Expression
B in Boolean Expression
I in Identifier
N in Number

```
P ::= K.
K ::= begin D; C end
D ::= D ; D | const I = N | var I
C ::= C ; C | I := E | if B then C else C endif| while B do C endwhile | K
B ::= true | false | E = E | not B
E ::= E + E | E - E | E * E | E / E | (E) | I:= E | I | N
I ::= x | y | z | u | v
N ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Assume that all programs written in this language take 2 inputs that are found in variables x and y before the program begins execution and produce one output found in variable z after the program execution is over. Notice that both arithmetic and boolean expressions have side-effects due to the rule E ::= I := E.

To evaluate a program in this language, write a predicate program_eval(P, X, Y, Z), where P is the parse tree of the program, X & Y are the inputs and Z is the output after the evaluation. Use program/3 (from DCG parsing) to create your parse tree and program_eval/4 to run it.

For example, your program should accept the following query:

```
?- program(P, [begin, var, z, ; , var, x, ;, z, :=, x, end, .], []), write(P),
program_eval(P, 2, 3, Z).
```

and return Z = 2.

Test your program with the following 8 cases. You may use the initial values for x and y to be x=2, y=3, thus to output z.

1. begin var z; var x; z:=x end.
2. begin var x; var y; var z; z:=x+y end.

3. begin var x; var y; var z; z:=(z:=x+2)+y end.
4. begin var x; var y; var z; if x=y then z:=1 else z:=0 endif end.
5. begin var x; var y; var z; if x = 0 then z:=x else z:=y endif end.
6. begin var x; var y; var z; if not x=y then z:=x else z:=y endif end.
7. begin var x; var z; z:=0; while not x=0 do z := z+1; x:=x-1 endwhile end.
8. begin var x; var y; var z; z:=1; u:=x; while not u = 0 do z :=z*y; u:=u-1 endwhile end.