Assignment 2 - Parsing using Definite Clause Grammar (DCG)

Total Points = 50

Instructions:

1. You must use SWI Prolog to implement these problems
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 **assignDCG.pl**. This file should contain your code for the problems in the assignment separated by comments 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 **assignDCG.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 blackboard notes.

Q1. Grammar (20 points)

Eliminate **left recursion** and incorporate **precedence** rules for expressions in the following grammar:

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 | I | N
I ::= x | y | z | u | v
N ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Q2. Parse Tree (30 points)

Program the resulting grammar as a Definite Clause Grammar (DCG). Your grammar should produce parse trees for the input programs. Assume the top-level predicate to be program/3

i.e., program(ParseTree, InputListOfTokens, [])

An example input program:

```
begin
  const x = 8;
  var y;
  var z;
  z := 0;
  if x = y + 2 then z := 5 else z := 3 endif;
  while not x = z do
   z := z + 2
  endwhile
end.
```

The above program will be fed as a list of tokens to program/3. This list of tokens is shown below.

[begin, const, x, =, 8, ;, var, y, ;, var, z, ;, z, :=, 0, ;, if, x, =, y, +, 2, then, z , := , 5, else, z, :=, 3, endif, ;, while, not, x, =, z, do, z, :=, z, +, 2, endwhile, end, .]

So, your query will be:

? - L = [begin, const, x, =, 8, ;, var, y, ;, var, z, ;, z, :=, 0, ;, if, x, =, y, +, 2, then, z , := , 5, else, z, :=, 3, endif, ;, while, not, x, =, z, do, z, :=, z, +, 2, endwhile, end, .],
program(P, L, []).

After successful execution of this query, **P** will be bound to the parse tree for this program.