For this homework assignment, submit your answers on paper at the start of the lecture on the due date.

Late homework assignments will not be accepted, unless you have a valid written excuse (medical, etc.). You must do this assignment alone. No team work or "talking with your friends" will be accepted. No copying from the Internet. Cheating means zero.

==

Consider the following context-free grammar for a small programming language:

```
    B -> { L }

    L -> D ; L
    L -> S ; L
    L -> eps

    D -> INT ID = E

    S -> IF ( E ) B ELSE B
    S -> ID = E

    E -> T E'

    E' -> + T E'
    E' -> eps

    T -> F T'

    T' -> * F T'
    T' -> eps

    F -> ( E )
    F -> ID
    F -> NUM
```

where {, }, ;, INT, ID, =, IF, (,), ELSE, +, *, and NUM are tokens, and eps represents the empty string epsilon. B is the start symbol of the grammar and represents a block of code, between curly braces. A block of code is then a list L of definitions and statements, in any order. A definition D is simply the definition of an integer variable which is initialized using an arithmetic expression E. A statement S is either an IF statement (with one block of code for the "then" part and one block of code for the "else" part) or the assignment of an arithmetic expression E to a variable. Arithmetic expressions follow the same grammar that we have used in class (except for the extra production F -> NUM, where NUM is a token representing a natural number).

1) For each nonterminal in the grammar above we define an attribute called 'time' which represents the maximum amount of time necessary for a computer to execute the corresponding code, expressed as a number of clock cycles. To simplify things, tokens and epsilon do not have such a 'time' attribute.

For each grammar production in the grammar above, write an attribute computation rule that computes the value of the 'time' attribute for the corresponding nonterminal. Your rules should be based on the following assumptions:
- reading a number takes one clock cycle;
- reading a variable (using the variable's value in an expression) takes one clock cycle;
- writing a variable (using an assignment to the variable's name) takes one clock cycle;
- allocating memory for a new integer variable takes one clock cycle;
- doing an addition takes one clock cycle;
- doing a multiplication takes two clock cycles;
- testing whether the value of an expression is true or false and making a decision based on the result of the test takes a total of one clock cycle.

Also give a clear explanation in English of each rule, explaining why you wrote the rule you way you did.

2) Is the ’time’ attribute synthesized or inherited? Explain why.

3) Write the parse tree for the following program:

```
{int x = y * 3;}
```

then compute the value of the ’time’ attribute for all the nonterminals in the parse tree.

What is then the maximum running time of that program?

4) How would you modify the attribute computation rules of Question 1 to compute the minimum running time of a program?

5) The attribute computation rules you wrote in Question 1 always compute a finite maximum running time for all programs written in our small programming language. Would the same be true if you tried to write the same kind of attribute computation rules for a general-purpose programming language like C? Why or why not? What does this mean for the computational power of our small programming language compared to the computational power of a general-purpose programming language like C?

Do your answers change if you compare our small programming language with the database query language SQL (limiting yourself to the data manipulation part of the original version of SQL, i.e. statements like INSERT, UPDATE, DELETE, and SELECT, and therefore ignoring later extensions to the language that allow the programmer to create his own functions)? What does this tell you about the computational power of SQL?