Programming languages (TC-2006)

Project 01

In this project, you will use the strategies seen in class to design and implement both the scanner and parser for a simple toy language. This project may be solved in teams of at most three members. If you want, you can submit this project individually.

1 The programming language

The toy programming language to develop is a variation (and oversimplification) of assembly. Our version of the language will handle a drastically reduced number of lexical elements, which are described as follows:

INTEGER A sequence of digits.

REGISTER The symbol `#` followed by only one of the following letters: `A`, `B`, `C`, and `D`.

OPERATION One of the following reserved words: `SUM`, `SUB`, `MUL`, and `DIV`.

ASSIGNMENT The reserved word `MOV`.

EOF The symbol `;`.

Please note that any other symbol not defined before must be considered illegal by the system and produce an error. For simplicity, assume that all the lexical elements in the input string are separated by blank spaces (do not forget to consider tabs and the "end of line" character as spaces). Blank spaces must be ignored by your system.

Regarding the syntactic component of the language, the programming language has the following rules:

1. An OPERATION (`SUM`, `SUB`, `MUL`, and `DIV`) must be followed by two registers. For example: `SUM #A #B`. It is not allowed for an operation to involve an INTEGER directly (it must be assigned to a register to be used).
2. An ASSIGNMENT (`MOV`) can take an INTEGER or a REGISTER as first operand. However, the second operand must be a REGISTER. For example: `MOVE 12 #A` or `MOV #B #D`.
3. Only at the end of the file, the symbol `;` must be present. This is to indicate the end of the file.

For example, the following would be a valid program for this language:

```
MOV 5 #A
MOV 10 #B
SUM #A #B
MOV #B #C
MUL #C #A
MOV #C #D
MOV 2 #A
SUB #D #A
;
```

Please consider that your system will read the input from a text file.

2 Lexical analysis (50%)

Generate the DFA and transition matrix for the lexical components defined before. With this information, implemement a lexical analyzer for the programming language. Remember that your system must generate the list of tokens when the lexical analysis is over.

3 Syntactic analysis (50%)

Design a grammar that properly describes the syntactic rules of the described programming language. With that grammar, implement the syntactic analyzer by using the recursive descent technique seen in class. Remember that the input for the syntactic analyzer is the list of tokens generated by the lexical analyzer.

Deliverables

Implement a program that integrates the scanner and parser for the language described [a]. Prepare a ZIP file that contains your codes and a brief PDF document (at most two pages) with the DFA and grammar designed, and submit it to Canvas. If you worked as part of a team, only one of the teammates should submit the solution. **Please, do not submit other formats but ZIP**.

[a] Your program must be implemented in one of the following programming languages: C++, C#, Java or Python. If you want to implement the scanner in a different programming language, please discuss it with the instructor first.

I promise to apply my knowledge, strive for its development, and not use unauthorized or illegal means to complete this activity, following the Tecnológico de Monterrey Student Code of Honor.