DEPARTMENT OF COMPUTER SCIENCE
CARLOS III UNIVERSITY OF MADRID

Computer Science
Language Processors

Rules

- The duration of the test is **2.0 hours**
- Questions will not be answered during the test
- One cannot re-enter the classroom after leaving it
- The answers must be written using a pen (not a pencil)

Design a recursive descent parser to evaluate sentences of the language represented by the following grammar:

```
E ::= T | T + E | T - E
T ::= identifier
T ::= number
T ::= ( E )
```

1. **Transform the grammar appropriately.**

2. **Calculate the First and Follow sets of the modified grammar.**

3. **Write in C pseudo-code the functions that you think that are necessary to analyze each symbol of the grammar. Consider as already defined the functions of the Lexical Analyzer, and the functions ParseNumber, ParseAdd, ParseSubstr, ParseLParen and ParseRParen to parse the token number, and the literals +, -, (, and).**

Solution:

The grammar can be formulated as follows:

```
E ::= TS
S ::= λ | + E | - E
T ::= identifier | number | ( E )
```

The First and FOLLOW sets are:

	FIRST	FOLLOW
E	num, id, (	$,)
S	λ, + -	$,)
T	num, id, (	$,), +, -

David Griol Barres, Antonio Berlanga de Jesús, Jesús García Herrero, Juan Manuel Alonso Weber

A Parse function is built for each Non-Terminal. For each of the right parts of the production rules, the sequence of symbols will result in a sequence of calls to its corresponding Parse functions. It is necessary to manage in a specific way the nullable Terminals (S), and those that generate several right parts (T).

E ::= TS

There are not dilemmas.

S is nullable:

⇨ Analize Follow(S) = { $,) }

```
ParseE () {
      ParseT () ;
      if (token ∉ { $, ) })
            ParseS () ;
}
```

T ::= id | num | (E)

There are dilemmas

⇨ First(T) = { num, id, (}

```
ParseT () {
      if (token == T_IDENTIFIER)
            ParseId () ;
      else if (token == T_NUMBER)
            ParseNum () ;
      else if (token == '(') {
            ParseLParen () ;
            ParseE () ;
            ParseRParen () ;
      } else error () ;
}
```

S ::= λ | + E | - E

There are dilemmas:

⇨ First(S) = { λ, + - }

λ already managed

```
ParseS () {
      if (token == '+') {
            ParseAdd () ;
            ParseE () ;
      } else if (token == '-') {
            ParseSub () ;
            ParseE () ;
      } else error () ;
}
```

```
ParseId () {
      MatchToken (T_IDENTIFIER) ;
      token == rd_token () ;
}
```