Relatório de APS

Manuela Castilla
17/06/2021

Introdução:

Para aprofundar mais o conhecimento sobre as linguagens de computação (vocabulário, gramática, e etc), como APS cada aluno teria que criar uma linguagem nova e fazer todos os processo: código-fonte, análise léxico, análise sintático, análise semântica e por fim geração do código. Para esse trabalho, em vez de escolher um código-fonte já existente, cada aluno teria que inventar sua própria linguagem. Assim, para esse trabalho foi escolhido implementar a linguagem C junto com o espanhol.

Implementação:

Antes de qualquer coisa primeiro é preciso decidir a EBNF:

```
BEGIN = "{" COMMAND "}";
BLOCK = COMMAND | BLOCK COMMAND;
COMMAND = ( λ | ASSIGNMENT | PRINT |  LOOP-STATEMENT | IF-STATEMENT), ";"
;
ASSIGNMENT = IDENTIFIER, "=", OREXPRESSION ;
PRINT = "imprimir", "(", OREXPRESSION, ")", ";" ;
LOOP-STATEMENT = ( WHILE-STATEMENT | FOR-STATEMENT) ;
WHILE-STATEMENT =  "todavia", "(", CONDITION_MORE, ")", "{", BEGIN ,"}" ;
FOR-STATEMENT = "por", "(" TYPE, ASSIGNMENT, ";", IDENTIFIER, "<",
"IDENTIFIER, ";", IDENTIFIER, "++" ")", "{", BLOCK ,"}" ;
IF-STATEMENT = "se", "(", OREXPRESSION, ")", BEGIN ;
IF-STATEMENT = "se", "(", OREXPRESSION, ")", BEGIN, "seno" BEGIN;
EXPRESSION = TERM, { ("+" | "-"), TERM } ;
CONDITION_MORE = (CONDITION | CONDITION_MORE);
CONDITION = (EXPRESSION, LOGICAL, EXPRESSION);
OREXPRESSION = ANDEXPRESSION, { "o", ANDEXPRESSION};
ANDEXPRESSION = EQUEEXPRESSION, { "y", EQEXPRESSION};
EQUEEXPRESSION = RELEXPRESSION, { "lo_mismo", RELEXPRESION};
RELEXPRESSION = EXPRESSION, { ("mas_grande" | "menor"), EXPRESSION}
ELSE-STATEMENT = "seno", "{", (BLOCK| IF-STATEMENT), "}"
TERM = FACTOR, { ("*" | "/"), FACTOR };
FACTOR = ("+" | "-" | "queno"), FACTOR | "(", EXPRESSION, ")" | NUMBER;
IDENTIFIER = LETTER, { LETTER | DIGIT | "_" } ;
NUMBER = DIGIT, { DIGIT } ;
LETTER = ( a | ... | z | A | ... | Z ) ;
DIGIT = ( 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 ) ;
```

Podemos ver acima que foi apenas decidido utilizar números inteiros, porém podia adicionar este feature caso fosse necessário.
Dado isso é possível ter o exemplo da linguagem criada:

```
{
    x = 1;
    todavia(x mas_grande 5){
        x = x + 1;
        imprimir(x);
    }
    se (x lo_mismo 5){
        imprimir(10);
    }
}
```

Etapas:

I. Pré-Processamento

Para o Pré-Processamento foi tirado todos os comentários. Assim, antes de tokenizar qualquer coisa. Foi usado uma função do *rply* na qual pega o regex inserido e ignora o que foi identificado.

```
# comentarios
self.lexer.ignore(r"//.*?//")
```

II. Léxico

Após limpar o código o próximo passo é tokenizar os símbolos do código fonte. Para isso foi utilizado a biblioteca *LexerGenerator* do *rply.* Com ele é possível adicionar o símbolo (detectado através do *regex)* e qual é o seu significado.

Por exemplo dado o código fonte:

```
{
    x = 1;
    imprimir(x);
}
```

Para anotar os símbolos é feito através da seguinte maneira:

```
self.lexer.add('OPEN_PAREN', r'\(')
```

Com isso, obtemos o seguinte tokenizador:

```
Token('OPEN_BRACES', '{')
Token('IDENTIFIER', 'x')
Token('EQUAL', '=')
Token('NUMBER', '1')
Token('SEMI_COLON', ';')
Token('PRINT', 'imprimir')
Token('OPEN_PAREN', '(')
Token('IDENTIFIER', 'x')
Token('CLOSE_PAREN', ')')
Token('SEMI_COLON', ';')
Token('CLOSE_BRACES', '}')
```

III. Sintática/ Parser

Com os símbolos tokenizados podemos junto com a EBNF é possível criar a regra de produção. O programa apenas acontece caso a cadeia do código-fonte faça sentido. Dado o exemplo de printar acima, temos então o seguinte parser:

```
@self.pg.production('println : PRINT OPEN_PAREN parseOREXPR CLOSE_PAREN SEMI_COLON')
# @self.pg.production('println : PRINT OPEN_PAREN variable expression CLOSE_PAREN SEMI_COLON')
def println(p):
    return Print(p[2])
```

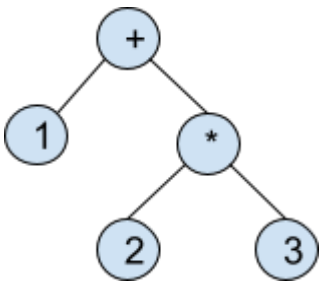
Assim, ao programa receber o código fonte, irá tokenizar e ser inserido na lista *p*. Cada elemento da lista é um token na ordem que vem (primeiro token é o token “imprimir” o segundo seria o abre parênteses e assim segue). Caso os tokens não baterem com a sequência, o programa não irá compilar.

falar de como um chama o outro que chama o outro

IV. AST (Árvore Sintática Abstrata)

Para organizar em que ordem vai ser feito o código, ou seja, durante seu processamento, os objetos que são da classe Node, estes são enviados para a *Árvore Sintática Abstrata*. Por exemplo, caso a função fosse:

```
funcion main(){
        imprimir(1 + 2 * 3);
}
```



V. Evaluate

Para destrinchar essa árvore existe o método Evaluate() e ao chamar ela, todos os nós vão ser resolvidos para então obter o resultado.

Assim, utilizando o exemplo acima, ao fazer Evaluate nela, o resultado final será 7.

VI. Conclusão

Com isso, temos todas as etapas para criar um compilador: dado um código fonte ocorre o pré-processamento, análise léxica, sintática e semântica. Próximo passo seria a geração e otimização do código. Com isso então, temos nossa linguagem criada.