Faculdade de Ciências e Tecnologia
Da Universidade de Coimbra
Departamento de Engenharia Informática
Licenciatura em Engenharia Informática
Compiladores

Relatório
Projeto de Compiladores 2018/19
Compilador para a linguagem deiGo

João Pedro Santos Rodrigues – 2016225773
Cláudio André Ventura Alves – 2016225581

Introdução:

O Projeto consiste no desenvolvimento de um compilador para a linguagem deiGo. No desenvolvimento de um compilador são necessárias quatro etapas fundamentais: análise lexical, análise sintática, análise semântica e geração de código.

Análise Sintática

Na análise sintática é suposto verificar se os tokens estão organizados da maneira correta consoante a gramática da linguagem. É necessário criar uma gramática e verificar se os tokens seguem a ordem de acordo com a mesma. Seguidamente, após verificar que não existem erros constrói-se uma estrutura de dados que neste caso é Árvore de Sintaxe Abstrata.

(i) Gramática re-escrita

Precedência (Mais relevante de baixo para cima, tal como o parser assume):
%left OR
%left AND
%left LT GT EQ NE LE GE
%left PLUS MINUS
%left STAR DIV MOD

Gramática:
S: Program

```
Program: PACKAGE ID SEMICOLON Declarations
         ;

Declarations: Declarations VarDeclaration SEMICOLON
            | Declarations FuncDeclaration SEMICOLON
            |
            ;

VarDeclaration: VAR VarSpec
              | VAR LPAR VarSpec SEMICOLON
              ;

VarSpec: ID MoreIDs Type
         ;

MoreIDs: MoreIDs COMMA ID
       |
       ;

Type: INT
    | FLOAT32
    | BOOL
    | STRING
    ;

OptType: Type
       |
       ;

FuncDeclaration: FUNC ID LPAR Parameters RPAR OptType FuncBody
               ;

Parameters: ID Type MoreParameters
          |
```

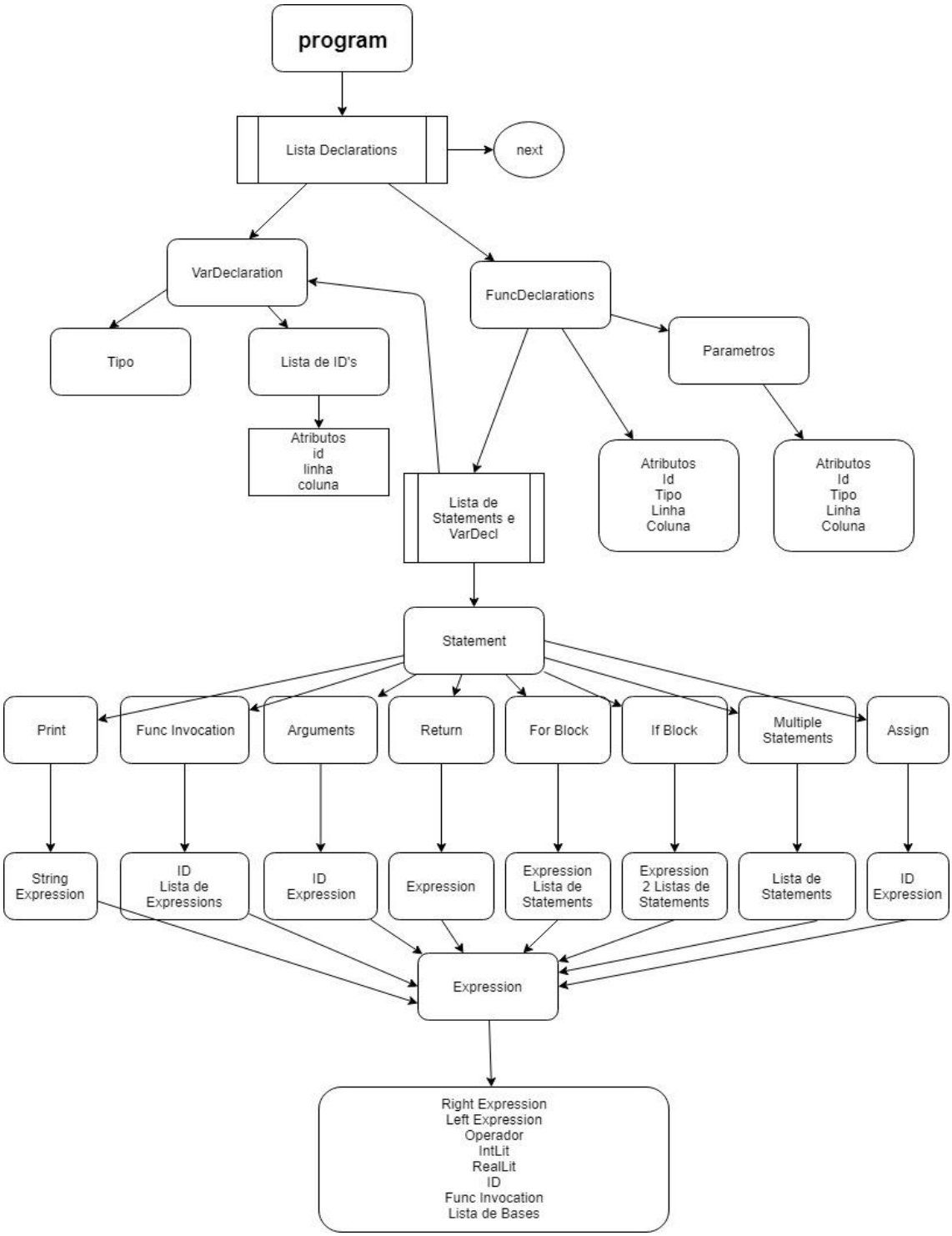
```
MoreParameters: MoreParameters COMMA ID Type
                |
                ;

FuncBody: LBRACE VarsAndStatements RBRACE
        ;

VarsAndStatements: VarsAndStatements OptVarDecStat SEMICOLON
                 |
                 ;

OptVarDecStat: Statement
             | VarDeclaration
             |
             ;

Statement: PRINT LPAR PrintExpression RPAR
         | FuncInvocation
         | ParseArgs
         | RETURN OptExpression
         | FOR OptExpression LBRACE MultipleStatements RBRACE
         | IF Expression LBRACE MultipleStatements RBRACE OptElse
         | LBRACE MultipleStatements RBRACE
         | ID ASSIGN Expression
         ;

OptElse: ELSE LBRACE MultipleStatements RBRACE
       |
       ;

MultipleStatements: MultipleStatements Statement SEMICOLON
                  |
                  ;                                                    ;

OptExpression: Expression
             |
             ;

PrintExpression: Expression
               | error
               | STRLIT
               ;

ParseArgs: ID COMMA BLANKID ASSIGN PARSEINT LPAR CMDARGS LSQ
Expression RSQ RPAR
         | ID COMMA BLANKID ASSIGN PARSEINT LPAR error RPAR
         ;




FuncInvocation: ID LPAR Fexpr RPAR
```

```
                    | ID LPAR error RPAR
                    ;

Fexpr: Expression Optexpr
       |
       ;

Optexpr: COMMA Expression Optexpr
         |
         ;
Expression: BaseExpression Texpression
            |  Expression OR Expression
            |  Expression AND Expression
            |  Expression LT Expression
            |  Expression GT Expression
            |  Expression EQ Expression
            |  Expression NE Expression
            |  Expression LE Expression
            |  Expression GE Expression
            |  Expression PLUS Expression
            |  Expression MINUS Expression
            |  Expression STAR Expression
            |  Expression DIV Expression
            |  Expression MOD Expression
            ;

BaseExpression:  BaseExpression PLUS
                 | BaseExpression MINUS
                 | BaseExpression NOT
                 |
                 ;


Texpression: INTLIT
             | REALLIT
             | ID
             | FuncInvocation
             |   LPAR Expression RPAR
             ;
```

As opções técnicas da reconstrução da gramática passam por tentar retirar a ambiguidade e a recursividade à esquerda, acrescentando termos e fatores.

Na análise lexical é necessário verificar se todas as palavras do código-fonte pertencem à linguagem, se pertencerem são lhes atribuídas tokens que facilitarão as etapas seguintes, se não pertencerem é necessário emitir erros lexicais.

(ii) estruturas de dados da AST e da tabela de símbolos



A nossa tabela de símbolos é representada por uma lista ligada de “var declarations” e “func declarations”, em cada “func declarations” estão os seus parâmetros e a lista de “var declarations”.

Na análise semântica, tendo já um AST correta, basta verificar a coerência dos tipos em todas as operações do código (ParseArgs, operadores, return…)

(iii) Geração de Código

Nesta meta da compilação foi necessário gerar llvm a partir da AST sintática e semanticamente correta. Para conseguir realizar esta meta, tivemos de estudar a geração de llvm a partir de programas em C e organizar os prints de código llvm com o percorrer da AST.