

Departamento de Engenharia Informática

Relatório do Projeto

**Compilador para a linguagem deiGo**

**Compiladores (COMP)**

Marco Pais Nº 2019218981

Tiago Oliveira Nº 2019219068

**Índice**

[**Reescrever a Gramática** 2](#_Toc90138116)

[**Algoritmos e Estruturas de dados** 3](#_Toc90138117)

# **Reescrever a Gramática**

Foi nos fornecida, inicialmente, uma gramática da linguagem em notação EBNF, sendo esta ambígua, necessitando assim de várias modificações para tornar possível implementar, corretamente, a linguagem na ferramenta YACC. As ambiguidades inicialmente existentes foram identificadas através da identificação de diversos conflitos manifestados na execução do programa, sendo estes conflitos “*shift-reduce*” e “*reduce-reduce*”.

A primeira fonte de ambiguidades a ser identificada e corrigida foi encontrada nas produções relacionadas com expressões (**Expr**), sendo necessário adicionar regras de determinação de associatividade e de precedência de operadores. As regras de associatividade podem ser obtidas no YACC através do uso dos comandos **%left**, **%right** e **%nonassoc**. Já para a precedência foi necessário organizar as declarações de associatividade, pois a precedência aumenta de baixo para cima.

Tendo, ainda, em conta a precedência de operadores da linguagem foi ainda necessário acrescentar uma regra relativa às *if-else clauses*, que não possuem regras de associatividade definidas. No YACC isso pode ser obtido através do comando **%nonassoc**.

A imagem seguinte é referente às regras de precedência implementadas.

Uma imagem com texto

Descrição gerada automaticamente

De seguida, foram efetuadas várias modificações na gramática relativas:

* À implementação de ciclos que produzem listas de declarações, variáveis e/ou expressões. Para isso foram implementadas produções recursivas.
* À implementação de regras que identificam e lidam com casos em que uma dada construção da gramática é opcional.
* À criação de uma nova produção (**IDaux**), devido à necessidade adicionar, algumas vezes, o *token* **ID** à Árvore de Sintaxe Abstrata (AST).

Por fim, foram utilizados comandos do YACC, **%destructor**, que permitem limpar a *stack* caso existam erros sintáticos na execução do programa.

# **Algoritmos e Estruturas de dados**

No desenvolvimento das metas 2 e 3 foram implementados, maioritariamente, diversos algoritmos que permitem a pesquisa de elementos nas diversas estruturas de dados e/ou permitem percorrer essas mesmas estruturas de dados.

As estruturas de dados implementadas foram árvores, listas ligadas e alguns vetores.

A primeira estrutura a ser implementada foi a Árvore de Sintaxe Abstrata (AST), sendo que essa estrutura tem nós que são criados assim que o YACC encontra uma sequência de carateres que corresponde a uma regra definida na gramática. A estrutura guarda o valor do *token* lido numa variável (char \* valor)