# CES-41: Compiladores

## Laboratório 4: Tabela de símbolos e de analisador semântico para uma linguagem de programação utilizando a ferramenta Yacc

Cássio dos Santos Sousa, Renan Pablo Rodrigues da Cruz

Professor: Fábio Carneiro Mokarzel

06 de outubro de 2015

# 1. Introdução

Já foi possível criar, em laboratórios anteriores, um analisador léxico (com uso da ferramenta *Flex*) e um analisador sintático (com uso da ferramenta *Yacc*) para a Linguagem COMP-ITA 2015.

Para este laboratório, como o título alerta, o objetivo é o de construir uma tabela de símbolos e um analisador semântico para a mesma linguagem com o uso das mesmas ferramentas, de tal forma que o programa resultante seja capaz de imprimir o conteúdo da tabela de símbolos e as devidas mensagens de erros para programas quaisquer utilizados como entrada.

Este laboratório e os seguintes são mais complexos que os anteriores, de tal forma que seu desenvolvimento pode ser feito agora por duplas de alunos, contanto que essa dupla seja mantida até a entrega do último laboratório da disciplina (de um total de seis).

# 2. Resultados

## 2.1. Atividades realizadas

A primeira das atividades foi a revisão do código escrito no laboratório anterior, o qual realizava a análise sintática e, caso não apresentasse erros, apresentava o código de entrada no formato *pretty printer*, removendo comentários e respeitando tabulações e espaçamentos de cada símbolo presente no código. A revisão deu foco a uma impressão menos espaçada das informações do código, algo mais próximo do que é visto na linguagem C.

A segunda delas foi a adequação do código à Prática 3 (quando foi proposto o Laboratório 4) e à Aula 6 (que fazia o tratamento teórico do mesmo tema). Não foi possível apenas copiar e colar o código que estava presente, pois eles eram válidos para linguagens diferentes da linguagem COMP-ITA 2015. O foco foi o de justamente atentar às necessidades de cada trecho de código nas apresentações e quais adaptações eram necessárias para encaixá-los no nosso código.

A terceira atividade foi a verificação da impressão correta da tabela de símbolos e da análise semântica para um código que, a princípio, não possuísse erros sintáticos na linguagem COMP-ITA 2015. O código em questão foi justamente aquele dado como exemplo nas especificações da linguagem. A saída do código implementado para este laboratório imprime, primeiramente, o código inserido em conjunto com erros sintáticos e semânticos, e se não houver mais impedimentos, a tabela de símbolos.

Foi interessante perceber que aquele mesmo código, apesar de não ter erros sintáticos (o que foi útil para o laboratório anterior), possuía erros semânticos. Em nome disso, foi construído um código que tivesse estes mesmos erros corrigidos, como exemplo de entrada correta (presente em **`Tests/1_Correct/lab04test1`**, e output em **`Tests/1_Correct/lab04results1`**).

A quarta atividade deu foco à criação de testes e revisão concomitante do código. Cada teste inserido tentou demonstrar um cenário de erro semântico de acordo com as especificações da

linguagem COMP-ITA 2015. Estes testes foram separados em pastas internas à pasta **Tests**, com nome e numeração adequados para facilitar a busca. O input está presente no formato **lab04test#**, e o output correspondente está presente em **lab04results#** na mesma pasta, semelhante ao que foi feito para os exemplos na atividade anterior.

# 2.2. Formato da Tabela de Símbolos

Manteve-se o formato da Aula 6 e da Prática 3 com respeito ao número limite da tabela hash (23 classes possíveis). Após a impressão do título da tabela, se uma das classes possuir símbolos, é impressa a numeração da classe seguida de cada um dos símbolos, um para cada linha.

O trecho a seguir foi tirado de uma das classes presentes em **Tests/1_Correct/lab04results1**, para ilustrar o formato da tabela de símbolos.

```
Classe 7:
  (c, IDVAR, CARACTERE, 0, 1, ##main)
  (ntab, IDVAR, INTEIRO, 1, 1, ##global)
  (palavra, IDVAR, CARACTERE, 1, 1, ##global, EH ARRAY
    ndims = 1, dimensoes: 10)
```

Para cada símbolo, são impressas as informações presentes no mesmo: sua identificação (presente na variável **cadeia**), seu tipo de identificador (**IDGLOB, IDVAR, IDPROG, IDPROC, IDFUNC** ou vazio – **IDGLOB** foi acrescentado por sua relevância no código), o tipo da variável (**NAOVAR, INTEIRO, LOGICO, REAL, CARACTERE**), flags que verificam se o símbolo foi, respectivamente, inicializado e referenciado (1 se foi), e o escopo do símbolo (o qual é esperado que seja vazio apenas para o símbolo **global**).

Se uma variável for indexada, é impresso **EH ARRAY** e, na linha seguinte, o número de dimensões e quais seriam elas numericamente.

# 2.3. Detalhes notados

Uma das coisas notadas durante a implementação de código foi que a função **`ProcuraSimb()`**, utilizada para verificar se um símbolo já está presente na tabela de símbolos, possuía duas utilizações: a primeira, antes de se instanciar uma variável (normalmente seguida de **`InsereSimb()`** caso o símbolo não esteja presente ainda). A segunda, apenas para verificação (apenas para ver se o símbolo verificado já foi inserido). Ela foi então separada em duas funções: **`ProcuraSimbParaInstanciar()`** e **`ProcuraSimbParaUsar()`**.

Uma outra coisa que se percebeu foi que é possível que uma função e um procedimento terminem suas tarefas sem retornar nada, dado que um Statement vazio pode ser utilizado num ReturnStat. Isso pode não trazer problemas para a main e para procedimentos, que não retornam valores, mas isso traz graves problemas para funções, pois a checagem pedida como teste semântico foi só aquela na qual uma função termina seus statements com return. Por conta do tempo e da não-especificação deste caso teste, ele foi desconsiderado.

Não só isso, passar um subprograma como parâmetro de outro subprograma era uma notação confusa, pois acontecia erro sintático quando se utilizava notações como f1(f2()) por conta dos parênteses, e o caso f1(f2) recaía na declaração de um argumento com mesmo nome de um módulo. Este caso de teste foi desconsiderado.

# 2.4. Casos de teste

Na pasta **`Tests`** dentro da pasta de códigos, existem os seguintes testes, acompanhados de seus respectivos resultados. Tentou-se utilizar o código exemplo o máximo possível nos testes:

- **`1_Correct:`** código presente na linguagem COMP-ITA com quaisquer erros semânticos corrigidos, tomado como correto e como referência para os demais testes;

- **`2_Undeclared_identifier:`** código para verificar erros associados a identificadores não declarados;

- **`3_Module_named_as_global_variable:`** código para verificar erros associados a módulos que possuam o mesmo nome que uma variável global;

- **`4_Indexed_variable_value_equals_zero:`** código para verificar erros associados a dimensões inteiras de variáveis indexadas com valor zero;

- **`5_Indexed_variable_value_lower_than_zero:`** código para verificar erros associados a dimensões inteiras de variáveis indexadas com valor menor que zero;

- **`6_Identifier_doubly_declared_in_module:`** código para verificar erros associados a identificadores declarados mais de uma vez em um mesmo módulo;

- **`7_Scalar_variable_as_array:`** código para verificar erros associados a variáveis escalares que apresentem subscritos, tentando se comportar como variáveis indexadas;

- **`8_Return_in_procedure_followed_by_expression:`** código para verificar erros associados ao retorno errôneo de uma expressão ao final de um procedimento;

- **`9_Return_in_function_followed_by_nothing:`** código para verificar erros associados à falta de uma expressão sendo retornada ao final de uma função;

- **`13_For_header_incorrect_type:`** código para verificar erros associados à utilização de uma variável não escalar do tipo inteiro ou caractere no cabeçalho de um **`for`**;

- **`14_For_expressions_with_incorrect_type:`** código para verificar erros associados à utilização de uma variável de tipo incorreto em uma das três expressões de um comando **`for`**;

- **`15_If_while_repeat_not_using_logic:`** código para verificar erros associados à utilização de uma variável que não seja relacional ou lógica no cabeçalho de um **`if`** ou **`while`**, e no encerramento de um **`repeat`**.

# 3. Conclusões

Foi possível avançar bastante na construção do compilador esperado para a linguagem COMP-ITA 2015. A análise semântica foi uma tarefa bem mais complicada que os laboratórios anteriores, e sua complexidade pôde ser notada no grande número de casos teste que foram considerados para este laboratório e a transição bem menos linear dos códigos e explicações feitas em aula para os códigos em Lex/Yacc do laboratório.

Espera-se que este laboratório tenha servido de preparativo para os demais, por meio não só de seu código, mas também de sua complexidade dentro da temática de compiladores.