**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.

A quinta e última atividade foi a revisão de comentários e disposição de átomos e outras informações nos códigos. Como esta parte é não-funcional, e o relatório já cobriria boa parte das explicações relevantes, esta atividade foi deixada por último.

**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. Decisões de design**

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()**.

**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;
* **10\_Code\_used\_for\_recursivity:** código para verificar erros associados à tentativa de se utilizar recursão na linguagem;
* **11\_Subprogram\_used\_as\_parameter:** código para verificar erros associados à tentativa de se utilizar um subprograma como parâmetro de outro subprograma;
* **12\_Subprogram\_used\_as\_argument:** código para verificar erros associados à tentativa de se utilizar um subprograma como argumento de outro subprograma;
* **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 o laboratório.

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