TRABALHO DE COMPILADORES: IMPLEMENTANDO ANALISADORES LÉXICO E SINTÁTICO, TABELA DE SÍMBOLOS E TRATAMENTO DE ERROS

Compiladores I

`Professor:` Mário Luiz Rodrigues Oliveira - `mario.luiz@ifmg.edu.br`
Vinícius Alves de Araújo - Matrícula: 0011941 - `viniaraujo568@gmail.com`
Vinícius Morais dos Santos - Matrícula: 0002864 - `viniciusms2907@gmail.com`

O presente documento tem como finalidade, enfatizar as definições e tratamentos(métodos) utilizados, além de apresentar as dificuldades encontradas ao longo da implementação do projeto.

Na realização do trabalho, utilizamos os códigos apresentados em aula como modelo para o desenvolvimento dos analisadores léxico e sintático. Optamos por utilizar a linguagem de programação Python e por organizar todo o funcionamento do programa em 6 arquivos, sendo 5 deles para classes e um para o arquivo principal(*main*). As classes criadas foram:

- ARCHIVE: classe encarregada de abrir e fechar o arquivo do código que se deseja realizar a compilação, além de gerar o arquivo de saída para a tabela de símbolos.
- TOKEN: responsável pela estrutura definida pelos integrantes para um token a ser identificado no arquivo de entrada.
- TOKENCLASS: mantém as informações sobre todos os tipos de tokens permitidos pelo compilador.
- LÉXICO: responsável por verificar cada caracter do arquivo de entrada e agrupar, transformando-os em lexema, para assim poder identificar como um dos tipos de tokens definidos na classe TokenClass.
- SINTÁTICO: encarregado de verificar a ordem de ocorrência dos lexemas encontrados pelo analisador léxico, para examinar se condiz com as regras de produção da gramática.

A classe desenvolvida para o analisador léxico conta com três procedimentos e um dicionário das palavras reservadas pela linguagem. O procedimento *getChar* retorna um caracter do arquivo passado como entrada, ou se houver algo no buffer, dando prioridade para consumir o que está presente no buffer. *ungetChar* foi criada para armazenar um caracter no buffer, ela é chamada quando realiza a leitura de um caracter no arquivo, porém o mesmo não será utilizado no momento, logo, para não perder a informação, salva no buffer. Por último, o procedimento *getToken*, que tem seu funcionamento pensado como estados de um autômato finito, onde cada estado é encarregado de identificar um conjunto de caracteres; o último caracter que for identificado, finalizando o lexema, especifica a que classe de token pertence.

O analisador sintático conta com dois procedimentos para iniciar a verificação se o programa de entrada foi escrito conforme as regras de produção da gramática da linguagem e um para consumir e verificar tokens, nomeados respectivamente de *parser* e *consume*. Os demais procedimentos foram nomeados e desenvolvidos conforme as regras de produção especificadas no documento do enunciado do trabalho.

Tanto o método pânico, para tratamento de erros, quanto a tabela de símbolos foram implementadas na classe do analisador sintático. As informações necessárias para a tabela foram identificadas no procedimento *consume*. No mesmo procedimento também encontra-se a forma como foi tratada quando houver a ocorrência de erros no código do programa de entrada, foram criadas dois dicionários, um para os *frist* e o outro para os *follow* de cada regra da gramática, estes obtidos através do GTK(Grammar Tool Kit), programa disponibilizado para verificar se a gramática do trabalho era LL(1). Por fim, foi escolhido utilizar apenas o dicionário de *follow*, com a abordagem de quando consumir um token, verificar se ele era o token esperado, caso não fosse, iria consumir tudo até encontrar um token que condiz com um dos presentes no dicionário de *follow's*. A partir desse token, assumiria que o programa estava correto e voltava a fazer a verificação normal. Por assumir que após um determinado ponto estaria correto, o erro pode ser propagado, ocasionando em outros erros ou identificando falsos erros, porém acertando as linhas onde se esperava uma falha. Essa etapa foi a que apresentou ser o maior impecílho no trabalho.

Referências

[1] AHO, A. V. et al. Compiladores. 2 ed. São Paulo: Pearson Addison-Wesley, 2008.