Analisador Léxico
O despertar de uma nova linguagem de programação

Allen H. M. Santos[1]

[1]Engenharia de Computação – Universidade Estadual de Feira de Santana (UEFS)
Caixa Postal 44.036-900 – Feira de Santana – BA – Brasil

[2]Departamento de Ciências Exatas (DEXA - UEFS)

allenhichard@hotmail.com

Resumo. *Este relatório técnico descreve o processo de desenvolvimento do primeiro problema do MI – Compiladores do período 2017-2. O problema propõe o desenvolvimento de um analisador léxico como parte inicial de um projeto complexo de compilador. A solução para os resultados adquiridos foi construída na Linguagem Java na versão 8 utilizando o sistema operacional Windows.*

1. Analisador Léxico

O analisador léxico é a parte inicial de uma análise feita pelo compilador. Sua principal característica é a leitura de caracteres de entrada produzindo uma sequência de tokens baseados nas expressões regulares que define a linguagem de programação.

Os tokens são classificados de acordo com a Tabela 1, e armazenados em uma tabela de símbolos para uma posterior análise, conhecida como análise sintática. No decorrer deste relatório será exemplificado como os tokens são gerados e como os mesmos são classificados.

Tabela 1. Estrutura Léxica da Linguagem

Palavras reservadas	class, final, if, else, scan, print, int, float, bool, true, false, string
Identificador	Letra {Letra | Dígito | _}
Número	[-] Espaço Dígito {Dígito}[.Dígito{Dígito}]
Dígito	[0..9]
Letra	[a..z] | [A..Z]
Operadores Aritméticos	+,-,*,/,%
Operadores relacionais	!=, =, <, <=, >, >=
Operadores Lógicos	!, &&, | |
Comentários	// - comentário de linha, /*comentário de bloco*/
Delimitadores	; , ( ) { } [ ] :
Cadeia de Caracteres	”{Letra | Dígito | Símbolo | \”}”
Símbolo	ASC II de 32 a 126 (exceto ACS II 34)
Espaço	{ASC II 9 | ASC II 10 | ASC II 13 | ASC II 32}

2. Funcionamento - Análise Linear

De forma geral, o analisador léxico desenvolvido foi baseado nas estruturas de análises já existentes, facilitando a analogia do funcionamento nesta fase, evitando fundir análises

posteriores. Para a implementação da estrutura gramatical foi utilizado o **Regex** (Regular Expression), pela simplicidade e agilidade em gerar autômatos para cada expressão regular utilizada, além de fornecer um completo sistema de validação de dados de entrada.

O sistema recebe uma lista de arquivos, onde cada arquivo possui diversos algoritmos compostos por uma sequência de caracteres. São esses caracteres que iremos analisar e de forma cuidadosa avaliar e classificar nos padrões da linguagem.

Todos os passos do desenvolvimento foram pensados para serem altamente adaptativos, ou seja, se quiséssemos mudar algo pontual dentro da linguagem, facilmente poderia ser alterado.

2.1. Leitura

O analisador, após receber a lista de arquivos, organiza por fila e manda para o sistema arquivo por arquivo. Para cada arquivo o sistema faz as requisições das linhas contidas nele e deixa em modo de espera os demais arquivos.

2.2. Bufferizando

Um sistema de buffers é utilizado para armazenar os dados temporariamente. O analisador possui dois buffers e podemos classificá-los como: buffer de linha e buffer de lexemas. O buffer de linha é utilizado para salvar as linhas dos arquivos e auxiliar o buffer de lexemas. O buffer de lexemas apenas guarda uma única palavra do buffer de linha por vez, como podemos ver na Tabela 2.

Tabela 2. Sistema de Buffer

Buffer de Linha	**int numero = 0;**
Buffer de Lexema	**int**
Buffer de Lexema	**numero**
Buffer de Lexema	**=**
Buffer de Lexema	**0**
Buffer de Lexema	;

2.3. Classificação

Um lexema é qualquer sequência de caracteres que tenha um significado dentro dos padrões da linguagem definida na Tabela 1. Alguns caracteres foram utilizados para quebrar os lexemas dentro do buffer de linha, sendo eles: espaços; quebra de linha; delimitadores; aspas duplas e qualquer um dos operadores.

Após o lexema ser definido, o sistema de classificação receberá seu buffer e irá verificar em qual estrutura léxica melhor se encaixa. Utilizando os dados da Tabela 2, os lexemas foram classificados conforme a Tabela 3, onde foram associados: o nome do lexema; seu tipo e em qual linha o mesmo aparece no arquivo texto. A Tabela 3 é conhecida como **Tabela de Símbolos**.

3. Limitações

De acordo com os testes efetuados e com os requisitos solicitados para o problema, todos os possíveis erros foram tratados e corrigidos ao longo do desenvolvimento. Agora se

Tabela 3. Tabela de Símbolos

Lexema	Token	Linha
int	**Palavra reservada**	**1**
numero	**Identificador**	**1**
=	**Operador relacional**	**1**
0	**Número**	**1**
;	**Delimitador**	**1**

analisarmos o compilador completo, o sistema léxico tratado neste relatório não consegue definir se um código está sintaticamente ou semanticamente correto, ficando para as próximas etapas sanar tais limitações.

4. Padrões de Entrada e Saída

O padrão de entrada é bem simples. Dentro do diretório raiz do sistema foi inserido uma pasta com o nome **entrada**, nesta pasta, o usuário pode colocar quantos arquivos.txt desejar. Cada arquivo pode conter códigos diversos sem restrição alguma, ou seja, o usuário pode inserir qualquer sequência de caractere. Para cada arquivo o analisador irá ler a sequência de caractere contida no mesmo e classificará dentro da estrutura da gramática ou erro gramatical.

Para cada arquivo de entrada, o analisador léxico classifica os lexemas e armazena-os na tabela de símbolos, após a finalização de cada arquivo, a tabela de símbolos é registrado em um arquivo de saída, ou seja, para cada arquivo de entrada existe um arquivo de saída com seus respectivos lexemas. O local dos arquivos de saída é o mesmo definido pelo padrão de entrada.

A principal característica do arquivo de saída é salvar uma mensagem caso o mesmo não possuir erro baseado em sua estrutura gramatical. O arquivo de saída está organizado em três colunas: "<Nome do Token, Nome do Lexema, Linha na qual pertence>".

5. Particularidades

Nesta sessão, vamos analisar algumas das principais decisões de projetos levadas pelo analisador léxico.

5.1. Dupla análise

Às vezes, o analisador léxico não pode simplesmente ler um dado no arquivo e classificá-lo, existem alguns casos que é necessário olhar um pouco a frente no código ou até mesmo voltar um pouco para determinar o lexema. Exemplo: quando encontramos o operador "<" precisamos olhar se o próximo valor é o "=", o que classificaria o mesmo como lexema "<=". No caso dos operadores foi levando em consideração que não pode haver espaços entre eles.

5.2. Número Negativo

Uma expressão com o operador aritmético "-" mais uma sequência de digito pode ser considerada como duas expressões, levando a classificá-las de maneira distintas. Pensando

em alguma análise posterior o sistema foi baseado em números negativos, diminuindo o número de tokens gerado pela linguagem. O número negativo é composto por seu Sinal "-" e um número real, podendo ter qualquer tipo dos espaços definido pela linguagem entre eles.

5.3. Comentário de Bloco

Os comentários contidos nos arquivos serão ignorados pela analisador. Existem dois tipos de comentário e para cada um deles um comportamento diferente. Para o comentário de linha // tudo que vir depois destes dois caracteres será desconsiderando e o sistema irá chamar a próxima linha. O comentário de bloco ignora tudo que estiver entre /* "ignora" */, neste caso o fechar pode está em qualquer lugar do arquivo, se o */ for encontrado tudo que vir depois, o analisador levará em consideração na análise.

Com base na decisão de projeto, quando o analisador identifica o /* mas não consegue encontrar o */ não é considerado um erro léxico, mas o sistema irá ignorar o restante do código.

6. Tratamento de Erros

Nem sempre tudo que está escrito no arquivo texto é considerado correto dentro da estrutura léxica da linguagem, levando ao buffer de lexema a não classificação, chamamos essa não classificação de erro léxico. Dentro deste analisador todo e qualquer erro é classificado em quatro grupos: string mal formada; identificador mal formado; número mal formado e erro de símbolo.

6.1. String mal formada

A string mal formada ocorre de duas maneiras diferentes. Na primeira, uma aspa dupla é aberta e no decorrer da linha na qual pertence não encontra o fechamento da mesma. Na segunda maneira a verificação é mais robusta, porque mesmo com o fechamento da aspa dupla acontecendo na mesma linha, o sistema irá capturar o que está entre as aspas e irá analisar nos padrões da linguagem, caso algum caractere dentro da string não pertencer a linguagem.

6.2. Identificador mal formada

Um identificador mal formado acontece quando uma sequência de caracteres não pode ser classificada conforme a Tabela 1. Quando a estrutura retornar falso, o sistema de erro recebe como parâmetro o primeiro caractere do buffer de lexema, se esse caractere for uma letra, o mesmo irá receber a classificação de identificador mal formado. O que gera um identificador mal formado é um lexema começar com uma letra e existir um ou mais símbolos não esperados dentro de buffer exceto o "_".

6.3. Número mal formada

Seguindo os mesmos princípios utilizados para o identificador mal formado, quando a estrutura da linguagem retornar falso e o primeiro caractere do buffer de lexema for um número, será classificado como número mal formado. O que gera um número mal formado é o mesmo ter caractere como letras ou símbolos não desejados em sua sequência exceto o "(. ponto)".

6.4. Erro de Símbolo

Levando em consideração todas as decisões de projeto tomadas até então, o erro de símbolo é considerado sempre que um caractere especial for encontrado fora de uma cadeia de caractere ou dentro de uma cadeia de caractere conter símbolos foda da estrutura gramatical.