UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIAS DE COMPUTAÇÃO

Trabalho Prático 1

Analisador léxico

Disciplina: SCC0217 Linguagens de programação e compiladores
Professor: Dr. Diego Raphael Amancio

Aluno:	**NºUSP:**
Bruno Daniel Sanches Silva	8066544
Carlos Humberto S. Baqueta	7987456
Cláudio César Domene Júnior	7987310

26/04/2015

1. Introdução

O papel do compilador é gerar um código com uma linguagem semanticamente equivalente a um código submetido em uma outra linguagem, e normalmente o compilador é utilizado para traduzir uma linguagem de fácil compreendimento para a linguagem de máquina específica de um processador e sistema operacional.

O processo de transformação do código-fonte no código de saída é divido em várias etapas, descritas na imagem ao lado. Nesse contexto, este primeiro trabalho tem como objetivo a implementação da primeira fase de um compilador: a análise léxica, que consiste em analisar o código de um programa-fonte que segue às regras de produção da LALG e produzir uma sequência de tokens (símbolos léxicos), além de identificar erros léxicos que possam estar contidos no código do programa-fonte de entrada.

2. Descrição do projeto

No que tange a primeira fase de um compilador, a análise léxica (*scanning*), vale ressaltar que a mesma objetiva: (1) ler o fluxo de caracteres de entrada do programa fonte; (2) agrupar as sequências significativas, chamadas lexemas. Para cada lexema, o analisador léxico produz como saída um token no formato **<nome-token, valor_atributo>**, que é passado para a fase subsequente, a análise sintática (*parsing*). A estrutura básica de um analisador léxico para uma dada linguagem pode ser conferida na figura 1, a seguir.

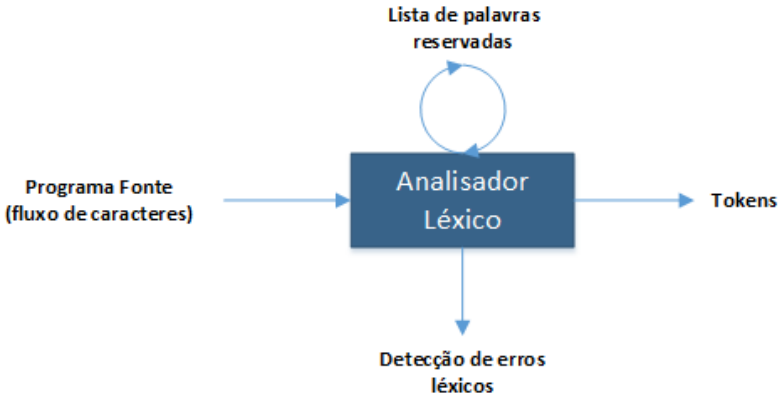

Figura 1 - Diagrama da estrutura básica do analisador léxico.

3. Implementação

Para implementar o projeto, utilizou-se a ferramenta Lex, que permite especificar um analisador léxico definindo expressões regulares para descrever padrões para os tokens, juntamente com código em linguagem C, para implementação do programa principal e estruturas auxiliares, como o dicionário de palavras reservadas.

4. Decisões de projeto

Durante a implementação do trabalho, algumas decisões de projeto foram tomadas:

- Foram tratados quatro tipos de erro:
  - Má formação de número inteiro: apenas erros de símbolos que começavam com dígitos, exemplo: 12asd, 1@3;
  - Má formação de ponto flutuante: erros de símbolos que inicializavam com dígitos e possuíam um ponto separador, exemplo: 1.@90, 34.3@34;
  - Tamanho máximo do identificador excedido (o tamanho máximo escolhido foi de 50 caracteres);
  - Qualquer outro erro de má formação da cadeia foi tratado como "Caracter desconhecido".

- Foi decidido que os operadores unários mais e menos (**'+'** e **'-'**), de sinal numérico, e os operadores aritméticos adição e subtração (**'+'** e **'-'**), serão tratados indistintamente na fase léxica, ou seja, a distinção desses caracteres será feita na fase seguinte, se necessária. Tal decisão foi devido ao fato de que é necessário informações de contexto mais abrangentes do que as expressões regulares são capazes de prover para se distinguir essas duas classes.

- Foi decidido que a estrutura de dados a ser utilizada para consulta de palavras reservadas seria uma TRIE. A principal razão pela qual optamos por essa abordagem é o fato de que funções *hash* não são 100% imunes à colisão, ou seja, a utilização de apenas uma *hash* para tal finalidade teria um comportamento não-determinístico, no sentido de que poderia atribuir como sendo palavra reservada uma cadeia que, na realidade, não é. Além disso, a TRIE é uma estrutura que possibilita buscas eficientes ($O(\iota)$, onde ι é o tamanho da maior palavra na TRIE). Mais informações em [1].

5. Instruções para a execução

1. Para compilar e executar o código do trabalho, digite no terminal do linux:

$ **bash ./script.sh**

2. O script irá lhe requerer o nome do arquivo de entrada. Forneça o nome da entrada (ex.: **test.in** ou ./**tests/test.in**);

3. Para sair, simplesmente digite **sair.**

Obs.: Alternativamente, também é possível compilar usando o makefile provido, digitando **make compile** no terminal. Além disso, é possível executar os pré-testes que realizamos através do comando **make run**.

6. Referências

[1] http://en.wikipedia.org/wiki/Trie

[2] LOUDEN, K.C. **Compiladores – Princípios e Praticas**. Ed. Thomson Pioneira, 2004.

[3] AHO,A.V., LAM, MONICA S., SETHI, RAVI, ULLMAN, J. D. **Compilers - Principles, Techniques and Tools**, Ed. Addison Wesley, 2nd edition, 2006.