Centro Universitário de Belo Horizonte – Uni-BH
Instituto de Engenharia e Tecnologia – IET
Curso: Ciência da Computação
Disciplina: Matemática Discreta e Teoria da Computação
Professor: Moisés Henrique Ramos Pereira

Trabalho Prático de Teoria da Computação – Implementação de um AFD

Um compilador é um programa de computador que transforma um programa em uma linguagem fonte (chamado de código-fonte) para uma linguagem alvo (linguagem objeto, linguagem de máquina, código-objeto).

A linguagem fonte deve ser uma linguagem formal. Sendo assim, ela consiste de um alfabeto e possui regras bem definidas para a formação de programas. Os símbolos de um alfabeto $\sum$ de uma linguagem fonte (uma linguagem de programação, por exemplo), bem como as derivações resultantes da combinação desses símbolos (terminais), são chamados *tokens*. Os *tokens* de Java, por exemplo, são identificadores, comando *if*, constantes reais, constantes inteiras, operadores, etc.

Os *tokens* da linguagem, por sua vez, são conjuntos de lexemas que são instâncias daquele *token*. Exemplificando, o *token* para abrir chaves só tem um lexema que é o próprio símbolo *{*. Já o *token* identificador possui vários lexemas: *nome*, *idade*, etc. Todos os nomes que atribuímos às variáveis são lexemas do *token* identificador.

Os padrões de formação dos lexemas de cada *token* são expressões regulares que determinam como avaliar se um lexema é de um determinado *token*. Em C, por exemplo, o *token* identificador possui o seguinte padrão de formação: *ID* = (_ + letra)(_ + letra + digito)*

Dado que o padrão de formação dos lexemas de um *token* são expressões regulares, os lexemas podem ser reconhecidos por meio de Autômatos Finitos. Este trabalho consiste em implementar um AFD que percorra um arquivo fonte e encontre os *tokens* presentes no arquivo. Os tokens que devem ser reconhecidos com seus respectivos padrões de formação de lexemas são dados abaixo:

Token	**Padrão de Formatação**
ID – Identificador	($)(letra)(_ + letra + digito)*
NumHex – Número Hexadecimal	(digito + ‘A’ + ‘B’ + ‘C’ + ‘D’ + ‘E’ + ‘F’)$^+$
NumReal – Número Real	(digito)$^+$(,)(digito)$^+$
NumInt – Número Inteiro	(digito)$^+$
Cadeia – Cadeia de Caracteres	(“)(letra + digito + símbolo)*(”)
Op – Operador	(‘+’ + ‘-’ + ‘*’ + ‘/’ + ‘%’ + ‘**’ + ‘&’ + ‘|’ + ‘=’)

Na definição acima, letra é o conjunto das letras maiúsculas e minúsculas do alfabeto latino, dígito é o conjunto dos 10 dígitos numéricos e símbolo é o conjunto dos símbolos do teclado, exceto aspas.

Durante o reconhecimento dos *tokens*, espaços em branco, tabulações e quebras de linha devem ser descartados. Logo, o programa deve funcionar como o exemplo a seguir:

Entrada	**Saída**
$valor = 5,6** 1B + 324+ “Goku”%1	ID Op NumReal Op NumHex Op NumInt Op Cadeia Op NumInt

O que fazer?

Você deverá criar um AFD que reconheça os lexemas de cada *token*. Deverá criar um programa que implemente esse AFD. Uma recomendação é que as transições do reconhecimento de cada *token* disparadas do estado inicial sejam completadas nos respectivos estados finais, ou seja, que exista um estado final para cada reconhecimento. No final de cada reconhecimento ou depois de processar toda a entrada, o programa volta para o estado inicial.

Seu programa deve solicitar para o usuário uma string de entrada e apresentar a saída com os *tokens* reconhecidos ou os erros identificados. O programa deve informar em qual posição (coluna) da string de entrada está o erro e qual o tipo de erro. Note que, para esse caso, os erros possíveis são de “caractere desconhecido” e de “cadeia não fechada”.

O trabalho poderá ser implementado em Pascal, C/C++, Java ou Python. Implementações noutras linguagens não serão aceitas.

O que entregar?

Você deverá entregar um AFD desenhado na ferramenta JFLAP, exemplos de entradas corretas e incorretas (pelo menos, 3 de cada) e o código-fonte do seu AFD implementado.

Considerações gerais

1. O trabalho poderá ser feito por grupos de até três pessoas.

2. Bibliografia recomendada:

   AHO, Alfred V. ; SETHI, Ravi ; ULLMAN, Jeffrey D.; LAM, Monica S. Compiladores: Princípios, Técnicas e Ferramentas . 2a ed. São Paulo: Pearson, 2008.

   LOUDEN, Kenneth C. Compiladores: princípios e práticas . Thomson, 2004.

3. Trabalhos copiados da internet ou de outros colegas sofrerão sanções (perdas parcial ou total dos pontos do trabalho).

4. O trabalho vale 20 pontos.