Avaliação Simulada de Compiladores

Unidades:

1. Introdução a compiladores
2. Análise Léxica

Questões:

Parte conceitual

1. (a) Que são aspectos léxicos e sintáticos de uma linguagem de programação? (b) Podemos dizer que existe uma hierarquia de abstração entre esses aspectos? Justifique.
2. A estrutura léxica de uma linguagem de programação é obrigatoriamente suportada por uma linguagem regular? Justifique.
3. (a) Qual a vatangem de se estruturar uma linguagem de programação em sublinguagens conforme a hierarquia de Chomsky? (b) Qual a conveniência de se definir construções léxicas através de linguagens regulares e construções sintáticas através de linguagens livres do contexto? (c) Que viriam a ser as construções pertencentes a sublinguagens 1 e 0 na hierarquia de Chomsky, no contexto de linguagens de programação?
4. O que distingue formalmente gramáticas regulares e expressões regulares?
5. Podemos dizer que a estrura léxica dos palíndromos pode ser descrita por uma gramática regular? Justifique.
6. Qual a vantagem de se abstrair analisadores léxicos na forma de autômatos finitos não-determinísticos?
7. De que modo podemos simular autômatos não-determinísticos na implementação de reconhecedores utilizando a linguagem C? Justifique.
8. Pode haver a necessidade de empregar algoritmos de “backtracking” na implementação de analisadores léxicos baseados em expressões regulares? Justifique ou dê um exemplo de tal situação.
9. Podemos dizer que um analisador léxico de linguagens como ALGOL, Pascal etc é abstraído por um autômato finito estruturado? Justifique ou exemplifique.
10. (a) Defina tokens no contexto de linguagens de programação. (b) Que vem a ser um lexema?

Parte prática

1. Números como 1234, 0123, 0x2F são, respectivamente, inteiros decimais, octais e hexadecimais, ficando o 0 (zero) trivialmente um decimal. (a) Escreva uma expressão regular, usando a notação **lex** do Unix, para descrever qualquer uma das quatro possibilidades. (b) Esboce o grafo de um autômato finito para esta expressão regular. (c) Escreva a tabela de transição deste autômato.
2. (a) Implemente uma função C, com o protótipo **`token_t chkint (FILE *src)`**, que retorne os tokens DEC, OCT, HEX (constantes C) para os casos descritos no exercício anterior. (b) Podemos dizer que este programa simula um NFA? Justique.

3. Há a necessidade de se criar um estado default, obtido por uma transição vazia a partir do estado inicial? Se afirmativo, qual a utilidade deste artifício no contexto de implementação de analisadores léxicos (lexers)?
4. (a) Escreva uma expressão regular que descreva números inteiros em algarismos romanos. (b) A partir desta espressão, implemente um lexer que identifique uma string como sendo ou não um inteiro romano. Obs: a casa dos milhares só vai até MMM.
5. *****Implemente um lexer que recuse concatenações proibidas na lingua portuguesa (BR).
6. Escreva uma expressão regular para descrever números ponto flutuante.
7. *****Faça um upgrade no classificador numérico do Exercício 2 para que este retorne, além dos já existentes, os tokens SFR (simples fracionário), SCN (notação científica).
8. Mostre por argumentos que a sintaxe de instruções da família i386 dos processadores Intel define uma linguagem regular.
9. *(a) Invente uma métrica para definir distâncias lexicográficas entre duas strings ASCIZ quaisquer. (b) Implemente uma função C com tal proósito.
10. Pesquise e mostre a referência bibliográfica para a definição de distância de edição entre duas strings.