Disciplina: Compiladores - 2023.2
Professora: Lis Custódio

Projeto de compilador

Etapa 1: Análise Léxica

Introdução

A primeira etapa do trabalho consiste em implementar um analisador léxico para a linguagem Lua Simplificada, que converterá uma entrada fluxo de caracteres em um fluxo de tokens. Embora esses programas geralmente sejam mais bem escritos usando um gerador de lexer (por exemplo, ML-Lex ou Flex), para esta tarefa você escreverá um scanner do zero, partindo do código discutido em sala de aula.

Convenções léxicas

A linguagem Lua Simplificada possui quatro classes de tokens: identificadores, delimitadores e operadores, números e literais de string. Os tokens podem ser separados por espaços em branco e/ou comentários.
Identificadores em Lua Simplificada podem ser qualquer sequência de letras, dígitos e sublinhados, não começando com um dígito. Os identificadores diferenciam maiúsculas de minúsculas (por exemplo, foo é diferente de Foo).

Os seguintes identificadores são reservados como palavras-chave:

and break do else elseif
end false for function if
in local nil not or
repeat return then true until while

A Lua Simplificada também possui uma coleção de delimitadores e operadores, que são os seguintes:

```
+   -   *   /   ^   =
~=  <=  >=  <   >   ==
(   )   {   }   [   ]
;   :   ,   .   ..
```

Os números em Lua Simplificada são inteiros e seus literais são escritos em notação decimal (sem sinal).

Os literais de string são delimitados por aspas duplas correspondentes e podem conter os seguintes sequências de espaçamento:

\a — bell (ASCII code 7)
\b — backspace (ASCII code 8)

\f — form feed (ASCII code 12)
\n — newline (ASCII code 10)
\r — carriage return (ASCII code 13)
\t — horizontal tab (ASCII code 8)
\v — vertical tab (ASCII code 11)
\\ — backslash
\" — quotation mark

Os comentários começam em qualquer lugar fora de uma string com um hífen duplo (--). Se o texto imediatamente depois de -- é diferente de [[, o comentário é um comentário curto, que vai até o final da linha. Caso contrário, é um comentário longo, que vai até o]]. Comentários longos podem ser executados para várias linhas e pode conter pares [[/]] aninhados.

Requerimentos

1 - Defina formalmente, através de expressões/definições regulares, a sintaxe de cada um dos tipos de lexemas a serem extraídas do texto-fonte pelo analisador léxico, bem como de cada um dos espaçadores e comentários.

2 - Converta cada uma das expressões regulares em autômatos finitos com saída nos estados, que emita como saída a lexema encontrada ao abandonar cada um dos estados finais para iniciar o reconhecimento de mais uma lexema do texto.

3 - Agrupe/converta os autômatos finitos em uma sub-rotina, escrita na linguagem de programação C/C++ (com base no código dado em sala de aula), que para cada lexema lido, deve retornar o nome-token e o atributo correspondentes.

Obs1: Por conveniência, um lexema consistindo de uma cadeia específica tal como ' ; ' ou IF , será tratado como tendo como nome-token a própria cadeia, mas nenhum atributo. Um lexema tal como um identificador, M, terá como nome-token ID e um atributo consistindo da sua linha na tabela de símbolo.
Obs2: Classificar os lexemas reconhecidos em tokens retornando as constantes definidas no início do código ou os códigos ASCII para caracteres simples.

4- Crie um programa principal, que chame repetidamente a sub-rotina construída, e a aplique sobre um arquivo do tipo texto contendo o texto-fonte a ser analisado. **Após cada chamada, esse programa deve imprimir o token referente ao lexema lido.**

Faça o programa parar quando o programa principal receber do analisador léxico uma lexema especial indicativo da ausência de novas lexemas no texto de entrada.

5 - Relate o funcionamento do analisador léxico construído, incluindo no relatório: descrição teórica do programa; descrição da sua estrutura; descrição de seu funcionamento; descrição dos testes realizados e das saídas obtidas.