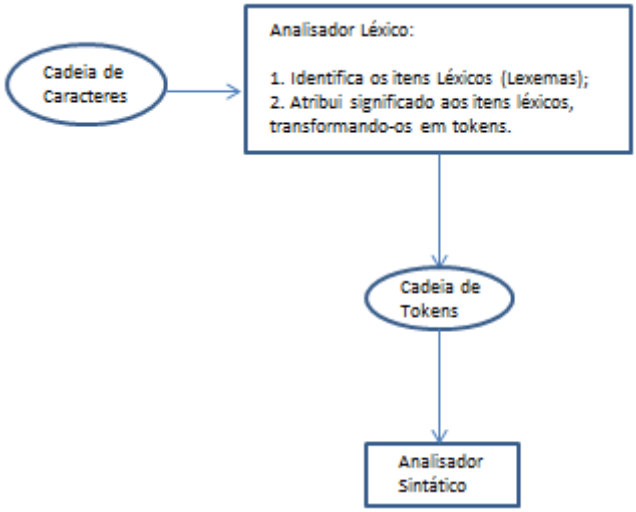
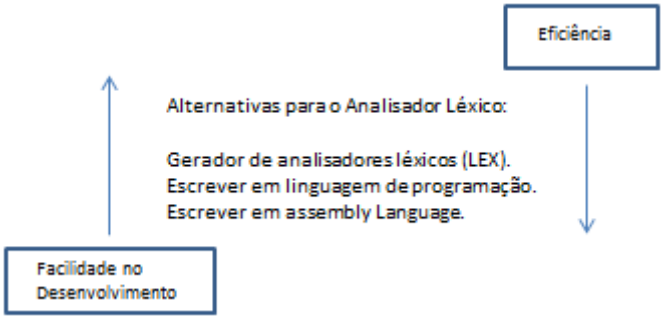
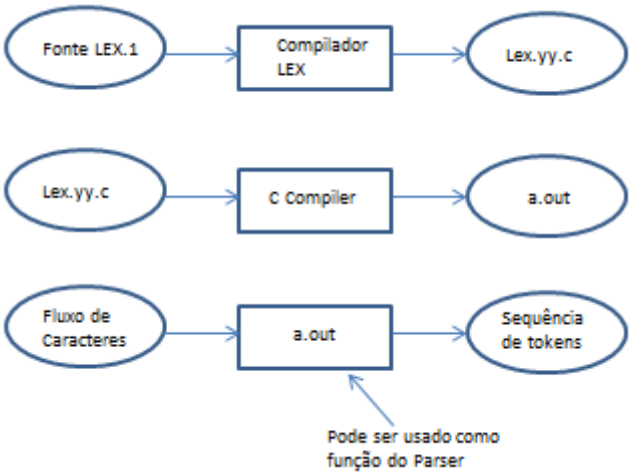
Analisador Léxico



Alternativas:



Compilador LEX (Linguagem LEX)



Exemplo de especificação em LEX (Aho; Lam; Sethi; Ullman):

```
%{
    /* definições de constantes manifestas
    LT, LE, EQ, NE, GT, GE,
    IF, THEN, ELSE, ID, NUMBER, RELOP */
%}

/* definições regulares */
delim      [ \t\n]
ws         {delim}+
letter     [A-Za-z]
digit      [0-9]
id         {letter}({letter}|{digit})*
number     {digit}+(\.{digit}+)?(E[+-]?{digit}+)?

%%

{ws}       {/* nenhuma ação e nenhum retorno */}
if         {return(IF);}
then       {return(THEN);}
else       {return(ELSE);}
{id}       {yylval = (int) installID(); return(ID);}
{number}   {yylval = (int) installNum(); return(NUMBER);}
"<"        {yylval = LT; return(RELOP);}
"<="       {yylval = LE; return(RELOP);}
"="        {yylval = EQ; return(RELOP);}
"<>"       {yylval = NE; return(RELOP);}
">"        {yylval = GT; return(RELOP);}
">="       {yylval = GE; return(RELOP);}

%%

int installID() {/* função para instalar o lexema, cujo
                    primeiro caractere é apontado por yytext,
                    e cujo tamanho é yyleng, na tabela de
                    símbolos, e retorna um apontador para lá */
}

int installNum() {/* semelhante a installID, mas coloca constantes
                     numéricas em uma tabela separada */
}
```

Declarações de Conjuntos

Regras de identificação de Tokens

Procedimentos em Linguagem C

FIGURA 3.23 Programa Lex para os tokens da Figura 3.12.

Compilador Lex:

Três partes separadas pela sequência “%%”:

a) Definições de expressões regulares:
   Definição de constantes e de conjunto de cadeias na forma de expressões regulares especificando as regras de formação dos lexemas.
b) Regras de formação de tokens (regras de tradução):
   Cada regra é um par da forma: Padrão { Ação }
   Cada padrão é uma expressão regular que pode usar as definições de conjuntos regulares da primeira sessão.
c) Funções auxiliares em C.
   Código em C, especificando funções com ações adicionais. Podem ser compiladas separadamente e carregadas no analisador léxico. Por exemplo, se for detectado um identificador as seguintes ações são realizadas:

   i) Chama a função install_ID para colocar o lexema na tabela de símbolos do programa;
   ii) Armazena o índice onde o lexema foi incluído na variável global yylval para ser usado pelo analisador sintático. Para fazer isto, a função install_ID tem à disposição as variáveis yytext e yyleng. A primeira é um apontador para o início do lexema e a segunda é a quantidade de caracteres.
   iii) O token ID é retornado ao analisador sintático.

A parte do código LEX acima, apresenta texto delimitado por %{ e %}. Tudo que estiver contido não é tratado pelo compilador Lex e é copiado diretamente para o arquivo lex.yy.c. É comum colocar definições de constantes com a diretiva #define da linguagem C. Estas constantes servem para associar o lexema encontrado ao respectivo token.

Conflitos no LEX: Quando vários prefixos da cadeia de entrada casam com mais que um padrão, as seguintes regras são aplicadas:

a) Sempre escolher um prefixo longo a um prefixo mais curto;
b) Se for possível casar o prefixo mais longo a mais que um padrão, leve em consideração o que ocorre primeiro na especificação LEX.

Analisador Léxico

Aspectos do analisador Léxico (Lexan)

1. Difere do AEF: O Lexan reconhece um lexema, mesmo sem terminar a análise da cadeia de entrada como ocorre com um AEF. É preciso entender logicamente que final de arquivo corresponde ao fato separador de dois lexemas.
2. Resolver o problema de perder um símbolo da cadeia de entrada entre ativações consecutivas do Lexan.
3. Resolver certo assincronismo entre entrada e saída no sentido de que o último item léxico de uma linha poderia ser detectado somente depois da leitura e impressão da próxima linha.
4. Adotar três parâmetros para o programa principal
   a) Arquivo de entrada contendo a cadeia de entrada com a sequência de tokens.
   b) Arquivo de saída com mensagens e outras informações.
   c) Arquivo binário para obter a TST.
5. A função Lexan() deve retornar o seguinte:

   Token, ou seja, o significado do lexema representado pelo índice da TST correspondente ao item identificado. Este valor indica se o lexema é um identificador, palavra reservada, número, operador ou constante, etc.

6. A função Lexan() deve ter parâmetros de retorno, ou seja, eles são apontadores para tipos. De outra forma, pode-se retornar uma "struct" composta do token citado acima e dos elementos a seguir:

   a) Item Léxico: cadeia de caracteres referente ao token.
   b) Valor numérico do inteiro.
   c) Valor numérico do tipo float.
   d) O conteúdo armazenado na TST referente ao índice retornado. Não tem utilidade para fins de compilador. Apenas para verificação do funcionamento do Lexan.

7. Adotar indiferentemente apóstrofos e aspas como delimitadores de constantes alfanuméricas, mas eles devem ser corretamente casados.
8. Comandos de controle do compilador:
   a) #list_tst - não tem default
   b) #list_token_on - não tem default
   c) #list_token_off - default off
   d) #list_tnt - não tem default
   e) #list_tgrf - não tem default
   f) #list_source_on - default on com número da linha
   g) #list_source_off

   Independentemente, da diretiva LIST_SOURCE_ON, imprimir a linha com a mensagem de erro.

Variáveis estáticas do Lexan ou globais:

a) List_token
b) List_source

9. Variáveis globais:
   a) Posição i (coluna) do vetor que armazena a linha corrente do arquivo de entrada.
   b) Posição i_inicial (coluna inicial) correspondente ao início do lexema atual. Serve para mensagens de erro.
   c) Número da linha no arquivo fonte.
   d) Arquivos de entrada e saída.
   e) Seguir o diagrama de transições do AEF.

10. Incluir manualmente, os seguintes elementos na parte especial da TST:
    ID, NUMBER, FLOAT, ALPHA, EOF defina constantes para eles.
    Obs: LAMBDA (-1) não é um símbolo de entrada, mas um símbolo da gramática que será visto mais tarde.
11. A identificação do que é ou não uma palavra reservada ou símbolo da linguagem é feita pela chamada à função do primeiro trabalho.
12. Adotar comentários da linguagem C.
13. Escrever o AEF que cumpre os papéis do Lexan e implementá-lo na forma vista em aula.
14. Se for necessário ter em mente uma Linguagem de Programação para decidir sobre a construção do AEF, assuma itens léxicos compatíveis com C standard.
15. Na entrada do Lexan, apontar para o primeiro caractere do arquivo de entrada. Quando retorna do Lexan, o apontador para a cadeia de entrada deve estar na posição imediatamente seguinte ao item reconhecido, independentemente do caractere que seja ele.
16. Programa chamador:
    a) Abre e fecha arquivos de entrada e saída.
    b) Retorna do Lexan apontando para a posição (i + 1), onde i é a última posição do item mais recentemente reconhecido.
17. Convenções:
    a) Número em ponto flutuante pode iniciar ou terminar com ponto.
    b) Os delimitadores de cadeias de caracteres podem ser indistintamente apóstrofos ou aspas, mas devem ser casadas corretamente.
    c) As cadeias de constantes alfanuméricas não podem ultrapassar uma linha.

18. Algumas funções do Léxico:
    a) Brancos são tratados como um único símbolo branco.
    b) Remoção de tabs e newlines.
    c) Identifica palavras reservadas.
    d) Identifica símbolos reservados.
    e) Identifica as constantes numéricas e alfanuméricas.

f) Forma os identificadores (variáveis, funções, procedimentos, rótulos, campos, tipos, constantes simbólicas de enumerações).
g) Trata comentários de linha e multilinhas (Podem ser tratados pelo pré-processador.
h) Upper/lower case.
i) Detecção de erros léxicos.
j) Precisa de uma tabela de palavras reservadas e símbolos da linguagem.
k) Retorna o token (valor lógico associado ao lexema).

19. Ler uma linha por vez. Se ativa opção correspondente, imprime a linha.
20. Se ocorrer um erro, mesmo que inativa a opção de impressão do código fonte, imprime a linha e aponta o erro na coluna onde foi encontrado.
21. Obter um caractere da linha por vez.
22. Apontador de erros é diferente do apontador corrente da cadeia de entrada, ou seja, são necessários dois apontadores para a cadeia de entrada.
23. Erros:
    a) Símbolo inexistente no alfabeto.
    b) Diretiva de compilação inexistente.
    c) Comentários sem fechamento.
    d) Constante alfanumérica sem encerramento.
24. Não perder símbolo vistoriado (look ahead)
25. Parâmetros do programa chamador do lexan:
    a) Nome do arquivo de entrada.
    b) Nome do arquivo de saída que contem informações da atuação do analex.
26. Quando é IDENT? Quando é palavra reservada?
27. Leitura de linha e devolução do caractere lido.
    Ler uma linha. Quando necessário, devolver um caractere.
    Sai do lexan com o apontador para o próximo caractere e com o caractere correspondente.
    Apontador de erros. Difere do apontador de entrada.
28. Funções da biblioteca stdio.h.
    c = getchar();
    unget(c);
    fgetc(file);
    Bufferização: Linha atual em análise. Buffer de 4096 ou 2048 bytes.

29. Definição de constantes (#define):
    - NUMBER
    - ALPHA
    - IDENT
    - EOF
    - FLOAT
30. Par produtor / consumidor (lexan x parser)
31. Lexan:
    - Retorna a entrada da TST
    - Parâmetro de saída com a cadeia

32. Estrutura possível:

```
estado = 0;
While (1){
 Switch(estado){
    Case 0: c = le_char();
   If (c == 'c') estado = 1;
.
.
.
```

33. No arquivo de teste: Incluir o comentário de única linha e o multilinhas.
34. O Analisador Léxico implementa o AEF que reconhece os itens léxicos. Deste modo, é necessário adicionar outros estados e transições ao autômato de exemplo:
    a) No estado zero, incluir transições para os símbolos: espaço em branco, tabulações, mudanças de linha e EOF para arquivos vazios.
    b) Strings delimitados por apóstrofos ou aspas.
    c) Diretivas de Compilação. Convencionaremos que elas começam com o símbolo "#" e seguem uma cadeia de letras e números. O delimitador final é a quebra de linha.
    d) Símbolos especiais: Avança mais dois caracteres e consulta a TST com os três caracteres. Se for encontrado, o lexema está formado e retorna da mesma forma como ocorre com outros tokens. Se não for encontrado, devolve o terceiro caractere e repete o procedimento para a cadeia composta por dois caracteres. Se ainda assim, não for encontrado na TST, devolve o segundo símbolo e repete o procedimento somente com o primeiro símbolo. Neste ponto, se for encontrado na TST, o token está formado. Em caso negativo, é uma situação de erro (símbolo inválido) e deve retornar ao estado zero.