1a)

|  |  |  |
| --- | --- | --- |
| Lexema | Classe | Padrão Utilizado |
| function | Palavra-chave | Próprio lexema |
| max | identificador | Cadeia de caracteres |
| ( | pontuação | Próprio lexema |
| i | identificador | Cadeia de caracteres |
| , | pontuação | Próprio lexema |
| j | identificador | Cadeia de caracteres |
| : | pontuação | Próprio lexema |
| integer | Palavra-chave | Próprio lexema |
| ) | pontuação | Próprio lexema |
| : | pontuação | Próprio lexema |
| integer | Palavra-chave | Próprio lexema |
| ; | Pontuação | Próprio lexema |
| { retorna o maior dos inteiros entre i e j } | Comentário | Sequencia de caracteres delimitada por { e } |
| begin | Palavra-chave | Próprio lexema |
| if | Palavra-chave | Próprio lexema |
| i | Identificador | Cadeia de caracteres |
| > | Operador relacional | Próprio lexema |
| j | Identificador | Cadeia de caracteres |
| then | Palavra-chave | Próprio lexema |
| max | Identificador | Cadeia de caracteres |
| := | Operador atribuição | Próprio lexema |
| i | Identificador | Cadeia de caracteres |
| else | Palavra-chave | Próprio lexema |
| max | Identificador | Cadeia de caracteres |
| := | Operador atribuição | Próprio lexema |
| j | Identificador | Cadeia de caracteres |
| end | Palavra-chave | Próprio lexema |
| ; | Pontuação | Próprio lexema |

1b)

|  |  |  |
| --- | --- | --- |
| Lexema | Classe | Padrão Utilizado |
| int | Palavra-chave | Próprio lexema |
| max | identificador | Cadeia de caracteres |
| ( | pontuação | Próprio lexema |
| i | identificador | Cadeia de caracteres |
| , | pontuação | Próprio lexema |
| j | identificador | Cadeia de caracteres |
| ) | pontuação | Próprio lexema |
| int | Palavra-chave | Próprio lexema |
| i | identificador | Cadeia de caracteres |
| , | pontuação | Próprio lexema |
| j | identificador | Cadeia de caracteres |
| ; | pontuação | Próprio lexema |
| /\* retorna o maior dos inteiros entre i e j \*/ | comentário | Sequencia de caracteres delimitada por /\*e \*/ |
| { | Delimitador de escopo | Próprio lexema |
| return | Palavra-chave | Próprio lexema |
| i | identificador | Cadeia de caracteres |
| > | Operador relacional | Próprio lexema |
| j | identificador | Cadeia de caracteres |
| ? | Operador ternário 1 | Próprio lexema |
| i | Identificador | Cadeia de caracteres |
| : | Operador ternário 2 | Próprio lexema |
| j | Identificador | Cadeia de caracteres |
| } | Delimitador de escopo | Próprio lexema |

1c) No primeiro programa, em Pascal.

2a)

<Delim><Dec><Var, 1><Delim><Int><Var, 2><Delim><Iint><Var, 3><Delim><Int><Var,4><Delim><Int> <Delim><ALG><LER><Var, 1><LER><Var, 2><LER><Var, 3><SE><Var, 1><OpRel, “>” ><Var, 2><ENT><INI><ATR>

<Var, 2><A><Var, 4><ATR><Var, 1><A><Var, 2><ATR><Var, 4><A><Var, 1><FIM><SE><Var, 1><OpRel, “>” >

<Var, 3><ENT><INI><ATR><Var, 3><A><Var, 4><ATR><Var, 1><A><Var, 3><ATR><Var, 4><A><Var, 1><FIM><SE>

<Var, 2><OpRel, “>” ><Var, 3><ENT><INI><ATR><Var, 3><A><Var, 4><ATR><Var, 2><A><Var, 3><ATR>

<Var, 4><A><Var, 2><FIM><IMP><Var, 1><IMP><Var, 2><IMP><Var, 3>

|  |  |
| --- | --- |
| índice | lexema |
| 1 | numero1 |
| 2 | numero2 |
| 3 | numero3 |
| 4 | aux |

2b)

<Delim> <Dec> <Var, 1> <Delim> <Int> <Var, 2> <Delim> <Int> <Var, 3> <Delim> <Int> <Delim> <Alg> <Ler> <Var, 1> <Se> <Var, 1> <OpRelIgual> <NumI, 4> <Ent> <Ini> <Atr> <NumI, 4> <A> <Var, 2> <Atr> <NumI, 4> <A> <Var, 3> <Fim> <Sen> <Se> <Var, 1> <OpRelIgual> <NumI, 5> <Ent> <Ini> <Atr> <NumI, 5> <A> <Var, 2> <Atr> <NumI, 5> <A> <var, 3> <Fim> <Sen> <Ini> <Atr> <Var, 6> <OpArit, “\*”> <Var, 1> <A> <Var, 2> <Atr> <Var, 1> <OpArit, “\*”> <AP> <Var, 1> <OpArit, “\*”> <Var, 1> <A> <Var, 7> <Fim> <Imp> <var, 1> <Imp> <Str, 8> <Imp> <Var, 3> <Var, 9> <Str, 10> <Imp> <Var, 1> <Imp> <Str, 11> <Var, 12> <Imp> <Str, 10>

|  |  |
| --- | --- |
| índice | lexema |
| 1 | numero |
| 2 | potencia2 |
| 3 | potencia3 |
| 4 | 0 |
| 5 | 1 |
| 6 | numbero |
| 7 | potencia33 |
| 8 | ‘ ao quadrado é igual a ‘ |
| 9 | IMPRIMIRS |
| 10 | ‘\n’ |
| 11 | ‘ ao cubo é igual a ‘ |
| 12 | IMPRIMIRpotencia3 |

2c) Sim, o programa b.

3) Linha 8: Não reconhece o lexema “!”. Linha 17: Não reconhece o lexema “&”

Explicações: Linha 2: “/” é um operador aritmético válido. Reconhece duas vezes.

Linha 3: reconhece uma constante numérica 123 e depois uma variável.

Linha 5: “=” é um operador relacional válido. Reconhece duas vezes.

Linha 7: “>”, “<=” e “>” são operadores relacionais válidos.

Linha 12: Comentários devem ter pelo menos um símbolo de porcentagem, o resto é ignorado.

Linha 15: mesmo caso da Linha 3.

4) – **Humanos entendem melhor** os programas que seguem nosso modelo de linguagens (palavras/vocabulários vs frases/gramática)

- **Facilita a implementação**. Permite dividir o problema em partes menores, dando foco específico em diferentes partes do problema (dividir-para-conquistar)

- **Facilita análise sintática**. Análise léxica é um problema mais simples (formalismo de linguagens regulares). Análise sintática, sem se considerar nomes (trabalho do léxico), também é facilitada (formalismo de linguagens livres de contexto).

- **Eficiência**. É possível otimizar tarefas de leitura, como por exemplo criar um buffer de entrada.

- **Portabilidade**. Permite “reaproveitar” apenas uma das partes. Também facilita a manutenção.

5) Na análise léxica é necessário ler uma longa sequência de caracteres, geralmente de um arquivo. Ler os caracteres, um a um, do disco, é menos eficiente do que ler porções maiores e trabalhar na memória.

Na análise léxica é comum uma implementação do tipo tentativa e erro, pois é mais vantajosa do que uma busca em paralelo por todas as possibilidades de casamento. Assim, cada tentativa errada demanda que haja um retrocesso dos caracteres lidos. Se fosse feita leitura do disco, isso não seria possível facilmente. Já na memória basta fazer um gerenciamento de ponteiros para que isso seja feito facilmente.

No retrocesso, existe um caso onde pode haver perda de informações. Caso uma tentativa ultrapasse o limite superior do buffer (simples), o mesmo é recarregado totalmente. Se for necessário retroceder além do limite inferior do buffer, a informação anterior já não estará mais lá. Por isso, trabalha-se com um buffer duplo, onde cada metade é recarregada de forma alternada. Para isso, no entanto, é necessário cuidado para não recarregar a mesma metade do buffer duas vezes, caso haja a situação de retrocesso além dos limites, como citado acima.

6) Permite que o programador atue em um nível mais alto de abstração. Com isso, ele pode focar seu trabalho em identificar corretamente os padrões, sem se preocupar com detalhes como retrocesso, buffer duplo, leitura do arquivo, etc. Além disso, hoje em dia os geradores produzem código quase tão eficiente (ou mais eficiente) do que aqueles construídos à mão.

7) a) ‘a’ (‘A’..’Z’)\* ‘a’

b) (‘0’..’9’)\* (‘0’|’2’|’4’|’6’|’8’)

c) (‘1’? (‘0’ (‘1’)\* ‘0’) \* )\*

d) (‘1’? (‘00’)\* )\*

e) Não é uma linguagem regular

f) (‘1’|’3’|’5’|’7’|’9’)\* (‘0’|’2’|’4’|’6’|’8’)\*

g) consoante: ‘ b’|’c’|’d’|’f’|...|’z’

cadeia: consoante\* ‘a’ consoante\* ‘e’ consoante\* ‘i’ consoante\* ‘o’ consoante\* ‘u’ consoante\*

h) (‘/\*’) .\*? (‘\*/’)