TRABALHO nº 2: implementação do analisador léxico

Implementar o analisador léxico de forma que reconheça os *tokens* especificados para a linguagem 2021.1 levando em consideração as especificações feitas no trabalho nº 1. Deve-se também implementar estratégias para a recuperação e tratamento de erros léxicos (símbolos que não fazem parte da linguagem, bem como sequências de símbolos que não obedecem às regras de formação dos *tokens* especificados).

Utilizar o JavaCC como ferramenta geradora do analisador léxico!!!

ENTRADA:

- conjunto de caracteres, podendo ser um arquivo texto ou um texto de um editor de textos (do ambiente de compilação), contendo o programa a ser analisado.

SAÍDA:

- **lista de *tokens*** contendo:
  - o lexema;
  - o número da linha;
  - o número da coluna (início);
  - a categoria e o número da categoria do *token*, de acordo com a tabela de símbolos terminais específica para cada linguagem

OU

- **mensagens de erro** indicando a ocorrência de erro(s) léxico(s). Neste caso, indicar o(s) **erro**(s) fazendo um diagnóstico de boa qualidade, ou seja, emitindo uma mensagem adequada, tal como *identificador inválido*, *comentário não finalizado*, *símbolo inválido*, etc., bem como a **linha** onde ocorreu.

OBSERVAÇÕES: desconsiderar espaços em branco, tabulação à esquerda e comentários.

DATA LIMITE: **29/03 (segunda-feira) postar no material didático, até as 12h**
29/03 (segunda-feira) será a **DEFESA DO TRABALHO**

ENTREGAR:

- ✓ a tabela de símbolos terminais (*token*, código do *token*, descrição do *token*); **PDF**
- ✓ a estrutura dos comentários de linha e de bloco; **PDF**
- ✓ a lista de mensagens de erro (código do erro e descrição do erro); **PDF**
- ✓ arquivo do JavaCC com a especificação léxica; **PDF**
- ✓ cópias do programa fonte e do programa executável postado no material didático (indicar como o analisador léxico deve ser usado). **Gerar o .JAR**
- ✓ **COMPACTAR todos os arquivos com o nome da equipe**

Serão levadas em consideração a **qualidade das mensagens de erro e a qualidade da interface do programa,** ou seja, deve-se projetar um ambiente de compilação conforme especificação preliminar anexa.

A nota da implementação do analisador léxico será composta da seguinte forma:

✓ ambiente de compilação:	20%
✓ reconhecimento de palavras reservadas:	10%
✓ reconhecimento de identificadores:	10%
✓ reconhecimento de constantes inteiras:	10%
✓ reconhecimento de constantes reais:	10%
✓ reconhecimento de constantes literais:	05%
✓ reconhecimento de símbolos especiais:	05%
✓ reconhecimento de comentários (linha e bloco):	10%
✓ tratamento de erros (mensagens):	20%

TRABALHO EM EQUIPE

ESPECIFICAÇÃO DA LINGUAGEM 2021.1

Forma geral de um programa

:- *comentário*
program {
<declaração de constantes e variáveis>
<corpo do programa>
}
identificador

- **:-** *comentário* é facultativo
- *comentário* corresponde a um comentário (constante literal) acerca do programa.
- *identificador* corresponde ao identificador do programa e é opcional

Forma geral da declaração de constantes e variáveis

define {
not variable
<tipo> **is** <lista de identificadores> <valor> **.**
variable
<tipo> **is** <lista de identificadores> **.**
}

- <tipo> **is** <lista de identificadores> <valor> **.** e
  <tipo> **is** <lista de identificadores> **.** podem ocorrer uma ou mais vezes;
- <tipo> pode ser **natural**, **real**, **char** ou **boolean**;
- em <lista de identificadores> deve existir no mínimo um identificador e, caso existam mais identificadores, os mesmos serão separados uns dos outros por uma vírgula (**,**); no caso da declaração de variáveis, cada identificador poderá ser seguido por **[** *constante numérica inteira* **]**, indicando uma variável indexada unidimensional, cujos índices variam no intervalo de 1 até a constante numérica especificada;
- <valor> pode ser um valor inteiro, real ou literal, compatíveis com os tipos **natural**, **real** e **char**, respectivamente;
- a declaração de constantes é precedida de **not variable**;
- a declaração de variáveis é precedida de **variable**;
- a declaração de constantes pode preceder a declaração de variáveis ou a declaração de variáveis pode preceder a declaração de constantes e isto ocorrerá apenas uma única vez;
- a declaração de constantes pode não existir, caso não seja utilizada nenhuma constante no programa;
- a declaração de variáveis pode não existir, caso não seja utilizada nenhuma variável no programa;
- a declaração de constantes e variáveis pode não existir, caso não seja utilizada nenhuma constante ou variável no programa.

Forma geral do corpo do programa

execute {
<lista de comandos>
}

- em <lista de comandos> deve existir no mínimo um comando.

Forma geral do comando de atribuição

set <expressão> **to** <lista de identificadores> **.**

- em <lista de identificadores> deve existir no mínimo um e caso existam mais identificadores de variáveis, os mesmos estão separados uns dos outros por uma vírgula (**,**);
- a <lista de identificadores> deve se referir a identificadores de variáveis;
- <expressão> pode ser qualquer expressão aritmética, relacional ou lógica envolvendo identificadores e/ou constantes do tipo **natural, real**, **char** ou **boolean**;
- o resultado da avaliação de <expressão> deve ser um valor do mesmo tipo (ou de tipo compatível) da lista de identificadores.

Forma geral do comando de entrada de dados

get { <lista de identificadores> **} .**

- em <lista de identificadores> deve existir no mínimo um e, caso existam mais identificadores de variáveis, os mesmos serão separados uns dos outros por uma vírgula (**,**).

Forma geral do comando de saída de dados

put { <lista de identificadores e/ou constantes> **} .**

- em <lista de identificadores e/ou constantes> deve existir no mínimo um identificador de constante/variável ou uma constante (numérica ou literal) e, caso existam mais identificadores de constantes/variáveis e/ou constantes, os mesmos serão separados uns dos outros por uma vírgula (**,**).
- as constantes e os conteúdos dos identificadores serão apresentados no dispositivo padrão de saída.

Forma geral do comando de seleção

verify <expressão>
is true { <lista de comandos> **}**
is false { <lista de comandos> **} .**

- <expressão> pode ser qualquer expressão relacional ou lógica envolvendo identificadores e/ou constantes do tipo **natural, real**, **char** ou **boolean**;
- o resultado da avaliação de <expressão> deve ser um valor lógico (**true** ou **false**);
- as cláusulas **is true** e **is false** só podem ocorrer uma vez, em qualquer ordem e são opcionais, mas deve existir no mínimo uma delas.

Forma geral do comando de repetição

loop
{ <lista de comandos> **}**
while <expressão> **is true .**

while <expressão> **is true do**
{ <lista de comandos> **} .**

- <expressão> pode ser qualquer expressão relacional ou lógica envolvendo identificadores e/ou constantes do tipo **natural, real**, **char** ou **boolean**;
- o resultado da avaliação de <expressão> deve ser um valor lógico (**true** ou **false**);
- os comandos da estrutura de repetição serão repetidos sempre que o resultado da avaliação da expressão for **true**.

Operadores:

a) aritméticos: **+ - * / **** (potência) **%** (divisão inteira) **%%** (resto da divisão inteira)
b) relacionais: **==** (igual), **!=** (diferente), **<** (menor), **>** (maior), **<=** (menor igual) e **>=** (maior igual)
c) lógicos: **&** (e), **|** (ou) e **!** (não)

Podem ser usados para agrupar as expressões aritméticas, relacionais ou lógicas os parênteses **(** e **)**.

São constantes lógicas: **true** (verdadeiro) e **false** (representando falso).

As palavras reservadas podem ser escritas com letras minúsculas e/ou maiúsculas.

Todos os comandos são finalizados com um ponto.