TRABALHO nº 2: implementação do analisador léxico

Implementar o analisador léxico de forma que reconheça os *tokens* especificados para a linguagem 2024.2, 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*, ...

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

DATA LIMITE: **06/09 (sexta-feira) postar no AVA Univali até 12h, impreterivelmente.**
06/09 (sexta-feira) será a **DEFESA DO TRABALHO.**

ENTREGAR:

- ✓ a tabela de símbolos terminais (*token*, código do *token*, descrição do *token*); **PDF e impresso**
- ✓ a estrutura dos comentários de linha e de bloco; **PDF e impresso**
- ✓ a lista de mensagens de erro (código do erro e descrição do erro); **PDF e impresso**
- ✓ arquivo do JavaCC com a especificação léxica; **PDF e impresso**
- ✓ cópias do programa fonte e do programa executável (indicar como o analisador léxico deve ser usado).
- ✓ **Gerar o .JAR (sob pena de receber nota zero)**

usar o IntelliJ

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 anexa.

Não deve ser implementado nada relativo à análise sintática, sob pena de receber nota zero.

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 2024.2

Forma geral de um programa

```
make  identificador
      <declaração de constantes e/ou variáveis>
      <lista de comandos>
end.
```

- *identificador* corresponde ao identificador do programa e é opcional;
- em <lista de comandos> deve existir ao menos um comando.

Forma geral da declaração de constantes e/ou variáveis

- a declaração de constantes é precedida de **const**;
- a declaração de variáveis é precedida de **var**;
- 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 da declaração de constantes

```
const
      <tipo> : <lista de identificadores> = <valor> .
end;
```

- <tipo> **:** <lista de identificadores> = <valor> **.** pode ocorrer uma ou mais vezes;
- <tipo> pode ser **int**, **real** ou **char**, correspondentes a um valor inteiro, real ou literal, respectivamente;
- em <lista de identificadores> deve existir no mínimo um e, caso existam mais identificadores, deverão ser separados uns dos outros por uma vírgula (**,**);
- <valor> pode ser um valor inteiro, real ou literal, compatíveis com os tipos **int**, **real** e **char**, respectivamente.

Forma geral da declaração de variáveis

```
var
      <tipo> : <lista de identificadores> .
end;
```

- <tipo> **:** <lista de identificadores> **.** pode ocorrer uma ou mais vezes;
- <tipo> pode ser **int**, **real**, **char** ou **bool**, correspondentes a um valor inteiro, real, literal ou lógico, respectivamente;
- em <lista de identificadores> deve existir no mínimo um e, caso existam mais identificadores, deverão ser separados uns dos outros por uma vírgula (**,**).

Forma geral do comando de atribuição

```
<expressão> -> identificador .
```

- <expressão> pode ser qualquer expressão aritmética, relacional ou lógica envolvendo identificadores e/ou constantes do tipo/compatíveis com os tipos **int**, **real**, **char** ou **bool**;
- o resultado da avaliação de <expressão> deve ser um valor do mesmo tipo (ou de tipo compatível) com o do identificador.

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, deverão ser 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, deverão ser separados uns dos outros por uma vírgula (**,**).

Forma geral do comando de seleção

```
if  <expressão>  then
     <lista de comandos>
else
     <lista de comandos>
end .
```

- <expressão> pode ser qualquer expressão aritmética, relacional ou lógica envolvendo identificadores e/ou constantes do tipo/compatíveis com os tipos **int**, **real**, **char** ou **bool**;
- o resultado da avaliação de <expressão> deve ser um valor lógico (**true** ou **false**);
- caso o resultado da avaliação de <expressão> seja **true**, os comandos associados a cláusula **if** serão executados; caso seja **false**, os comandos associados à cláusula **else** serão executados;
- a cláusula **else** é opcional.

Forma geral do comando de repetição

```
while  <expressão>  do
      <lista de comandos>
end .
```

- <expressão> pode ser qualquer expressão aritmética, relacional ou lógica envolvendo identificadores e/ou constantes do tipo/compatíveis com os tipos **int**, **real**, **char** ou **bool**;
- o resultado da avaliação de <expressão> deve ser um valor lógico (**true** ou **false**);
- sempre que o resultado da avaliação de <expressão> for **true**, a lista de comandos será executada; quando a avaliação de <expressão> resultar em um valor **false**, a repetição é interrompida.

São operadores:

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

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

São constantes lógicas: **true** e **false**.

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

Todos os comandos são finalizados com um ponto.