TRABALHO FINAL – parte 2: implementação do analisador léxico

Implementar o **analisador léxico** de forma que identifique, nos programas escritos na linguagem 2020.2, os *tokens* corretos, levando em consideração as especificações/correções feitas no trabalho nº1. Deve-se implementar também **tratamento de erros** léxicos, quais sejam: símbolos que não fazem parte da linguagem em questão bem como sequências de símbolos que não obedecem às regras de formação dos *tokens* especificados.

Na implementação do analisador léxico pode ser utilizada qualquer ferramenta para geração de compiladores (GALS, JavaCC, etc.) que gere analisadores sintáticos do tipo descendente (recursivo ou preditivo tabular).

Entrada	– A entrada para o analisador léxico é um conjunto de caracteres, isto é, o programa fonte do editor do compilador.
Saída	– Caso o botão **compilar** seja pressionado (ou a tecla de atalho correspondente), a ação deve ser: executar a análise léxica do programa fonte e apresentar a saída. Um programa pode ser compilado com sucesso ou apresentar erros. Em cada uma das situações a saída deve ser: 1ª situação: programa compilado com sucesso ✓ **lista de *tokens***, na área reservada para mensagens, contendo, para cada *token* reconhecido, a linha onde se encontra, a sua classe (por extenso) e o lexema, nessa ordem, conforme exemplo; ✓ **mensagem** (*programa compilado com sucesso*), na área reservada para mensagens, indicando que o programa não apresenta erros. As classes possíveis para os *tokens* são: palavra reservada, identificador, constante `int`, constante `float`, constante `str`, símbolo especial. 2ª situação: programa apresenta erros ✓ **mensagem**, na área reservada para mensagens, indicando que o programa apresenta erro. Neste caso, indicar a linha onde ocorreu o erro e a descrição do erro, emitindo uma mensagem adequada. Tem-se que: • para símbolo inválido: deve ser apresentado, além da mensagem (*símbolo inválido*) e da linha onde ocorreu o erro, o símbolo não reconhecido; • para constante `str` inválida: devem ser apresentadas a mensagem (*constante* `str` *inválida ou não finalizada*) e a linha onde ocorreu o erro; • para comentário de bloco inválido ou não finalizado: devem ser apresentadas a mensagem (*comentário de bloco inválido ou não finalizado*) e a linha onde ocorreu o erro, sendo que, nesse caso, deve ser apresentada a linha onde inicia o comentário. As mensagens geradas por ferramentas, como o GALS, devem ser alteradas. Caso seja pressionado o botão **compilar** e o editor não contenha nenhuma sequência de caracteres ou contenha apenas caracteres de formatação (espaço em branco, quebra de linha ou tabulação), deve ser apresentada a mensagem *nenhum programa para compilar* na área reservada para mensagens.

OBSERVAÇÕES:

- As palavras reservadas da linguagem 2020.2 são: `and or not if elif else for in range while end false true input int float str print`. As palavras reservadas devem ser especificadas como casos especiais de `identificador`, definido no trabalho nº1.
- Os símbolos especiais da linguagem 2020.2 são: `( ) == != < <= > >= + - * / // % , : = += -=` Símbolos diferentes dos especificados nesse item constituem erro léxico.
- Os comentários (de bloco e de linha) e os caracteres de formatação (espaços em branco, quebra de linha, tabulação) devem ser reconhecidos, porém ignorados. Ou seja, não devem ser apresentados como saída do analisador léxico. Isso deve ser especificado na própria ferramenta que será usada para gerar o analisador léxico, no arquivo com as especificações léxicas (no caso do GALS, arquivo com extensão `.gals`). Comentários de bloco que não seguem o padrão de formação especificado devem ser diagnosticados como erro léxico.
- As **mensagens de erro** devem ser conforme nos exemplos abaixo:
  Erro na linha 1 – `@` símbolo inválido
  Erro na linha 1 – constante str inválida ou não finalizada
  Erro na linha 1 – comentário de bloco inválido ou não finalizado
  No 1º exemplo, o símbolo `@` não é símbolo especial da linguagem, portanto caracteriza um erro léxico. Nesse caso, o símbolo foi apresentado na mensagem de erro. Nos demais exemplos foram apresentadas a linha e a mensagem de erro. Ou seja, as sequências não reconhecidas não foram apresentadas na mensagem de erro. Observa-se que para comentários de bloco inválidos ou não finalizados deve ser apresentada a linha onde inicia o comentário.
- As especificações feitas no trabalho nº1 (e já corrigidas) devem usadas para implementação do analisador léxico. Observa-se que essas especificações devem ser adaptadas à notação da ferramenta que será utilizada para gerar o analisador léxico. Além disso, trabalhos desenvolvidos usando especificações diferentes daquelas elaboradas pela equipe no trabalho nº1 receberão nota 0.0 (zero). Qualquer alteração nas especificações feitas no trabalho nº1 devem ser acordadas com a professora.
- A implementação do analisador léxico, bem como da interface do compilador, deve ser disponibilizada no AVA (na aba COMPILADOR), na "**pasta" da sua equipe**. Deve ser disponibilizado um **arquivo compactado** (com o nome: `lexico`) contendo: o código fonte, o executável e o arquivo com as especificações léxicas (no GALS, arquivo com extensão `.gals`).

- Na avaliação do analisador léxico serão levadas em consideração: a correta especificação dos *tokens*, conforme trabalho nº1; a qualidade das mensagens de erro, conforme descrito acima; o uso apropriado de ferramentas para construção de compiladores.

DATA: entregar o trabalho até às 23h do dia 02/10/2020 (sexta-feira). Não serão aceitos trabalhos após data e hora determinados.

EXEMPLOS DE ENTRADA / SAÍDA

EXEMPLO 1: sem erro léxico

ENTRADA

```
linha
   1 [
   2 isso é um comentário
   3 ]
   4
   5 while   area   ==
   6
   7 01.0
```

SAÍDA (na área de mensagens)

```
linha   classe                lexema
5       palavra reservada     while
5       identificador         area
5       símbolo especial      ==
7       constante float       01.0


    programa compilado com sucesso
```

EXEMPLO 2: com erro léxico

ENTRADA

```
linha
   1 [
   2 isso é um comentário
   3 ]
   4 @
   5 while   area   ==
   6
   7 01.0
```

SAÍDA (na área de mensagens)

```
Erro na linha 4 - @ símbolo inválido
```