TRABALHO FINAL – parte 3: implementação do analisador sintático

Implementar o **analisador sintático** de forma que indique quais programas escritos na linguagem 2025.2 estão sintaticamente corretos, seguindo as orientações abaixo:
1º passo: efetue as correções/alterações nas especificações dos *tokens* e da gramática, conforme indicado na avaliação do trabalho no.3.

2º passo: implemente o analisador sintático, bem como o tratamento de erros sintáticos, conforme especificado abaixo.

Entrada	– A entrada para o analisador sintático é um conjunto de caracteres, isto é, o programa fonte do editor do compilador.
Saída	– Caso o botão **compilar** seja pressionado, a ação deve ser: executar as análises léxica e sintática 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 ✓ **mensagem** (*programa compilado com sucesso*), na área reservada para mensagens, indicando que o programa não apresenta erros. A **lista de *tokens*** não deve mais ser mostrada na área reservada para mensagens. 2ª situação: programa apresenta erros ✓ **mensagem**, na área reservada para mensagens, indicando que o programa apresenta erro. O erro pode ser **léxico** ou **sintático**, cujas mensagens devem ser conforme descrito abaixo. As mensagens geradas pelo GALS devem ser alteradas, conforme explicado em aula.

OBSERVAÇÕES:

- O tipo do analisador sintático a ser gerado é **LL (1)**.
- As mensagens para os **erros léxicos** devem ser conforme especificado na parte 2 do trabalho final.
- As mensagens para os **erros sintáticos** devem indicar a linha onde ocorreu o erro, o *token* encontrado (lexema) e o(s) símbolo(s) esperado(s), conforme explicado em aula e detalhado a seguir. Assim, tem-se alguns exemplos:

  linha 1: **encontrado** `EOF` **esperado** `expressão`
  linha 1: **encontrado** `area` **esperado** `(`

  Observa-se que:
  - quando for encontrado uma constante_`string` , como por exemplo `"olá mundo"`, a mensagem deve ser: **encontrado** `constante_string` **esperado ...**
  - quando for encontrado `$`, a mensagem deve ser: **encontrado** `EOF` **esperado ...**
  - para qualquer uma das demais classes de *tokens* encontradas, deve ser apresentado o lexema na mensagem de erro, como em: **encontrado** `area` **esperado ...** , **encontrado** `123` **esperado ...** , **encontrado** `;` **esperado ...**
  - quando for esperado fim de programa, a mensagem deve ser: **encontrado ... esperado** `EOF`
  - quando for esperado identificador, constante_`int`, constante_`float` ou constante_`string`, um deles, a mensagem deve ser, respectivamente: **encontrado ... esperado** `identificador`, **encontrado ... esperado** `constante_int`, **encontrado ... esperado** `constante_float`, **encontrado ... esperado** `constante_string`
  - quando for esperado `int`, `float`, `bool`, `string` e `list`, todos eles, a mensagem deve ser, respectivamente: **encontrado ... esperado** `tipo`
  - quando for esperado `int`, `float`, `bool`, `string` ou `list`, um deles, a mensagem deve ser, respectivamente: **encontrado ... esperado** `int`, **encontrado ... esperado** `float`, **encontrado ... esperado** `bool`, **encontrado ... esperado** `string`, **encontrado ... esperado** `list`
  - quando for esperado `int`, `float`, `bool` e `string`, todos eles, a mensagem deve ser, respectivamente: **encontrado ... esperado** `tipo primitivo`
  - as palavras reservadas e os símbolos especiais devem ser escritos nas mensagens de erro tal como definido na linguagem, ou seja, exatamente como o programador deve escrevê-los nos programas
  - para o não-terminal `<lista_de_expressoes>`, ou com outro nome, usado para definir essa estrutura sintática especificada no trabalho no.3, a mensagem deve ser: **encontrado ... esperado** `expressao`
  - para os não-terminais `<expressao>`, `<expressao_>`, `<valor>`, `<relacional>`, `<relacional_>`, `<aritmetica>`, `<aritmetica_>`, `<termo>`, `<termo_>`, `<fator>` e `<fator_>`, a mensagem deve ser: **encontrado ... esperado** `expressão`
  - para os demais não-terminais, a mensagem deve ser: **encontrado ... esperado** $símbolo_1$, $símbolo_2$, ... $símbolo_n$, conforme tabela de análise sintática (exemplificado a seguir);
  - todas as mensagens de erro geradas pelo GALS devem ser mantidas (em **comentário de linha** ao lado da mensagem alterada), MAS devem ser alteradas, seguindo as orientações dadas em aula.

Por exemplo, considerando o seguinte “trecho” da tabela de análise sintática (menu `Documentação > Tabela de Análise Sintática`):

	id	"="	"<-"	";"	","	pr_add	pr_begin	pr_bool	pr_delete	pr_do	pr_float	pr_if	pr_int	pr_list	pr_print	pr_read	pr_string
<programa>	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-
<lista_instrucoes>	1	-	-	-	-	-	-	1	-	1	1	1	1	1	1	1	1
<tipo>	-	-	-	-	-	-	-	7	-	-	7	-	7	8	-	-	7
<lista>	-	-	-	-	-	-	-	-	-	-	-	-	-	13	-	-	-
<lista_id>	14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
<atribuicao_manipulacao_listas>	-	22	22	-	-	23	-	-	23	-	-	-	-	-	-	-	-

As mensagens de erro para os não-terminais relacionados devem ser:

- para o não-terminal `<programa>`: **encontrado** ... **esperado** `begin`
- para o não-terminal `<lista_instrucoes>`: **encontrado** ... **esperado** `identificador do if print read tipo`
- para o não-terminal `<tipo>`: **encontrado** ... **esperado** `tipo`
- para o não-terminal `<lista>`: **encontrado** ... **esperado** `list`
- para o não-terminal `<lista_id>`: **encontrado** ... **esperado** `identificador`
- para o não-terminal `<atribuicao_manipulacao_listas>`: **encontrado** ... **esperado** `= <- add delete`
- para o não-terminal `<expressao>`: **encontrado** ... **esperado** `expressao`

- Observa-se que:
  (1) as especificações feitas no trabalho nº3 devem usadas para implementação do analisador sintático, caso contrário, será atribuída nota 0.0 (zero) ao analisador sintático;
  (2) todas as alterações/correções indicadas devem ser efetuadas, caso contrário, será atribuída nota 0.0 (zero) ao analisador sintático;
  (3) o trabalho nº3 deve ser devolvido para professora na 1ª aula de implementação do analisador sintático, caso contrário, será atribuída nota 0.0 (zero) ao analisador sintático.
- A implementação do analisador sintático, bem como da interface do compilador e do analisador léxico, deve ser disponibilizada no **AVA**, na aba **COMPILADOR**, na tarefa **“parte 3 - analisador sintático”**. Deve ser disponibilizado um **arquivo compactado** (com o nome: `sintatico`, seguido do número da equipe), contendo o projeto completo, incluindo:
  (1) **código fonte**;
  (2) **executável** (ou **.jar**);
  (3) **arquivo com as especificações léxica e sintática** (no GALS, arquivo com extensão `.gals`);
  (4) demais recursos.
  Basta um integrante da equipe postar o trabalho. Caso não sejam postados todos os arquivos solicitados ou não seja possível rodar o **executável** (ou **.jar**), será atribuída nota 0.0 (zero) ao analisador sintático.
- Na avaliação do analisador sintático serão levadas em consideração: a correta especificação do arquivo `.gals`, conforme as correções/alterações do trabalho nº3; a qualidade das mensagens de erro, conforme descrito acima; o uso apropriado de ferramentas para construção de compiladores. Observa-se que <u>todas</u> as mensagens de erro sintático geradas pelo GALS devem ser alteradas conforme especificado anteriormente e mantidas em comentário de linha ao lado da mensagem alterada, caso contrário, será atribuída nota 0.0 (zero) ao analisador sintático.

DATA: entregar o trabalho até às 23h do dia 31/10/2025 (sexta-feira).

EXEMPLOS DE ENTRADA / SAÍDA

EXEMPLO 1: com erro léxico

ENTRADA	SAÍDA (na área de mensagens)
linha **1:** begin **2:** int lado; **3:** **4:** read ("digite um valor para lado: , lado **5:** print ("o valor digitado foi: ", lado); **6:** end	linha 4: constante_string inválida

EXEMPLO 2: com erro sintático

ENTRADA	SAÍDA (na área de mensagens)
linha **1:** begin **2:** int lado; **3:** **4:** read ("digite um valor para lado: ", lado **5:** print ("o valor digitado foi: ", lado); **6:** end	linha 5: encontrado print esperado ,)

EXEMPLO 3: sem erro

ENTRADA	SAÍDA (na área de mensagens)
linha **1:** begin **2:** int lado; **3:** **4:** read ("digite um valor para lado: ", lado); **5:** print ("o valor digitado foi: ", lado); **6:** end	programa compilado com sucesso