TRABALHO FINAL – parte 3: implementação do analisador sintático

Implementar o **analisador sintático** de forma que indique quais programas escritos na linguagem 2023.2 estão sintaticamente corretos, seguindo as orientações abaixo:
1º passo: efetue correções, se for o caso, na especificação da gramática, conforme solicitado/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:

  Erro na linha 1 – **encontrado** `EOF` **esperado** `expressão`
  Erro na linha 1 – **encontrado** `_iarea` **esperado** `(`

  Observa-se que:
  - quando for encontrado ou esperado fim de programa ou fim de arquivo ou `$`, a mensagem deve ser encontrado (ou esperado) `EOF`
  - quando for encontrada ou esperada uma palavra reservada, a mesma deve ser escrita da seguinte forma: `do else false fun if in main out repeat true while`, conforme o caso
  - quando for encontrado ou esperado um símbolo especial, o mesmo deve ser escrito da seguinte forma: `& | ! , ; = : ( ) { } == != < > + - * /`, conforme o caso
  - quando for encontrada ou esperada uma constante, a mesma deve ser escrita da seguinte forma: `constante_int constante_float constante_string`, conforme o caso
  - quando for encontrado um identificador, deve ser escrito o lexema
  - quando for esperado um identificador, deve ser escrito: `identificador`
  - para o não-terminal `<lista_de_expressoes>`, ou com outro nome, usado para definir essa estrutura sintática especificada no trabalho no.2, a mensagem deve ser do tipo: **encontrado** ... **esperado** `expressão`
  - para os não-terminais `<expressao>`, `<expressao_>`, `<elemento>`, `<relacional>`, `<relacional_>`, `<aritmetica>`, `<aritmética_>`, `<termo>`, `<termo_>` e `<fator>`, a mensagem deve ser do tipo: **encontrado** ... **esperado** `expressão`
  - para os demais não-terminais, a mensagem deve ser do tipo: **encontrado** ... **esperado** $símbolo_1$ $símbolo_2$ ... $símbolo_n$, conforme tabela de análise sintática, exemplificado a seguir;
  - são exemplos de mensagens de erro **inadequadas**: `<lista_comandos>` inválido, esperado `cte_int inesperado` ou `$ inesperado`
  - todas as mensagens de erro geradas pelo GALS devem ser mantidas (em comentário de linha), MAS devem ser alteradas, conforme especificado.

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

	id	fun	if	in	out	repeat	while	","	";"	"="	")"
<programa>	-	0	-	-	-	-	-	-	-	-	-
<lista_instrucoes>	1	-	1	1	1	1	1	-	-	-	-
<lista_identificadores>	17	-	-	-	-	-	-	-	-	-	-
<lista_identificadores_>	-	-	-	-	-	-	-	19	18	18	18

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

- para o não-terminal `<programa>`: **encontrado** ... **esperado** `fun`
- para o não-terminal `<lista_instrucoes>`: **encontrado** ... **esperado** `identificador if in out repeat while`
- para o não-terminal `<lista_identificadores>`: **encontrado** ... **esperado** `identificador`
- para o não-terminal `<lista_identificadores_>`: **encontrado** ... **esperado** `, ; = )`
- para o não-terminal `<lista_de_expressoes>`: **encontrado** ... **esperado** `expressão`
- para o não-terminal `<expressao>`: **encontrado** ... **esperado** `expressão`

- A gramática especificada no trabalho nº3 (com as devidas correções) deve ser usada para implementação do analisador sintático. Além disso, trabalhos desenvolvidos usando especificações diferentes daquelas elaboradas pela equipe no trabalho nº3 receberão nota 0.0 (zero).
- A implementação do analisador sintático, bem como da interface do compilador e do analisador léxico, deve ser disponibilizada no **AVA** na tarefa **"parte 3 - analisador sintático"**, aba **COMPILADOR**. Deve ser disponibilizado um **arquivo compactado** (com o nome: `sintatico`, seguido do número da equipe), contendo: o projeto completo, incluindo o **código fonte**, o **executável** e o **arquivo com as especificações léxica e sintática** (no GALS, arquivo com extensão `.gals`) e demais recursos. Basta um integrante da equipe postar o trabalho. Caso não sejam postados os arquivos solicitados, será atribuída nota 0.0 (zero).
- Na avaliação do analisador sintático serão levadas em consideração: a correta especificação da gramática, conforme 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.

DATA: entregar o trabalho até às 23h do dia 23/10/2023 (segunda-feira).

EXEMPLOS DE ENTRADA / SAÍDA

EXEMPLO 1: com erro léxico

ENTRADA	SAÍDA (na área de mensagens)
linha **1:** fun main { **2:** _ilado, _iarea; **3:** in ("digite um valor para lado: , _ilado **4:** _iarea = _ilado * _ilado; **5:** out (_iarea); }	Erro na linha 3 - constante_string inválida

EXEMPLO 2: com erro sintático

ENTRADA	SAÍDA (na área de mensagens)
linha **1:** fun main { **2:** _ilado, _iarea; **3:** in ("digite um valor para lado: ", _ilado **4:** _iarea = _ilado * _ilado; **5:** out (_iarea); }	Erro na linha 4 - encontrado _iarea esperado , ; = :)

EXEMPLO 3: sem erro

ENTRADA	SAÍDA (na área de mensagens)
linha **1:** fun main { **2:** _ilado, _iarea; **3:** in ("digite um valor para lado: ", _ilado); **4:** _iarea = _ilado * _ilado; **5:** out (_iarea); }	programa compilado com sucesso