Trabalho Compiladores
Analisador léxico para Linguagem cic_2023.2

1 Instruções Gerais

1.2 O trabalho pode ser desenvolvido em qualquer linguagem, desde que respeite o fluxo do autômato (AFD) gerado e que **cada processamento seja feito usando leitura do código fonte, caracter a caracter**, não palavras completas.

1.3 O trabalho deve ser entregue no Classroom até **03/10 às 23:59**

A entrega é constituída de: códigos/projeto fonte, programa executável, relatório, arquivos com casos de testes (entrada) e saídas.

1.4 O trabalho deve ser apresentado a mim nas datas **05 e 09/10**

1.5 Trabalhos não apresentados não recebem nota.

2. Especificação do trabalho

O trabalho consiste na implementação de um analisador léxico para a linguagem de programação fictícia cic_2023.2 (especificada na seção 3). O programa deve receber como entrada um arquivo de texto com extensão .cic contendo o programa fonte escrito na linguagem cic_2023.2.

O analisador léxico deve:

- Reconhecer e retornar o token correspondente para palavras reservadas, identificadores, operadores, delimitadores e constantes seguindo autômato determinístico (AFD).
- Informar possíveis erros e onde eles se encontram
- Ser implementado como uma sub-rotina (função ou método) que, a cada chamada retorna o próximo token da sequência no código fonte. Você deve simular todas chamadas até o fim do arquivo. A implementação dessa rotina simula o analisador sintático, que utiliza e solicita tokens ao analisador léxico.
- Imprimir uma série de relatórios no fim do programa

IMPORTANTE: O analisador léxico possuirá funcionalidades bem definidas e independentes umas das outras, como leitura do arquivo de entrada, reconhecimento de palavras reservadas, impressão de mensagens de erro e geração das saídas. Implemente essas funcionalidades distintas através de funções separadas e projete as passagens de parâmetros a fim de evitar ao máximo o uso de variáveis globais.

Seção 3 – Linguagem cic_2023.2

3.1 Tipos de dados

Há 3 tipos de dados em cic_2023.2: moeda, cadeia e número

O token número (**TK_NUMERO**) pode inteiro ou flutuante (seja **num** um TK_NUMERO inteiro, o número flutuante é composto por **num.num**) e pode ser ou não apresentado por notação científica (neste caso, seja **num_float** um TK_NUMERO com ponto flutuante, a notação científica é composta de: **numenum** ou **num_floatenum**). Números não levam sinal, sua negação é jeita com o operador unário -. Sinal apenas é permitido na segunda parte do número em notação científica (ex:

10e-3). Os números em cic_2023.2 são descritos em sua base hexadecimal, portanto seus dígitos são [0-9A-F]. Números deve ter pelo menos 1 caracter alfanumérico.
Exemplos: **15**, **A1**, **A3e9**, **D.25Fe-9D**
Não reconhecidos: **. ,** **a1,** **234.5ee3,** **.324,** **234. , -D.25Fe-9.D**

O token moeda (**TK_MOEDA**) é composto por uma letra maiúscula, cifrão ($), e número na base decimal com 2 casas após o ponto.
Exemplos: **R$21314.00, U$2.90, A$2.99**
Não reconhecidos: **R$21314.0000, U$2.9, A$2., $13, R$12**

O token cadeia (**TK_CADEIA**) é uma sequência de caracteres delimitada por aspas duplas e totalmente contida em uma única linha. Uma cadeia aberta em uma linha e fechada em outra deve gerar erro léxico.
Exemplos: **“-.15”** , **“cic 2023\n”** e **“”**.

3.2 Identificadores de nomes de variáveis
Identificadores de variáveis(TK_ID) são formados por sequência de letras minúsculas e dígitos, dentro dos limitadores: <>. Um identificador deve ter ao menos 1 letra e não deve começar com dígito.
Exemplo: **<var1>, <a>, <a8a8a8>**
Não reconhecidos: **a, <1var>, <1>**

3.3 Operadores:

	Unários		Binários					
Aritméticos e lógicos	-	~	+	-	*	/	&	|
Relacionais			!=	=	>=	<=	>	<
Atribuição			:=					

Operadores devem ter um token específico para cada.

3.4 Comentários
Reconhecer comentários é uma tarefa do analisador léxico. Ele teve ter em seu autômoto, rotinas para ler, porém não retornar nenhum token, apenas ignorar.
Comentários de linhas são começados em # e comentários de bloco começados e terminados por 3 aspas simples.

Em caso de comentários abertos e nunca fechados, o programa deve informar o erro. Para continuar o processamento, ignore apenas a primeira e recomece a análise léxica pela seguinte.

3.5 Delimitadores
Os delimitadores são: virgula, abertura e fechamento de parênteses.
Abertura e fechamento de parênteses devem ser reconhecido como delimitadores separadamente, isto é, um token para cada.

3.6 Palavras reservadas

São compostas por pelo menos 2 letras minúsculas. Cada palavra reservada deve ter seu token correspondente. Essa estrutura de reconhecimento deve estar implementada no seu autômato.

Você deverá retornar um token diferente para cada palavra diferente. Esta parte pode estar implementada a através de algum mapeamento no seu código, não precisa estar refletido no autômato. Isto é, você não precisa de um autômato do tipo:

e1 → i → e2 → f → acc
→ e → e3 → l → e4 → s → e5 → e → acc
(caso if e else sejam palavras reservadas na linguagem)

Mas precisa de um autômato mais genérico, do tipo:

e1 → letra → e2 → letra ... → acc

Onde, no estado de aceitação, a correspondência para o token correto deve ser feito.

Lista de palavras reservadas da linguagem: programa, fim_programa, se, senao, entao, imprima, leia, enquanto.

3.7 Exemplos escritos na linguagem cic_2023.2

```
# Ex-01-correto.cic

programa
    imprima("Digite um nro")
    leia(<numero>)
    <soma> := <a> + 8D.3
    imprima(<raiz>)
fim_programa
```

```
# Ex-02-incorreto.cic

programa
    imprima("Digite um
    nro")
    leia(<1num>)
    <soma> := <a> + R$23.3
    imprima(<raiz>)
fim_programa
```

```
''' Ex-03-incorreto.cic

Programa
    leia(<a>)
    <soma> := <a> + R$23.34
    se <a> <= <b> então
        imprima(34F.34e23A)
    imprima("Fim prog"rama")
fim_programa
```

4. Saídas

Para cada código fonte, além de retornar tokens a medida que for solicitado pelo analisador sintático (aqui neste trabalho meramente simulado por chamadas no programa principal), o analisador léxico deve também informar a presença de erros e, ao final da execução, deve gerar alguns relatórios. Tais relatórios são saídas textuais e devem estar bem formatadas e fáceis de ler. Não é necessário usar qualquer outra forma de interface gráfica.

4.1 Código fonte com identificação de erro

Código fonte com linhas numeradas, mensagem e marcação (linha, coluna e apontamento) do erro, quando houver.
Exemplo:

```
[ 1] # Ex-02-incorreto.cic
[ 2]
[ 3] programa
[ 4] imprima("Digite um
     ------------------^
     Erro léxico na linha 4 coluna 19:
     Cadeia não fechada
[ 5]          nro")
     --------^
     -----------^
     Erro léxico na linha 5 coluna 8:
     Palavra reservada não encontrada
     Erro léxico na linha 5 coluna 12:
     Cadeia não fechada
[ 6] leia(<1num>)
     -----^
     Erro léxico na linha 6 coluna 7:
     Nome de variável mal formatado
[ 7] <soma> := <a> + R$23.3
     -----------------^
     Erro léxico na linha 7 coluna 18:
     Moeda mal formatada
[ 8] imprima(<raiz>)
[ 9] fim_programa
```

As mensagens e formatos podem ser ligeiramente modifcadas, desde que contenha os elementos obrigatórios descritos acima.

4.2 Lista de tokens reconhecidos

Deve ser apresentado em ordem de ocorrência no arquivo fonte. Apresentar sua linha, coluna, token e lexema. O Lexema só deve ser preenchido para tokens que permitirem diferentes lexemas. Para tokens com lexemas únicos, mostrar apenas linha e coluna da ocorrência.

LIN	COL	TOKEN	LEXEMA
3	1	TK_PROGRAMA	
4	1	TK_IMPRIMA	
	8	TK_AB_PAREN	
5	13	TK_FE_PAREN	
6	1	TK_LEIA	
	5	TK_AB_PAREN	
	12	TK_FE_PAREN	
7	1	TK_IDENT	<soma>
	8	TK_ATRIB	
	11	TK_IDENT	<a>

...

4.3 Somatório de tokens reconhecimentos
Ordene por quantidade de usos.
Veja exemplo para o código fonte `Ex-02-incorreto.cic`

```
+-------------------+--------+
| TOKEN             | Usos   |
+-------------------+--------+
|TK_IDENT           |      3 |
+-------------------+--------+
|TK_PROGRAMA        |      1 |
+-------------------+--------+
|TK_FIM_PROGRAMA    |      1 |
+-------------------+--------+
|TK_MOEDA           |      1 |
+-------------------+--------+
... (completar todos os tokens)
+-------------------+--------+
| TOTAL             |     16 |
+-------------------+--------+
```

5. Apresentação e Relatório
Deve ser entregue em formato pdf
Deve link acessível para o código do programa e imagem do autômato
Deve conter as expressões regulares para cada token e o autômato (um único autômato para toda a linguagem) usado como guia para desenvolver o analisador léxico, representado de duas maneiras: a) desenho do grafo, com estados e transições e b) tabela de transição.
Deve conter a entrada e saída 3 casos de testes fornecidos com essa especificação.

A apresentação será em sala de aula, com duração máxima de 10 minutos. O aluno deve apresentar apenas a mim, não para a sala completa. O aluno deverá executar o programa com algum caso de teste já conhecido por ele, e também um outro caso de teste inédito fornecido por mim na hora.

6. Critérios de avaliação
Os pesos ou penalidades para cada critério ainda serão definidos.

Corretude do autômato

Abordagem para leitura do código fonte:
- Navegação em arquivo fonte
- Navegação por estrutura computacional como array, listas, matriz e etc…
- Leitura e comparação de palavras completas **(penalidade alta)**

Estrutura do código
- Modularizado e comentado
- Segue fluxo do autômato

Saídas
- Informação de erros
- Lista de tokens
- Sumário de tokens

Reconhecimentos corretos:
- Numeros
- Moedas
- identificadores
- Cadeias
- Comentários
- Demais

Relatório
- deve conter resumo, autômato, codigo fonte, entradas e saídas