Trabalho Compiladores

Analisador léxico para Linguagem cic_2024.1

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 em etapas**, de acordo com datas no classroom.

   A entrega final 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 em uma das datas: **25/04, 29/04 e 02/05**

   A divisão ocorrerá em sala de aula

   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_2024.1 (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_2024.1.

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_2024.1

3.1 Tipos de dados

Há 5 tipos de dados em cic_2024.1: data, cadeia, endereço, inteiro e ponto flutuante

3.1.1 Int

O token inteiro (**TK_INT**) é composto por um ou mais dígitos e é um número não negativo, isto é, não possui sinal.

Exemplos:

15, 5, 0, 2314535452542

3.1.2 Float

O token de ponto flutuante (**TK_FLOAT**) possui obrigatoriamente um ponto (.), só pode ter até 3 dígitos antes do ponto (.).

Para expressar um valor maior, deve ser escrito em notação científica. Na notação científica, o sinal positivo deve ser omitido.

Um ponto sozinho, não representa um número inteiro.

O ponto flutuante não possui sinal.

Exemplos:

.123, 0.123, 23.23, 134.5, 123., 123.212314, 123.13141e10, 1.0e-12, .9e-12

Não reconhecidos:

., -10.1, 12345.0, 12314., 1.0e+12

3.1.3 Data

O token data (**TK_DATA**) é composto por 3 partes, dia, mês e ano, separados por / ou -. Para dia e mês, usar 2 dígitos, para ano, 4 dígitos.

O separados deve ser o mesmo em todas as partes de uma data.

Exemplos:

11/11/2000, 11-11-2000, 99/99/9999

Não reconhecidos:

11-11/2000, 11/11-2000, 01/01/22, 1/01/2000

3.1.4 Endereço

O token endereço (**TK_END**) Possui um dígito hexadecimal (0-9A-F), o caractere “x” e um ou mais dígitos hexadecimais (0-9A-F).

Exemplo:

0x123, 0x0, 3x3F, FxAAAAAAAAA, 3x3

Não reconhecidos:

00x34, GxF, 3x3f, x434

3.1.5 Cadeia

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 ou funções

Identificadores de variáveis (**TK_ID**) são formados por sequência de letras minúsculas e maiúsculas alternadas, iniciando por minúscula. Um identificador deve ter ao menos 2 letras. Não é permitido dígitos.

Exemplo: **vArIaVeI, nUmErO**

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 deve ter em seu autômato, rotinas para ler, porém não retornar nenhum token, apenas ignorar.

Comentários de linhas são começados em # e comentários de bloco começados por <<< e terminados por >>>.

Em caso de comentários abertos e nunca fechados, o programa deve informar o erro.

3.5 Outros

Delimitadores são: dois pontos, abertura e fechamento de parênteses.

Abertura e fechamento de parênteses devem ser reconhecidos 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.

Você deverá retornar um token diferente para cada palavra diferente. Esta parte pode estar implementada 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 para reconhecer a palavra reservada if

→e → e3 → l → e4 → s → e5→ e → acc para reconhecer a palavra reservada else

(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 feita.

Lista de palavras reservadas da linguagem: rotina, fim_rotina, se, senao, imprima, leia, para, enquanto.

3.7 Exemplos escritos na linguagem cic_2024.1

```
# Ex-01-correto.cic

rotina
        imprima("Digite um nro")
        leia(nUmErO)
        nUm <== nUm <> 123
        imprima(123.e10)
fim_rotina
```

```
# Ex-02-incorreto.cic

rotina
        imprima("Digite um
    nro")
        NuM <== 11/11/2000 + 11-11-20
        imprima(Ax23423)
fim_rotina
```

```
<<< Ex-03-incorreto.cic

rotina
      leia(<a>)
>>>
      enquanto nUm <= .09e-123:
         imprima(99/12-1999)
   imprima("Fim prog"rama")
fim_rotina
```

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] rotina
[ 4]    imprima("Digite um
     --------------------^
     Erro linha 4 coluna 22: Cadeia não fechada
[ 5]     nro")
     ----^
     -------^
     Erro linha 5 coluna 5: palavra reservada nao encontrada
     Erro linha 5 coluna 8: Cadeia não fechada
[ 6]    NuM <== 11/11/2000 + 11-11-20
     ---^
     ------------------------^
     Erro linha 6 coluna 3
     Erro linha 6 coluna 25: data mal formatada
[ 7]    imprima(Ax23423)
[ 8] fim_rotina
```

As mensagens e formatos podem ser ligeiramente modificados, desde que contenham 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_ROTINA	
4	4	TK_IMPRIMA	
	11	TK_ABRE_PAR	
5	9	TK_FECHA_PAR	
6	8	TK_ATRIB	
	12	TK_DATA	11/11/2000
	23	TK_MAIS	
7	4	TK_IMPRIMA	
	11	TK_ABRE_PAR	
	12	TK_END	Ax23423
	19	TK_FECHA_PAR	
8	1	TK_FIN_ROTINA	

4.3 Somatório de tokens reconhecimentos

Ordene por quantidade de usos.

Veja exemplo de parte dos tokens para o código fonte Ex-02-incorreto.cic

TOKEN	USOS
TK_IMPRIMA	2
TK_ABRE_PAR	2
TK_FECHA_PAR	2
TK_ROTINA	1

...

5. Apresentação, Entrega e Relatório

1ª etapa: Entrega do autômato e expressão regular que reconheça todos os elementos da linguagem. Para cada estado de aceitação, anote qual token será retornado. (15% da nota)

Atenção para que a imagem tenha símbolos legíveis!

2ª etapa: Escolher pelo menos 2 dentre os elementos de tipo de dados para ser implementado e reconhecido pelo analisador léxico. Deve ser entregue o código, devidamente comentado, e exemplos de testes (entradas e saídas do token reconhecido).
Nesta etapa, não é necessário que reconheça uma sequência de palavras/tokens. Todos os testes podem ser feitos com apenas uma palavra. (15% da nota)

3ª etapa:

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:

- Números

- Endereços

- Datas

- Identificadores

- Cadeias

- Comentários

- Demais

Relatório

- deve conter resumo, autômato, código fonte, entradas e saídas