Implementação de um analisador léxico

Atividade I

Universidade Federal do ABC

Atualização

- 26/06:
  - adicionado comentário sobre limite de tamanho de literal de uma string;
  - corrigido exemplo de erro;
  - corrigido caso de teste 1;
- 17/06:
  - alteração na ER do número de ponto flutuante, de acordo com o conversado em sala;
  - atualização de caso de testes (baixe o zip novamente);
  - adicionada *dica* sobre identificação de literais booleanos e palavras reservadas;

Descrição

Na primeira atividade você deve desenvolver um analisador léxico para Javalette[1]. Esta linguagem pode ser vista como um subconjunto de C.

Veja um exemplo de código nessa linguagem (espaços em branco realçados para facilitar entendimento):

```
1  int␣main␣(␣)␣{
2  ␣␣␣␣int␣i␣=␣0␣;
3  ␣␣␣␣while␣(␣i␣<␣10)␣{
4  ␣␣␣␣␣␣␣␣if␣(␣i␣%␣2␣==␣0)␣printInt(␣i␣)␣;
5  ␣␣␣␣␣␣␣␣i++␣;
6  ␣␣␣␣}
7  ␣␣␣␣return␣0␣;
8  }
```

Nesta etapa, seu programa receberá (como argumento pela linha de comando) o caminho de um arquivo com um código-fonte e deverá imprimir na saída padrão uma tabela com um token por linha e 4 colunas (separadas por @) contendo: tipo do token, lexema, linha e coluna.

Os tipos de tokens e suas respectivas expressões regulares são:

Token	ER
INTEIRO	[0-9]+
PONTO_FLUTUANTE	([0-9]*.[0-9]+([eE][+-]?[0-9]+)?|[0-9]+.[0-9]*)
STRING	“[^"\n]*”
BOOL	(true|false)
IDENTIFICADOR	[a-zA-Z][a-zA-Z0-9_]*
PALAVRA_RESERVADA	(while|if|else|return|int|bool|double|void)
PONTO_VIRGULA	;
ABRE_PARENTESES	(
FECHA_PARENTESES	)
ABRE_CHAVES	{

[1] http://www.cse.chalmers.se/edu/course/TDA283_Compiler_Construction/project/#javalette

Token	ER
FECHA_CHAVES	}
VIRGULA	,
ATRIBUICAO	=
OPERADOR	(&&|||+|-|*|/|%)
OPERADOR_RELACIONAL	(==|<=|>=|!=|<|>)
INCREMENTO	(++|--)

Note que a definição de uma string não engloba strings que contenham aspas (mesmo se escapadas) e não permite strings em múltiplas linhas, ou seja, caracteres de quebra de linha dentro da string. Seu léxico pode considerar que um literal string poderá ter no máximo 200 caracteres. Como delimitador você deve considerar qualquer caractere equivalente a um espaço em branco, ou seja, espaços/quebras de linha/tabs. Além disso, você deve **ignorar comentários** de uma linha (estilo C++ //) e de múltiplas linhas (estilo C /* */). Cada identificador pode ter no máximo **20 caracteres**. Note que palavras reservadas e literais de booleanos são identificadores válidos. Considere a estratégia discutida em sala de aula em que esses tokens são inicialmente classificados como identificadores e posteriormente (i.e., no código) avaliados em relação a uma tabela pré-definida.

Para o programa exemplo, a saída desejada é (note que o @ está suprimido devido ao formato de tabela para facilitar a visualização):

TOKEN	LEXEMA	LINHA	COLUNA
PALAVRA_RESERVADA	int	1	1
IDENTIFICADOR	main	1	5
ABRE_PARENTESES	(	1	10
FECHA_PARENTESES	)	1	11
ABRE_CHAVES	{	1	13
PALAVRA_RESERVADA	int	2	5
IDENTIFICADOR	i	2	9
ATRIBUICAO	=	2	11
INTEIRO	0	2	13
PONTO_VIRGULA	;	2	15
PALAVRA_RESERVADA	while	3	5
ABRE_PARENTESES	(	3	11
IDENTIFICADOR	i	3	12
OPERADOR_RELACIONAL	<	3	14
INTEIRO	10	3	16
FECHA_PARENTESES	)	3	18
ABRE_CHAVES	{	3	20
PALAVRA_RESERVADA	if	4	9
ABRE_PARENTESES	(	4	12
IDENTIFICADOR	i	4	13
OPERADOR	%	4	15
INTEIRO	2	4	17
OPERADOR_RELACIONAL	==	4	19
INTEIRO	0	4	22
FECHA_PARENTESES	)	4	23
IDENTIFICADOR	printInt	4	25
ABRE_PARENTESES	(	4	33
IDENTIFICADOR	i	4	34
FECHA_PARENTESES	)	4	35
PONTO_VIRGULA	;	4	37
IDENTIFICADOR	i	5	9

TOKEN	LEXEMA	LINHA	COLUNA
INCREMENTO	++	5	10
PONTO_VIRGULA	;	5	13
FECHA_CHAVES	}	6	5
PALAVRA_RESERVADA	return	7	5
INTEIRO	0	7	12
PONTO_VIRGULA	;	7	14
FECHA_CHAVES	}	8	1

Note que é esperado o cabeçalho e uma quebra de linha também no último pacote.

No caso da identificação de um erro em um lexema, você deverá separar o caractere ilegal em outro pacote com o token do tipo ERRO, reconhecendo as cadeias anteriores e posteriores à ele. Por exemplo, considere o seguinte código:

```
1  in#t␣main␣()␣{}
```

A saída esperada para esse código é:

TOKEN	LEXEMA	LINHA	COLUNA
IDENTIFICADOR	in	1	1
ERRO	#	1	3
IDENTIFICADOR	t	1	4
IDENTIFICADOR	main	1	6
ABRE_PARENTESES	(	1	11
FECHA_PARENTESES	)	1	12
ABRE_CHAVES	{	1	14
FECHA_CHAVES	}	1	15

Em texto puro (sem suprimir os @), o resultado seria:

```
TOKEN@LEXEMA@LINHA@COLUNA
IDENTIFICADOR@in@1@1@
ERRO@#@1@3@
IDENTIFICADOR@t@1@4@
IDENTIFICADOR@main@1@6@
ABRE_PARENTESES@(@1@11@
FECHA_PARENTESES@)@1@12@
ABRE_CHAVES@{@1@14@
FECHA_CHAVES@}@1@15@
```

Seu léxico deve ser feito em linguagem C. A equipe de desenvolvimento deverá ter no máximo 3 integrantes. Em conjunto com o código, deverá ser entregue um relatório de 2 páginas (no máximo) descrevendo a abordagem, o autômato finito determinístico correspondente e as principais funções implementadas. O trabalho deverá ser entregue até o dia **30/06** . O uso de geradores de analisadores léxico (Flex, JFlex etc) está **proibido**. No entanto, o aluno é encorajado a conhecer essas ferramentas para entender seu funcionamento. A entrega deve ser feita via **TIDIA**, sendo a submissão composta por um arquivo PDF e um arquivo C.

Seu código deve compilar sem apresentar nenhum **erro** ou **aviso** com a seguinte linha de comando:

```
gcc -o lex ARQUIVO.c -std=c99 -Wall -g
```