Universidade Estadual de Londrina

DC Departamento de Computação

5COP093 - Compilador C: Analisador Léxico

Utilizando a ferramenta Flex, implemente um analisador léxico para um **subconjunto** de *tokens* da linguagem C. Para cada *token* reconhecido, você deverá imprimir a classificação do mesmo. A classificação dos *tokens* que você deve seguir encontra-se a seguir:

```
Identificadores: IDENTIFIER
Inteiros: NUM_INTEGER
Octais:   NUM_OCTAL
Hexadecimal: NUM_HEXA
String:  STRING
Caracter: CHARACTER

void     VOID
int      INT
char     CHAR
return   RETURN
break    BREAK
switch   SWITCH
case     CASE
default  DEFAULT
do       DO
while    WHILE
for      FOR
if       IF
else     ELSE
typedef  TYPEDEF
struct   STRUCT
+        PLUS
-        MINUS
*        MULTIPLY
/        DIV
%        REMAINDER
++       INC
--       DEC
&        BITWISE_AND
|        BITWISE_OR
~        BITWISE_NOT
^        BITWISE_XOR
!        NOT
&&       LOGICAL_AND
```

```
||        LOGICAL_OR
==        EQUAL
!=        NOT_EQUAL
<         LESS_THAN
>         GREATER_THAN
<=        LESS_EQUAL
>=        GREATER_EQUAL
>>        R_SHIFT
<<        L_SHIFT
=         ASSIGN
+=        ADD_ASSIGN
-=        MINUS_ASSIGN
;         SEMICOLON
,         COMMA
:         COLON
(         L_PAREN
)         R_PAREN
{         L_CURLY_BRACKET
}         R_CURLY_BRACKET
[         L_SQUARE_BRACKET
]         R_SQUARE_BRACKET
?         TERNARY_CONDITIONAL
#         NUMBER_SIGN
->        POINTER
printf    PRINTF
scanf     SCANF
define    DEFINE
exit      EXIT
```

Identificadores: os identificadores podem ser iniciados com letras maiúsculas e minúsculas e o caractere *underscore*. A partir do segundo símbolo, números de 0 até 9 também podem aparecer na formação do identificador.

Devem ser removidos:

```
Espaços em branco
Comentários de uma linha:  (//)
Comentários de múltiplas linhas:  (/* ...  */)
```

Também devem ser detectados comentários de múltiplas linhas que são iniciados e não estão finalizados. Os *tokens* reconhecidos devem ser impressos um por linha. Quando um erro for detectado, deve-se mostrar a linha e a coluna onde o erro ocorreu.

Exemplo de entrada:

```
if(@)
{
   a
}
```

Saída esperada:

```
IF
L_PAREN
error:lexical:1:4: @
R_PAREN
L_CURLY_BRACKET
IDENTIFIER(a)
R_CURLY_BRACKET
```

Observe que na saída esperada, a mensagem de erro apresentada foi:

```
error:lexical:1:4: @
```

onde o primeiro número indica a linha onde o erro ocorreu e o segundo número indica a coluna onde o erro ocorreu. No exemplo dado, o erro ocorreu na linha 1, coluna 4. Observe também que deve ser impresso o caractere que causou o erro léxico. Mesmo quando ocorrerem erros, o processo de reconhecimento dos *tokens* não para, continuando até que se atinja o fim do arquivo de entrada.

Observe o exemplo a seguir:

```
if(1)
{
/* isto eh um
comentario iniciado
e nao terminado
```

Saída esperada:

```
IF
L_PAREN
NUM_INTEGER(1)
R_PAREN
L_CURLY_BRACKET
error:lexical:3:1: unterminated comment
```

No exemplo apresentado, o arquivo termina com um comentário de bloco que não foi fechado. Tal tipo de erro deve ser apresentado com a mensagem padrão do exemplo. Observe também que mesmo o arquivo contendo 3 linhas de comentário de bloco não terminado, a mensagem de erro irá mostrar a linha onde o comentário se inicia. No caso do exemplo, o comentário se inicia na linha 3. Mesmo que o comentário possua inúmeras linhas, o erro deve apontar a linha onde o comentário é iniciado. Tal como outros erros léxicos, também deve-se informar a coluna onde o erro ocorreu. No exemplo apresentado, o erro ocorreu na coluna 1.

A seguir é apresentado um exemplo maior de arquivo de entrada, bem como sua respectiva saída.

Entrada:

```
if(@)
{
   a;
   printf("Adeus mundo \"cruel!\"")
   //Oi, eu sou um comentario de linha
   a->[666] += 0x34 + 07 << !2;
     "picanha" != '@' + de_boi;
   /* oi, eu sou um
      comentario de bloco */
   for(;;){$} +45=-78,0X78+07;
}
comment++;/*Isto eh um
comentario sem fim...
```

Saída esperada:

```
IF
L_PAREN
error:lexical:1:4: @
R_PAREN
L_CURLY_BRACKET
IDENTIFIER(a)
SEMICOLON
PRINTF
L_PAREN
STRING(Adeus mundo \"cruel!\")
R_PAREN
IDENTIFIER(a)
POINTER
L_SQUARE_BRACKET
NUM_INTEGER(666)
R_SQUARE_BRACKET
ADD_ASSIGN
NUM_HEXA(0x34)
PLUS
NUM_OCTAL(07)
L_SHIFT
NOT
NUM_INTEGER(2)
SEMICOLON
STRING(picanha)
NOT_EQUAL
CHARACTER(@)
PLUS
IDENTIFIER(de_boi)
SEMICOLON
FOR
```

```
L_PAREN
SEMICOLON
SEMICOLON
R_PAREN
L_CURLY_BRACKET
error:lexical:10:12: $
R_CURLY_BRACKET
PLUS
NUM_INTEGER(45)
ASSIGN
MINUS
NUM_INTEGER(78)
COMMA
NUM_HEXA(0X78)
PLUS
NUM_OCTAL(07)
SEMICOLON
R_CURLY_BRACKET
IDENTIFIER(comment)
INC
SEMICOLON
error:lexical:12:11: unterminated comment
```

Especificações de Entrega

O trabalho deve ser entregue no *AVA* em um arquivo com o nome `lexico.l`.

A entrega deve ser feita exclusivamente no *AVA* até a data/hora especificada. Não serão aceitas entregas atrasadas ou por outro meio que não seja o *AVA*.

Os arquivos entregues serão compilados da seguinte forma:

```
$flex ./lexico.l
$gcc lex.yy.c -o lexico
```

Desta forma, certifique-se que o seu código pode ser compilado/executado corretamente com os comandos apresentados.

O programa gerado deve ler as suas entradas da entrada padrão do sistema e imprimir as saídas na saída padrão do sistema. Um exemplo de execução para uma entrada chamada `teste.c` seria a seguinte:

```
$./lexico < teste.c
```

IMPORTANTE: Arquivos ou programas entregues fora do padrão receberão nota **ZERO**. Entende-se como arquivo fora do padrão aquele que tenha um nome diferente de `lexico.l`. Entende-se como programa fora do padrão aquele que apresentar erro de compilação, que não ler da entrada padrão, não imprimir na saída padrão, por exemplo. Uma forma de verificar se seu arquivo ou programa está dentro das especificações é testar o mesmo com o `script` de testes que é fornecido no *AVA*. Se o seu arquivo/programa **não** funcionar com o `script`, significa que ele está **fora** das especificações e, portanto, receberá nota **ZERO**.