Centro Universitário de Belo Horizonte – UniBH
UC: Teoria da Computação e Compiladores
Professor: Gustavo Alves Fernandes

Exercício Prático I – Análise Léxica

Nesta etapa, você deverá implementar um analisador léxico para a linguagem ***Mini-Cond***, cuja descrição encontra-se nas próximas páginas. Seu analisador léxico deverá ser implementado conforme visto em sala de aula, com o auxílio de um Autômato Finito Determinístico (disponível em anexo). Ele deverá reconhecer um lexema e retornar, a cada chamada, um *token* de acordo com o lexema encontrado.

Para facilitar o exercício, o Lexer está implementado em linguagem Python. Na implementação há: 1) uma Tabela de Símbolos (TS). Essa tabela contém, inicialmente, **todas as palavras reservadas** da linguagem. À medida que novos *tokens* (Identificadores) forem sendo reconhecidos, esses deverão ser consultados na TS antes de serem cadastrados. **Somente palavras reservadas e identificadores serão cadastrados na TS**. Não é permitido o cadastro de um mesmo *token* mais de uma vez na TS. 2) a classe Token, que caracteriza um *token*.

O Lexer imprime a lista de todos os *tokens* reconhecidos, assim como imprime o que está cadastrado na Tabela de Símbolos. A impressão dos *tokens* é formatada como: **`<nome_do_token, lexema, linha, coluna>.`**

Além de reconhecer os *tokens* da linguagem, o Lexer detecta possíveis erros e os reporta ao usuário. O programa informa o erro e o local onde ocorreu (linha e coluna). Porém a contagem de linha e coluna não foi implementada. Você deverá implementar essa *feature*. Os erros verificados são: i) caracteres não esperados em um padrão ou inválidos. Se o analisador léxico encontrar mais do que um erro léxico, o processo de compilação é interrompido.

Espaços em branco, tabulações, quebras de linhas e comentários não são *tokens*, ou seja, são descartados/ignorados pelo referido analisador.

O que entregar?

Você deverá entregar nesta etapa:

- Um vídeo comentando a sua implementação e mostrando testes comprovando a implementação.
- Submeter seu código fonte para apreciação do professor.

Para avaliar a correção, o programa deverá exibir os *tokens* reconhecidos e o local de sua ocorrência, os *tokens* armazenados na tabela de símbolos, bem como os erros léxicos gerados, se acontecerem.

Regras:

- O trabalho poderá ser realizado individualmente, em dupla ou trio.
- Não é permitido o uso de ferramentas para geração do analisador léxico.
- A implementação poderá ser realizada em uma das linguagens C, C++, C#, Java, Ruby, Python, Javascript ou PHP.
- Trabalhos total ou parcialmente iguais receberão avaliação nula.
- Se o analisador léxico não compilar ou apresentar erros de execução durante a fase de teste realizada pelo professor, a avaliação será nula.
- Após a data definida para entrega, nenhum trabalho será recebido.

A linguagem *Mini-Cond*

```
PROGRAMA  → CMD EOF
CMD       → if E then { CMD } |
            if E then { CMD } else { CMD } |
            print T; | ATRIB CMD
ATRIB     → id = T;
E         → T OP T | T
OP        → < | <= | == | != | > | >=
T         → id | num
```

Padrões:

```
id: [A – Za – z]([A – Za – z] | [0 – 9])*
num: [0 – 9]+
```

Os demais nomes de token são apresentados no código.

Atenção: EOF significa “Fim de Arquivo”. A identificação de fim de arquivo já está implementada.

Exemplo: Programa válido para essa linguagem:

```
var1 = 10;
var2 = 10;
if var1 != 10 then {
     print var1;
}
else {
     if 10 == var2 then {
          print var2;
     }
     else {
          print 10;
     }
}
```