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

Trabalho Prático I – Análise Léxica

Descrição do trabalho

Nesta etapa, você deverá implementar um analisador léxico para a linguagem ***Mini Logo***, 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. Ele deverá reconhecer um lexema e retornar, a cada chamada, um *token* de acordo com o lexema encontrado.

Para facilitar a implementação, uma Tabela de Símbolos (TS) deverá ser usada. Essa tabela conterá, 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.

Seu Analisador Léxico deverá imprimir a lista de todos os *tokens* reconhecidos, assim como imprimir o que está cadastrado na Tabela de Símbolos. A impressão dos *tokens* deverá ser formatada como: **`<nome_do_token, lexema, linha, coluna>.`**

Além de reconhecer os *tokens* da linguagem, seu analisador léxico deverá detectar possíveis erros e reportá-los ao usuário. O programa deverá informar o erro e o local onde ocorreu (linha e coluna). Os erros verificados serão: i) caracteres desconhecidos (não esperados em um padrão ou inválidos), ii) *string* não fechada antes de quebra de linha e iii) *string* não-fechada antes do fim de arquivo. Se o analisador léxico encontrar mais do que um erro léxico, o processo de compilação deve ser interrompido.

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

Pontos extras (3 pontos):

É possível implementar um processo de recuperação de erro, para que o analisador léxico possa verificar a maior quantidade de código possível. Para isso o Modo Pânico deverá ser implementado na recuperação de erros, conforme visto em aula. Porém, se o número de erros ultrapassar o limite de 5 erros, o programa deverá abortar a análise.

Cronograma e Valor

O trabalho vale 30 pontos no total. Ele deverá ser entregue por etapas, pelo ulife, sendo a primeira etapa correspondendo ao Analisador Léxico, conforme consta na tabela abaixo.

Etapa	**Data de entrega**	**Valor**
Analisador Léxico	05/06/2022	10 pontos
Analisador Sintático	A definir	10 pontos
Analisador Semântico	A definir	10 pontos

O que entregar?

Você deverá entregar nesta etapa: Programa com todos os arquivos fonte.

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 ou em dupla.
- 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 testes realizada pelo professor junto ao aluno, a avaliação será nula.
- Após a data definida para entrega, nenhum trabalho será recebido/avaliado.

Descrição dos Padrões de Formatação

Os padrões de formação das constantes ‘number’, ‘literal’ e dos identificadores ‘id’ da linguagem são descritos abaixo:
- ***id***: deve iniciar com uma letra seguida de 0 ou mais produções de letras e/ou dígitos.
- ***number***: cadeia numérica contendo 1 ou mais produções de dígitos.
- ***literal (string)***: deve iniciar e finalizar com o caractere aspas duplas (") contendo entre eles uma sequência de 1 ou mais produções de letras, dígitos e/ou símbolos da tabela ASCII – exceto o próprio caractere aspas.
- ***EOF*** é o código de fim de arquivo.
- Aspas simples na gramática apenas destacam os terminais e os diferencia dos não-terminais, isto é, **não** fazem parte da gramática.
- Apenas comentários de uma linha são permitidos e são identificados por qualquer quantidade de caracteres entre um par de chaves ‘{‘ e ‘}’.

A linguagem *Mini Logo*

```
Program             → 'program' literal Decl Block EOF
Decl                → ':'id ';' Decl | ε
Block               → 'begin' StatementList 'end'
StatementList       →  Statement ';' StatementList | Statement
Statement           → 'turn' term 'degrees' | 'forward' term |
                      'repeat' term 'do' Block | 'print' literal |
                      AssignmentStatement | IfStatement
AssignmentStatement → ':'id Expr
IfStatement         → 'if' Expr 'do' Block
Expr                → Expr '+' Expr |  Expr '-' Expr |
                      Expr '*' Expr | Expr '/' Expr | term
term                → number | ':'id
```

Exemplo de programa em Mini Logo:

```
program "HelloMiniLogo"
:var;
begin
  print "movendo para frente";
 :var 10;
  forward :var;
  print "iniciando loop";
  repeat 3 do
  begin
    turn 90 degrees;
    forward var
  end;
  print "fim"
end
```