Centro Universitário de Belo Horizonte – Uni-BH
Instituto de Engenharia e Tecnologia
Curso: Ciência da Computação
Disciplina: Compiladores
Professor: Moisés Henrique Ramos Pereira

Trabalho Prático I – Análise Léxica

Descrição do trabalho

Nesta etapa, você deverá implementar um analisador léxico para a linguagem *Pyscal* cuja descrição encontra-se na próxima página.

Seu analisador léxico deverá ser implementado conforme visto em sala de aula, com o auxílio de um autômato finito. Ele deverá reconhecer um terminal e retornar, a cada chamada, um objeto da classe *Token*, representando o token reconhecido e seu lexema (quando necessário).

Para facilitar a implementação, uma tabela de símbolos deverá ser usada. Essa tabela conterá, inicialmente, todas as palavras reservadas da linguagem. À medida que novos tokens ID forem sendo reconhecidos, esses deverão ser consultados na tabela de símbolos antes de serem retornados.

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), lembrando que em análise léxica tem-se 3 tipos de erros: caracteres desconhecidos, string não-fechada antes de quebra de linha e comentário não-fechado antes do fim de arquivo.

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

Na especificação da linguagem, os terminais de um lexema, bem como as palavras reservadas, estão entre aspas apenas como ilustração, ou seja, as aspas não fazem parte do terminal: a especificação de uma real String contendo aspas está no padrão de formatação do terminal ConstString.

O que entregar?

Você deverá entregar nesta etapa:

1. Autômato Finito Determinístico para reconhecimento dos tokens;

2. Programa com todos os arquivos fonte (será apresentado em data previamente marcada);

3. Testes realizados com programas corretos e errados (no mínimo, 3 certos e 3 errados).

Para avaliar a correção, o programa deverá exibir os tokens reconhecidos e o local de sua ocorrência, bem como os erros léxicos gerados.

A linguagem *Pyscal*

Programa → Classe EOF
Classe → "class" ID ":" ListaFuncao Main "end" "."
DeclaraID → TipoMacro ID ";"
ListaFuncao → ListaFuncao Funcao | λ
Funcao → "def" TipoMacro ID "(" ListaArg ")" ":" (DeclaraID)∗ ListaCmd Retorno "end" ";"
ListaArg → Arg "," ListaArg | Arg
Arg → TipoMacro ID
Retorno → "return" Expressao ";" | λ
Main → "defstatic" "void" "main" "(" "String" "[" "]" ID ")" ":" (DeclaraID)∗ ListaCmd "end" ";"
TipoMacro → TipoPrimitivo "[" "]" | TipoPrimitivo
TipoPrimitivo → "bool" | "integer" | "String" | "double" | "void"
ListaCmd → ListaCmd Cmd | λ
Cmd → CmdIF | CmdWhile | CmdAtribui | CmdFuncao | CmdWrite | CmdWriteln
CmdIF → "if" "(" Expressao ")" ":" ListaCmd "end" ";"
| "if" "(" Expressao ")" ":" ListaCmd "else" ListaCmd "end" ";"
CmdWhile → "while" "(" Expressao ")" ":" ListaCmd "end" ";"
CmdWrite → "write" "(" Expressao ")" ";"
CmdWriteln → "writeln" "(" Expressao ")" ";"
CmdAtribui → ID "=" Expressao ";"
| ID "[" Expressao "]" "=" Expressao ";"
CmdFuncao → ID "(" (Expressao ("," Expressao)∗)? ")" ";"
Expressao → Expressao Op Expressao
| ID "[" Expressao"]" | ID
| ID "(" (Expressao ("," Expressao)∗)? ")"
| ConstInteger | ConstDouble | ConstString | "true" | "false"
| "vector" TipoPrimitivo "[" Expressao "]"
| OpUnario Expressao | "(" Expressao")"
Op → "or" | "and" | "<" | "<=" | ">" | ">=" | "==" | "!=" | "/" | "*" | "-" | "+"
OpUnario → "-" | "!"

Descrição dos Padrões de Formatação

Os padrões de formação das constantes e dos identificadores da linguagem são descritos abaixo:

- ID: deve iniciar com uma letra seguida de 0 ou mais produções de letras, dígitos e/ou caracteres _.
- ConstInteger: cadeia numérica contendo 1 ou mais produções de dígitos.
- ConstDouble cadeia numérica contendo 1 ou mais produções de dígitos, tendo em seguida um símbolo de ponto antes de 1 ou mais produções de dígitos.
- ConstString: deve iniciar e finalizar com o caractere " (aspas) contendo entre esses uma sequência de 0 ou mais produções de letras, dígitos e/ou símbolos.
- EOF é o código de fim de arquivo.

A leitura deve ser feita com as funções "integer_input()", "double_input()", "string_input()". Os símbolos referidos na constante String (ConstString) indica qualquer caractere imprimível ASCII.
Os comentários de bloco e de linha seguem o padrão de Java.