

Trabalho Prático I – Análise Léxica e Tabela de Símbolos

Descrição do trabalho

Nesta etapa, você deverá implementar um analisador léxico para a linguagem *Pyscal* cuja descrição encontra-se na página 3.

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.

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 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), 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, ou seja, as aspas não fazem parte do terminal: a especificação de uma String contendo aspas está no padrão de formatação do terminal ConstString.

Cronograma e Valor

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

Etapa	Data de entrega	Valor	Multa por atraso
Analisador Léxico e Tabela de símbolos	11/04/2017	10 pontos	2pts/dia
Analisador Sintático	A definir	10 pontos	2pts/dia
Analisador Semântico	A definir	10 pontos	2pts/dia

O que entregar?

Você deverá entregar nesta etapa:

1. Autômato Finito Determinístico para reconhecimento dos tokens;
2. Todos os arquivos fonte;
3. Relatório técnico contendo explicações do propósito de todas as classes, métodos ou funções da implementação, assim como testes realizados com programas corretos e errados (no mínimo, 3 certos e 3 errados). Os resultados deverão apresentar a saída do Analisador Léxico (a sequência de tokens identificados e o local de sua ocorrência) correspondente, os símbolos instalados na Tabela de Símbolos, bem como os erros léxicos encontrados.

Regras:

O trabalho poderá ser realizado individualmente ou em dupla.

Não é permitido o uso de ferramentas para geração do analisador léxico.

A implementação deverá ser realizada em uma das linguagens C, C++, Java ou Python.

A recuperação de erro deverá ser em Modo Pânico, conforme discutido em sala. Mensagens de erros correspondentes devem ser apresentadas, indicando a linha de ocorrência do erro.

Trabalhos total ou parcialmente iguais receberão avaliação nula.

Ultrapassados cinco (5) dias, após a data definida para entrega, nenhum trabalho será recebido.

Gramática da linguagem *Pyscal*

Programa → Classe **EOF**

Classe → "class" ID ":" ListaFuncao Main "end" "."

DeclararID → TipoMacro ID ";"

ListaFuncao → ListaFuncao Funcao | ϵ

Funcao → "def" TipoMacro ID "(" ListaArg ")" ":" (DeclararID)* ListaCmd Retorno "end" ";"

ListaArg → Arg "," ListaArg | Arg | ϵ

Arg → TipoMacro ID

Retorno → "return" Expressao ";" | ϵ

Main → "defstatic" "void" "main" "(" "String" "[" "]" ID ")" ":" (DeclararID)* ListaCmd "end" ";"

TipoMacro → TipoPrimitivo "[" "]" | TipoPrimitivo

TipoPrimitivo → "bool" | "integer" | "String" | "double" | "void"

ListaCmd → ListaCmd Cmd | ϵ

Cmd → CmdIF | CmdWhile | CmdAtribui | CmdFuncao | CmdWrite | CmdWriteln

CmdIF → "if" "(" Expressao ")" ":" ListaCmd "end" ";"
| "if" "(" Expressao ")" ":" ListaCmd "eles" ":" 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 que representa fim de arquivo.

Os símbolos referidos na constante String (ConstString) indica qualquer caractere imprimível ASCII. Os comentários de bloco e de linha seguem o padrão de Java.