Centro Universitário de Belo Horizonte – Uni-BH
Departamento de Ciências Exatas 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 *JaCa+* 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 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 reconhecidos, esses deverão ser incluídos 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 seu local (linha e coluna).

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

O que entregar?

Você deverá entregar nesta etapa:

1. Autômato 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 *JaCa+*

Programa → Classe EOF
Classe → "public" "class" ID "{" ListaMetodo Main "}"
DeclaracaoVar → Tipo ID ";"
ListaMetodo → ListaMetodo Metodo | λ
Metodo → Tipo ID "(" ListaParam ")" "{" (DeclaracaoVar)∗ ListaCmd Retorno }
ListaParam → Param ", " ListaParam | Param | λ
Param → Tipo ID
Retorno → "return" Expressao ";" | λ
Main → "public" "void" "main" "(" "String" ID "[" "]" ")" "{" (DeclaracaoVar)∗ ListaCmd "}"
Tipo → TipoPrimitivo"[" "]" | TipoPrimitivo
TipoPrimitivo → "boolean" | "int" | "String" | "double" | "void"
ListaCmd → ListaCmd Cmd | λ
Cmd → "{" ListaCmd "}"
| CmdIF | CmdWhile | CmdAtrib | CmdFunc | CmdCout | CmdCoutEndl
CmdIF → "if" "(" Expressao ")" Cmd
| "if" "(" Expressao ")" Cmd "else" Cmd
CmdWhile → "while" "(" Expressao ")" Cmd
CmdCout → "cout" "<<" Expressao ";"
CmdCoutEndl → "cout" "<<" Expressao "<<" "endl" ";"
CmdAtrib → ID "=" Expressao ";"
| ID "[" Expressao "]" "=" Expressao";"
CmdFunc → ID "(" (Expressao ("," Expressao)∗)? ")"
Expressao → Expressao Op Expressao
| ID "[" Expressao "]" | ID
| ID "(" (Expressao ("," Expressao)∗)? ")"
| ConstInteira | ConstReal | ConstString | "true" | "false"
| "new" TipoPrimitivo "[" Expressao "]"
| OpUnario Expressao | "(" Expressao")"
Op → "||" | "&&" | "<" | "<=" | ">" | ">=" | "==" | "!=" | "/" | "*" | "-" | "+"
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 _.
- ConstInteira: cadeia numérica contendo 1 ou mais produções de dígitos.
- ConstReal: cadeia numérica contendo 1 ou mais produções de dígitos, tendo em seguida um símbolo de ponto à frente de 0 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.

A leitura deve ser feita com as funções "cinInt()", "cinDouble()", "cinString()", "cinBoolean()".
Os símbolos referidos na constante String (ConstString) indica qualquer caractere da tabela ASCII.
Os comentários seguem o padrão de Java.