Centro Universitário de Belo Horizonte – Uni-BH
Curso: Ciência da Computação
Disciplina: Compiladores
Professor: Gustavo Alves Fernandes

Trabalho Prático I – Análise Léxica e Tabela de Símbolos

Descrição do trabalho

Nesta etapa, você deverá implementar um analisador léxico para a linguagem ***PasC*** cuja descrição encontra-se nas páginas 3 e 4.

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 objeto da classe *Token*, representando o token reconhecido 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 forem sendo reconhecidos, esses deverão ser consultados na TS antes de serem cadastrados e retornados. Somente palavras reservadas e identificadores serão cadastrados na TS. Não é permitido o cadastro de um mesmo token mais de uma vez na TS.

Resumindo, seu Analisador Léxico deverá imprimir a lista de todos os tokens reconhecidos, assim como mostrar o que está cadastrado na Tabela de Símbolos. Na impressão dos tokens, deverá aparecer a tupla <nome, lexema> assim como linha e coluna do token.

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 (não esperados ou inválidos), 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 gramática do *PasC*, os terminais de um lexema, bem como as palavras reservadas, estão entre aspas duplas para destacá-los, ou seja, **as aspas não são tokens**.

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	06/04/2017	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. Uma figura apresentando o Autômato Finito Determinístico para reconhecimento dos tokens, conforme visto em sala de aula (dê uma olhada na ferramenta JFLAP: http://www.jflap.org/);

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 programas testes deverão ser definidos de acordo com a gramática do *PasC*. Os resultados deverão apresentar a saída do Analisador Léxico (a sequência de tokens identificados e o local de sua ocorrência) e 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++, C#, Java, Ruby 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 e coluna 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.

Em anexo (pasta: *lexer_exemplo*) segue um exemplo de uma Gramática, AFD, programas de exemplo, e a saída dos Tokens. **ATENÇÂO**: a gramática do exemplo não tem relação com a gramática do *PasC*.

Gramática da linguagem *PasC*

prog	→ “program” “id” body
body	→ decl-list “{“ stmt-list “}”
decl-list	→ decl “;” decl-list | ε
decl	→ type id-list
type	→ “num” | “char”
id-list	→ “id” | “id” “,” id-list
stmt-list	→ stmt “;” stmt-list | ε
stmt	→ assign-stmt | if-stmt | while-stmt | read-stmt | write-stmt
assign-stmt	→ “id” “=” simple_expr
if-stmt	→ “if” “(“ condition “)” “{“ stmt-list “}” | “if” “(“ condition “)” “{“ stmt-list “}” “else” “{“ stmt-list “}”
condition	→ expression
while-stmt	→ stmt-prefix “{“ stmt-list “}”
stmt-prefix	→ “while” “(“ condition “)”
read-stmt	→ “read” “id”
write-stmt	→ “write” writable
writable	→ simple-expr | “literal”
expression	→ simple-expr | simple-expr relop simple-expr
simple-expr	→ term | simple-expr addop term
term	→ factor-a | term mulop factor-a
factor-a	→ factor | “not” factor
factor	→ “id” | constant | “(“ expression “)”
relop	→ “==” | “>” | “>=” | “<” | “<=” | “!=”
addop	→ “+” | “-” | “or”
mulop	→ “*” | “/” | “and”
constant	→ “num_const” | “char_const”

Padrões para números, caracteres, literais e identificadores do *PasC*

digit	= [0-9]
letter	= [A-Z | a-z]
id	= letter (letter | digit)*
literal	= pelo menos um dos 256 caracteres do conjunto ASCII entre aspas duplas
char_const	= um dos 256 caracteres do conjunto ASCII entre aspas simples
num_const	= digit$^+$ (“.” digit$^+$)?

Nomes para os tokens:
Operadores:

OP_EQ: ==	OP_GE: >=	OP_MUL: *
OP_NE: !=	OP_LE: <=	OP_DIV: /
OP_GT: >	OP_AD: +	OP_ASS: =
OP_LT: <	OP_MIN: -	

Símbolos:

SMB_OBC: {	SMB_COM: ,
SMB_CBC: }	SMB_SEM: ;
SMB_OPA: (	
SMB_CPA:)	

Palavras-chave: KW: program, if, else, while, write, read, num, char, not, or, and

Identificadores: ID

Literal: LIT

Constantes: CON_NUM: num_const e CON_CHAR: char_const

Outras características de *PasC*

- As palavras-chave de *PasC* são reservadas;
- Toda variável deve ser declarada antes do seu uso;
- A linguagem possui comentários de mais de uma linha. Um comentário começa com "/*" e
- deve terminar com "*/";
- A linguagem possui comentários de uma linha. Um comentário começa com "//";
- A semântica dos demais comandos e expressões é a tradicional do Pascal, exceto que "=" é utilizado no comando de atribuição, "==" é operador relacional que verifica se os operandos são iguais, e "!=" é operador relacional que verifica se os operandos são diferentes;
- Os tipos numeral e caractere não são compatíveis;
- A linguagem não é *case-sensitive;*
- Cada tabulação, deverá contar como 3 espaços em branco;