UNIVERSIDADE CATÓLICA DOM BOSCO – UCDB

Engenharia de Computação – 7º semestre
Segundo Trabalho de Compiladores I
Prof. Marcelo Silva Cintra
Primeiro Semestre de 2007

2 Analisador Sintático

O objetivo do trabalho é construir um analisador sintático para a linguagem Pascal. A descrição da linguagem a ser reconhecida pelo analisador é dada na seção 2.3.

Para o reconhecimento da linguagem, deve ser construído um analisador sintático descendente recursivo sem recuperação de erros. Quando um erro for detectado, o analisador deve emitir uma mensagem de erro explicativa, mostrando o tipo de erro detectado e a linha que o erro ocorreu e terminar a execução do programa.

O trabalho deverá ser feito em grupo de no máximo 2 integrantes. O grupo, obrigatoriamente, deverá ser o mesmo do trabalho anterior. Somente quem optar em não continuar trabalhando em grupo na segunda fase poderá desenvolver o trabalho individualmente. Esta opção implica no desenvolvimento individual até o fim do semestre. Qualquer tentativa de fraude detectada será punida com a nota zero 0 (zero).

2.1 Prazo para entrega

O trabalho deverá ser apresentado na aula do dia 11/06/2007.

2.2 Avaliação

A avaliação será feita mediante entrevista com o grupo e entrega do projeto ao professor.

2.3 Descrição da Linguagem Pascal

Na descrição a seguir, os terminais estão em negrito. Os terminais que são símbolos especiais (+, -, *, ...) estão em negrito e tamanho 14. Os símbolos não terminais estão em itálico. As formas sentenciais que estiverem entre { } significa zero ou mais ocorrências da forma sentencial. Seja a e b $\in$ (N U T)*, (a | b) significa a ocorrência de a ou de b. As formas sentenciais que estiverem entre colchetes [ ] podem ser opcionais, ou seja, indicam uma ou nenhuma ocorrência da forma.

Atenção, qualquer erro que for identificado na gramática favor reportar para o professor. Revisões da gramática contarão com o conceito C do cálculo da média.

program ::= **program identifier** [**(** *identifier_list* **)**] **;** *block* **.**

block ::= { [*label_declaration_part*]
[*const_declaration_part*]
[*type_declaration_part*]
[*var_declaration_part*]
[*subroutine_declaration_part*] }
compound_statement

label_declaration_part ::= **label number** { **,** *number* } **;**

const_declaration_part ::= **const** *const_definition* { **;** *const_definition* } **;**

const_definition ::= **identifier** **=** *const*

type_declaration_part ::= **type** *type_definition* { **;** *type_definition* } **;**

type_definition ::= **identifier** **=** *type*

type ::= **^** **identifier**
| **array** **[** *simple_type* { **,** *simple_type* } **]** **of** *type*
| **set of** *simple_type*
| **record** *field_list* **end**
| *simple_type*

simple_type ::= **identifier**
| **(** **identifier** { **,** *identifier* } **)**
| *const* **..** *const*

const ::= **string**
| [**+** | **-**] **identifier**
| [**+** | **-**] **number**

field_list ::= [*identifier_list* **:** *type*] { **;** [*identifier_list* **:** *type*] }

var_declaration_part ::= **var** *var_declaration* { **;** *var_declaration*} **;**

var_declaration ::= *identifier_list* **:** *type*

identifier_list ::= **identifier** { **,** **identifier** }

subroutine_declaration_part ::= { *procedure_declaration* **;** | *function_declaration* **;** }

procedure_declaration ::= **procedure identifier** [*formal_parameters*] **;** *block*

function_declaration ::= **function identifier** [*formal_parameters*] **:** **identifier** **;** *block*

formal_parameters ::= **(** *param_section* { **;** *param_section* } **)**

param_section ::= [**var**] *identifier_list* **:** **identifier**
| **function** *identifier_list* **:** **identifier**
| **procedure** identifier_list

compound_statement ::= **begin** *labeled_statement* { **;** *labeled_statement* } **end**

labeled_statement ::= [**number** **:**] *statement*

statement ::= *assign_statement*
| *procedure_call*
| *if_statement*
| *case_statement*
| *while_statement*
| *repeat_statement*
| *for_statement*
| *with_statement*
| *goto_statement*
| *compound_statement*
| ε

assign_statement ::= **identifier** [*infipo*] := *expr*

procedure_call ::= **identifier** [**(** *expr_list* **)**]

if_statement ::= **if** *expr* **then** *statement* [**else** *statement*]

while_statement ::= **while** *expr* **do** *statement*

repeat_statement ::= **repeat** *statement* { **;** *statement* } **until** *expr*

for_statement ::= **for identifier** *infipo* **:=** *expr* (**to** | **downto**) *expr* **do** *statement*

with_statement ::= **with identifier** *infipo* { **,** **identifier** *infipo* } **do** *statement*

case_statement ::= **case** *expr* **of** *case* { **;** *case* } **end**

case ::= (**number** | **identifier**) **:** *statement*

goto_statement ::= **goto number**

infipo ::= **[** *expr* { **,** *expr* } **]** *infipo*
| **.** **identifier** *infipo*
| ^ *infipo*
| ε

expr_list ::= *expr* { **,** *expr* }

expr ::= *simple_expr* [*relop simple_expr*]

relop ::= = | < | > | <> | >= | <= | **in**

simple_expr ::= [+|-] *term* { *addop term* }

addop ::= **+** | **-** | **or**

term ::= *factor* { *mulop factor* }

mulop ::= ***** | **/** | **div** | **mod** | **and**

factor ::= **identifier** *infipo*
| **number**
| **string**
| **identifier** [**(** *expr_list* **)**]
| **(** *expr* **)**
| **not** *factor*