UFMG — ICEX — DCC
Departamento de Ciência da Computação
Compiladores
Profa. Mariza Andrade da Silva Bigonha
Trabalho Prático - Valor: 40 pontos
Data de Entrega Final: 19 de junho de 2018
Trabalho em dupla

---

1 Compilador para a linguagem *Simple*

Considere a *Simple* definida pela seguinte gramática:

program	::=	**program** identifier ";" decl_list compound_stmt
decl_list	::=	decl_list " ;" decl
	|	decl
decl	::=	dcl_var
	|	dcl_proc
dcl_var	::=	ident_list ":" type
ident_list	::=	ident_list " ," identifier
	|	identifier
type	::=	**integer**
	|	**real**
	|	**boolean**
	|	**char**
dcl_proc	::=	tipo_retornado PROCEDURE identifier espec_parametros corpo
tipo_retornado	::=	**integer**
	|	**real**
	|	**boolean**
	|	**char**
	|	vazio
corpo	::=	":" decl_list ";" compound_stmt id_return
	|	vazio
id_return	::=	identifier
	|	vazio
espec_parametros	::=	"(" lista_parametros ")"
lista_de_parametros	::=	parametro
	|	lista_de_parametros , parametro
parametro	::=	modo type : identifier
modo	::=	**value**
	|	**reference**
compound_stmt	::=	**begin** stmt_list **end**
stmt_list	::=	stmt_list ";" stmt
	|	stmt
stmt	::=	assign_stmt
	|	if_stmt
	|	repeat_stmt
	|	read_stmt
	|	write_stmt

	|	compound_stmt
	|	function_ref_par
assign_stmt	::=	identifier " :=" expr
if_stmt	::=	**if** cond **then** stmt
	|	**if** cond **then** stmt **else** stmt
cond	::=	expr
repeat_stmt	::=	**repeat** stmt_list **until** expr
read_stmt	::=	**read** "(" ident_list ")"
write_stmt	::=	**write** "(" expr_list ")"
expr_list	::=	expr
	|	expr_list " ," expr
expr	::=	*Simple*_expr
	|	*Simple*_expr RELOP *Simple*_expr
*Simple*_expr	::=	term
	|	*Simple*_expr ADDOP term
term	::=	factor_a
	|	term MULOP factor_a
factor_a	::=	"−" factor
	|	factor
factor	::=	identifier
	|	constant
	|	" (" expr ")"
	|	NOT factor
	|	function_ref_par
function_ref_par	::=	variable " (" expr_list ")"
variable	::=	*Simple*_variable_or_proc
*Simple*_variable_or_proc	::=	identifier
constant	::=	integer_constant
	|	real_constant
	|	char_constant
	|	boolean_constant
boolean_constant	:=	**false** | **true**

Considere as seguintes convenções léxicas:

1. Identificadores são definidos pelas seguintes expressões regulares:

letter	::=	A | B | ··· Z
	|	a | b | ··· z
digit	::=	**0** | **2** | **3** | **4** |**5** | **6** | **7** | **8** | **9**
identifier	::=	letter (letter | digit)⋆

Na implementação pode-se limitar o tamanho do identificador.

2. Constantes são definidas da seguinte forma:

$$
\begin{array}{rcl}
\text{unsigned\_integer} & ::= & \text{digit} \quad \text{digit}\star \\
\text{sign} & ::= & + \mid - \mid \mathcal{E} \\
\text{scale\_factor} & := & \texttt{"E"} \quad \text{sign} \quad \text{unsigned\_integer} \\
\text{unsigned\_real} & ::= & \text{unsigned\_integer} \quad (\ \mathcal{E} \ \mid \ \texttt{"."} \quad \text{digit}\star)(\ \mathcal{E} \ \mid \ \text{scale\_factor}) \\
\text{integer\_constant} & ::= & \text{unsigned\_integer} \\
\text{real\_constant} & ::= & \text{unsigned\_real} \\
\text{char\_constant} & ::= & \texttt{"'"} \quad \text{caractereASCII} \quad \texttt{"'"}
\end{array}
$$

3. Os operadores de relação (RELOP's) são:

$$\text{RELOP} \quad ::= \quad = \mid < \mid \leq \mid > \mid \geq \mid != $$

4. Os operadores de adição (ADDOP's) são:

$$\text{ADDOP} \quad ::= \quad + \mid - \mid \textbf{or}$$

5. Os operadores (MULOP's) são:

$$\text{MULOP} \quad ::= \quad \star \mid / \mid \textbf{div} \mid \textbf{mod} \mid \textbf{and}$$

6. A linguagem *Simple* é um subconjunto bastante reduzido das linguagens imperativas mais comuns. A sua semântica supomos ser óbvia, mas os casos de dúvida prevalesce a semântica do Pascal por ser mais *Simple*s.

RESULTADOS DESEJADOS

A sua tarefa é construir um **compilador** para a *Simple* e um interpretador de quádruplas (ou usar o interpretador TAM disponível na página do curso), de forma a ser possível executar programas em *Simple*. O compilador deve gerar código para a linguagem intermediária de quádruplas, cujas instruções disponíveis devem ser definidas conforme a necessidade.

Além da listagem do programa, exibindo compilação e execução de exemplos, a documentação do projeto também deverá ser produzida. Esta documentação deverá, **no mínimo**, incluir o seguinte:

Análise Léxica:
definição dos tokens e estruturas de dados usadas.

Análise Sintática:
Comentário sobre o método escolhido e as modificações efetuadas na gramática dada,se houverem, visando a eliminação de conflitos e ambiguidades. Comente também sobre o que foi feito sobre a recuperação de erros.

Com o objetivo de simplificar o seu trabalho, não é exigido que se faça "recuperação de erros" , isto é, erros sintáticos podem ser considerados fatais. Entretanto, a mensagem de erro correspondente deve ser expressa.

Erros decorrentes do mal uso de tipos, inconsistência do número de parâmetros em uma chamada de procedimento e uso de variáveis não declaradas **não** interferem com o funcionamento do analisador sintático. Erros deste tipo não devem descontinuar a compilação.

Tabela de Símbolos:
Organização, método de acesso e atributos dos símbolos.

Front-End:
Construção de um **front-end** para a linguagem **Simple** definida, gerando código intermediário, incluindo a descrição da estrutura de dados; a definição dos atributos usados e especificação das rotinas semânticas implementadas. As implementações devem ser obrigatoriamente em Java ou C++.

Tradutor:
Construção de um **Tradutor** da linguagem intermediária gerada pelo *front-end* de **Simple** para o código da máquina virtual **TAM**, de forma a ser possível compilar e executar programas em *Simple*. O interpretador das instruções de **TAM** está disponível na página desse Curso.

Saída do Interpretador:
Compilador Integrado. Listagem de programas exemplos, da tabela de símbolos, do código gerado (quádruplas) e do resultado da execução do programa.