Especificação Fase 2

Bacharelado em Ciência da Computação
Universidade Federal de São Carlos
Campus Sorocaba
Compiladores

08 de Abril de 2015

1 Segunda fase

A segunda fase do trabalho consiste na implementação de um compilador que realiza a análise léxica, sintática e semântica da gramática da linguagem, além de gerar código em linguagem C.

2 Entrega

Data: **05/05/15**
Você deve entregar todos os arquivos `.java` com a mesma estrutura do Compilador 9, isto é, com o pacote AST, Lexer com as análises léxica, sintática e emissão de erros separadas. No diretório tests, você deve enviar todos os testes realizados.

3 Gramática

Esta seção define a gramática da linguagem. Ao longo de todas as fases deste trabalho, a gramática utilizada será uma versão simplificada da linguagem de programação Pascal, baseada em [1].

Em cada linha há a descrição de uma regra de produção da gramática. As palavras-chave da linguagem estão em letras maiúsculas. Os símbolos da linguagem são mostrados entre apóstrofos. Os símbolos não-terminais da gramática são aqueles descritos por palavras em letras minúsculas.

Uma sequência de símbolos entre `[` e `]` é opcional, enquanto que uma sequência de símbolos entre `{` e `}` pode ser repetida zero ou mais vezes. Qualquer sequência de caracteres no arquivo fonte encontrado entre `{` e `}` deve ser tratado como um comentário.

É importante notar que o símbolo `.` presente na quarta regra de produção do símbolo não-terminal `factor` representa uma *string* de tamanho variável. Essa *string* pode conter qualquer caractere menos o apóstrofo, pois este é um símbolo da linguagem.

```
prog          ::= PROGRAM pid ';' body '.'

body          ::= [dclpart] compstmt

dclpart       ::= VAR dcls

dcls          ::= dcl {dcl}

dcl           ::= idlist ':'  type ';'

idlist        ::= id {',' id}

type          ::= stdtype | arraytype

stdtype       ::= INTEGER | REAL | CHAR | STRING

arraytype     ::= ARRAY '[' intnum '..'  intnum ']' OF stdtype

compstmt      ::= BEGIN stmts END

stmts         ::= stmt {';' stmt} ';'

stmt          ::= ifstmt
```

```
                | whilestmt
                | assignstmt
                | compstmt
                | readstmt
                | writestmt
                | writelnstmt
ifstmt        ::= IF expr THEN stmts [ELSE stmts] ENDIF
whilestmt     ::= WHILE expr DO stmts ENDWHILE
assignstmt    ::= vbl ':=' expr
readstmt      ::= READ '(' vblist ')'
writestmt     ::= WRITE '(' exprlist ')'
writelnstmt   ::= WRITELN '(' [exprlist] ')'
vblist        ::= vbl {',' vbl}
vbl           ::= id ['[' expr ']']
exprlist      ::= expr {',' expr}
expr          ::= simexp [relop expr]
simexp        ::= [unary] term {addop term}
term          ::= factor {mulop factor}
factor        ::= vbl
                | num
                | '(' expr ')'
                | ''.''
id            ::= letter {letter | digit}
pid           ::= letter {letter | digit}
num           ::= ['+' | '-'] digit ['.']  {digit}
intnum        ::= digit {digit}
relop         ::= '=' | '<' | '>' | '<=' | '>=' | '<>'
addop         ::= '+' | '-' | OR
mulop         ::= '*' | '/' | AND | MOD | DIV
unary         ::= '+' | '-' | NOT
```

4 Instruções

O seu compilador deve se basear no Compilador 9 da apostila. Isto é, a partir de um programa descrito na gramática acima, o seu compilador deve fazer a análise léxica, sintática e semântica e gerar código em linguagem C. Observe que seu programa deve:

- gerar uma mensagem de erro no caso de uma situação que o compilador não esperava (erro léxico, sintático ou semântico presente no código fonte);
- caso contrário, gerar um arquivo de saída de extensão `.c` com o código em C equivalente ao código do arquivo de entrada.

Abaixo são apresentados três exemplos com erros e um sem erros. Depois de cada exemplo é descrito o erro (quando há) ou mostrada a conversão correta para a linguagem C.

```
PROGRAM exemplo01ErroLexico;
    VAR a, b : INTEGER;
    BEGIN
        IF a > b THEN
            WRITE'a eh maior que b'); { # }
        ENDIF;
    END.
```

Erro: após o comando `write` o símbolo `(` era esperado.

```
PROGRAM exemplo02ErroSintatico;
    VAR a, b : INTEGER;
    BEGIN
        IF a > b THEN
            WRITE('a eh maior que b');
        END; { # }
    END.
```

Erro: uso de `end` ao invés de `endif`.

```
PROGRAM exemplo03ErroSemantico;
    VAR a, b : INTEGER;
    BEGIN
        IF a > c THEN { # }
            WRITE('a eh maior que b');
        ENDIF;
    END.
```

Erro: a variável `c` não foi declarada.

```
PROGRAM exemplo04Correto;
    VAR a, b : INTEGER;
    BEGIN
        IF ( a > b ) THEN
            WRITE('a eh maior que b');
        ENDIF;
    END.
```

Exemplo sem erros. Um compilador correto deve gerar o seguinte código em C:

```
#include <stdio.h>

int main() {
    int a, b;
```

```
    if ( a > b ) {
        printf("a eh maior que b");
    }

    return 0;
}
```

Referências

[1] Alfred Aho, Monica Lam, Ravi Sethi e Jeffrey Ullman. *Compilers: Principles, Techniques and Tools.* Addison-Wesley, Technical, 1986.