Faculdade de Ciências Exatas e da Engenharia

Teoria das Linguagens e Compiladores

(2015/2016)

Enunciado do Projecto

1. Objectivos

O objectivo do projecto é realizar um *front-end* para uma linguagem, que efectue a análise lexical e sintáctica da referida linguagem.

2. A Notação BNF (Backus Naur Form)

A notação BNF foi introduzida por John Backus para descrever a sintaxe da linguagem ALGOL 58, sendo posteriormente modificada por Peter Naur na definição da sintaxe da linguagem ALGOL 60. Esta notação, após sofrer algumas alterações e extensões, possui uma grande aceitação e tornou-se na mais utilizada na descrição da sintaxe das linguagens de programação.

O BNF e as suas extensões podem ser descritas da seguinte forma:

- Os meta símbolos do BNF são os seguintes:

  `::=` significa "é definido como"
  `|` significa "ou"
  `<>` são utilizados para definir os nomes das regras sintácticas

- As partes opcionais são incluídas entre os meta símbolos **`[`** e **`]`**, exemplo:

```
<IfStatement> ::= if <BooleanExpression> then <StatementSequence>
                  [ else <StatementSequence> ]
                  endif
```

- As partes repetitivas (zero ou mais vezes) são incluídas entre os meta símbolos **`{`** e **`}`**, exemplo:

```
<Identifier> ::= <Letter> { <Letter> | <Digit> }
```

- Os símbolos terminais de um único caracter são incluídos entre aspas (") para os distinguir dos meta símbolos.

- Nalguns dos textos mais recentes os símbolos terminais são escritos a **bold**, eliminando-se a utilização dos meta-símbolos **<>** para definir os nomes das regras sintácticas (também designados por símbolos não terminais). Por exemplo:

```
IfStatement ::= if BooleanExpression then StatementSequence
                [ else StatementSequence ]
                endif
```

A definição da notação BNF utilizando a própria notação BNF é a seguinte:

```
Syntax       ::= { Rule }
Rule         ::= Identifier "::=" Expression
Expression   ::= Term { "|" Term }
Term         ::= Factor { Factor }
Factor       ::= Identifier         |
                 QuotedSymbol       |
                 "(" Expression ")" |
                 "[" Expression "]" |
                 "{" Expression "}"
Identifier   ::= Letter { Letter | Digit }
Letter       ::= "a" | "b" | ... | "z" |
                 "A" | "B" | ... | "Z"
Digit        ::= "0" | "1" | ... | "9"
QuotedSymbol ::= """ Caracter """
```

(Nota: O símbolo com três pontos "..." não pertence à notação BNF, sendo unicamente utilizado por questões de economia de espaço.)

3. O BNF da Linguagem do projecto

Em seguida, apresenta-se o BNF da linguagem:

```
Module      ::= MODULE ident ";" [ ImportList ] {Definition} {DeclSeq} Body
                ident "." .

ImportList  ::= IMPORT ident [ ":=" ident ] {"," ident [ ":=" ident ]} ";".

Definition  ::= DEFINITION ident [ DEFINES Qualident ]
                {PROCEDURE ident [ FormalPars ] ";"} END ident.

DeclSeq     ::= CONST {ConstDecl ";"}
                | TYPE {TypeDecl ";"}
                | VAR {VarDecl ";"}
                | {ProcDecl ";"}.

ConstDecl   ::= IdentDef "=" ConstExpr.

TypeDecl    ::= IdentDef "=" Type.

VarDecl     ::= IdentList ":" Type.
```

```
ProcDecl     ::= PROCEDURE ProcHead ";" {DeclSeq} Body ident.

ProcHead     ::= [ SysFlag ] [ "*" | "&" ] IdentDef [ FormalPars ].

SysFlag      ::= "[" ident "]".

FormalPars   ::= "(" [ FPSection {";" FPSection} ] ")" [ ":" Qualident ].

FPSection    ::= [ VAR ] ident {"," ident} ":" Type.

Type         ::= Qualident
                 | RECORD [ SysFlag ] [ "(" Qualident ")" ] [ FieldList ] END
                 | POINTER [ SysFlag ] TO Type
                 | OBJECT [ SysFlag ] [ "(" Qualident ")" ] [ IMPLEMENTS Qualident ]
                 {DeclSeq} Body
                 | PROCEDURE [ SysFlag ] [ FormalPars ].

FieldDecl    ::= [ IdentList ":" Type ].

FieldList    ::= FieldDecl {";" FieldDecl}.

Body         ::= StatBlock | END.

StatBlock    ::= BEGIN [ "{" IdentList "}" ] [ StatSeq ] END.

StatSeq      ::= Statement {";" Statement}.

Statement    ::= [ Designator ":=" Expr
                 | Designator
                 | IF Expr THEN StatSeq {ELSIF Expr THEN StatSeq}[ELSE StatSeq]
                 END
                 | CASE Expr THEN Case {"|" Case} [ELSE StatSeq] END
                 | WHILE Expr DO StatSeq END
                 | FOR ident ":=" Expr TO Expr [BY ConstExpr] DO StatSeq END
                 | LOOP StatSeq END
                 | WITH Qualident ":" Qualident DO StatSeq END
                 | EXIT
                 | RETURN [Expr]
                 | AWAIT "(" Expr ")"
                 | StatBlock ].

Case         ::= [ CaseLabels { "," CaseLabels } ":" StatSeq ].

CaseLabels   ::= ConstExpr [ ".." ConstExpr ].

ConstExpr    ::= Expr.

Expr         ::= SimpleExpr [ Relation SimpleExpr ].

SimpleExpr   ::= Term {MulOp Term}.
```

```
Term        ::= [ "+"|"-" ] Factor {AddOp Factor}.

Factor      ::= Designator
              | number
              | string
              | NIL
              | Set
              | "(" Expr ")"
              | "~" Factor.

Set         ::= "{" [Element {"," Element}] "}".

Element     ::= Expr [ ".." Expr ].

Relation    ::= "=" | "#" | "<" | "<=" | ">" | ">=" | IN | IS.

MulOp       ::= "*" | DIV | MOD | "/" | "&" .

AddOp       ::= "+" | "-" | OR .

Designator  ::= ident { "." ident | "[" ExprList "]" | "^" | "(" ExprList ")" }.

ExprList    ::= Expr {"," Expr}.

IdentList   ::= IdentDef {"," IdentDef}.

Qualident   ::= [ ident "." ] ident.

IdentDef    ::= ident ["*"|"-"].

Identifier  ::= Letter {Letter | Digit | "_"}.

Letter      ::= "A" |"B" | … | "Z" | "a" | "b" | …"z".

String      ::= """{Character}""".

Number      ::= Integer | Real.

Integer     ::= Digit {Digit}| Digit {HexDigit} "H".

Real        ::= Digit {Digit} "." {Digit} [ScaleFactor].

ScaleFactor ::= ("E"|"D") [ "+"|"-" ] Digit {Digit}.

HexDigit    ::= Digit | "A" |"B" | "C" | "D" |"E" | "F".

Digit       ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
```

4. A Linguagem

A linguagem deverá ser estendida. A sintaxe pretendida é ilustrada através dos seguintes exemplos:

Exemplo1:

```
MODULE Example;
IMPORT Commands, Files;
PROCEDURE CreateFile*(context : Commands.Context);
VAR
filename : Files.FileName;
file : Files.File; writer : Files.Writer;
ch : CHAR;
BEGIN
IF context.arg.GetString(filename) THEN
file := Files.New(filename);
IF (file # NIL) THEN
Files.OpenWriter(writer, file, 0);
context.arg.Char(ch);
REPEAT
context.arg.Char(ch);
writer.Char(ch);
UNTIL (ch = X);
writer.Update;
Files.Register(file);
ELSE
context.error.String("Could not create file"); context.error.Ln;
END;
ELSE
context.error.String("Expected filename argument"); context.error.Ln;
END;
END CreateFile;
END Example.
```

Exemplo2:

```
MODULE Arrays;
VAR
a: ARRAY 32 OF ARRAY 20 OF INTEGER;
b: POINTER TO ARRAY OF INTEGER;
PROCEDURE Print(x: ARRAY OF INTEGER);
VAR i: INTEGER;
BEGIN
i := 0;
WHILE i<LEN(x) DO
INC(i);
END;
END Print;
END Arrays.
```

Observações Gerais

Os comentários são iniciados com a sequência de símbolos **(*** e terminam com a sequência ***)**, e podem abranger mais de uma linha. Para comentários de uma linha, também pode ser utilizada a sequência de símbolos `//` para indicar o início do comentário.

A linguagem é *case insensitive.*

5. Trabalho a Desenvolver

As tarefas que são necessárias realizar são as seguintes:

1. Analisar o BNF da linguagem.
2. Definir o analisador sintático para a linguagem utilizando a ferramenta GOLD Parser.
3. Estender a linguagem com as alterações propostas.

Data de Entrega: 26 de maio

BOM TRABALHO