

# DISEÑO DE COMPILADORES

## EBNF Mini-Pascal

CAMPUS QUERÉTARO

---

### 1. Programs and blocks

```
programa ::= <program-heading> ";" <program-block> "."
program-heading ::= program <identifier> [ "(" <identifier-list> ")" ]
program-block ::= [<constant-declaration-part>]
                 [<variable-declaration-part>]
                 <statement-part>
constant-declaration-part ::= constant <constant-definition> ";"
                           { <constant-definition> ";" }
constant-definition ::= <identifier> "=" <constant>
variable-declaration-part ::= var <variable-declaration> ";" { <variable-declaration> ";" }
variable-declaration ::= <identifier-list> ":" <type>
statement-part ::= begin <statement-sequence> end "."
```

### 2. Statements

```
statement-sequence ::= <statement> { ";" <statement> }
statement ::= [ <simple-statement> | <structured-statement> ]
simple-statement ::= [ <assignment-statement> | <IO-statement> ]
assignment-statement ::= <identifier> "=" <expression>
IO-statement ::= [ <writeln-statement> | <readln-statement> ]
writeln-statement ::= "writeln" [ "(" <write-parameter-list> ")" ]
write-parameter-list ::= "(" parameter { "," <parameter> } ")"
parameter ::= <expression> | <identifier> |
readln-statement ::= "readln" [ "(" <identifier-list> ")" ]
structured-statement ::= <compound-statement> | <repetitive-statement> |
                        <conditional-statement>
compound-statement ::= begin <statement-sequence> end "."
repetitive-statement ::= <while-statement> | <repeat-statement> | <for-statement>
while-statement ::= while <expression> do <statement>
```

```

repeat-statement ::= repeat <statement-sequence> until <expression>

for-statement ::=
    for <identifier> “:=” <expression> (to|downto)
    <expression> do <statement>

conditional-statement ::= <if-statement>

if-statement ::= if <expression> then <statement> [else <statement> ]

```

### 3. Expressions

```

expression ::= <simple-expression> [ <relational-operator> <simple-expression> ]

simple-expression ::= [ <sign> ] <term> { <addition-operator> <term> }

term ::= <factor> { <multiplication-operator> <factor> }

factor ::= <identifier> | <number> | <string> |
    “(” <expression> “)” | “not” <factor>

relational-operator ::= “=” | “<” | “<” | “<=” | “>” | “>=”

addition-operator ::= “+” | “-” | “or”

multiplication-operator ::= “*” | “/” | “div” | “mod” | “and”

```

### 4. Variable and Identifier Categories

```

type ::= integer | real | boolean | string

identifier ::= <letter> { <letter> | <digit> }

```

### 5. Low Level Definitions

```

identifier-list ::= <identifier> { “,” <identifier> }

number ::= <integer-number> | <real-number>

integer-number ::= <digit-sequence>

real-number ::= <digit-sequence> “.” [ <unsigned-digit-sequence> ] [ <scale-factor> ] |
    <digit-sequence> <scale-factor>

scale-factor ::= ( “E” | “e” ) <digit-sequence>

unsigned-digit-sequence ::= <digit> { <digit> }

digit-sequence ::= [ <sign> ] <unsigned-digit-sequence>

sign ::= “+” | “-”

letter ::= [ “A” - “Z” ] | [ “a” - “z” ]

digit ::= [ “0” - “9” ]

```

```
string ::= " " <string-character> { string-character } " "  
string-character ::= <any-character-except-quote> | "  
constant ::= [ <sign> ] ( <identifier> | <number> ) | <string>
```