

# **CS4100/5100 COMPILER DESIGN PROJECT LANGUAGE SPECIFICATION**

## **Part A- Fall 2020**

The first stage of the language to be parsed by the compiler project, **P20**, is described below and in the CFG provided. It is the instructor's Pascal-like creation for this course for instructional purposes.

### **LEXICAL FEATURES- related to the scanner/lexical analyzer**

The Lexical part of the language is as described in the Lexical Analyzer assignment. The properly-functioning GetNextToken function from that assignment will exactly identify the tokens of P20.

### **SYNTAX FEATURES- related to the parser/syntax analyzer**

The Language Specification will be broken into Parts A and B, in order to divide the syntax analysis assignment into more manageable pieces. For Part A, an entire P20 program will allow a single BEGIN/END block of assignment statements using expressions containing identifiers and numeric constants. Virtually all of the code developed for Part A will be directly used in Part B of the assignment.

## PROJECT CFG

### PART-A

Notation: In the CFG below, the following conventions are used:

- 1) Anything prefaced by a \$ is a terminal token (symbol or reserved word); anything inside of <> pointy brackets is a non-terminal
- 2) An item enclosed in '[' , ']' square braces is optional
- 3) An item enclosed in '{' , '}' curly braces is repeatable; '\*' is '0 or more times', while '+' is '1 or more times'
- 4) Vertical bars, '|', are OR connectors; any one of the items they separate may be selected
- 5) Note that all named elements of the form \$SOMETHING are token codes for terminals which are defined for this language and returned by the *lexical analyzer*.

**NOTE: A program, below, must have a unique identifier for its name, which cannot appear as an identifier anywhere else within this program**

```
<program>      ->      $UNIT  <prog-identifier>  $SEMICOLON  <block>
                                   $PERIOD
```

**NOTE: A block, below, has a required 'BEGIN', at least one statement, and 'END'.**

```
<block>        ->      $BEGIN <statement>  { $SEMICOLON  <statement> } *
                                   $END
```

The rules <prog-identifier> and <variable> below are defined to provide for easy identifier checking to prevent misuse or re-use of any identifier during code generation later. Implement exactly as given!

```
<prog-identifier> -> <identifier>
```

#### Statements

```
<statement>->
```

*The only <statement> type for part A is 'assignment statement'*  
<variable> \$COLON-EQUALS <simple expression>

```
<variable>      ->      <identifier>
```

```
<simple expression> -> [<sign>] <term> {<addop> <term>}*
```

```
<addop>         ->      $PLUS | $MINUS
```

```
<sign>          ->      $PLUS | $MINUS
```

```
<term>          ->      <factor> {<mulop> <factor> }*
```

```
<mulop>         ->      $MULTIPLY | $DIVIDE
```

```

<factor>      ->      <unsigned constant> |
                        <variable> |
                        $LPAR    <simple expression>    $RPAR

<unsigned constant>->  <unsigned number>

<unsigned number>->    $FLOAT | $INTTYPE
                        **note: as defined for Lexical

<identifier>    ->    $IDENTIFIER
                        **note: as defined for Lexical, which is
                        <letter>  {<letter> |<digit> | $ | _ }*

```