Documentation for Final Delivery

CSCI 465 – Fall 2020

Derek Trom, Elena Corpus

9 December 2020

Table of Contents

1. Assumptions for Delivery 2:

a. Compile the variable declarations (integer only, so no type checking)

b. Handle simplified expressions

c. Handle Assignment statements

d. Handle I/O calls (at least for reading and writing numbers).

2. Current Status:

a. **IO Module**: Currently completed and up too standards in order to feed into scanner.

b. **Scanner**: Can currently create lexemes for each feature in the requirements.

c. **Parser**: The parser is recursive in nature with one (1) lookahead value, or LL(1). This means that the semantic analysis will have to be tightly coupled with the parser in the near future to allow for the confirmation of correctness.

d. **Symbol Table**: Currently has linked-list and support for scopes.

3. IO Module:

a. Responsible for reading input the pascal file and writing output to the scanner.

b. The IO Module reads from the file character by character and is able to put the character back into a file stream.

c. This gets the lexemes from the scanner needed for the lexical analysis and receives errors from the classes for the Lexer.

4. **Scanner**:

a. Is responsible for the lexical analysis in the program.

b. This scans the input from the IO module and translates the input into lexemes to be used by the parser.

5. **Parser**:

a. Using Flex and Bison as automated generators for lexical analysis and parsing will be used to verify the scanner and parser.

b. In addition, the current parser generates an abstract syntax tree.

6. Symbol Table:

a. This keeps track of all the lexemes and its values.

b. The parser is now able to add entries into the symbol table, with context, verify that there are no conflicting symbols, and create multiple levels of symbol tables for different blocks within the program, e.g. functions, procedures, etc.

c. Print statements are shown for each entrance into a function call for proof of concept and its error messages as well.

7. Intermediate Code Generation / Code Generator:

a. Now generates machine-dependent code

b. Using the abstract syntax tree to generate the MIPS code

8. Syntax

```
start   : program

program : PROGRAM ID SEMICOLON block PERIOD

block   : constant-definition-part

          /*type-definition-part*/

          variable-declaration-part

          procedure-and-function-declaration-part

          statement-part


constant-definition-part      : CONST constant-definition | ε

constant-definition           : ID constant-definition-variable EQUALS
constant-no-id SEMICOLON constant-definition-recursive

constant-definition-variable  : COMMA ID constant-definition-variable
| ε

constant-definition-recursive : constant-definition | ε


type-definition-part      : TYPE type-definition | ε

type-definition           : ID EQUALS type type-definition-recursive

type-definition-recursive : SEMICOLON type-definition | ε


variable-definition-part      : VAR variable-declaration | ε

variable-definition           : ID variable-definition-variable COLON
type SEMICOLON variable-definition-recursive | ε
```

```
variable-definition-variable  : COMMA ID variable-definition-variable
| ε

variable-definition-recursive : variable-definition | ε


type             : simple-type /*| array-type*/

array-type       : ARRAY LBRACK index-type RBRACK OF simple-type

index-type       : ID | index-range

index-constant   : sign INTVAL | CHARVAL | sign constant-name

index-range      : index-constant DOTDOT index-constant

simple-type      : STRING | INTEGER | REAL | CHAR

constant-name    : ID

sign             : ADD | MINUS | ε


procedure-and-function-definition-part : procedure-declaration
SEMICOLON | function-declaration SEMICOLON | ε

procedure-declaration : PROCEDURE ID

                        LPAREN formal-parameters RPAREN SEMICOLON

                        block

                        procedure-and-function-definition-part


formal-parameters           : ID formal-parameters-variable COLON type

formal-parameters-variable  : COMMA ID formal-parameters-variable | ε


function-declaration  : FUNCTION ID

                        LPAREN formal-parameters RPAREN

                        COLON type SEMICOLON

                        block

                        procedure-and-function-declaration-part


statement-part      : compound-statement
```

```
compount-statement  : BEGIN statement statement-recursive SEMICOLON
END

statement           : simple-statement | structured-statement

statement-recursive : SEMICOLON statement statement-recursive | ε

simple-statement    : assignment-statement | procedure-statement |
application | read-statement | write-statement


assignment-statement      : variable ASSIGN expression

procedure-statement       : ID

application               : ID LPAREN expression application-recursive
RPAREN

application-recursive     : COMMA expression application-recursive | ε

read-statement            : READ read-statement-part | READLN read-
statement-part

read-statement-part       : LPAREN ID read-statement-recursive RPAREN

read-statement-recursive  : COMMA ID read-statement-recursive | ε

write-statement           : WRITE write-statement-part | WRITELN
write-statement-part

write-statement-part      : LPAREN expression write-statement-
recursive RPAREN

write-statement-recursive : COMMA expression write-statement-recursive
| ε


structured-statement  : compound-statement | if-statement | while-
statement | for-statement

if-statement          : IF expression THEN statement if-statement-else

if-statement-else     : ELSE statement | ε

while-statement       : WHILE expression DO statement

for-statement         : FOR ID ASSIGN expression for-statement-to
expression DO statement

for-statement-to      : TO | DOWNTO


expression            : simple-expression expression-relational
```

```
simple-expression     : sign term expression-add
expression-add        : add-term term expression-add | ε
add-term              : ADD | MINUS | OR

term                  : factor term-mult
term-mult             : mult-term factor term-mult | ε
mult-term             : MULT | IDIV | DIV | AND
factor                : application | variable | constant | NOT factor

expression-relational : relational-operator simple-expression | ε
relational-operator   : EQ | NEQ | LT | LTE | GTE | GT

variable              : ID | ID LBRACK expression RBRACK
paramenter-identifier : ID

constant              : constant-no-id | sign constant-identifier
constant-no-id        : constant-number | sign constant-identifier |
CHARVAL | STRINGVAL
constant-number       : sign INTEGERNO | sign REALNO
constant-identifier   : ID
```