|  |
| --- |
| Frostburg State University |
| COSC 470 Compiler Project: User Manual |
| *PL/SQL Scripting Language Compiler* |
|  |
| **Daniel Herold** |
| **5/17/2014** |

|  |
| --- |
| Version 2 of my compiler project for COSC 470 in the Spring 2014 semester. |

Contents

[List of Tables 2](#_Toc388111231)

[List of Figures 2](#_Toc388111232)

[Introduction 4](#_Toc388111233)

[How to Start 4](#_Toc388111234)

[Data Types 5](#_Toc388111235)

[Operators 5](#_Toc388111236)

[Condition Statements 6](#_Toc388111237)

[Loops 6](#_Toc388111238)

[Print Statements 7](#_Toc388111239)

[User Input 7](#_Toc388111240)

[Comments 7](#_Toc388111241)

[Appendix A: Syntax Diagram i](#_Toc388111242)

[Block i](#_Toc388111243)

[Declarations i](#_Toc388111244)

[Declare\_rest i](#_Toc388111245)

[Default i](#_Toc388111246)

[Data\_type ii](#_Toc388111247)

[Characters ii](#_Toc388111248)

[Size ii](#_Toc388111249)

[Size\_option ii](#_Toc388111250)

[Numbers iii](#_Toc388111251)

[Compound\_statement iii](#_Toc388111252)

[Optional\_statements iii](#_Toc388111253)

[Statement\_list iii](#_Toc388111254)

[Statement iv](#_Toc388111255)

[Lefthandside iv](#_Toc388111256)

[Righthandside v](#_Toc388111257)

[Expression v](#_Toc388111258)

[Simple\_expression v](#_Toc388111259)

[Term v](#_Toc388111260)

[Factor vi](#_Toc388111261)

[Relop vii](#_Toc388111262)

[Addop vii](#_Toc388111263)

[Mulop viii](#_Toc388111264)

[Appendix B: Error Messages viii](#_Toc388111265)

[Type Mismatch Exception viii](#_Toc388111266)

[End of File Exception viii](#_Toc388111267)

[Max ID Size Error viii](#_Toc388111268)

[Max INT Size Exception viii](#_Toc388111269)

[Max SMALLINT Size Exception viii](#_Toc388111270)

[Index ix](#_Toc388111271)

# List of Tables

[Table 1: Data Types 5](#_Toc388103996)

[Table 2: Arithmetic Operators 5](#_Toc388103997)

[Table 3: Relational Operators 5](#_Toc388103998)

[Table 4: Print Statements 7](#_Toc388103999)

# List of Figures

[Figure 1: block i](#_Toc388104516)

[Figure 2: declarations i](#_Toc388104517)

[Figure 3: declare\_rest i](#_Toc388104518)

[Figure 4: default i](#_Toc388104519)

[Figure 5: data\_type ii](#_Toc388104520)

[Figure 6: characters ii](#_Toc388104521)

[Figure 7: size ii](#_Toc388104522)

[Figure 8: size\_option ii](#_Toc388104523)

[Figure 9: numbers iii](#_Toc388104524)

[Figure 10: compound\_statement iii](#_Toc388104525)

[Figure 11: optional\_statements iii](#_Toc388104526)

[Figure 12: statement\_list iii](#_Toc388104527)

[Figure 13: statement iv](#_Toc388104528)

[Figure 14: lefthandside iv](#_Toc388104529)

[Figure 15: righthandside v](#_Toc388104530)

[Figure 16: expression v](#_Toc388104531)

[Figure 17: simple\_expression v](#_Toc388104532)

[Figure 18: term v](#_Toc388104533)

[Figure 19: factor vi](#_Toc388104534)

[Figure 20: relop vii](#_Toc388104535)

[Figure 21: addop vii](#_Toc388104536)

[Figure 22: mulop viii](#_Toc388104537)

# Introduction

In order to complete the course COSC 470, Compiler Design and Implementation, we were tasked with creating a compiler for a LALR grammar for a PL/SQL scripting language. I used the Java programming language to implement my compiler. I made use of the following: a lexical analyzer, a semantic analyzer, and an intermediate code generator. The code generator output is sent through the Intermediate Code mini-Assembler, which was written in Java, written by Truman Parks Boyer in June 2003. This program translates mini-assembler instruction files into a file of raw integer quadruples. The output from mini is sent through the mini-Intermediate Code Engine (mICE), also developed by Truman Parks Boyer. mICE runs the code and runs all the commands from the source program.

# How to Start

The first thing one must do to use my compiler, is unzip the file *Compiler\_Working\_djherold0.zip*. Then import the source files into a Java-based IDE, preferably Netbeans. Now a program must be written to put through the compiler. The following is a simple example of the syntax of the language:

DECLARE

my\_int NUMBER() := 2;

BEGIN

DBMS\_OUTPUT.PUT\_LINE(my\_int)

END;

$

The DECLARE section is where all variables are declared. They are case sensitive. Variables cannot be declared in the body of the code. Variables are declared as follows:

identifier TYPE ;

or they can be declared and assigned:

identifier TYPE assignment\_op value ;

The maximum size of an identifier is 20 characters. Identifiers may contain underscores (\_), letters, and digits, and must start with a letter.

BEGIN starts a block and END terminates a block. The last statement in a block does not need a semi-colon at the end of it. Anywhere NULL is placed, a semi-colon must follow (i.e. NULL;).

The body of the code, DBMS\_OUTPUT.PUT\_LINE(my\_int), tells the system to output the value of the variable my\_int to the screen and also write to a new line. The program must end with a dollar sign ($).

# Data Types

This scripting language supports 6 data types:

|  |  |  |
| --- | --- | --- |
| BOOLEAN | True/false |  |
| INT | Numbers go here | Do not need to declare size, cannot be more than 8 digits |
| NUMBER | This type is for a number | Do not need to declare size |
| SMALLINT | Integers of small sizes | Cannot be more than 4 digits |
| POSITIVE | Positive numbers only | Cannot be negative |
| CHAR | Single characters | Single characters, in apostrophes (‘ ‘) |

Table : Data Types

Size is the maximum number of digits that variable can hold. If variables do not have a size declared, the parentheses must still follow, as in the above example program.

# Operators

The following operators are supported for arithmetic operations on INT, NUMBER, SMALLINT, and POSITIVE data types:

|  |  |
| --- | --- |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| MOD | Modulus |

Table : Arithmetic Operators

The Following relational operators are supported:

|  |  |
| --- | --- |
| < | Less than |
| <= | Less than or equal to |
| = | Equal to |
| >= | Greater than or equal to |
| > | Greater than |
| <> | Not equal to |

Table : Relational Operators

The reserved word NOT negates numbers or Boolean values.

# Condition Statements

Conditional statements are supported by my compiler and have the form:

IF condition THEN

Statement

END IF

Condition represents a Boolean relation (e.g. a < b), and statement is the thing to carry out if the condition is true. If there is only 1 statement, BEGIN and END reserved words are not required surrounding the statement, and a semi-colon is not needed at the end of the statement. However, if the block contains more than 1 statement, it requires the reserved words BEGIN and END, following the form:

IF condition THEN

BEGIN

Statement1;

Statement2

END;

END IF

# Loops

This language supports WHILE loops, which check the condition at the beginning of each loop and execute the statements if the condition is true. Loops have the following form:

WHILE condition LOOP

Statement

END LOOP

As with conditional statements, if there are multiple statements in the body of a loop they must have the BEGIN and END reserved words, as follows:

WHILE condition LOOP

BEGIN

Statement1;

Statement2

END;

END LOOP

# Print Statements

The language allows several print statements to be made:

|  |  |
| --- | --- |
| DBMS\_OUTPUT.PUT | Prints a value and stays on the line |
| DBMS\_OUTPUT.PUT\_LINE | Prints a value and moves to a new line |
| DBMS\_OUTPUT.NEW\_LINE | Moves to a new line |

Table : Print Statements

# User Input

The language supports user input: wherever an ampersand (&) appears before variables, it represents user input. For example, &my\_num means my\_num will be a variable whose value is determined by user input.

# Comments

My compiler supports 2 forms of comments:

|  |  |
| --- | --- |
| -- | Single line comments; only last for the remainder of the line the where they appear |
| /\* \*/ | Multiple line comments; start at the opening (/\*) and continue until the terminator is reached (\*/) |

# Appendix A: Syntax Diagram

## Block



Figure : block

Block ::= declarations compound\_statment '$'

## Declarations

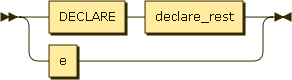


Figure : declarations

declarations ::= DECLARE declare\_rest

| e

## Declare\_rest

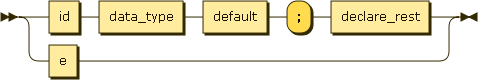


Figure : declare\_rest

declare\_rest ::= id data\_type default ';' declare\_rest

| e

## Default



Figure : default

default ::= ':=' righthandside

| e

## Data\_type

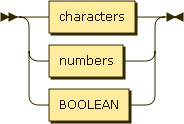


Figure : data\_type

data\_type ::= characters

| numbers

| BOOLEAN

## Characters



Figure : characters

characters ::= CHAR

## Size

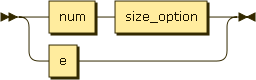


Figure : size

size ::= num size\_option

| e

## Size\_option

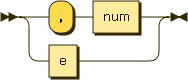


Figure : size\_option

size\_option ::= ',' num

| e

## Numbers

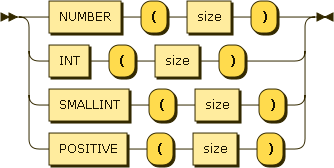


Figure : numbers

numbers ::= NUMBER '(' size ')'

| INT '(' size ')'

| SMALLINT '(' size ')'

| POSITIVE '(' size ')'

## Compound\_statement



Figure : compound\_statement

compound\_statement ::= BEGIN optional\_statements END ';'

## Optional\_statements

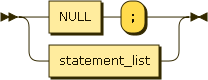


Figure : optional\_statements

optional\_statements ::= NULL ';'

| statement\_list

## Statement\_list

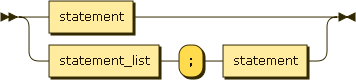


Figure : statement\_list

statement\_list ::= statement

| statement\_list ';' statement

## Statement

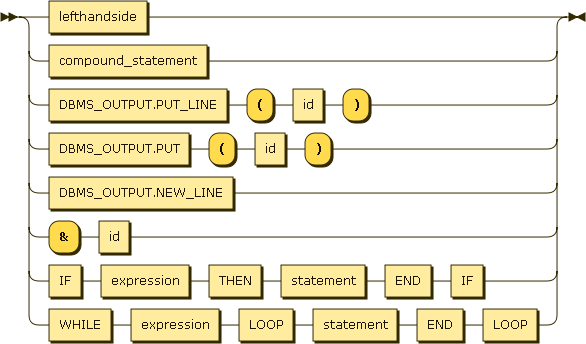


Figure : statement

statement ::= lefthandside

| compound\_statement

| DBMS\_OUTPUT.PUT\_LINE '(' id ')'

| DBMS\_OUTPUT.PUT '(' id ')'

| DBMS\_OUTPUT.NEW\_LINE

| '&' id

| IF expression THEN statement END IF

| WHILE expression LOOP statement END LOOP

## Lefthandside



Figure : lefthandside

lefthandside ::= id ':=' righthandside

## Righthandside

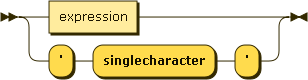


Figure : righthandside

righthandside ::= expression

| "'" 'singlecharacter' "'"

## Expression

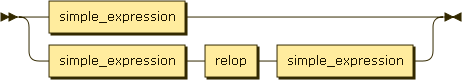


Figure : expression

expression ::= simple\_expression

| simple\_expression relop simple\_expression

## Simple\_expression

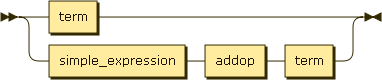


Figure : simple\_expression

simple\_expression ::= term

| simple\_expression addop term

## Term

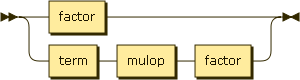


Figure : term

term ::= factor

| term mulop factor

## Factor

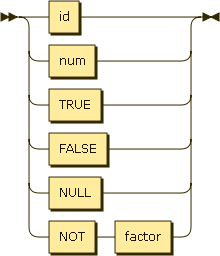


Figure : factor

factor ::= id

| num

| TRUE

| FALSE

| NULL

| NOT factor

## Relop

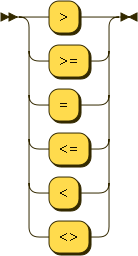


Figure : relop

relop ::= '>'

| '>='

| '='

| '<='

| '<'

| '<>'

## Addop

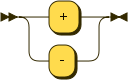


Figure : addop

addop ::= '+'

| '-'

## Mulop

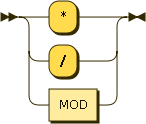


Figure : mulop

mulop ::= '\*'

| '/'

| MOD

# Appendix B: Error Messages

## Type Mismatch Exception

This error is thrown when a variable is not assigned the correct type, or if TRUE is trying to be added to 5.

## End of File Exception

This error is thrown when scanning in the source file and reaching the end of file before seeing the dollar sign ($).

## Max ID Size Error

This error is thrown when an identifier is used that is more than 20 characters long.

## Max INT Size Exception

This error is thrown when an INT has more than 8 digits in it.

## Max SMALLINT Size Exception

This error is thrown when a SMALLINT has more than 4 digits.

# Index

Comments, 7

Conditional Statements, 6

Data Types, 5

Error Messages, viii

Loops, 6

Operators, 5

Print Statements, 7

Syntax Diagrams, i

User Input, 7