COSC 470 Final Project

Mini-PL/SQL Compiler

User Manual

Developed by:

Ryan Kasprzyk

Release Candidate 3

Date of Release:

5/14/2012

Contents

[Introduction 3](#_Toc356289707)

[Acknowledgments 3](#_Toc356289708)

[Getting Started 3](#_Toc356289709)

[Data Types 5](#_Toc356289710)

[Operators 6](#_Toc356289711)

[Conditional Statements 7](#_Toc356289712)

[Loops 8](#_Toc356289713)

[Print Statements 8](#_Toc356289714)

[User Input 9](#_Toc356289715)

[Comments 9](#_Toc356289716)

[Appendix A – Syntax Diagram i](#_Toc356289717)

[Block i](#_Toc356289718)

[Declarations i](#_Toc356289719)

[Declare\_rest i](#_Toc356289720)

[Type i](#_Toc356289721)

[Default i](#_Toc356289722)

[Data\_type i](#_Toc356289723)

[Characters ii](#_Toc356289724)

[Size ii](#_Toc356289725)

[Numbers ii](#_Toc356289726)

[Compund\_statement ii](#_Toc356289727)

[Optional\_statements ii](#_Toc356289728)

[Statement\_list ii](#_Toc356289729)

[Statement iii](#_Toc356289730)

[Lefthandside iii](#_Toc356289731)

[Righthandside iv](#_Toc356289732)

[Casting iv](#_Toc356289733)

[Simple\_expression iv](#_Toc356289734)

[Term v](#_Toc356289735)

[Factor v](#_Toc356289736)

[Relop vi](#_Toc356289737)

[Addop vi](#_Toc356289738)

[Mulop vi](#_Toc356289739)

[Appendix B – Error Messages vii](#_Toc356289740)

[Index viii](#_Toc356289741)

[Table 1 - Data Types 5](#_Toc356289742)

[Table 2 - Arithmetic Operators 6](#_Toc356289743)

[Table 3 - Relational Operators 7](#_Toc356289744)

[Table 4 - Print Statements 8](#_Toc356289745)

# Introduction

As the semester project for COSC 470: Compiler Design and Implementation, we have implemented a basic compiler of a small scripting language, based on our understanding of the course material. Using a number of design tools and the class material, we have created a front-end compiler, namely one that translates our programming language into a set of intermediate code that will be executed using a back-end compiler called micE (Mini-Intermediate Code Engine). Parse tables were generated based on the language grammar supplied using GNU-Bison, and open-source version of YACC. This compiler is distributed under the GNU General Public License and may be freely distributed and modified, except for profit. Any section of this compiler may be used with credit.

## Acknowledgments

Thanks to the GNU Project for providing GNU Bison free of charge for Windows and Linux distributions. Designed and written in the Java version 6 API from Oracle Corporation using Netbeans IDE from Oracle Corporation. All technologies and programs are used under the Common Development and Distribution License v1.0 and the GNU General Public License v2.

# Getting Started

To use this compiler, begin by unzipping the download file *Compiler\_Final\_rbkasprzyk0.zip* to a directory you can easily find, and import the source files into whichever Java-based IDE you may be using (Netbeans or Eclipse, generally). Once you have done this, it's time to write a program. To begin understanding the syntax of the language, let us begin with a relatively simple program:

DECLARE

my\_int NUMBER(2):=2;

BEGIN

DBMS\_OUTPUT.PUT(my\_int);

END;\

Now we will discuss this code to help understand how this language works. The first thing you may notice is the word DECLARE. In this language, all variables must be declared at the start of the program. Declaring variables within the main body of the code will throw and exception. Variable declarations must be of the form:

identifier TYPE

where applicable, an assignment may also be made, but all variables must be declared with a name and type. Once the first BEGIN is encountered, declarations may no longer be made and the program will begin executing commands. The BEGIN and END reserved words function here as open ({) and close (}) braces respectively function in languages like Java and C. All lines with the exception of BEGIN and DECLARE must end with a semicolon (;). Within the body of the code, the line DBMS\_OUTPUT.PUT(my\_int) tells the system to print the value of my\_int to the screen. Further print statements will be discussed in later sections. Lastly, the program itself must end with a backslash (\).

# Data Types

This programming language supports 5 data types:

|  |  |  |
| --- | --- | --- |
| BOOLEAN | Contains true or false values |  |
| POSITIVE | Contains positive numbers, including 0 | Must declare size |
| NUMBER | Contains positive and negative values | Must declare size |
| CHAR | Contains single characters |  |
| VARCHAR | Contains strings of alphanumeric and special characters | Must declare size |

Table - Data Types

In the cases of those with size attributes, the size indicates the maximum number of digits/characters that may be held in the object.

Explicit casts allowed are as follows:

* VARCHAR2 may be converted to CHAR is the length of VARCHAR is 1
* CHAR may be converted to NUMBER, POSITIVE, or VARCHAR
* NUMBER may be converted to VARCHAR2, POSITIVE (if the value is positive), or CHAR (if the value is a single digit)
* POSITIVE may be converted to VARCHAR2, NUMBER, or CHAR (if the value is a single digit)

In addition, several implicit casting operations are allowed:

* A NUMBER may freely be assigned the value of a POSITIVE
* A POSITIVE may be assigned the value of a NUMBER of the currents value is greater than or equal to 0
* A VARCHAR may freely be assigned the value of a CHAR
* A CHAR may be assigned the value of a CHAR if the current value of the VARCHAR is a single character or the VARCHAR has size 1

# Operators

The following operators are supported for arithmetic operations on POSITIVE and NUMBER objects:

|  |  |
| --- | --- |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Modular (remainder) division |

Table - Arithmetic Operators

The following comparative operations are also 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 may be used to indicate a negative number, or to negate the value of a numeric or Boolean construct.

# Conditional Statements

Conditional IF statements are supported by this compiler and must be of the form:

IF BEGIN condition THEN

Statements

END IF;

Where *condition* signifies a single Boolean relational condition and *statements* signifies the operations to be carried out if the condition is true. Please note that while simply placing one operation in the statements section requires no additional code, putting multiple operations in the statements section requires it to be enclosed in a BEGIN-END pair, of the form:

IF BEGIN condition THEN

BEGIN

Operation1;

Operation2;

END;

END IF;

# Loops

This programming language supports only while loops, which execute based on a starting condition and check whether to continue execution by checking the condition at the beginning of every pass through the loop. Similar to conditional statements, loops are of the form:

WHILE condition LOOP

Statements

END LOOP;

An additional similarity to conditionals is that no additional code is needed if the loop only executes one operation within the body of the loop. If multiple operations are to be executed in the body of the loop, they must be enclosed in a BEGIN-ELSE pair, such as:

WHILE condition LOOP

BEGIN

Operation1;

Operation2;

END;

END LOOP;

# Print Statements

The following statements will print the contents of a given identifier when invoked:

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

Table - Print Statements

# User Input

Wherever the program should be setting the value of a variable based on user input, simply preface the identifier with an ampersand (&). For example:

&my\_int;

Informs the compiler that the next user input should be stored in the variable my\_int.

# Comments

Single line comments must be prefaced with two dashes (--). This signals the compiler that the remainder of the line is to be ignored.

Multi-line comments must begin with the opening string (/\*). When the compiler sees this it will disregard all data it sees in the input file until it reaches the closing string (\*/).

# Appendix A – Syntax Diagram

### Block



[block](http://railroad.my28msec.com/rr/ui#block) ::= [declarations](http://railroad.my28msec.com/rr/ui#declarations) [compound\_statement](http://railroad.my28msec.com/rr/ui#compound_statement) '!'

### Declarations



[declarations](http://railroad.my28msec.com/rr/ui#declarations) ::= [DECLARE](http://railroad.my28msec.com/rr/ui#DECLARE) [declare\_rest](http://railroad.my28msec.com/rr/ui#declare_rest) | e

### Declare\_rest



[declare\_rest](http://railroad.my28msec.com/rr/ui#declare_rest) ::= [identifier](http://railroad.my28msec.com/rr/ui#identifier) [type](http://railroad.my28msec.com/rr/ui#type) ';' [declare\_rest](http://railroad.my28msec.com/rr/ui#declare_rest) | e

### Type



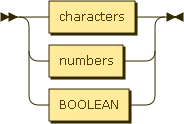
[type](http://railroad.my28msec.com/rr/ui#type) ::= [data\_type](http://railroad.my28msec.com/rr/ui#data_type) [default](http://railroad.my28msec.com/rr/ui#default)

### Default



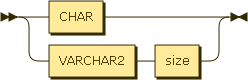
[default](http://railroad.my28msec.com/rr/ui#default) ::= ':=' [righthandside](http://railroad.my28msec.com/rr/ui#righthandside) | e

### Data\_type



[data\_type](http://railroad.my28msec.com/rr/ui#data_type) ::= [characters](http://railroad.my28msec.com/rr/ui#characters) | [numbers](http://railroad.my28msec.com/rr/ui#numbers) | [BOOLEAN](http://railroad.my28msec.com/rr/ui#BOOLEAN)

### Characters



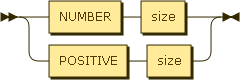
[characters](http://railroad.my28msec.com/rr/ui#characters) ::= [CHAR](http://railroad.my28msec.com/rr/ui#CHAR) | [VARCHAR2](http://railroad.my28msec.com/rr/ui#VARCHAR2) [size](http://railroad.my28msec.com/rr/ui#size)

### Size



[size](http://railroad.my28msec.com/rr/ui#size) ::= '(' [num](http://railroad.my28msec.com/rr/ui#num) ')'

### Numbers



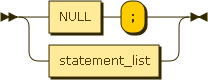
[numbers](http://railroad.my28msec.com/rr/ui#numbers) ::= [NUMBER](http://railroad.my28msec.com/rr/ui#NUMBER) [size](http://railroad.my28msec.com/rr/ui#size) | [POSITIVE](http://railroad.my28msec.com/rr/ui#POSITIVE) [size](http://railroad.my28msec.com/rr/ui#size)

### Compund\_statement



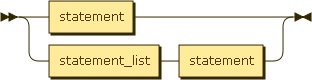
[compound\_statement](http://railroad.my28msec.com/rr/ui#compound_statement) ::= [BEGIN](http://railroad.my28msec.com/rr/ui#BEGIN) [optional\_statements](http://railroad.my28msec.com/rr/ui#optional_statements) [END](http://railroad.my28msec.com/rr/ui#END) ';'

### Optional\_statements



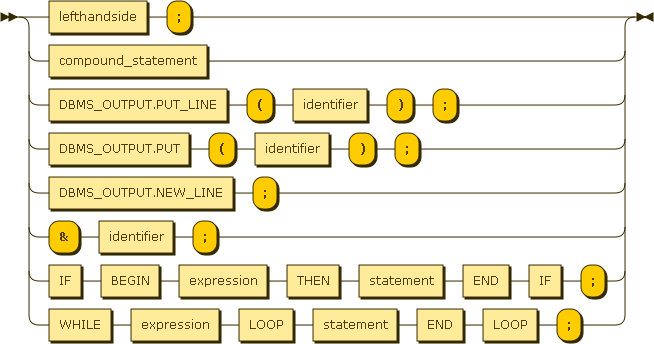
[optional\_statements](http://railroad.my28msec.com/rr/ui#optional_statements) ::= [NULL](http://railroad.my28msec.com/rr/ui#NULL) ';' | [statement\_list](http://railroad.my28msec.com/rr/ui#statement_list)

### Statement\_list



[statement\_list](http://railroad.my28msec.com/rr/ui#statement_list) ::= [statement](http://railroad.my28msec.com/rr/ui#statement) | [statement\_list](http://railroad.my28msec.com/rr/ui#statement_list) [statement](http://railroad.my28msec.com/rr/ui#statement)

### Statement



[statement](http://railroad.my28msec.com/rr/ui#statement) ::= [lefthandside](http://railroad.my28msec.com/rr/ui#lefthandside) ';'

| [compound\_statement](http://railroad.my28msec.com/rr/ui#compound_statement)

| [DBMS\_OUTPUT.PUT\_LINE](http://railroad.my28msec.com/rr/ui#DBMS_OUTPUT.PUT_LINE) '(' [identifier](http://railroad.my28msec.com/rr/ui#identifier) ')' ';'

| [DBMS\_OUTPUT.PUT](http://railroad.my28msec.com/rr/ui#DBMS_OUTPUT.PUT) '(' [identifier](http://railroad.my28msec.com/rr/ui#identifier) ')' ';'

| [DBMS\_OUTPUT.NEW\_LINE](http://railroad.my28msec.com/rr/ui#DBMS_OUTPUT.NEW_LINE) ';'

| '&' [identifier](http://railroad.my28msec.com/rr/ui#identifier) ';'

| [IF](http://railroad.my28msec.com/rr/ui#IF) [BEGIN](http://railroad.my28msec.com/rr/ui#BEGIN) [expression](http://railroad.my28msec.com/rr/ui#expression) [THEN](http://railroad.my28msec.com/rr/ui#THEN) [statement](http://railroad.my28msec.com/rr/ui#statement) [END](http://railroad.my28msec.com/rr/ui#END) [IF](http://railroad.my28msec.com/rr/ui#IF) ';'

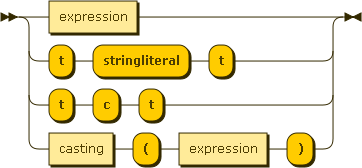
| [WHILE](http://railroad.my28msec.com/rr/ui#WHILE) [expression](http://railroad.my28msec.com/rr/ui#expression) [LOOP](http://railroad.my28msec.com/rr/ui#LOOP) [statement](http://railroad.my28msec.com/rr/ui#statement) [END](http://railroad.my28msec.com/rr/ui#END) [LOOP](http://railroad.my28msec.com/rr/ui#LOOP) ';'

### Lefthandside



[lefthandside](http://railroad.my28msec.com/rr/ui#lefthandside) ::= [identifier](http://railroad.my28msec.com/rr/ui#identifier) ':=' [righthandside](http://railroad.my28msec.com/rr/ui#righthandside)

### Righthandside



[righthandside](http://railroad.my28msec.com/rr/ui#righthandside) ::= [expression](http://railroad.my28msec.com/rr/ui#expression)

| 't' 'stringliteral' 't'

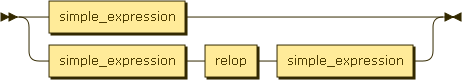
| 't' 'c' 't'

| [casting](http://railroad.my28msec.com/rr/ui#casting) '(' [expression](http://railroad.my28msec.com/rr/ui#expression) ')'

### Casting



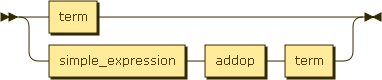
[casting](http://railroad.my28msec.com/rr/ui#casting) ::= [data\_type](http://railroad.my28msec.com/rr/ui#data_type)



[expression](http://railroad.my28msec.com/rr/ui#expression) ::= [simple\_expression](http://railroad.my28msec.com/rr/ui#simple_expression)

| [simple\_expression](http://railroad.my28msec.com/rr/ui#simple_expression) [relop](http://railroad.my28msec.com/rr/ui#relop) [simple\_expression](http://railroad.my28msec.com/rr/ui#simple_expression)

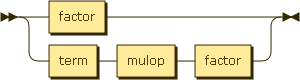
### Simple\_expression



[simple\_expression](http://railroad.my28msec.com/rr/ui#simple_expression) ::= [term](http://railroad.my28msec.com/rr/ui#term)

| [simple\_expression](http://railroad.my28msec.com/rr/ui#simple_expression) [addop](http://railroad.my28msec.com/rr/ui#addop) [term](http://railroad.my28msec.com/rr/ui#term)

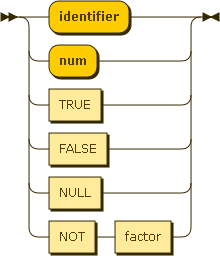
### Term



[term](http://railroad.my28msec.com/rr/ui#term) ::= [factor](http://railroad.my28msec.com/rr/ui#factor)

| [term](http://railroad.my28msec.com/rr/ui#term) [mulop](http://railroad.my28msec.com/rr/ui#mulop) [factor](http://railroad.my28msec.com/rr/ui#factor)

### Factor



[factor](http://railroad.my28msec.com/rr/ui#factor) ::= 'identifier'

| 'num'

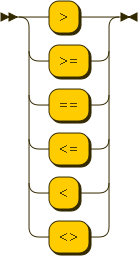
| [TRUE](http://railroad.my28msec.com/rr/ui#TRUE)

| [FALSE](http://railroad.my28msec.com/rr/ui#FALSE)

| [NULL](http://railroad.my28msec.com/rr/ui#NULL)

| [NOT](http://railroad.my28msec.com/rr/ui#NOT) [factor](http://railroad.my28msec.com/rr/ui#factor)

### Relop



[relop](http://railroad.my28msec.com/rr/ui#relop) ::= '>'

| '>='

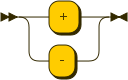
| '=='

| '<='

| '<'

| '<>'

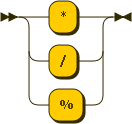
### Addop



[addop](http://railroad.my28msec.com/rr/ui#addop) ::= '+'

| '-'

### Mulop



[mulop](http://railroad.my28msec.com/rr/ui#mulop) ::= '\*'

| '/'

| '%'

# Appendix B – Error Messages

**GrammarException** – This error is thrown by the parser when it encounters input that does not match the rules of the grammar.

**TypeMismatchException** – This error is thrown when an attempt is made to assign invalid data to a variable (e.g. assign TRUE to a NUMBER).

**InvalidTokenException** – This error is thrown when the token scanner encounters an undefined reserved word.

**SizeMismatchException** – This error is thrown when attempting to assign too large a value into a variable (e.g. attempting to store the number 999 into a number of size 2).

**UndeclaredIdentifierException** – This error is thrown when an attempt is made to declare a variable outside the declarations section of the code.

**InvalidIdentifierException** – This error is thrown when the token scanner sees an identifier longer than 20 characters or containing illegal characters.

# Index

Comments, 9

Conditional Statements, 7

Data Types, 5

Error Messages, vii

Loops, 8

Operators, 6

Print Statements, 8

Syntax Diagrams, i

User Input, 9