LANGUAGE SPECIFICATIONS

Case Sensitive:

The language is case sensitive. All keywords when used should be in capital letters.

Language Type:

The language is a Strongly Typed language. Variables when declared require a type. Once a type is assigned it cannot be changed. Supported variable types are

CHARACTER	1 byte unsigned
INTEGER	4 bytes signed
BOOLEAN	1 byte Unsigned
STRING	1 byte sequence

Variable Scope:

Variables are accessible within their defined scope. A variable is not recognized outside the scope it is defined in.

Variable Naming:

Variable names cannot start with:

Two underscores (__ variable name)
Digits

Variable names cannot be repeated (even when defined in different scopes)

Variables cannot have spaces in between.

LANGUAGE MANUAL

Keyword LET:

LET is used to declare variables.

Example:

```
LET x AS INTEGER
LET Y AS CHARACTER
LET name AS STRING = "name"
LET myVal AS INTEGER = -560
```

Strings must be initialized as well as declared.
Declaration and assignment can also be done at the same time.

Keyword SET:

After variable is declared using LET keyword, SET is used to change values of a variable. The variable can also be assigned the result of an arithmetic expression or a returned value from a CALL.

Example:

```
SET x = 8
SET y = (a * 20) + ((23 ^ 2) – x + 9)
SET z = z + 1
SET sum = sumOfInt(23, z)
```

STRING variables cannot be changed using SET. They are immutable.

Keyword PRINT and PRINTLINE:

PRINT and PRINTLINE is used to print to the screen.

Example:

```
PRINT “The value of variableName: ”
PRINTLINE variableName
PRINTLINE “Hello, World!”
```

PRINTLINE changes a line after printing the value, while PRINT does not.

Keyword INPUT:

INPUT keyword is used to gain input from user.

Example:

```
INPUT variable
INPUT anotherVar
```

The datatype of the variable is resolved by the compiler itself and there's no need to specify what kind of variable is the destination of input.

Keyword IF:

IF keyword is used for logical expressions. Logical operators are supported such as AND, OR, NOT, <, <=, >, >=, ==, != are supported.

IF causes a new scope to be defined, and the scope must be closed using ENDIF statement.

Example:

```
IF (X==0)
	PRINTLINE "X is 0"
ENDIF
```

```
IF(Y<9 AND ((X == 5) AND (NOT (Y < 5)))
	IF(TRUE AND FALSE)
		PRINTLINE "This will never be printed"
	ELSE
		PRINTLINE "This will always be printed"
	ENDIF
ENDIF
```

Keyword METHOD :

METHOD is used for procedures. The syntax is as follows:

```
METHOD <name> TAKES <argName1> AS <type> [REF], <argName2> AS <type>, RETURNS <type>
      [ METHOD CODE ]
END METHOD
```

Example:

```
METHOD simpleMethod RETURNS NOTHING
      PRINTLINE “I am simple!”
END METHOD
```

```
METHOD SUM TAKES X AS INTEGER, Y AS BOOLEAN, RETURNS NOTHING
      PRINTLINE “EXAMPLE”
END METHOD
```

```
METHOD myMethod RETURNS NOTHING
      IF (someVar >= 5)
            RETURN
      ELSE
            PRINTLINE “SomeVar is less than 5.”
      ENDIF
ENDMETHOD
```

```
METHOD SWAP TAKES VarA AS INTEGER REF, VarB AS INTEGER REF, RETURNS NOTHING
      LET temp AS INTEGER
      SET temp = VarA
      SET VarA = VarB
      SET VarB = temp
ENDMETHOD
```

```
METHOD isNegative TAKES someVar AS INTEGER, RETURNS BOOLEAN
      LET retVal AS BOOLEAN

      IF (someVar < 0)
            RETURN TRUE
      ELSE
            RETURN FALSE
      ENDIF
ENDMETHOD
```

A method can take any number of arguments, and arguments can be passed by value or by reference using REF keyword.

STRING type variables are always passed as reference, regardless of the usage of REF keyword. But since they are immutable, their value can not be changed.

Recursion is also supported.

Keyword CALL:

Call is used to call a method. Syntax is as follows:

```
CALL <method name> (parameter1, parameter2, ...)
```

Example:

```
CALL SUM (2, -9)
CALL SUM (varA, varB)

LET medianValue AS INTEGER
SET medianValue = CALL median(10, 20, 30)
```

Keyword RETURN:

Keyword **RETURN** is used to return a value from a procedure.
Syntax is as follows:

```
RETURN <value>
```

RETURN is not necessary. In case a function returns a value, not writing RETURN will cause a default value of that type to be returned instead without any warning.

Example:

```
METHOD abc TAKES X AS INTEGER, RETURNS INTEGER
	PRINTLINE “HELLO”
	RETURN X
END METHOD
```

```
METHOD absolute TAKES X AS INTEGER, RETURNS NOTHING
	IF (X >= 0)
		RETURN
	ELSE
		LET retVal AS INTEGER
		SET retVal = X * -1
		RETURN retVal
	ENDIF
END METHOD
```

```
METHOD abc TAKES X AS INTEGER, RETURNS INTEGER
       PRINTLINE “HELLO”
       RETURN 0
END METHOD
```

Keyword PAUSE:

Keyword pause is used to pause the screen.

Example:

```
PAUSE
```

This causes a string to be shown: “Press any key to continue . . .”
and the program execution pauses until a key is pressed. Pressing a key causes the program to continue with the execution.

Keyword NEWLINE:

Keyword NEWLINE is used to print a new line on the screen which essentially causes the next print statement to skip the current line and start printing at next line.

Example:

```
NEWLINE
```

Keyword COMMENT:

COMMENT is used to tell compiler the lines that are to be ignored.

Example:

```
PRINTLINE “hello”
COMMENT this is a commented line and will be ignored
PRINTLINE “hello again”
```

USER MANUAL

To compile the code, write it in a text file.

Then use the command line to call the compiler as follows:

```
COAL_Compiler.exe FileToCompile.txt Output.exe
```