Análisis Sintáctico

Dado un programa en QB64, su tarea consiste en realizar el análisis sintáctico sobre dicho código. Nos enfocaremos en los errores sintácticos generados. En este documento se especifica la manera correcta de mostrar las salidas.

En el caso de que el programa se encuentre bien formado con respecto a las reglas especificadas de QB64 se debe escribir la siguiente salida.

El analisis sintactico ha finalizado correctamente.

Note que no se permiten tildes en el mensaje de salida.

En caso contrario, es decir, si se encontró un error sintáctico, se debe **abortar** el análisis e escribir únicamente el primer error encontrado.

Consideraciones gramaticales

A continuación se presentan algunas reglas gramaticales que debe tener en cuenta a la hora de realizar el análisis sintáctico:

- Un programa escrito en QB64 debe tener la siguiente estructura:

```
' Modulo principal
' cero o mas instrucciones

variable = 10

for i = 0 to 50
      print "Esto hace parte del modulo principal"
next i


' Procedimientos y funciones
' cero o mas procedimientos y funciones

sub sub1
      print "Procedimiento1"
end sub

' Aqui solo puede ser declarado nuevas funciones
' o nuevos procedimientos
```

```
function f1
      print "Funcion1"
end function
```

Si un programa no respeta la estructura mencionada se considera un error sintáctico.

Declaración y uso de variables

Como se había especificado en el documento de QB64, la declaración de variables se hace mediante la palabra reservada DIM o con el uso de sufijos.

Cuando se usa DIM también se puede especificar que una variable es global colocando la palabra reservada SHARED.

En el caso de la declaración usando DIM se debe tener la siguiente estructura gramatical:

```
DIM Y AS LONG
DIM X(expresion, expresion, expresion) AS STRING
DIM A, B, C(expresion) AS SINGLE
DIM SHARED global, g(expresion, expresion) AS DOUBLE
```

Mientras que en el caso de sufijos, la declaración puede ser realizada en cualquier parte del código: en sentencias de input, print o incluso en el llamado de una función. Lo importante es que respete la estructura del documento de especificación de QB64 y su sufijo correspondiente.

```
a& = 1000000
b! = 1.1
c = 2.0
d% = 1000
e# = 10.000000042
f$ = “string”
```

La declaración de constantes se hace usando la palabra reservada CONST y la declaración con sufijos. Tiene la siguiente sintaxis:

```
CONST INF& = 10000000
```

INPUT, PRINT

Los comandos para ingresar e imprimir datos por consola están dados por la siguiente formal gramatical:

```
INPUT nombre$
INPUT nombre$, edad
PRINT nombre$
PRINT edad; nombre$; 2017
```

Cuando se usa INPUT existe la posibilidad de capturar varios datos desde consola separando los argumentos por comas.

El input solo tiene permitido capturar valores dentro de variables. Es decir, algo como lo siguiente no es viable:

```
INPUT nombre$, 8, nombre    ' Un valor no esta permitido
```

En el caso de PRINT deben ir separados por punto y coma.

NOTA IMPORTANTE:

Las sentencias PRINT e INPUT no pueden ir vacías, es decir, siempre se debe capturar algo o imprimir algo. Adicionalmente, para evitar ambigüedades en el lenguaje una sentencia PRINT **no** puede terminar en ';'.

Estructuras de control

IF

```
IF expresion THEN
    ' comandos
ELSEIF expresion THEN
    ' comandos
ELSE
    ' comandos
END IF
```

Donde *expresion* es cualquier forma de comparación usando operadores relacionales y lógicos entre variables y lexemas de los distintos tipos de dato, o simplemente una de esas variables o un lexema.

Debe terminar con END IF y posiblemente tener un ELSE en la parte final. Puede haber cero o más ELSEIF antes de ELSE.

SELECT … CASE

La estructura de control de selección tiene la siguiente forma:

```
SELECT CASE Name$
    CASE "Ted"
       ' comandos
    CASE "Mike"
       ' comandos
    CASE ELSE
       ' comandos
END SELECT
```

Los select pueden comparar todos los tipos de dato primitivos. Siempre debe estar indicada una **variable** en el inicio del SELECT pues ésta es la que se va a comparar. Es decir, no es permitido un entero por ejemplo. Sin embargo, para la comparación en cada CASE sólo están permitidos lexemas de tokens de los tipo de datos primitivos.

Finalmente es opcional hacer un caso default usando CASE ELSE.

WHILE

```
WHILE expresion
    ' comandos
WEND
```

expresion sigue siendo lo mismo que lo dicho en el caso del IF.

DO … WHILE

```
DO
    ' comandos
LOOP WHILE expresion
```

DO … UNTIL

```
DO
   ' comandos
LOOP UNTIL expresion
```

FOR

```
FOR i = 1 TO 10
    PRINT i
NEXT
```

Para hacer aumentos específicos, diferentes a 1, se utiliza la palabra reservada **STEP**.

La palabra reservada NEXT debe ir al fin de cada estructura FOR.

```
FOR i = expresion TO expresion STEP expresion
    FOR j = expresion TO expresion
        PRINT i + j
    NEXT
NEXT
```

El tipo de variables que pueden ser declaradas para recorrer el for son **únicamente** las numéricas (single, double, integer, long) y solo pueden ser variables, es decir, una posición de un arreglo no está permitida.

NOTA IMPORTANTE:

Para realizar el aumento del FOR usando la palabra reservada NEXT **no** se debe incluir la variable de aumento. Es decir, algo como: “NEXT id” sería interpretado como el uso de la palabra reservada NEXT seguido por el llamado a un procedimiento con identificador “id”.

Procedimientos

Inician con la palabra reservada **SUB** y finalizan con **END SUB.** Luego de SUB, debe ir el nombre del procedimiento, luego los parámetros que el procedimiento debe recibir.

Ejemplo de un procedimiento sin parámetros:

```
SUB procedimiento
    PRINT “Ejemplo sin parametros”
END SUB
```

Ejemplo de un procedimiento con parámetros:

```
SUB procedimiento (nombre$, edad)
    PRINT nombre$; edad;
END SUB
```

Funciones

Tienen la misma estructura gramatical que los procedimientos.

Ejemplo de una función sin parámetros:

```
FUNCTION funcion!
    PRINT “Ejemplo sin parametros”
    funcion! = 3.4
END FUNCTION
```

Ejemplo de una función con parámetros:

```
FUNCTION funcion% (nombre$, edad)
    PRINT nombre$; edad;
END FUNCTION
```

Las funciones, a excepción de los procedimientos, pueden llevar un sufijo. Indicando así el tipo de retorno de la función, por eso precisamente no está permitido para los procedimientos.

Llamados a funciones y procedimientos

El llamado a procedimientos se realiza de la siguiente manera:

```
procedimiento 5, 6      ' Procedimiento que recibe dos parametros
                        ' Ambos parametros por referencia
procedimiento (a), b    ' Primer parametro por valor, segundo por
                        ' referencia
```

Note que la noción de referencia y valor debería estar simplemente para las variables. Sin embargo, es posible enviar expresiones sin paréntesis que las encierre (en el enfoque de referencia) y es completamente válido. También es posible enviar expresiones por valor.

El llamado a funciones se realiza de la siguiente manera:

```
a = funcion(4, 0) ' Funcion que recibe dos parametros
a = funcion       ' Funcion que no recibe parametros
```

El llamado de funciones se realiza **exclusivamente** en expresiones mientras que los procedimientos deben ser llamados fuera de expresiones.

```
procedimiento 5, (a)

variable = funcion(4)

if funcion(10) * funcionsinparametros then
    print
end if

if procedimiento 5, 15 then    ' Error sintactico
    print
end if

funcion(14, 17)                ' Error sintactico
```

El paso de arreglos como argumentos a una función o procedimiento se comporta de la misma manera pero tiene una estructura adicional:

```
dim a(5, 6) as integer
dim x(5, 6, 7, 8) as integer

a(3, 2) = 10

test (a(  )), x ( ), "skdfj"

sub test (a ( ) as integer, b() as integer, x$)
    print a(3, 2)                 ' Imprime 10
end sub
```

Tanto en el envío como en la declaración del arreglo en la firma de la función, para cualquier dimensión de una matriz o arreglo se debe colocar el id (SIN SUFIJO) seguido de un paréntesis que abre y luego otro que cierra.

NOTA IMPORTANTE:

A la hora de llamar procedimientos de la siguiente manera se presenta una ambigüedad:

```
id1 id2          ' Llamado al procedimiento con id1 y parametro id2
id3              ' Llamado al procedimiento con id3
```

Esto también puede ser interpretado como el llamado a tres procedimientos diferentes sin parámetros.

Así que para fines de la práctica se garantiza que **no** hay casos en los que un llamado a un procedimiento tenga como primer parámetro un id **por referencia**.

Entonces el llamado a un procedimiento podrá ser efectuado de maneras similares a las siguientes:

```
id 5, id       ' Llamado al procedimiento nombrado id, parametros 5 y id
id             ' Llamado al procedimiento nombrado id (ya no hay
               ' ambiguedad en la siguiente linea porque el primer
               ' parametro no puede ser un identificador por referencia)
id
id (id)
id (id), id
id (a()), a()  ' Los arreglos por referencia tampoco quedan permitidos
               ' en el primer parametro

id id id id    ' Cuatro llamados consecutivos a un procedimiento
```

Errores sintácticos

Los errores sintácticos se producen cuando el programa no respeta la estructura gramatical mencionada. Cuando se presenta alguno **se debe reportar y abortar el análisis**.

El formato para reportar un error sintáctico es el siguiente:

```
<fila:columna> Error sintactico: se encontro: {lexema del token encontrado}; se esperaba: {lista de símbolos/tokens esperados}.
```

Donde:

- `fila` y `columna` son los números de línea y columna donde se detectó el error.
- `lexema del token encontrado`: corresponde al lexema encontrado que no se esperaba encerrado entre comillas simples.
- `lista de símbolos/tokens esperados`: corresponde a la lista de tokens esperados o los lexemas esperados cuando estos tokens representen un único símbolo del lenguaje. Los elementos de la lista deben ir separados por comas y encerrados entre comillas simples. Por ejemplo: ';', ')', ','.

Ejemplo:

Entrada	**Salida**
`array(5, 8) ! 2`	`<1,12> Error sintactico: se encontro: '!'; se esperaba: '='.`

Consideraciones para reportar qué se encontró y qué se esperaba:

- Cuando es necesario reportar palabras reservadas se debe hacer imprimiendo la palabra encontrada en minúscula.
- Para los lexemas de los tipo de dato primitivos se debe hacer con la palabra valor_tipodedato. Mostrado en la siguiente tabla:

token	**lo que se imprime**
token_integer	valor_entero
token_long	valor_long
token_string	valor_string
token_single	valor_single
token_double	valor_double

- Cuando se reportan símbolos del lenguaje se deben imprimir sin ninguna alteración. Ej. ';', '$', '!', '<', '<>'.
- Cuando se reportan operadores que no son símbolos sino palabras, se debe hacer igual que con las palabras reservadas, en minúscula.

- Cuando hay más de un símbolo/token esperado la lista debe reportarse **lexicográficamente ordenada** (basado en ASCII) y separada por comas.
- Cuando se reporta un id se debe hacer con la palabra 'identificador'. Ej: "<12, 15> Error sintactico: se encontro 'identificador'; se esperaba '+'."

Notas

- El editor de QB64 no acepta cosas de este estilo:

```
if a
     then
else

end
            if
```

  Sin embargo, en la práctica ese tipo de programas serán válidos.

- Las expresiones pueden ser de cualquier tipo de dato. Es decir, multiplicar un integer con un string no está mal en términos sintácticos. Más adelante el analizador semántico se encargará de manejar ese tipo de cosas.

Expresiones

Existe en QB64 una serie de operadores, los cuales pueden ser divididos en binarios y unarios. Los operadores binarios como su palabra lo dice deben operar dos expresiones mientras que los unarios solamente una (a la derecha del operador en este caso).

```
a = expresion OP_BINARIO expresion
b = OP_UNARIO expresion
c = expresion OP_BINARIO (OP_UNARIO (expresion))
d = expresion OP_BINARIO OP_UNARIO expresion
```

Las **expresiones** pueden ser rodeadas por una cantidad **arbitraria** de paréntesis ('(' ')'). Desde que haya la misma cantidad abriendo y cerrando está bien formado sintácticamente.

Colocar paréntesis sin agrupar ninguna expresión **NO** está permitido. Las únicas excepciones son cuando se especifica un arreglo o matriz de cualquier número de dimensiones en los parámetros de un procedimiento o una función, o cuando se envían como argumentos. Además, la sintaxis para llamar una función sin argumentos es **solamente un identificador**.

```
dim a(5, 6) as integer
```

```
a(3, 2) = 10

test a(), "skdfj"

esto_es_un_id = esto_es_una_expresion

sub test (a() as integer, x$)
    print a(3, 2)                ' Imprime 10
end sub
```

```
a() = 0                    ' Error sintactico
(a) = 0                    ' Error sintactico
```

Cosas como la siguiente están también mal formadas sintácticamente:

```
a = 5 10                  ' Error sintactico
```

Esto se da claramente porque entre expresiones debe haber un operador binario.

Los operadores binarios son los enunciados la especificación de QB64 y que son equivalentes a los de cualquier lenguaje de programación (Ej: xor, =). Los operadores unarios son ‘not’ y ‘-’ (signo menos).

Ejemplos

Entrada	**Salida**
' Nada	El analisis sintactico ha finalizado correctamente.

Entrada	**Salida**
dim shared a as integer a = 1 print a increase (a) print a sub increase (a) a = a + 1 end sub	El analisis sintactico ha finalizado correctamente.

Entrada	Salida
a& = 1000000 b! = 1.1 c = 2.0 d% = 1000 e# = 10.000000042 f$ = "string" input a& input g$ input h(5, 6), i$(1, 1, 1) print a; b	El analisis sintactico ha finalizado correctamente.

Entrada	Salida
if 5 else then end if	<2,1> Error sintactico: se encontro: 'else'; se esperaba: '*', '+', '-', '/', '<', '<=', '<>', '=', '>', '>=', '^', 'and', 'or', 'then', 'xor'.