PC4 Propuesta formal del lenguaje Singularity

Bryan Ulate[1], Christian Rodríguez[1], Gabriel Gálvez[1] y Jostin Álvarez[1]

[1]Quantum Refraction

Agosto 27, 2020

1. Lista de tokens y su utilidad:

Token	Regex	Utilidad
SET	set	Permite asignar el valor a una palabra
TO	to	Se utiliza en conjunto con el set para especificar el valor
AS	as	Se utiliza en conjunto con el set para crear listas o matrices
LIST	list	Permite creer listas
MATRIX	matrix	Permite crear matrices de tamaño m x n
AT	at	Permite señalar una posición en un arreglo
READ	read	Permite leer una variable de la entrada estándar
PRINT	print	Permite hacer una impresión en la salida estándar
IF	if	Se utiliza para manejar el control de flujo
OTHERWISE	otherwise	Se utiliza en control de flujo en caso de que no se cumpla la condición del if
BEGIN	begin	Abre un bloque de código. Pueden ser funciones, ciclos, condicionales, entre otros.
END	end	Cierra un bloque de código. Es el opuesto al begin.
WHILE	while	Indica repetición del bloque de texto siguiente
COUNTING	counting	Se utiliza para decir que la variable siguiente será un índice, similar a un ciclo for
DEFINE	define	Permite definir funciones o procedimientos al sucederse de la palabra
FUNCTION	function	Utilizado para definir una función, al suceder la palabra `define`
DIFFERENT	different	Se utiliza en conficionales para determinar si dos valores son distintos
ANSWER	answer	Marca el final de una función. Permite indicar su valor de retorno
CALL	call	Se utiliza para llamar una función
WITH	with	Se utiliza en conjunto con la palabra `parameters` o `arguments`
PARAMETERS	parameters	Se utiliza para indicar los parámetros que se utilizarán al llamar a una función
ARGUMENTS	arguments	Se utiliza para definir cuáles argumentos necesita una función al ser llamada
NOT	not	Es el operador para la negación lógica
AND	and	Se utiliza en los condicionales como un operador lógico
OR	or	Se utiliza como el operador lógico de disyunción
XOR	xor	Se utiliza como el operador lógico de o exclusivo
ADDITION	+	Operador de suma
SUBSTRACTION	-	Operador de resta
MULTIPLICATION	*	Operador de multiplicación
DIVISION	/	Operador de división
EQUALS	=	Operador de comparación
GEQ	>=	Operador mayor o igual
LEQ	<=	Operador menor o igual

GREATER	>	Operador de mayor
GREATER	greater than	Equivalente al otro operador de mayor
LESS	<	Operador de menor
LESS	less than	Equivalente al otro operador de menor
OPEN_PARENTHESIS	(	Utilizado para delimitar parámetros de funciones o establecer la precedencia de operaciones
CLOSE_PARENTHESIS	)	Utilizado para delimitar parámetros de funciones o establecer la precedencia de operaciones
STRING	["][^"]*["]	Utilizado para delimitar una hilera de caracteres.
HASH	#	Utilizado para comentarios de una línea.
INTEGER	[0-9]+	Valores enteros para realizar operaciones aritméticas.
FLOAT	[0-9]+\.[0-9]+	Valores en punto flotante para realizar operaciones aritméticas.
IDENTIFIER	[a-zA-Z_] [a-zA-Z0-9_]*	Para atrapar los nombres de variables dados por el usuario.

Cuadro 1: Tokens, regex que los detectan y su utilidad

2. Ejemplos de código

2.1. Código de ejemplo que imprime una matriz de asteriscos:

```
define function print_asterisks with arguments n
begin
    # Recorre las filas de la matriz
    while row counting from 0 to n
    begin
        # Recorre las columnas de la matriz
        while column counting from 0 to n
        begin
            print "*"
        end
        print "\n"
    end
end

define function start
begin
    read to n
    call print_asterisks with parameters (n)
end
```

2.2. Código de ejemplo con un arreglo:

```
define function start
begin
    read to n
    set numbers as list
    while i counting from 0 to n
    begin
        read to numbers at i
    end
```

```
    set acum to 0
    while i counting from 0 to n
    begin
        set acum to acum + numbers at i
    end
    print acum
end
```

2.3. Código de ejemplo con una matriz:

```
define function start
begin
    set accumulator_matrix as matrix of size 3 by 3
    while row counting from 0 to 3
    begin
        while col counting from 0 to 3
        begin
            set accumulator_matrix at[row,col]  to row + col
        end
    end

    while row counting from 0 to 3
    begin
        while col counting from 0 to 3
        begin
            print(acumulator_matrix at [row, col])
            print(",␣")
        end
        print("\n")
    end
end
```

3. Error lógico que se detectará:

3.1. Recursión sin condición de parada

Si dentro de una función se hace un llamado a sí misma, y no hay un condicional dentro de ella, sabemos que esta es una recursión sin condición de parada que se ejecutaría infinitamente. Esto es un error y podemos reportarlo como tal. Por ejemplo:

```
define function bad_recursion
begin
    call bad_recursion
end

define function start
begin
    call bad_recursion
end
```

4. Actividades realizadas

Cuadro 2: Actividades realizadas para la primera parte del proyecto

Actividad	**Miembros responsables**
Inclusión de tokens y regex que reconocen esos tokens	Jostin, Christian, Gabriel, Bryan
Resolver la compilación del código a partir de un ejemplo	Gabriel
Organizar los archivos generados por Bison y Flex en una carpeta	Christian
Creación de ejemplos	Bryan y Gabriel
Tabla con tokens	Jostin, Christian, Gabriel, Bryan
Syntax highlighting en LaTeX	Bryan