Tecnológico de Monterrey

Evidencia Final Compiladores: Desarrollo de herramienta de soporte al proceso de análisis de imágenes (Documentación)

Luis Javier Karam Galland - A01751941

Sebastián González Villacorta - A01029746

Andreína Isabel Sanánez Rico - A01024927

Fecha de entrega: 2/5/2024

TC3002B.201

Profesor: José Mondragón

Reglas de Gramática

Las reglas de gramática proporcionadas definen la estructura sintáctica de un lenguaje de programación. Comenzando con la definición del símbolo inicial y avanzando a través de asignaciones, expresiones matemáticas, llamadas de función, y comparaciones lógicas, estas reglas abarcan una amplia gama de construcciones gramaticales. Además, incluyen características avanzadas como expresiones condicionales "if-else" y expresiones ternarias. Cada regla especifica la sintaxis y atributos asociados, proporcionando un marco claro para la comprensión y análisis del código escrito en este lenguaje.

Regla 0:

- Descripción: Esta regla define el símbolo inicial de la gramática.
- BNF:
    - S' ::= assignment

Regla 1:

- Descripción: Esta regla define una asignación de una expresión o un flujo a una variable.
- BNF:
    - assignment(attr) ::= VARIABLE SETTO expression(attr) | VARIABLE SETTO flow(attr)
    - attribute ::= {'type': 'assignment', 'variable': string, 'expression': expression_attr | flow_attr}

Regla 2:

- Descripción: Esta regla define un flujo conectado a una variable.
- BNF:
    - flow(attr) ::= VARIABLE CONNECT flow_functions(attr)
    - attribute ::= {'type': 'flow', 'variable': string, 'functions': flow_functions_attr}

Regla 3:

- Descripción: Esta regla define una secuencia de funciones de flujo conectadas.
- BNF:
    - flow_functions(attr) ::= flow_function_call(attr) CONNECT flow_functions(attr) | flow_function_call(attr)
    - attribute ::= {'type': 'flow_functions', 'functions': [flow_function_call_attr]}

Regla 4:

- Descripción: Esta regla define una llamada a una función de flujo.
- BNF:
    - flow_function_call(attr) ::= VARIABLE LPAREN params(attr) RPAREN
    - attribute ::= {'type': 'flow_function_call', 'function_name': string, 'params': params_attr}

Regla 5:

- Descripción: Esta regla define una llamada a una función de flujo sin conexión adicional.
- BNF:
    - flow_functions(attr) ::= flow_function_call(attr)

Regla 6:

- Descripción: Esta regla define una llamada a una función de flujo con sus parámetros.
- BNF:
    - flow_function_call(attr) ::= VARIABLE LPAREN params(attr) RPAREN

Regla 7:

- Descripción: Esta regla define una asignación de una expresión.
- BNF:
    - assignment(attr) ::= expression(attr
    - attribute ::= {'type': 'assignment', 'expression': expression_attr}

Regla 8:

- Descripción: Esta regla define una suma entre expresiones.
- BNF:
    - expression(attr) ::= expression(attr) PLUS term(attr)
    - attribute ::= {'type': 'expression', 'operator': 'PLUS', 'operands': [expression_attr, term_attr]}

Regla 9:

- Descripción: Esta regla define una resta entre expresiones.
- BNF:
    - expression(attr) ::= expression(attr) MINUS term(attr)
    - attribute ::= {'type': 'expression', 'operator': 'MINUS', 'operands': [expression_attr, term_attr]}

Regla 10:

- Descripción: Esta regla define una expresión como un término.
- BNF:
    - expression(attr) ::= term(attr)

Regla 11:

- Descripción: Esta regla define una expresión como una cadena de caracteres.
- BNF:
    - expression(attr) ::= string(attr)
    - attribute ::= {'type': 'expression', 'string': string_value}

Regla 12:

- Descripción: Esta regla define una cadena de caracteres.

- BNF:
    - string(attr) ::= STRING
    - attribute ::= {'type': 'string', 'value': string_value}

Regla 13:

- Descripción: Esta regla define un término como el producto de dos exponentes.
- BNF:
    - term(attr) ::= term(attr) TIMES exponent(attr)
    - attribute ::= {'type': 'term', 'operator': 'TIMES', 'operands': [term_attr, exponent_attr]}

Regla 14:

- Descripción: Esta regla define un término como el cociente de dos exponentes.
- BNF:
    - term(attr) ::= term(attr) DIVIDE exponent(attr)
    - attribute ::= {'type': 'term', 'operator': 'DIVIDE', 'operands': [term_attr, exponent_attr]}

Regla 15:

- Descripción: Esta regla define un término como un exponente.
- BNF:
    - term(attr) ::= exponent(attr)

Regla 10:

- Descripción: Esta regla define una expresión como un término.
    - BNF:
    - expression(attr) ::= term(attr)

Regla 11:

- Descripción: Esta regla define una expresión como una cadena de caracteres.
- BNF:
    - expression(attr) ::= string(attr)
    - attribute ::= {'type': 'expression', 'string': string_value}

Regla 12:

- Descripción: Esta regla define una cadena de caracteres.
- BNF:
    - string(attr) ::= STRING
    - attribute ::= {'type': 'string', 'value': string_value}

Regla 13:

- Descripción: Esta regla define un término como el producto de dos exponentes.
- BNF:
    - term(attr) ::= term(attr) TIMES exponent(attr)

- attribute ::= {'type': 'term', 'operator': 'TIMES', 'operands': [term_attr, exponent_attr]}

Regla 14:

- Descripción: Esta regla define un término como el cociente de dos exponentes.
- BNF:
  - term(attr) ::= term(attr) DIVIDE exponent(attr)
  - attribute ::= {'type': 'term', 'operator': 'DIVIDE', 'operands': [term_attr, exponent_attr]}

Regla 15:

- Descripción: Esta regla define un término como un exponente.
  - BNF:
  - term(attr) ::= exponent(attr)

Regla 16:

- Descripción: Esta regla define un exponente como la base elevada a la potencia de otro exponente.
- BNF:
  - exponent(attr) ::= factor(attr) EXP factor(attr)
  - attribute ::= {'type': 'exponent', 'base': factor_attr, 'exponent': factor_attr}

Regla 17:

- Descripción: Esta regla define un exponente como un factor.
- BNF:
  - exponent(attr) ::= factor(attr)

Regla 18:

- Descripción: Esta regla define un exponente como una expresión encerrada entre paréntesis.
- BNF:
  - exponent(attr) ::= LPAREN expression(attr) RPAREN
  - attribute ::= {'type': 'exponent', 'expression': expression_attr}

Regla 19:

- Descripción: Esta regla define un factor como un número.
- BNF:
  - factor(attr) ::= NUMBER
  - attribute ::= {'type': 'factor', 'value': number_value}

Regla 20:

- Descripción: Esta regla define un factor como una variable.
- BNF:
  - factor(attr) ::= VARIABLE

    - attribute ::= {'type': 'factor', 'variable': variable_name}

Regla 21:

- Descripción: Esta regla define una expresión como la comparación de dos expresiones utilizando el operador mayor que.
- BNF:
    - expression(attr) ::= expression(attr) GT expression(attr)
    - attribute ::= {'type': 'expression', 'operator': 'GT', 'operands': [expression_attr, expression_attr]}

Regla 22:

- Descripción: Esta regla define una expresión como la comparación de dos expresiones utilizando el operador menor que.
- BNF:
    - expression(attr) ::= expression(attr) LT expression(attr)
    - attribute ::= {'type': 'expression', 'operator': 'LT', 'operands': [expression_attr, expression_attr]}

Regla 23:

- Descripción: Esta regla define una expresión como la comparación de dos expresiones utilizando el operador mayor o igual que.
- BNF:
    - expression(attr) ::= expression(attr) GE expression(attr)
    - attribute ::= {'type': 'expression', 'operator': 'GE', 'operands': [expression_attr, expression_attr]}

Regla 24:

- Descripción: Esta regla define una expresión como la comparación de dos expresiones utilizando el operador menor o igual que.
- BNF:
    - expression(attr) ::= expression(attr) LE expression(attr)
    - attribute ::= {'type': 'expression', 'operator': 'LE', 'operands': [expression_attr, expression_attr]}

Regla 25:

- Descripción: Esta regla define una expresión como la comparación de dos expresiones utilizando el operador de igualdad.
- BNF:
    - expression(attr) ::= expression(attr) EQ expression(attr)
    - attribute ::= {'type': 'expression', 'operator': 'EQ', 'operands': [expression_attr, expression_attr]}

Regla 26:

- Descripción: Esta regla define una expresión como la comparación de dos expresiones utilizando el operador de desigualdad.
- BNF:
    - expression(attr) ::= expression(attr) NE expression(attr)
    - attribute ::= {'type': 'expression', 'operator': 'NE', 'operands': [expression_attr, expression_attr]}

Regla 27:

- Descripción: Esta regla define una expresión como la conjunción lógica de dos expresiones encerradas entre paréntesis.
- BNF:
    - expression(attr) ::= LPAREN expression(attr) RPAREN AND LPAREN expression(attr) RPAREN
    - attribute ::= {'type': 'expression', 'operator': 'AND', 'operands': [expression_attr, expression_attr]}

Regla 28:

- Descripción: Esta regla define una expresión como la disyunción lógica de dos expresiones encerradas entre paréntesis.
- BNF:
    - expression(attr) ::= LPAREN expression(attr) RPAREN OR LPAREN expression(attr) RPAREN
    - attribute ::= {'type': 'expression', 'operator': 'OR', 'operands': [expression_attr, expression_attr]}

Regla 29:

- Descripción: Esta regla define un factor como una llamada a una función.
- BNF:
    - factor(attr) ::= function_call(attr)

Regla 30:

- Descripción: Esta regla define una llamada a una función sin argumentos.
- BNF:
    - function_call(attr) ::= VARIABLE LPAREN RPAREN
    - attribute ::= {'type': 'function_call', 'function_name': string}

Regla 31:

- Descripción: Esta regla define una llamada a una función con argumentos.
- BNF:
    - function_call(attr) ::= VARIABLE LPAREN params(attr) RPAREN
    - attribute ::= {'type': 'function_call', 'function_name': string, 'params': params_attr}

Regla 32:

- Descripción: Esta regla define una lista de parámetros separados por comas.
- BNF:
    - params(attr) ::= params(attr) COMMA expression(attr)
    - attribute ::= {'type': 'params', 'expressions': [expression_attr]}

Regla 33:

- Descripción: Esta regla define un único parámetro.
- BNF:
    - params(attr) ::= expression(attr)

Rule 34:

- Descripción: Esta regla define una estructura condicional “if -else”
- BNF:
    - expression(attr) ::= IF LPAREN expression(attr) RPAREN COLON expression(attr) ELSE COLON expression(attr)
    - attribute ::= {'type': 'expression', 'condition': expression_attr, 'true_branch': expression_attr, 'false_branch': expression_attr}

Regla 35:

- Descripción: Esta regla define una expresión ternaria.
- BNF:
    - expression(attr) ::= LPAREN expression(attr) RPAREN TERNARY LPAREN expression(attr) RPAREN COLON LPAREN expression(attr) RPAREN
    - attribute ::= {'type': 'expression', 'condition': expression_attr, 'true_branch': expression_attr, 'false_branch': expression_attr}

Descripción de las Funciones Implementadas

Las funciones implementadas ofrecen una herramienta versátil para manipular imágenes y generar matrices NumPy de forma eficiente. Desde cargar imágenes desde rutas específicas hasta guardarlas en archivos, estas funciones aprovechan la biblioteca OpenCV (cv2). Además, permiten mostrar imágenes en ventanas y buscar funciones específicas dentro de OpenCV. También incluyen utilidades para generar matrices y vectores, ofreciendo flexibilidad en la manipulación de datos numéricos.

1. load_image(path): Esta función carga una imagen desde la ruta especificada utilizando la biblioteca OpenCV (cv2) y la devuelve como una matriz NumPy, que representa los píxeles de la imagen.
    - Parámetros:
        - path: Una cadena que especifica la ruta de la imagen que se va a cargar.

2. save_image(filename, image): Guarda la imagen representada por la matriz `image` en el archivo especificado por `filename`, utilizando la biblioteca OpenCV (cv2).
    - Parámetros:
        - filename: Una cadena que especifica el nombre del archivo en el que se guardará la imagen.
        - image: Una matriz NumPy que representa la imagen que se va a guardar.

3. show_image(img): Muestra la imagen representada por la matriz `img` en una ventana utilizando la biblioteca OpenCV (cv2). Espera hasta que se presione una tecla y luego cierra la ventana.
    - Parámetros:
        - img: Una matriz NumPy que representa la imagen que se va a mostrar.

4. search_cv2(function_name): Esta función intenta buscar una función en la biblioteca OpenCV (cv2) con el nombre especificado por `function_name`. Si la función existe, la devuelve; de lo contrario, devuelve `None`.
    - Parámetros:
        - function_name: Una cadena que especifica el nombre de la función que se va a buscar en la biblioteca OpenCV (cv2).

5. gen_matrix(a, b, *args): Genera una matriz NumPy de dimensiones `a` x `b` a partir de los elementos pasados como argumentos adicionales `*args`.
    - Parámetros:
        - a: Un entero que especifica el número de filas de la matriz.
        - b: Un entero que especifica el número de columnas de la matriz.
        - *args: Elementos adicionales que se utilizarán para llenar la matriz.

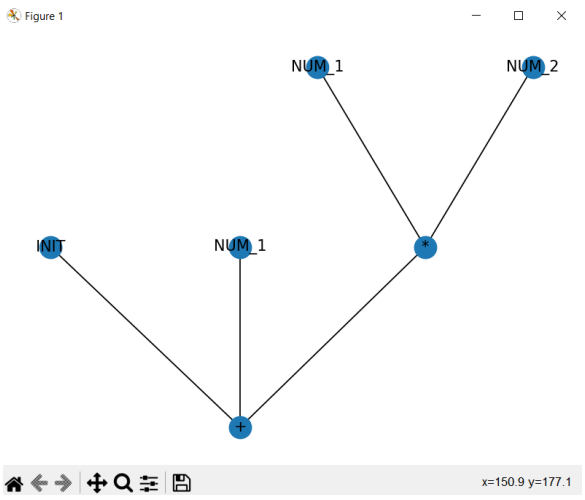
6. gen_vector(*args): Genera un vector NumPy a partir de los elementos pasados como argumentos.
    - Parámetros:
        - *args: Elementos que se utilizarán para crear el vector.

Demostración de expresiones

La demostración de expresiones ofrece ejemplos concretos de cómo se manejan operaciones, asignaciones y funciones. Desde la precedencia de operadores hasta la aplicación de filtros de OpenCV, cada caso ilustra aspectos clave del comportamiento del sistema, incluyendo la introducción de nuevas características como condicionales. Estos ejemplos brindan una visión concisa pero completa del funcionamiento del entorno de programación así como el árbol de sintaxis generado y todos sus caminos posibles.

Precedencia de operadores

input: 1+1*2

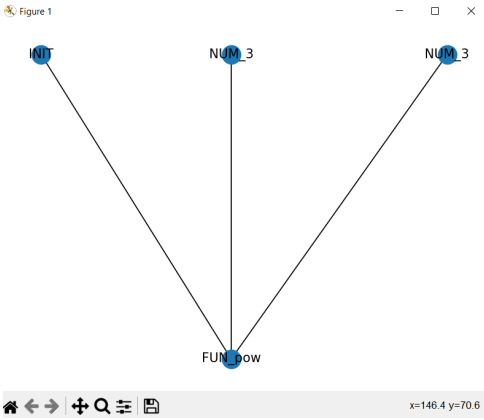



```
>1+1*2
TREE_RESULT:  3
```

Descripción: Se puede ver una vez que se inicializa la tarea, aunque se considera el NUM_1 para la suma, el árbol en vez de considerar NUM_1 otra vez para la suma en vez utiliza el resultado de la multiplicación entre NUM_1 y NUM_2 regresando como resultado correcto 3 en vez de 4.

Llamadas a funciones

input: pow(3,3)

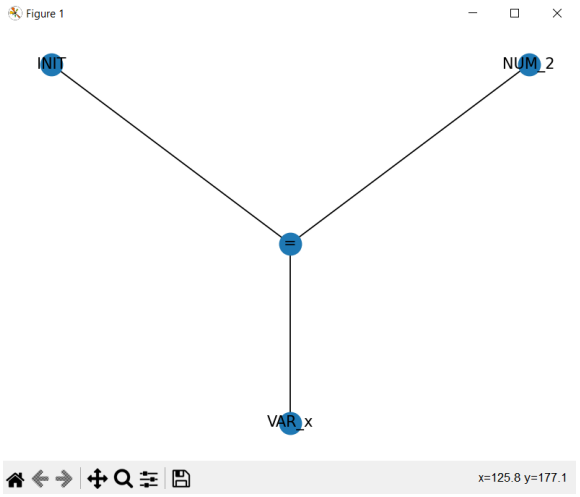



```
>pow(3,3)
TREE_RESULT:  [27.]
```

Descripción: En este caso se corre la FUN_pow en ambos NÚM 3 y regresa el resultado correcto 27.

Asignación de variables

input: x = 2

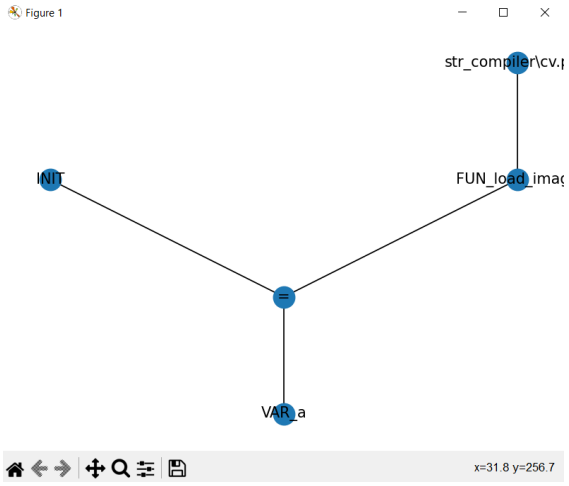



```
>x = 2
TREE RESULT:  2
```

Descripción: En este caso se le asigna el valor de NUM_2 a la variable VAR_x.

Implementación de flujos de imágenes

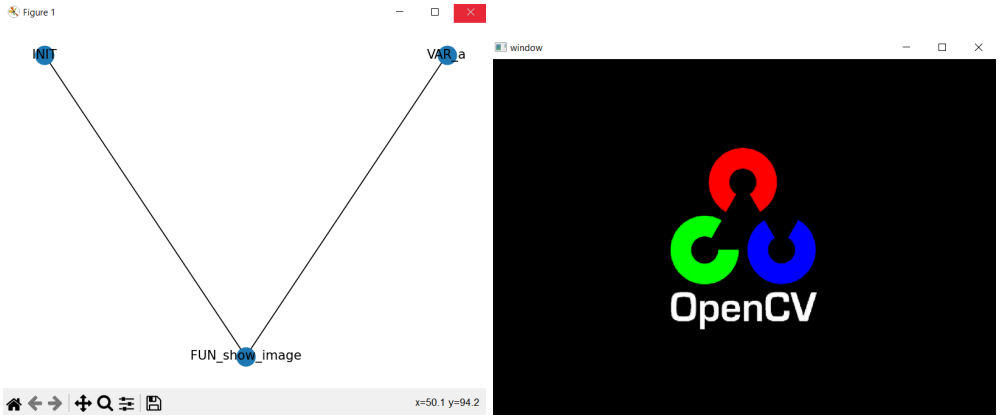
input: a = load_image(“compiler\cv.png”)




```
TREE_RESULT:  [[0 0 0]
 [0 0 0]
 [0 0 0]
 ...
 [0 0 0]
 [0 0 0]
 [0 0 0]]
```

Descripción: En este caso se le asigna el resultado de la función de FUN_load_image que carga el str_compiler\cv.png, a la variable VAR_a.

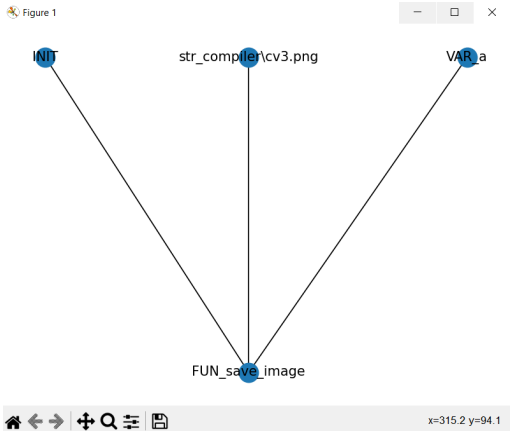
input: show_image(a)



```
TREE_RESULT:  [[0 0 0]
 [0 0 0]
 [0 0 0]
 ...
 [0 0 0]
 [0 0 0]
 [0 0 0]]
```

Descripción: la función FUN_show_image muestra la imagen guardada en VAR_a.

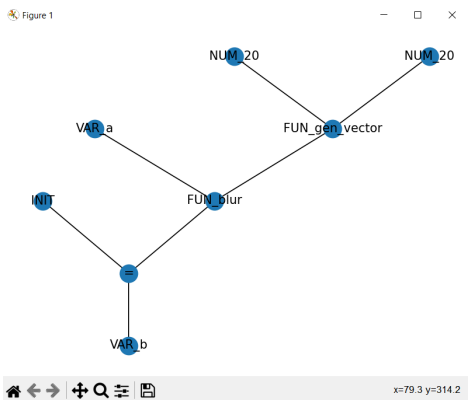
input: save_image("compiler\cv3.png",a)



Descripción: La función FUN_save_image guarda una copia de la imagen guardada en VAR_a con el path str_compiler\ y el nombre cv3.png.

Aplicación de filtros de Opencv

input: b = blur(a,gen_vector(20,20))



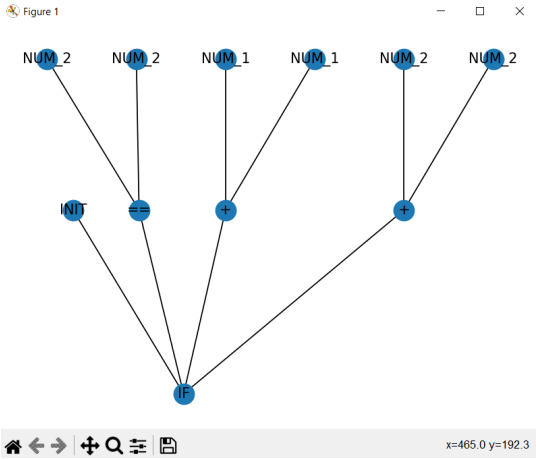
```
TREE_RESULT:  [[0 0 0]
 [0 0 0]
 [0 0 0]
 ...
 [0 0 0]
 [0 0 0]
 [0 0 0]]
```

Descripción: Se le asigna a la VAR_b el resultado de la función FUN_blur la cual toma una imagen VAR_a y un vector que se genera mediante la función FUN_gen_vector el cual toma dos números NUM_20.

Cada una de las nuevas características implementadas

Condicionales

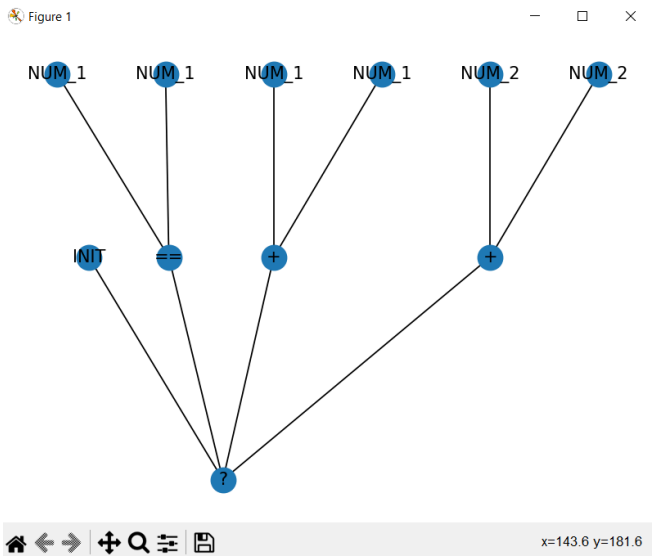
inputs: if(2==1):1+1 else: 2+2




```
>if(2==1): 1+1 else: 2+2
TREE_RESULT:  4
```

Descripción: La operación “if” recibe un valor booleano el cual en este caso se calcula con un operador de igualdad “==” que compara un valor NUM_1 con el valor NUM_2, y en el caso de que el resultado de este sea verdadero entonces se realiza la operación “+” con dos NUM_1. En el caso que el valor sea falso entonces se ejecuta la operación “else” la cual realiza la operación “+” con dos NUM 2.

input: (2==1) ? (1+1) : (2+2)




```
>(1==2) ? (1+1) : (2+2)
TREE_RESULT:  4
```

Descripción: La operación “?” toma un valor booleano que en este caso se calcula mediante el “==” que compara un valor NUM_1 y un valor NUM_2 , en el caso de que este sea verdadero entonces se corre la operación “+” que suma dos números NUM_1 y en caso de que este sea falso entonces se corre la operación “+” pero en este caso con dos números NUM_2.