NeuroPY

Tecnológico de Monterrey

A01250658

Juventino Gutierrez Romo

05 de Junio 2023

Índice

Índice 2
Descripción del proyecto 3
  Análisis de requerimientos 4
  Aprendizajes 4
  Avances: 5
Descripción del Lenguaje 6
Descripción del Compilador 8
  Librerías 8
  Analisis Lexico 8
  Análisis de Sintaxis 10
Descripción de la Máquina Virtual 25
Pruebas 26
Manual de Usuario 33

Descripción del proyecto

El propósito de nuestro proyecto es desarrollar e implementar nuestro propio lenguaje de programación, el cual hemos creado utilizando Python. El proceso comienza al recibir una entrada de texto que consiste en una secuencia de tokens que siguen un orden específico definido por las reglas del lenguaje. Luego, utilizamos la sintaxis diseñada para interpretar estos tokens y generar la estructura del programa.

Una vez que hemos realizado el análisis sintáctico, entramos en la fase de semántica, donde generamos el código intermedio. Este código actúa como una representación de las operaciones realizadas en el programa. Finalmente, nuestra máquina virtual interpreta este código intermedio y ejecuta las operaciones correspondientes.

En resumen, nuestro objetivo es crear un lenguaje de programación propio utilizando Python, donde podemos recibir una entrada de texto con tokens, analizar su sintaxis y generar código intermedio mediante la semántica. A través de nuestra máquina virtual, logramos interpretar y ejecutar las operaciones generadas en el código intermedio

Análisis de requerimientos

- Operaciones aritméticas y asignación.
- Expresiones comparativas.
- condiciones.
- ciclos For y While
- Gestión de memoria en ambientes anidados
- funciones
- recursividad
- Uso de Arreglos
- Input del usuario
- Output a la terminal
- Impresión de Errores

Aprendizajes

Durante este proyecto hubo demasiados problemas que al final se pudieron solucionar, el principal problema fue que mi compañero de proyecto no podía apoyarme, yo no sé sus razones pero básicamente me dejó el proyecto solo. Es fácil decir que es su problema pero la verdad es que si me afecto, tenía que esperarme para que el trabajo fuera parejo eso ocasionó que me fuera retrasando con las entregas y no ir al tanto de como quería. De todo eso aprendí que si puedes ser comprensivo con tus compañeros, pero es importante poner un límite ya que la única responsabilidad es conmigo de entregar este proyecto. Pero con el compilador aprendí demasiado a cómo debuggear ya que tenía muchos problemas con la generación de cuádruplos, pero de conceptos computacionales aprendí sobre estructura de datos y algoritmos y un mejor entendimiento sobre compiladores. Todavía hay muchas cosas por aprender, ya sea sobre compiladores, estructuras de datos o algoritmos, pero este proyecto me dio una base sólida para seguir explorando y profundizando en estos temas.

Avances:

16 Abril 2023: Empecé con el lenguaje patito y lo modifique para que sirva para mi compilador, ahí empecé a diseñar mi syntaxis.

23 Abril 2023: Agregamos cubo semántico y empezar a trabajar con la semántica

30 Abril 2023: Avance de semántica para trabajar con expresiones

7 Mayo 2023: Avance de generación de cuádruplos

14 Mayo 2023: Seguí trabajando en cuadruplos, impresión de cuádruplos

22 Mayo 2023: Trabaje en el Mapa de memoria de la máquina virtual

29 Mayo 2023: Avance de Máquina virtual

02 Junio 2023: Pruebas y documentación

Descripción del Lenguaje

Nombre del Lenguaje: NeuroPY

Descripción genérica:

Es un lenguaje de programación con el funcionamiento base como c,este permite:

- Crear y utilizar variables tipo int, float y char.
- Puede leer y escribir datos como int, float y char.
- Hace operaciones aritméticas, lógicas, relacionales.
- Cuenta con estatutos de Decisión.
- Declara Funciones retornables tipo int, float, char y no retornables como void.
- Maneja elementos no-tóxicos como arreglos

Errores:

1. **(Syntax error at line ...)** Este error ocurre cuando el analizador sintáctico encuentra una secuencia de tokens que no se ajusta a las reglas gramaticales del lenguaje. Por ejemplo, puede que falte un punto y coma al final de una sentencia, o que un paréntesis no tenga su par correspondiente.
2. **(The amount of parameters of the function... do not match with those of its call:)** Este error ocurre cuando se llama a una función con un número incorrecto de argumentos.

3. **(The variable ... was previously declared:)** Este error ocurre cuando se intenta declarar una variable que ya ha sido declarada antes en el mismo ámbito.
4. **(The parameters of the function... do not match in their types:)** Este error ocurre cuando los tipos de los parámetros pasados a una función no corresponden a los tipos de los parámetros definidos en la declaración de la función.
5. **(Error in type operations:)** Este error ocurre cuando se intenta realizar una operación en tipos de datos que no son compatibles, como la suma de un entero y una cadena.
6. **(The variable... was not declared in the non-conditional loop:)** Este error ocurre cuando se intenta utilizar una variable en un bucle que no ha sido declarada dentro de ese bucle.
7. **(The variable... must be an integer to be used in a non-conditional loop:)** Este error ocurre cuando se intenta utilizar una variable no entera como contador en un bucle.
8. **(The variable ... can't be used for a decision:)** Este error ocurre cuando se intenta utilizar una variable en una expresión condicional que no es de un tipo de dato que pueda evaluarse a verdadero o falso.
9. **(The returned value is not compatible with the function type:)** Este error ocurre cuando el tipo de valor que una función devuelve no coincide con el tipo de valor que se especificó en su definición.
10. **(Variable... must be declared before it is used:)** Este error ocurre cuando se intenta utilizar una variable antes de que haya sido declarada.
11. **(Assignment cannot be performed due to type compatibility:)** Este error ocurre cuando se intenta asignar un valor a una variable, pero el tipo del valor no coincide con el tipo de la variable.
12. **(Stack overflow of... constants y Stack overflow of... variables:)** Estos errores ocurren cuando se intenta almacenar más constantes o variables de las que la memoria puede manejar.
13. **(You cannot use the function... in an expression y You cannot use the function... in a statement:)** Estos errores ocurren cuando se intenta utilizar una función en un contexto donde no está permitido.

14. **(The function... was not declared:)** Este error ocurre cuando se intenta utilizar una función que no ha sido declarada previamente.
15. **(Variable... not declared:)** Este error ocurre cuando se intenta utilizar una variable que no ha sido declarada.
16. **(The array... has already been declared and The variable... has already been declared:)** Estos errores ocurren cuando se intenta declarar una variable o arreglo que ya ha sido declarado anteriormente.
17. **(The function... should not have return statements and the function... does not have a return statement:)** Estos errores ocurren cuando una función no tiene una declaración de retorno cuando debería, o tiene una cuando no debería.
18. **(The function... has already been declared:)** Este error ocurre cuando se intenta declarar una función que ya ha sido declarada previamente.

Descripción del Compilador

Librerías

Para desarrollar el compilador, se utilizó la biblioteca PLY (Python Lex-Yacc), que es una implementación en Python de las herramientas Lex y Yacc. PLY proporciona una manera sencilla y flexible de construir analizadores léxicos (lexers) y sintácticos (parsers) para el procesamiento de lenguajes de programación.

Analisis Lexico

Token	Pattern
PROGRAM	'Program'
VARIABLES	'Var'
FUNCTION	'Function'

MAIN	'Main'
RETURN	'Return'
READ	'Read'
WRITE	'Write'
IF	'If'
THEN	'Then'
ELSE	'Else'
WHILE	'While'
DO	'Do'
FROM	'From'
TO	'To'
SEMICOLON	';'
COMA	','
COLON	':'
L_BRACE	'{'
R_BRACE	'}'
L_PAREN	'('
R_PAREN	')'
L_BRACKET	'['
R_BRACKET	']'
EQUAL	'='
PLUS	'+'
MINUS	'-'
MULT	'*'

DIV	'/'
AND	'&'
OR	'|'
LESS	'<'
GREATER	'>'
LESSEQUAL	'<='
GREATEREQUAL	'>='
DIFF	'!='
EQUALTO	'=='
INT	'[-]?[0-9]+'
FLOAT	'[-]?[0-9]+([.][0-9]+)'
CHAR	'('[^']')'
STRING	'"[\w\d\s,.]*"'
INT	'Int'
FLOAT	'Float'
CHAR	'Char'
VOID	'Void'
ID	'([a-z][a-zA-Z0-9]*)'

Análisis de Sintaxis

Gramática Formal

Unset

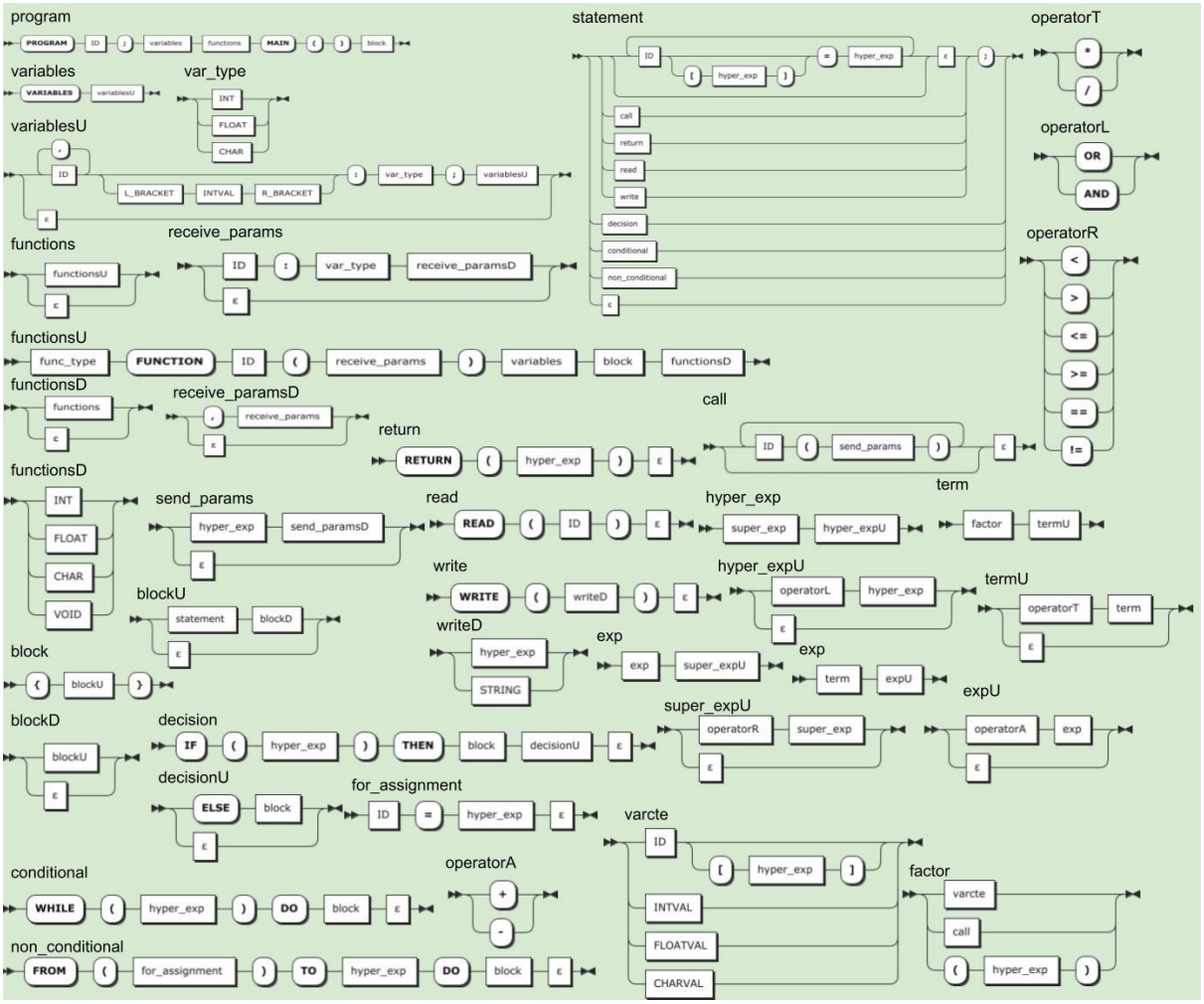
```
program    : PROGRAM ID SEMICOLON variables functions MAIN
L_PAREN R_PAREN block empty
```

```
variables  : VARIABLES variablesU
           | empty

variablesU   : variablesD
           | empty

variablesD   : ID COMA variablesD
           | ID COLON var_type SEMICOLON variablesU
           | ID L_BRACKET INTVAL R_BRACKET COLON var_type
SEMICOLON variablesU

functions  : functionsU
           | empty

functionsU   : func_type FUNCTION ID L_PAREN receive_params
R_PAREN variables block functionsD

functionsD   : functions
           | empty

func_type  : INT empty
           | FLOAT empty
           | CHAR empty
           | VOID empty

var_type   : INT empty
           | FLOAT empty
           | CHAR empty

receive_params : ID COLON var_type receive_paramsD empty
                | empty

receive_paramsD : COMA receive_params empty
                | empty

send_params  : hyper_exp send_paramsD empty
           | empty
```

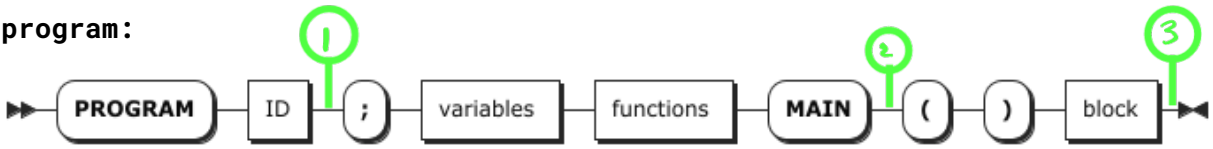
```
send_paramsD : COMA send_params empty
             | empty

block        : L_BRACE blockU R_BRACE empty

blockU       : statement blockD empty
             | empty

blockD       : blockU empty
             | empty

statement    : assignment SEMICOLON empty
             | call SEMICOLON empty
             | return SEMICOLON empty
             | read SEMICOLON empty
             | write SEMICOLON empty
             | decision empty
             | conditional empty
             | non_conditional empty
             | empty

assignment     : ID EQUAL hyper_exp assignment empty
             | ID L_BRACKET hyper_exp R_BRACKET EQUAL hyper_exp
assignment empty

call         : ID L_PAREN send_params R_PAREN call empty

return       : RETURN L_PAREN hyper_exp R_PAREN empty

read         : READ L_PAREN ID R_PAREN empty

write        : WRITE L_PAREN writeD R_PAREN empty

writeD       : hyper_exp empty
             | STRING empty

decision     : IF L_PAREN hyper_exp R_PAREN THEN block decisionU
empty
```

```
decisionU  : ELSE block empty
           | empty

conditional  : WHILE L_PAREN hyper_exp R_PAREN DO block empty

non_conditional : FROM L_PAREN for_assignment R_PAREN TO
hyper_exp DO block empty

for_assignment : ID EQUAL hyper_exp empty

operatorA  : PLUS empty
           | MINUS empty

operatorT  : MULT empty
           | DIV empty

operatorL  : OR empty
           | AND empty

operatorR  : LESS empty
           | GREATER empty
           | LESSEQUAL empty
           | GREATEREQUAL empty
           | EQUALTO empty
           | DIFF empty

hyper_exp  : super_exp hyper_expU

hyper_expU : operatorL hyper_exp empty
           | empty

super_exp  : exp super_expU

super_expU : operatorR super_exp empty
           | empty

exp        : term expU

expU       : operatorA exp
```

```
            | empty

term        : factor termU

termU       : operatorT term
            | empty

factor      : varcte empty
            | call empty
            | L_PAREN hyper_exp R_PAREN empty

varcte      : ID empty
            | ID L_BRACKET hyper_exp R_BRACKET empty
            | INTVAL empty
            | FLOATVAL empty
            | CHARVAL empty

empty       :
```



program:

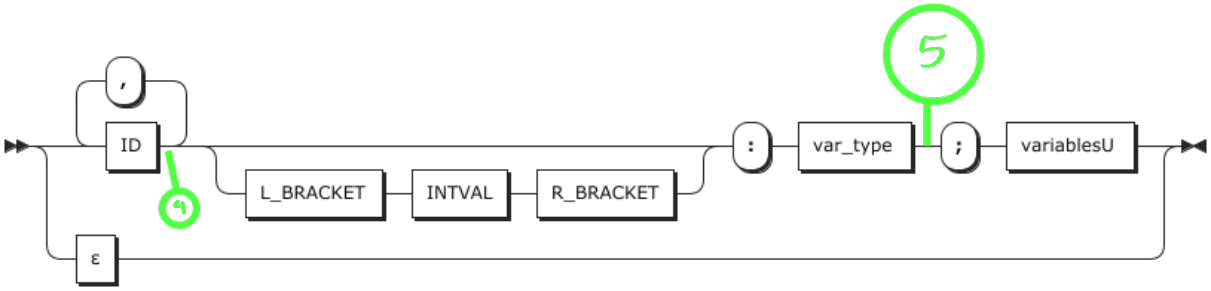


1 - Inicia el programa

2 - Inicial Main()

3 - Terminar el programa

variables:

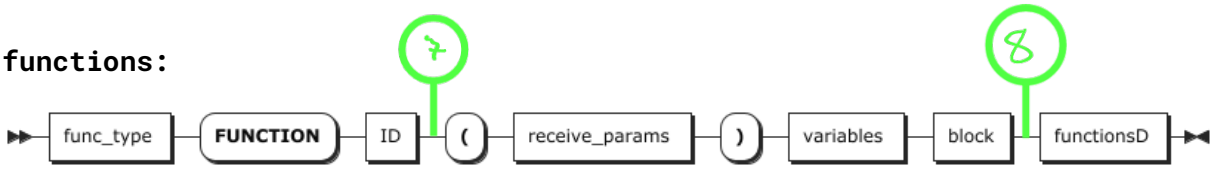


4 - Añade de Variable a Var Stack

5 - Añade variables a la tabla de variables

6 - Añade arreglos a la tabla de variables

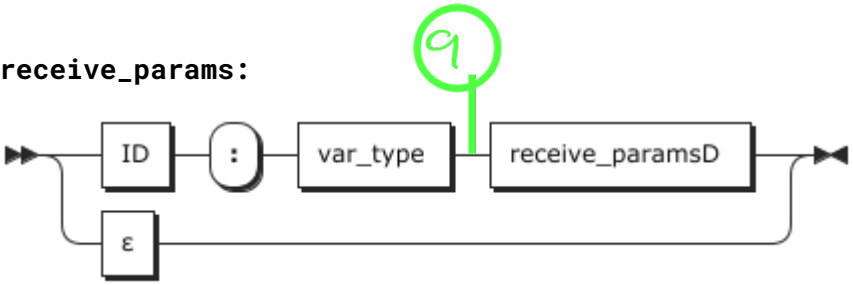
functions:



7 - Asignar a las variables globales el nombre y tipo de la función, respectivamente

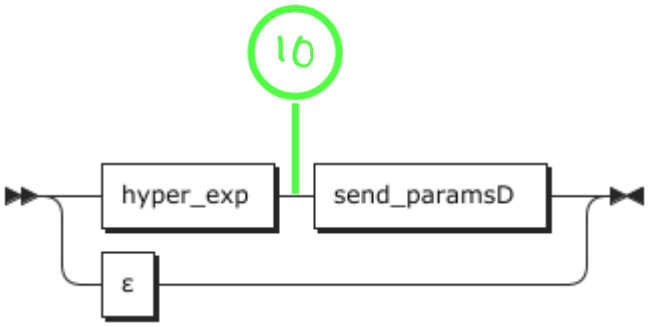
8 - En caso de que no haya otra función con el mismo nombre, se incluye la función en la tabla de variables.

receive_params:



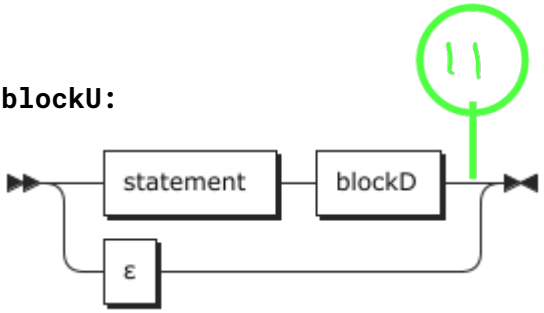
9-Recibe una variable como parámetro en una función y esta variable no existe ni en el contexto local ni como variable global, se añaden los tipos correspondientes a la tabla de variables

send_params:



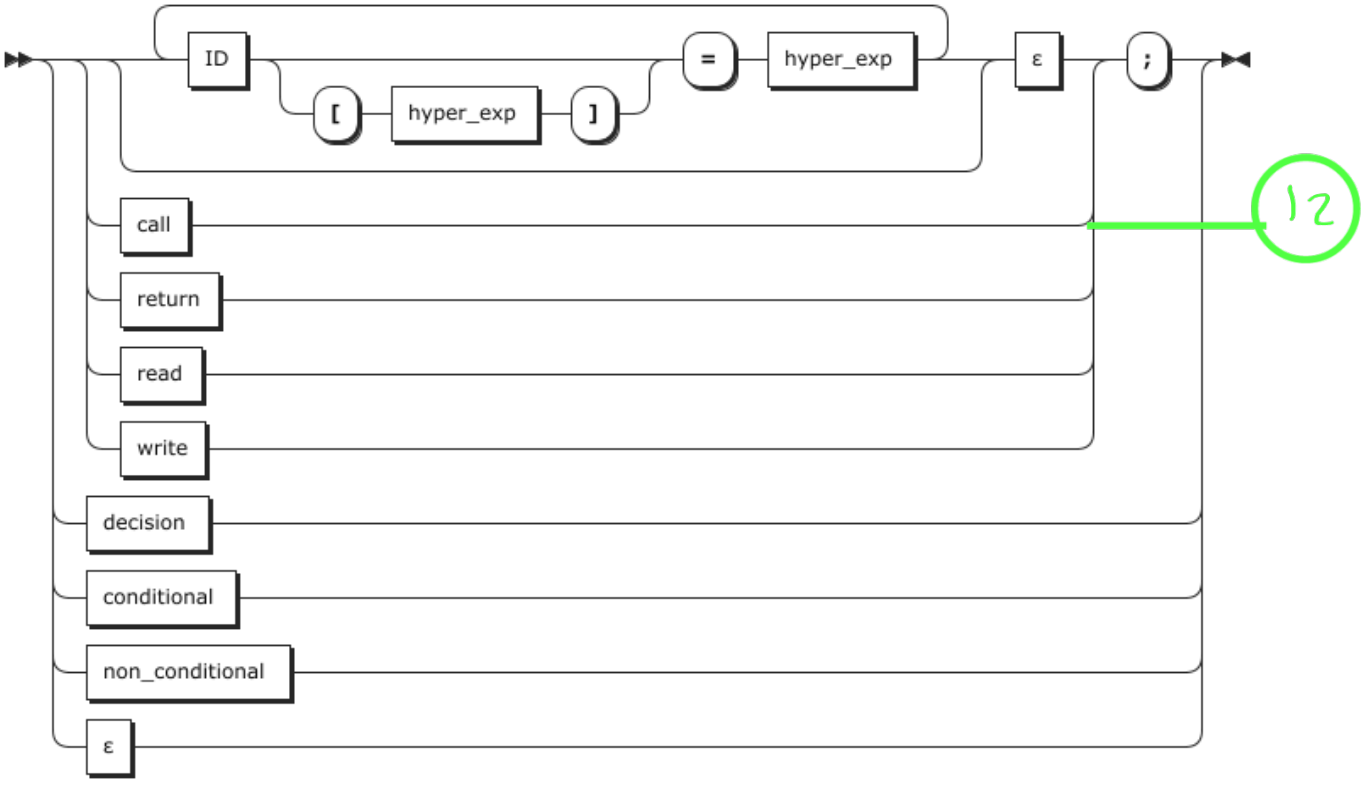
10 - Verifica si existe un en registro en la tabla de parámetros

blockU:



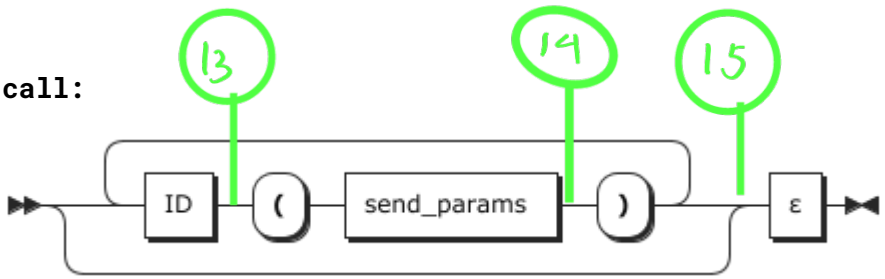
11 - Vacia las pilas

statement:



12- Usa una función en un estatuto que no es void

call:

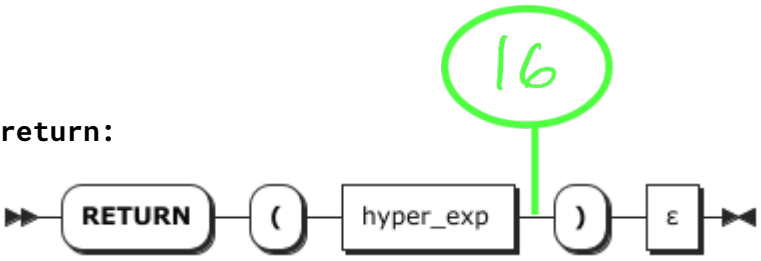


13 Llamada ERA

14 Valida si no se envían parámetros

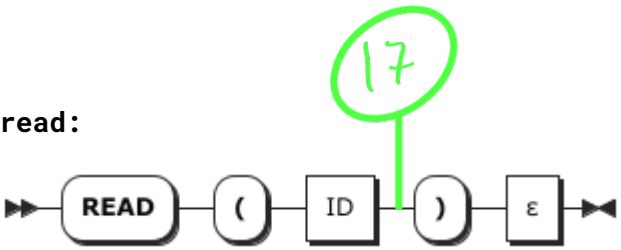
15 Llamada GOSUB

return:



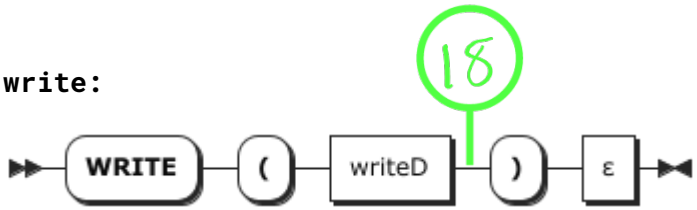
16 Verifica si la variable que se va a retornar tiene el mismo tipo que la función

read:



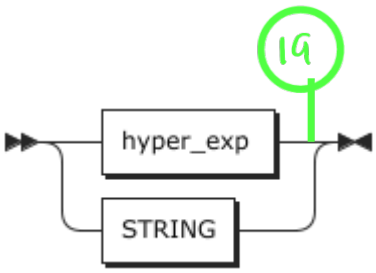
17 Válida si mi ID está entre las variables declaradas globales o locales

write:



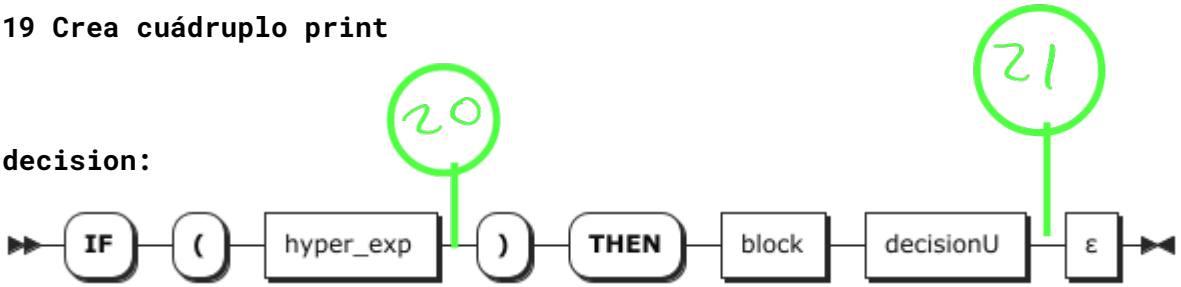
18 Crea cuádruplo print

writeD:



19 Crea cuádruplo print

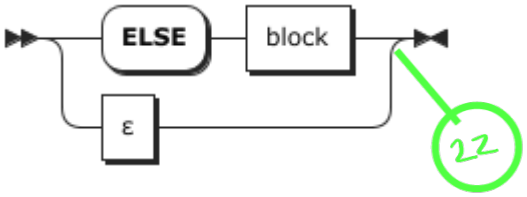
decision:



20 Valida el valor

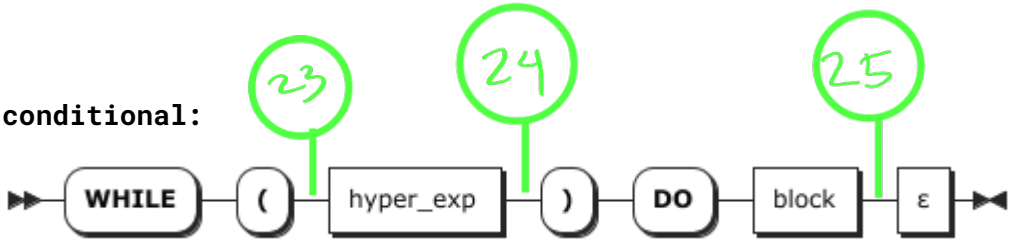
21 pop.jumpstack = len(cuadruplos)

decisionU:



22 Crea cuádruplo Goto

conditional:

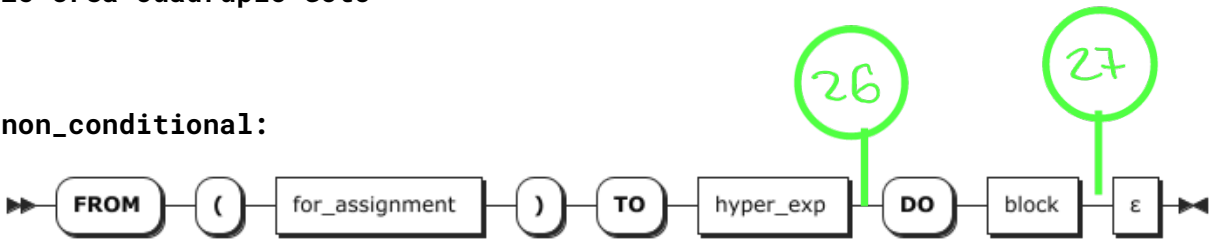


23 condicional antes

24 Crea cuádruplo GotoF

25 Crea cuádruplo Goto

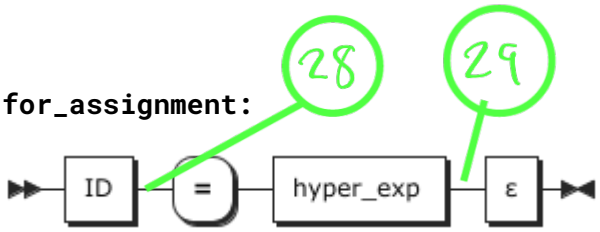
non_conditional:



26 OpStack.append(p[-1])

27 Crea cuádruplo Goto

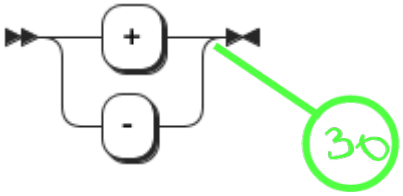
for_assignment:



28 OpStack.append(p[-1])

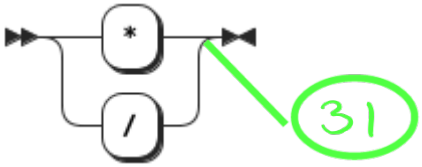
29 Genera un cuádruplo para la asignación

operatorA:



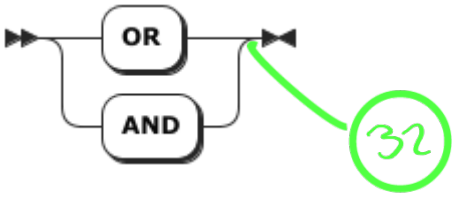
30 Genera un cuádruplo para la asignación

operatorT:



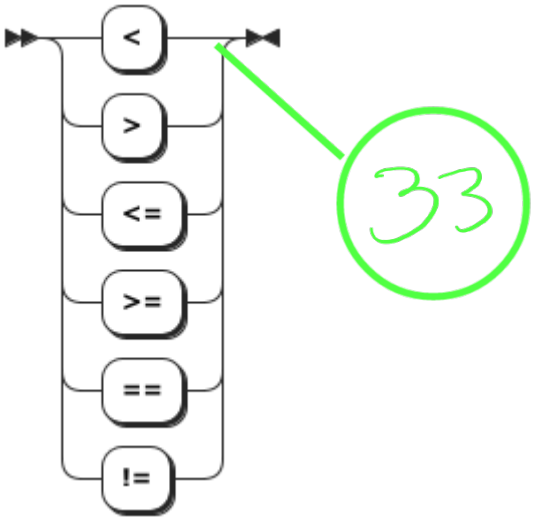
31 Genera un cuádruplo para la asignación

operatorL:



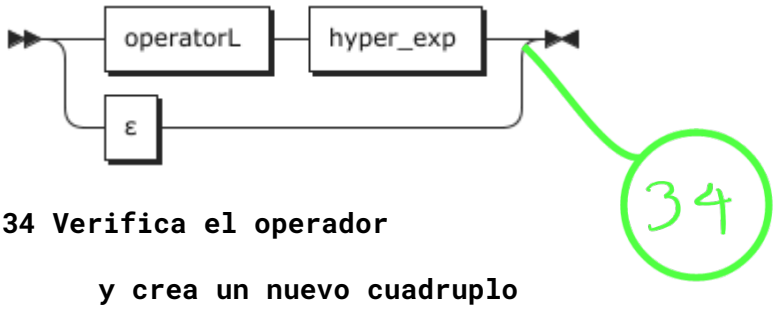
32 Genera un cuádruplo para la asignación

operatorR:



33 Genera un cuádruplo para la asignación

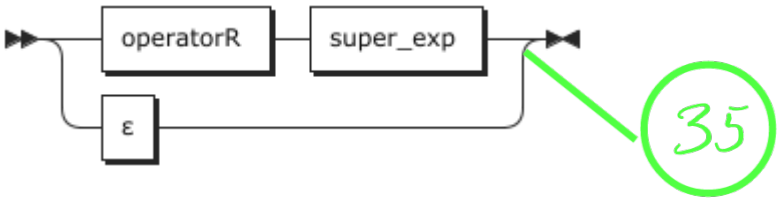
hyper_expU:



34 Verifica el operador

y crea un nuevo cuadruplo

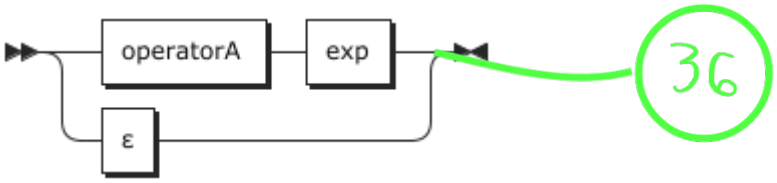
super_expU:



35 Verifica si hay elementos en la pila operadores

y agrega un nuevo objeto cuadruplo

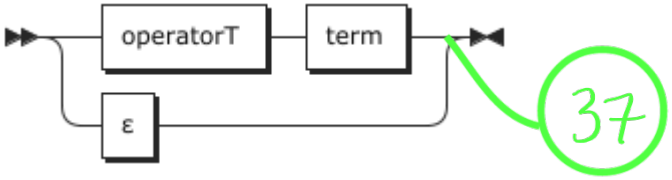
expU:



36 Verifica si hay elementos en la pila de operadores

y agrega un nuevo cuadruplo

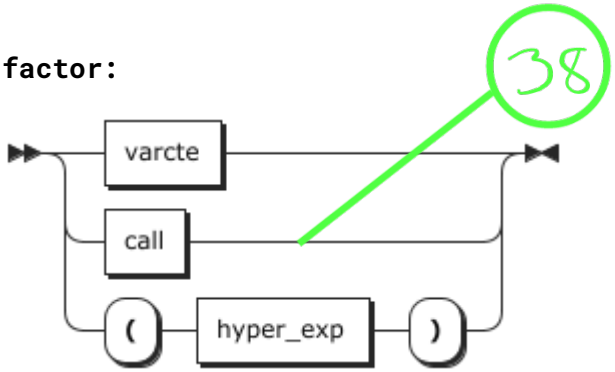
termU:



37 Verifica si hay elementos en la pila de operadores

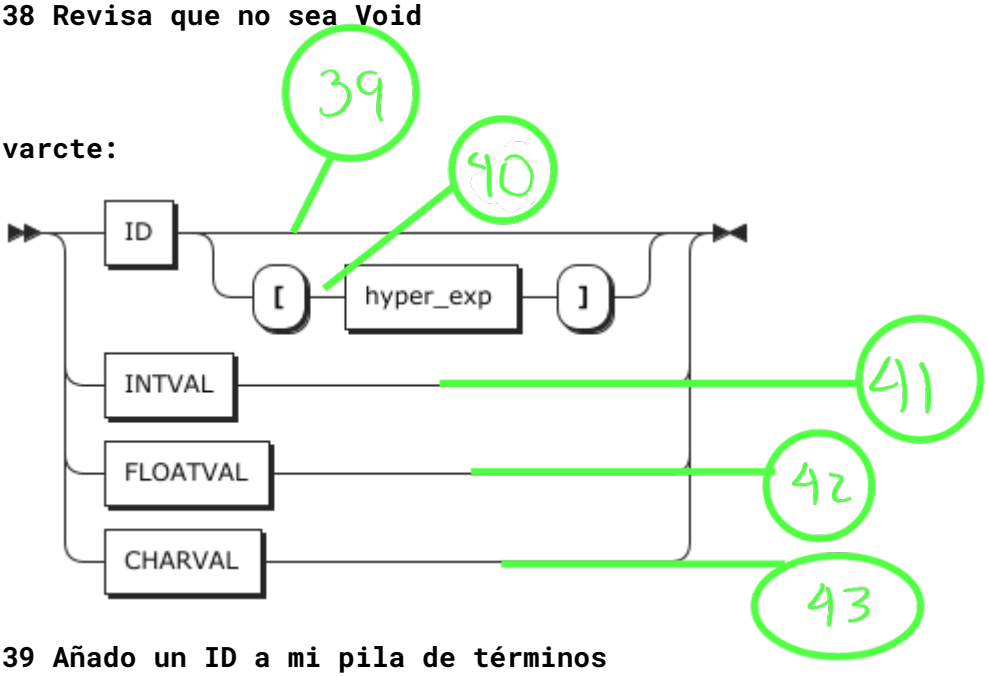
y agrega un nuevo cuadruplo

factor:



38 Revisa que no sea Void

varcte:



39 Añado un ID a mi pila de términos

40 Hace fondo falso

41 Añado ENTERO a la tabla de constantes

42 Añado Float a la tabla de constantes

43 Añado char a la tabla de constantes

Cubo Semantico

Type 1	Type 2	Operator	Resulting Type
Int	Int	+, -, *, /	Int
Int	Float	+, -, *, /	Float
Int	Float	==, !=	Int
Int	Char	-	Error
Float	Int	+, -, *, /	Float
Float	Float	+, -, *, /	Float
Float	Float	==, !=	Int
Float	Char	-	Error
Char	Int	-	Error
Char	Float	-	Error
Char	Char	==, !=	Int
Char	Char	+, -, *, /	Error

1. **Tabla de variables:** Esta es una estructura de datos que contiene información sobre todas las variables utilizadas en el programa. Para cada variable, se registra su nombre, su tipo de dato y su dirección de memoria. En este código, la tabla de variables es un diccionario que almacena las variables de la función junto con su tipo y su dirección de memoria.
2. **Tabla de constantes:** Esta es similar a la tabla de variables pero para constantes. La tabla de constantes es un diccionario que contiene constantes de diferentes tipos (Entero, Flotante, Caracter, String) junto con sus direcciones de memoria.
3. **Cuádruplos:** Los cuádruplos son una forma común de representar las instrucciones en la generación de código intermedio en un compilador. Cada cuádruplo contiene cuatro campos: operador, operando 1, operando 2 y resultado. Los cuádruplos representan una operación que se realizará. En este compilador, el array cuádruplo se llena con objetos de tipo cuádruplo que representan las instrucciones del programa.
4. **Pilas:** Las pilas se utilizan para varias funciones en este compilador, como mantener un seguimiento de las variables de ciclo , operadores , términos y tipos . Las pilas son útiles en la compilación porque permiten un acceso rápido y ordenado a la información, y se utilizan comúnmente en la resolución de expresiones y en la implementación de estructuras de control, como ciclos y condicionales.

En cuanto a la **arquitectura de la memoria**, este código tiene una división de la memoria en distintas secciones para diferentes tipos de datos y usos:

- Memoria global (para enteros, flotantes y caracteres).
- Memoria local (para enteros, flotantes y caracteres).
- Memoria constante (para enteros, flotantes y caracteres).
- Memoria para Strings.

Estos valores de memoria representan el comienzo de cada sección de memoria para cada tipo de dato. Durante la ejecución del programa, estas direcciones de memoria se utilizan para almacenar y recuperar valores.

Direcciones de memoria:

```
global_mem_int = 1000
global_mem_float = 2000
global_mem_char = 3000
local_mem_int = 4000
local_mem_float = 5000
local_mem_char = 6000
constant_mem_int = 7000
constant_mem_float = 8000
constant_mem_char = 9000
mem_strings = 10000
```

1000-1999 Int Global	2000-2999 Float Global	3000-3999 Char Global	4000-4999 Int Local	5000-5999 Float Local	6000-6999 Char Local	7000-7999 Int Constantes	8000-8999 Float Constantes	9000-9999 Char Constantes	1000 Strings Constantes

Descripción de la Máquina Virtual

Lenguaje y Librerías usadas:
La máquina virtual se desarrolló en windows y fue escrita en el lenguaje python.

La máquina virtual implementada tiene como objetivo ejecutar los cuádruplos generados durante la compilación del programa.

Administración de Memoria

1. Estructuras de datos para manejo de scopes:
    - local_variables: Es una lista que contiene diccionarios representando las variables locales en diferentes scopes. Cada diccionario almacena las direcciones de memoria y los valores asignados a las variables locales en ese scope específico.

- local_table_dictionary: Es un diccionario que almacena las variables locales en el scope actual. Las claves son las direcciones de memoria y los valores son los valores asignados a esas variables.
- variable_table: Es un diccionario que almacena información sobre las variables del programa. Contiene la información de las variables globales y locales, incluyendo las direcciones de memoria y los tipos de datos.

2. Asociación entre direcciones virtuales y reales:
    - Durante la ejecución, las direcciones de memoria virtuales utilizadas en la compilación se asocian con las direcciones de memoria reales. Para esto, se utilizan las variables super_variable_table y super_constant_table, que son diccionarios que mapean las direcciones de memoria virtuales a los valores reales.
    - Estas estructuras de datos permiten acceder a los valores almacenados en memoria durante la ejecución del programa, utilizando las direcciones de memoria virtuales proporcionadas en los cuádruplos.

En resumen, durante la ejecución de la máquina virtual, se utilizan estructuras de datos como listas y diccionarios para gestionar los scopes y las variables locales. Las direcciones de memoria virtuales se asocian con las direcciones de memoria reales mediante el uso de diccionarios. Esto permite acceder y asignar valores a las variables en tiempo de ejecución de manera eficiente y controlada.

1001-2000 Int Global	2001-3000 Float Global	3001-4000 Char Global	4001-5000 Int Local	5001-6000 Float Local	6001-7000 Char Local	7001-8000 Int Global	8001-900 Float Global	9001-1000 Char Global	1001-1100 String Global

Pruebas

Factorial (Iterative).neupy

Factorial (Iterative).neupy

```
1   Program factorialIterative;
2
3   Var
4       n : Int;
5       result : Int;
6
7   Int Function factorial(m : Int)
8   Var
9       i : Int;
10  {
11      result = 1;
12      From (i = 1) To m Do
13      {
14          result = result * i;
15      }
16      Return(result);
17  }
18
19  Main()
20  {
21      Read(n);
22      result = factorial(n);
23      Write(result);
24  }
25
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
3 ['<', 4002, 4001, 4003]
4 ['GOTOF', 4003, None, 10]
5 ['*', 1002, 4002, 4004]
6 ['=', 4004, None, 1002]
7 ['+', 4002, 7001, 4005]
8 ['=', 4005, None, 4002]
9 ['GOTO', None, None, 3]
10 ['RETURN', None, None, 1002]
11 ['ENDFUNC', None, None, None]
12 ['READ', None, None, 1001]
13 ['ERA', 'factorial', None, None]
14 ['PARAM', 1001, None, 'PARAM1']
15 ['GOSUB', 'factorial', None, None]
16 ['=', 1003, None, 1004]
17 ['=', 1004, None, 1002]
18 ['WRITE', None, None, 1002]
19 ['END', None, None, None]
> 9
40320
PS C:\Users\juven\Documents\GitHub\Compiler>
```

Fibonacci (Iterative).neupy

Fibonacci (Iterative).neupy

```
1   Program fibonacciIterative;
2   Var
3       n : Int;
4       fib : Int;
5   Int Function fibonacci(m : Int)
6   Var
7       i : Int;
8       fib1 : Int;
9       fib2 : Int;
10  {
11      fib1 = 0;
12      fib2 = 1;
13      If (n == 0) Then
14      {
15          Return(fib1);
16      }
17      From (i = 2) To n Do
18      {
19          fib = fib1 + fib2;
20          fib1 = fib2;
21          fib2 = fib;
22      }
23      Return(fib2);
24  }
25  Main()
26  {
27      Read(n);
28      fib = fibonacci(n);
29      Write(fib);
30  }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
9 ['+', 4003, 4004, 4007]
10 ['=', 4007, None, 1002]
11 ['=', 4004, None, 4003]
12 ['=', 1002, None, 4004]
13 ['+', 4002, 7002, 4008]
14 ['=', 4008, None, 4002]
15 ['GOTO', None, None, 7]
16 ['RETURN', None, None, 4004]
17 ['ENDFUNC', None, None, None]
18 ['READ', None, None, 1001]
19 ['ERA', 'fibonacci', None, None]
20 ['PARAM', 1001, None, 'PARAM1']
21 ['GOSUB', 'fibonacci', None, None]
22 ['=', 1003, None, 1004]
23 ['=', 1004, None, 1002]
24 ['WRITE', None, None, 1002]
25 ['END', None, None, None]
> 9
21
PS C:\Users\juven\Documents\GitHub\Compiler>
```

Linear Search.neupy

Linear Search.neupy

```
12  {
13      arr[0] = 3;
14      arr[1] = 4;
15      arr[2] = 2;
16      arr[3] = 7;
17      arr[4] = 4;
18      arr[5] = -2;
19      arr[6] = 2;
20      arr[7] = 17;
21      arr[8] = -9;
22      arr[9] = -8;
23
24      searchValue = 7;
25      found = 0;
26      i = 0;
27
28      From (i = 0) To 9 Do
29      {
30          If (arr[i] == searchValue) Then
31          {
32              found = 1;
33              value = arr[i];
34
35          }
36      }
37
38      If (found == 1) Then
39      {
40          Write("Element found");
41          Write(value);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
38 ['SUM_BASE', 1001, 1012, 1027]
39 ['==', '(1027)', 1013, 1028]
40 ['GOTOF', 1028, None, 45]
41 ['=', 7003, None, 1014]
42 ['VERIFY', 1012, 0, '10']
43 ['SUM_BASE', 1001, 1012, 1029]
44 ['=', '(1029)', None, 1015]
45 ['+', 1012, 7003, 1030]
46 ['=', 1030, None, 1012]
47 ['GOTO', None, None, 35]
48 ['==', 1014, 7003, 1031]
49 ['GOTOF', 1031, None, 53]
50 ['WRITE', None, None, 10001]
51 ['WRITE', None, None, 1015]
52 ['GOTO', None, None, 54]
53 ['WRITE', None, None, 10002]
54 ['END', None, None, None]
Element found
7
PS C:\Users\juven\Documents\GitHub\Compiler>
```

BubbleSort.neupy

BubbleSort.neupy

```
1   Program bubbleSort;
2
3   Var
4       arr[10] : Int;
5       n : Int;
6       i : Int;
7       j : Int;
8       temp : Int;
9
10  Main()
11  {
12      arr[0] = 3;
13      arr[1] = 4;
14      arr[2] = 2;
15      arr[3] = 7;
16      arr[4] = 4;
17      arr[5] = -2;
18      arr[6] = 2;
19      arr[7] = 17;
20      arr[8] = -9;
21      arr[9] = -8;
22
23      Write("unsorted");
24      From (i = 0) To 9 Do
25      {
26          Write(arr[i]);
27      }
28
29      From (i = 0) To 10 Do
30      {
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
3
4
2
7
4
-2
2
17
-9
sorted
-9
-2
2
2
3
4
4
7
17
PS C:\Users\juven\Documents\GitHub\Compiler>
```

Factorial (Recursive)

Factorial (Recursive).neupy

```
1   Program factorialRecursive;
2
3   Var
4       n : Int;
5       result : Int;
6
7   Int Function factorial(m : Int)
8   {
9       If (m == 0 | m == 1) Then
10      {
11          Return(1);
12      }
13      Else
14      {
15          Return(m * factorial(m - 1));
16      }
17  }
18
19  Main()
20  {
21      Read(n);
22      result = factorial(n);
23      Write(result);
24  }
25
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
6 ['GOTO', None, None, 14]
7 ['ERA', 'factorial', None, None]
8 ['-', 4001, 7002, 4005]
9 ['PARAM', 4005, None, 'PARAM1']
10 ['GOSUB', 'factorial', None, None]
11 ['=', 1003, None, 4006]
12 ['*', 4001, 4006, 4007]
13 ['RETURN', None, None, 4007]
14 ['ENDFUNC', None, None, None]
15 ['READ', None, None, 1001]
16 ['ERA', 'factorial', None, None]
17 ['PARAM', 1001, None, 'PARAM1']
18 ['GOSUB', 'factorial', None, None]
19 ['=', 1003, None, 1004]
20 ['=', 1004, None, 1002]
21 ['WRITE', None, None, 1002]
22 ['END', None, None, None]
> 4
24
PS C:\Users\juven\Documents\GitHub\Compiler>
```

Fibonacci (Recursive)

Fibonacci (Recursive).neupy

```
1   Program fibonacciRecursive;
2   Var
3       n : Int;
4       fib : Int;
5   Int Function fibonacci(m : Int)
6   {
7       If (m == 0) Then
8       {
9           Return(0);
10      }
11      Else {
12          If (m == 1) Then
13      {
14          Return(1);
15      }
16      Else
17      {
18          Return(fibonacci(m - 1) + fibonac
19      }
20      }
21  }
22  Main()
23  {
24      Read(n);
25      fib = fibonacci(n);
26      Write(fib);
27  }
28
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
13 ['=', 1003, None, 4005]
14 ['ERA', 'fibonacci', None, None]
15 ['-', 4001, 7003, 4006]
16 ['PARAM', 4006, None, 'PARAM1']
17 ['GOSUB', 'fibonacci', None, None]
18 ['=', 1003, None, 4007]
19 ['+', 4005, 4007, 4008]
20 ['RETURN', None, None, 4008]
21 ['ENDFUNC', None, None, None]
22 ['READ', None, None, 1001]
23 ['ERA', 'fibonacci', None, None]
24 ['PARAM', 1001, None, 'PARAM1']
25 ['GOSUB', 'fibonacci', None, None]
26 ['=', 1003, None, 1004]
27 ['=', 1004, None, 1002]
28 ['WRITE', None, None, 1002]
29 ['END', None, None, None]
> 9
34
PS C:\Users\juven\Documents\GitHub\Compiler>
```

Manual de Usuario

1. Instala Python
2. Descargar el repositorio
3. Abre el folder
4. Abre la ubicación del folder ‘compilador’.
5. Crea un archivo .neupy.
6. Correr la maquina virtual e introduce el nombre del archivo

Video:

https://youtu.be/kIGC3ZU46OE

Github:

https://github.com/juvengtz/Compiler