

Instituto Tecnológico de Costa Rica

Escuela de Computación

Ingeniería en computación

Compiladores e Intérpretes

Grupo 2

Profesor: Erika Marín

Análisis Sintáctico

Josué Arrieta Salas, 2014008153

Adrián López Quesada, 2014081634

Cartago

Miércoles 8 de junio

Índice

| | |
|---|----|
| Análisis de resultados | 3 |
| Casos de pruebas | 4 |
| Prueba 1: Funciones..... | 4 |
| Prueba 1.1 – Primeras pruebas con Funciones..... | 4 |
| Prueba 1.2 – Segundas pruebas con Funciones | 4 |
| Prueba 2: Expresiones..... | 5 |
| Prueba 2.1 – Primeras pruebas con Expresiones..... | 5 |
| Prueba 2.1 – Segundas pruebas con Expresiones | 6 |
| Prueba 3: Declaración de variables | 7 |
| Prueba 4: Asignaciones..... | 9 |
| Prueba 5: Print e Input..... | 9 |
| Prueba 6: While..... | 10 |
| Prueba numero 7: If | 11 |
| Prueba numero 8: For (Adrián)..... | 12 |
| Prueba numero 9: Try | 13 |
| Prueba numero 10: Break - Continue (Adrián) | 14 |
| Prueba numero 11: OOP | 14 |
| Prueba 11.1 – Estructura Básica | 14 |
| Prueba 11.2 – El ; | 15 |
| Gramática..... | 17 |
| ¿Cómo compilar y correr el Parser? | 23 |

Análisis de resultados

| Objetivo | Porcentaje de éxito (100%) |
|---|---|
| Programa recibe código fuente escrito en Python y analiza el archivo. | 100% |
| Se listan errores léxicos encontrados (por línea y el error). | 100%: objetivo del proyecto pasado. |
| El programa ante un error léxico se recupera de este y no despliega errores en cascada. | 100%: objetivo del proyecto pasado. |
| Se despliega lista de errores sintácticos: despliega línea, columna, y el token incorrecto. | 100% |
| Se manejan mensajes específicos para reportar errores sintácticos. | 100% |
| El programa ante un error sintáctico se recupera de este y no despliega errores en cascada. | 100%: se recupera en gran medida; sin embargo se ha de mencionar que en algunos casos es simplemente imposible recuperarse del error y continuar parseando. |
| Se parsean correctamente declaraciones de funciones. | 100% |
| Se parsean correctamente declaraciones de variables de tipo: int, float, list, string, boolean y char. | 100% |
| Se procesa correctamente la estructura de un programa funcional. | 100% |
| Se parsea correctamente la estructura de un programa orientado a objetos. | 100%: se toma en cuenta además, que debe venir al menos un atributo y un método para este tipo de archivos. |
| Se parsean correctamente todo tipo de expresiones: aritméticas y booleanas. Una llamada a función también se considera una expresión. | 100%: también se considera que las asignaciones son expresiones. |
| Se parsean correctamente todo tipo de asignaciones. También son válidas la creaciones de clases. | 100% |
| Se define la estructura de la función print e input. | 100% |
| Se parsean correctamente las estructuras de control: if, while y for. | 100% |
| En un bloque se pueden usar las sentencias break y continue. | 100% |
| Se parsea correctamente la estructura de control de errores: Try-Except-Finally. | 100% |

Casos de pruebas

Prueba 1: Funciones

Prueba 1.1 – Primeras pruebas con Funciones

Pruebas simples de funciones, todavía no estaban claras todas las reglas que debían tener las funciones. No se tenía tanto manejo de número de líneas y errores.

Objetivo: Probar reglas básicas de las funciones del lenguaje MyPython.

```
def pruebaFuncion (int x,int b):  
def pruebaFuncion ():  
def pruebaFuncion ()|
```

Resultado:

```
Writing code to src\Generado\Scanner\Lexer.java  
Error sintático en la línea 2. Identificador: def no reconocido.
```

Ya que no se habían definido “x” cantidad de funciones

Al definirlo para “x” cantidad de funciones:

```
Writing code to "src\Generado\Scanner\Lexer.java"  
Error sintático en la línea -1. Identificador: null no reconocido.
```

Porque falta el ultimo “:”

Al arreglar ese error en la prueba:

```
Ahora se procede a parsear:  
Parseo realizado exitosamente. De tipo Funcional.
```

Prueba 1.2 – Segundas pruebas con Funciones

Objetivo: realizar aún más tipos de prueba con funciones. En este caso se prueba fallos en la declaración de la función, falta de dos puntos o puntos y comas, y la estructura que debe mantener la función.

Se tiene el siguiente archivo de entrada:

```

def funcion1()
;                               #la acepta
def func( :
;                               #error dec funcion
def funcion2():

def funcion3():                #error estructura, asume que esta definiendo una
                                #funcion dentro del bloque de codigo
;                               #se recupera con el puntoComa

def funcion4()

    if True:                    #se recupera con estos dos puntos y no analiza el if
    ;
def funcion5():
    int variable
    if True:
    ;
    string variable #erro de estructura
;

```

Guiándose con los comentarios del archivo se esperan los siguientes resultados (Correctos):

```

Se procede a scannear:
Análisis léxico terminado.

Ahora se procede a parsear:
Parseo realizado exitosamente. De tipo Funcional.
Hay un error en la declaracion de la funcion. En el token: ; En la línea: 2 En la columna: 1
Hay un error en la declaracion de la funcion. En el token: : En la línea: 3 En la columna: 11
Estructura invalida (verifique puntoYcoma de estructura anterior). En el token: funcion3 En la línea: 7 En la columna: 5
Hay un error en la declaracion de la funcion. En el token: if En la línea: 13 En la columna: 2
Estructura invalida (verifique puntoYcoma de estructura anterior). En el token: variable En la línea: 19 En la columna: 9

```

Prueba 2: Expresiones

Prueba 2.1 – Primeras pruebas con Expresiones

Esta prueba fue realizada en el transcurso de la realización del proyecto, y es considerada precisamente por esa razón. En el momento que se realizó las pruebas, una asignación no era considerada una expresión (ahora sí lo es) para que se tome en cuenta.

Objetivo: probar todo tipo de expresiones en el lenguaje MyPython.

Se tiene el siguiente documento de texto de entrada:

```

1  2 + 2
2  []
3  [{"hola"}, [identificador]]
4  (a<=3) and (b==4)
5  2 and 8
6  -3
7  not []
8  2 + 3 * 4
9  True + False
10 ("string" * 4) // (id or (not True))
11 funcion(2,4,[lista],identificador)
12 'c'
13 """ las siguientes son errores """
14 b += 2
15 (("string") * 4) (//) id or True

```

Se espera errores en la línea 14 y en la línea 15, sin embargo las demás son expresiones válidas. Se tienen los siguientes resultados (incorrectos):

Ahora se procede a parsear:

Expresion reconocida

Hay un error en la expresión. Línea: 11. Columna: 12

Hay un error en la expresión. Línea: 11. Columna: 19

Hay un error en la expresión. Línea: 11. Columna: 21

Hay un error en la expresión. Línea: 14. Columna: 1

Hay un error en la expresión. Línea: 15. Columna: 18

Hay un error en la expresión. Línea: 15. Columna: 19

Se muestra que hay un error en la línea 11 debido a que las funciones no se contemplaron como expresiones. Se aplica un arreglo y al correr el mismo archivo se tienen los siguientes resultados (correctos):

Ahora se procede a parsear:

Expresion reconocida

Hay un error en la expresión. Línea: 14. Columna: 1

Hay un error en la expresión. Línea: 15. Columna: 18

Hay un error en la expresión. Línea: 15. Columna: 19

Prueba 2.1 – Segundas pruebas con Expresiones

Objetivo: realizar aún más tipos de prueba con expresiones. En este caso las asignaciones si son tomadas en cuenta como expresiones. Se tiene el siguiente archivo de entrada:

```

15     a = []
16     a = [{"hola"},[identificador]]
17     not []
18     2 + 3 * 4
19     True + False
20     funcion1(3+x , x=2, 3=z) #error por el 3=z
21     'a'
22     a + * b #error
23     (a==b) or (b<=2)

```

Resultados esperados: Se espera que haya errores en las líneas 22 y 20. Se obtuvieron los siguientes resultados (correctos):

```

Se procede a scannear:
Análisis léxico terminado.

Ahora se procede a parsear:
Parseo realizado exitosamente. De tipo OOP.
Hay un error en la expresión. En el token: = En la línea: 20 En la columna: 23
Hay un error en la expresión. En el token: * En la línea: 22 En la columna: 6

```

Prueba 3: Declaración de variables

Objetivo: probar todo tipo de declaración de variables en el lenguaje MyPython. Se tiene el siguiente código de entrada:

```

1  class clase:
2      int x,z,c
3      int 5      #error
4      float y
5      int "hola" #error
6      list z
7      intt y      #error
8      boolean a
9      char b
10     int a a #error
11     char z, #error
12     int z
13
14
15     def funcion():
16         print("hola mundo")
17     ;
18
19 ;
20
21 #codigo principal
22 int x
23 x = + 2

```

Resultados esperados: Se espera error en las líneas 3, 5, 7, 10 y 11. Se obtuvieron los siguientes resultados (correctos):

Se procede a scanear:

Análisis léxico terminado.

Ahora se procede a parsear:

Parseo realizado exitosamente. De tipo OOP.

Hay un error al declarar variables. En el token: 5 En la línea: 3 En la columna: 6

Hay un error al declarar variables. En el token: "hola" En la línea: 5 En la columna: 11

Hay un error al declarar variables. En el token: intt En la línea: 7 En la columna: 2

Hay un error al declarar variables. En el token: y En la línea: 7 En la columna: 7

Hay un error al declarar variables. En el token: a En la línea: 10 En la columna: 8

Hay un error al declarar variables. En el token: int En la línea: 12 En la columna: 2

El error de la línea 11 se muestra en la línea 12 debido a la naturaleza del error y el error surge al encontrar "int" en la línea 12 y no un identificador. Este mensaje es considerado correcto.

Prueba 4: Asignaciones

Objetivo: probar diferentes tipos de asignaciones posibles en el lenguaje MyPython. Se tiene el siguiente código de entrada:

```
Prueba.mypy X
1 variable **= 2
2 variable /= [lista,2,"hola"]
3 c -= Clase.Funcion()
4 funcion += 2 + funcion(variable,c,3)
5 x + 2 """ este es un error """
6 var = 0
7 c = Clase()
8
```

Resultados esperados: Se espera un error en la línea 5 ya que "x + 2" no es una asignación sino una expresión, sin embargo se espera que el compilador se recupere de este error. Se obtuvieron los siguientes resultados (correctos):

Ahora se procede a parsear:

Asignaciones Reconocidas

Hay un error en la asignación. Línea: 5. Columna: 1

Prueba 5: Print e Input

Objetivo: probar la estructura de las funciones print() e input() predefinidas. Se tiene el siguiente código de entrada:

```
14 input #error
15 2 + 2
16 input()
17 input(+) #error
18 input(x)
19 input("digite un valor", x) #error
20 input(x+x)
21
22 print(213+ 518- 345)
23 print(C)
24 print() #error
25 print(C - 123)
26 print #error
27 2 + 2
28 print("El valor es", x)
29 print("El resultado de la suma es ", suma(a,b,c))
```

Se espera que haya errores en las líneas 14, 17, 19,24 y 26. También se espera que se recupere del error y siga parseando. Se obtuvieron los siguientes resultados (incorrectos):

Se procede a scannear:
Análisis léxico terminado.

Ahora se procede a parsear:
Parseo realizado exitosamente. De tipo OOP.
Hay un error en input(). En el token: 2 En la línea: 15 En la columna: 2
Hay un error en la expresión. En el token:) En la línea: 17 En la columna: 9
Hay un error en la expresión. En el token: , En la línea: 19 En la columna: 25
Hay un error en la expresión. En el token: 2 En la línea: 27 En la columna: 2

Como se ha mencionado, algunos errores los muestra en la línea después (en el caso del error de la línea 14 y 26). Sin embargo el error de la línea 24 no se mostró. Print() es considerado correcto. Se aplica una corrección y se vuelve a correr el mismo archivo, y se obtienen los siguientes resultados (correctos):

Se procede a scannear:
Análisis léxico terminado.

Ahora se procede a parsear:
Parseo realizado exitosamente. De tipo OOP.
Hay un error en input(). En el token: 2 En la línea: 15 En la columna: 2
Hay un error en la expresión. En el token:) En la línea: 17 En la columna: 9
Hay un error en la expresión. En el token: , En la línea: 19 En la columna: 25
Hay un error en la expresión. En el token:) En la línea: 24 En la columna: 8
Hay un error en la expresión. En el token: 2 En la línea: 27 En la columna: 2

Prueba 6: While

Objetivo: Probar todas las funcionalidades del While y su recuperación de errores.

Se tiene el siguiente código de entrada:

```
while ():  
;  
while ():  
;  
else:  
;  
  
while(): #error  
  
while True:  
;  
  
while nt4:  
;  
  
while n<4 #error  
  
while Truee: #se recupera hasta aqui  
;|
```

Se obtuvieron los siguientes resultados (Correctos):

Ahora se procede a parsear:

Error: Se llego al final del archivo. Falto un ; o :

Hay un error en la expresionn del While. En el token: : En la linea: 13 En la columna: 10

Hay un error en la declaracion del While. En el token: while En la linea: 18 En la columna: 1

En el token: Se llego al final del archivo sin cerrar un bloque En la linea: -1 En la columna: -1

Los resultados son correctos, pero cabe destacar algunas observaciones. Se llego la captura de emergencia que se tiene por falta de un puntoYComa, esto es debido a que el while que le falta el puntoYComa tomo a todos los demás whiles como si fueran parte de su bloque, por lo tanto no se pudo recuperar con otro puntoYComa. Ademas se puede observar cómo no expreso error en el último while con el True, esto es correcto ya que el while anterior carece de los dos puntos, por lo que ignora todo hasta llegar a los siguiente dos puntos.

Prueba numero 7: If

Objetivo: Probar la estructura del if, elif y else, ifs anidados, ifs con y sin else y elifs, recuperación de puntos y comas, dos puntos, mal ubicación de estructuras else elif y expresiones no booleanas.

```
if True:
;
if True:
    if True:
        ;
    else:
        ;
;
if x>4:
;
elif True|:
;
if identificador:
;
if True
;
elif Truet: # se recupera aca
;
else:
;
else:
;
if True:

elif True:
;
```

Resultados obtenidos (Correctos):

Ahora se procede a parsear:

Parseo realizado exitosamente. De tipo Funcional.

Hay un error en la expresión. Debe ser booleana. En el token: : En la línea: 13 En la columna: 17

Hay un error en la declaración del If (revisar dos puntos). En el token: ; En la línea: 16 En la columna: 1

Estructura invalida (verifique puntoYcoma de estructura anterior). En el token: : En la línea: 21 En la columna: 5

Hay un error en la expresión. En el token: ; En la línea: 22 En la columna: 1

Estructura invalida (verifique puntoYcoma de estructura anterior). En el token: Triue En la línea: 25 En la columna: 6

Se puede apreciar como las primeras estructuras las acepta sin ningún problema. Los demás errores son claros de porque suceden, dándose la recuperación donde los comentarios especifican. El error de la línea 22 se debe a que no acepta el else sin un if por lo tanto se recupera en los dos puntos y queda el punto y coma para analizarlo como expresión. Además no muestra el error de True ya que por falta de punto y coma del último if, anula la expresión elif ya que no la acepta sin tener un if antes; se recupera en los dos puntos y utiliza el punto y coma del elif para terminar el parseo.

Prueba numero 8: For (Adrián)

Objetivo: probar la estructura del for, todas sus posibilidades, control de errores y el uso de Range e In.

Se usó el siguiente trozo de código:

```
for x in ident:
;
for x in range(8):
;
for x in "string":
;
for x in range(4,6):
;
for x range(5):
;
for in range(9):
;
for x range(8):
;
for t in range():
;
for t in range(9)

;
for t in range(10):
;
for t in x :

while():
;
```

Se obtuvieron los siguientes resultados (Correctos):

Ahora se procede a parsear:

Error: Se llego al final del archivo. Falto un ; o :

Hay un error, falta 'in'. En el token: range En la línea: 9 En la columna: 7

Hay un error en el for. En el token: in En la línea: 11 En la columna: 5

Hay un error, falta 'in'. En el token: range En la línea: 13 En la columna: 7

Hay un error en el range. En el token:) En la línea: 15 En la columna: 16

Erro falta parentesis range o puntoComa. En el token: ; En la línea: 19 En la columna: 1

En el token: Se llego al final del archivo sin cerrar un bloque En la línea: -1 En la columna: -1

Se puede apreciar como acepta todos los tipos de for y usos del range. Respecto a la parte de los errores, los identifica bien, pero es necesario aclararlos. En la línea 19 tira error, pero debido a la gramática es bastante complicado analizar donde si es por falta de paréntesis o dos puntos del for, por lo tanto, especifica al usuario que verifique ambos casos. Además, hay un error de EOF, debido al for de la línea 22, el cual no tiene punto y coma. El compilador ubica al while dentro del for y cierra el while pero falta cerrar el for.

Prueba numero 9: Try

Objetivo: probar la estructura para el control de errores Try-Except en MyPython. Se tiene el siguiente archivo de entrada:

```
14     try:
15         print("esto es un try")
16     ;
17     except Exception:
18         x = 2 + 2
19     ;
20     finally:
21         input(x)
22     ; #todo el bloque anterior se espera correcto
23
24
25     try          #error : faltan los ':' del try
26         print("esto es un try")
27     ;
28     except :     #error: no viene indentificador
29         input(x)
30     ;
```

Resultados esperados:

- Error en la línea 25 ya que faltan los ':' del try. Debe tirarlo en la línea 26 ya que ahí está el token que posee el error.
- Error en la línea 28 ya que falta un indentificador en el except.

Resultados obtenidos (correctos):

```
Se procede a scanear:  
Análisis léxico terminado.  
  
Ahora se procede a parsear:  
Parseo realizado exitosamente. De tipo OOP.  
Hay un error, en el try. Faltan dos puntos. En el token: print En la linea: 26 En la columna: 3  
Hay un error con la definición del except (revisar dos puntos o identificador). En el token: : En la linea: 28 En la columna: 9
```

Prueba numero 10: Break - Continue (Adrián)

Objetivo: Probar el uso del break y continue.

Debido a ser una prueba sencilla se presentan los resultados y el archivo en uno solo:

```
Se procede a scanear:  
Análisis léxico terminado.  
  
Ahora se procede a parsear:  
Parseo realizado exitosamente. De tipo Funcional.
```

| Archivo | Edición | Formato | Ver | Ayuda |
|---|---------|---------|-----|-------|
| <pre>def funcion(): break continue for ident in range(4,5): break continue ;</pre> | | | | |

No tiro ningún error, esto esta correcto ya que el análisis del break y el continue se va a desarrollar dentro del analizador semántico. En la parte del sintáctico puede aparecer en cualquier parte de un bloque de código.

Prueba numero 11: OOP

Prueba 11.1 – Estructura Básica

Objetivo: probar la estructura básica de la clase de un archivo MyPython. Se tiene el siguiente código de entrada:

```
Prueba.mypy x  
1 class clase  
2  
3  
4  
5 ;  
6  
7 #codigo principal  
8 int x  
9 x = + 2
```

Resultados esperados: se espera que surjan los errores:

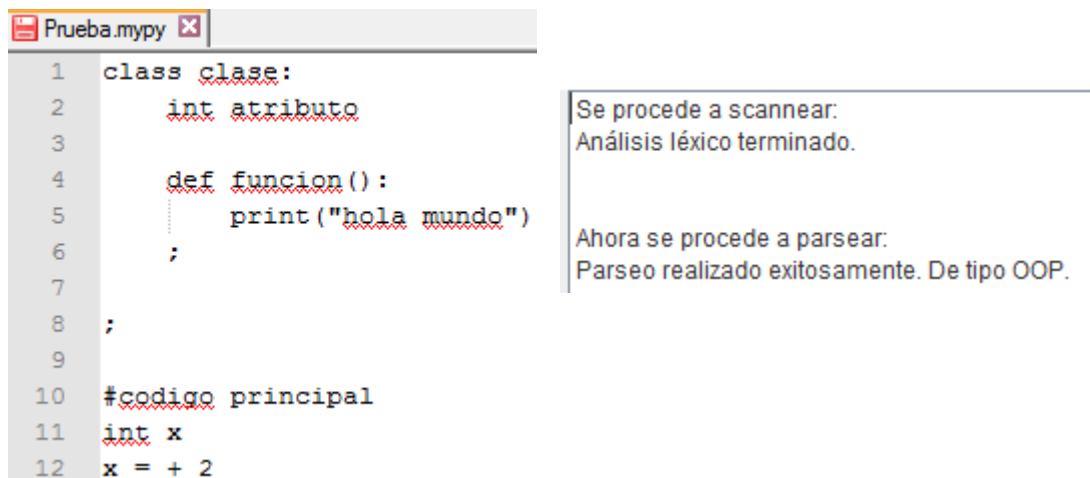
- Faltan los ':' en el header de la clase.
- Se queje de que no vengan variables ni métodos (tienen que venir al menos 1):

Resultados obtenidos (correctos):

| Se procede a scanear:
Análisis léxico terminado.

Ahora se procede a parsear:
Parseo realizado exitosamente. De tipo OOP.
Hay un error en el header del fuente de tipo OOP. En el token: ; En la línea: 5 En la columna: 1
debe venir al menos una variable y un método al definir la clase, en la línea: 5

Si se añade el siguiente archivo (con los errores corregidos), no hay errores sintácticos:



```
1 class clase:
2     int atributo
3
4     def funcion():
5         print("hola mundo")
6
7
8 ;
9
10 #codigo principal
11 int x
12 x = + 2
```

Se procede a scanear:
Análisis léxico terminado.

Ahora se procede a parsear:
Parseo realizado exitosamente. De tipo OOP.

Prueba 11.2 - El ;

Objetivo: probar cuando no viene el ; al definir una clase. Se tiene el siguiente archivo:

```

1  class clase:
2      int atributo
3
4      def funcion():
5          print("hola mundo")
6      ;
7
8
9
10 #codigo principal
11 int x
12 x = + 2

```

Resultados esperados: error sintáctico al no cerrar la clase con el ;. Resultados obtenidos (incorrectos) :

Couldn't repair and continue parse

Se aplica un parche que arregle el error y al correr el mismo archivo se obtiene (resultados correctos):

```

Se procede a scanear:
Análisis léxico terminado.

Ahora se procede a parsear:
Parseo realizado exitosamente. De tipo OOP.
Falto el ; de la clase principal en la línea: 4

```


Gramática

start with Programa

Programa ::= Funcional | OOP | error

-----**Funcional**-----

Funcional ::= DefFuncionesCodigoPrincipal

DefFunciones ::= DefFuncion DefFunciones |

DefFuncion ::= def identificador parenAbierto DefParametrosFact dosPuntos Variables
CualquierCosas puntoComa

| def identificador parenAbierto DefParametrosFact dosPuntos Variables CualquierCosas
errPuntoYComa

| def errDecFuncion dosPuntos Variables CualquierCosas puntoComa

| def identificador parenAbierto DefParametrosFact errDecFuncion dosPuntos

| def errDecFuncion puntoComa

| def parenAbierto errDecFuncion puntoComa

| def errPuntoYComa DefFuncion

| errDecFuncion DefFuncion

| errDecFuncion puntoComa

DefParametrosFact ::= Var identificador DefParametros | var errParametro1 DefParametros |
identificador errParametro1 DefParametros | parenCerrado

DefParametros ::= coma Var identificador DefParametros | parenCerrado | errParametro1

-----**OOP**-----

VariablesOOP ::= Variable Variables;

FuncionesOOP ::= DefFunciones2 puntoComa CodigoPrincipal

| DefFunciones2:c { : usado para definir un error: } CodigoPrincipal

| puntoComa:c { : usado para definir un error: } CodigoPrincipal

DefFunciones2 ::= DefFuncion DefFunciones2 | DefFuncion

FuncionesOOP2 ::= DefFunciones puntoComaCodigoPrincipal

| DefFunciones:c { : usado para definir un error : } CodigoPrincipal

OOP ::= HeaderOOP

BloqueOOP ::= VariablesOOP FuncionesOOP

| DefFuncion:c { : usado para definir un error : } FuncionesOOP2

| puntoComa:c { : usado para definir un error : } CodigoPrincipal

HeaderOOP ::= clas identificador dosPuntos BloqueOOP | clas errHeaderOOP BloqueOOP

-----Bloques-----

If ::= iif ExpresionB dosPuntos BloquePuntoComa Elif

| iif ExpresionB error dosPuntos BloquePuntoComa Elif

| iif errExpresionB dosPuntos BloquePuntoComa Elif

Elif ::= elif ExpresionB dosPuntos BloquePuntoComa Elif

| elif error dosPuntos BloquePuntoComa Elif | Else

Else ::= eelse dosPuntos BloquePuntoComa | eelse errElse dosPuntos BloquePuntoComa |

ForFact ::= ffor identificador in For dosPuntos BloquePuntoComa

| ffor errFor in For dosPuntos BloquePuntoComa

| ffor identificador in For errFor dosPuntos BloquePuntoComa

| ffor identificador errIn For dosPuntos BloquePuntoComa

For ::= STRING | identificador | Range | errFor

Range ::= range parenAbierto INT coma INT parenCerrado

| range parenAbierto INT parenCerrado | range errRange parenCerrado

| range parenAbierto INT error | range parenAbierto INT coma INT error | range errFor

While ::= wwhile ExpresionB dosPuntos BloquePuntoComa Else

| wwhile parenAbierto parenCerrado dosPuntos BloquePuntoComa Else

| wwhile error:e dosPuntos BloquePuntoComa Else

| wwhile ExpresionB error:e dosPuntos BloquePuntoComa Else

Try::= ttry dosPuntos CualquierCosas puntoComa Except

| ttry errTry BloquePuntoComa Except

| ttry dosPuntos CualquierCosas errExcept puntoComa

Except::= eexcept identificador dosPuntos BloquePuntoComa Finally

| eexcept error dosPuntos BloquePuntoComa Finally | errExcept2

Finally::= finally dosPuntos BloquePuntoComa

| finally errFinally dosPuntos BloquePuntoComa |

BloquePuntoComa ::= CualquierCosas puntoComa | errPuntoYComa puntoComa

-----**Expresiones**-----

Funcion::= identificador parenAbierto Parametros | identificador errParen parenCerrado

Parametros ::= Expresion Parametro | parenCerrado | errParen

Parametro ::= coma Expresion Parametro | parenCerrado | errParametro parenCerrado

Literal ::= INT|FLOAT|CHAR|STRING

Lista::= cuadradoAbierto ElementosLista cuadradoCerrado

ElementosLista::= Expresion MasElementosLista |

MasElementosLista::= coma Expresion MasElementosLista |

Expresion ::= Literal|Lista|identificador | parenAbierto Expresion parenCerrado | errExpresion

| parenAbierto errParen

| errParen parenCerrado;

Expresion ::= Expresion OpBinario Expresion

Expresion ::= OpUnario Expresion

Expresion ::= identificador punto Funcion

Expresion ::= ExpresionB

Expresion ::= FuncionPredefinida

ExpresionB ::= ffalse | ttrue | Funcion | opNot Expresion | Expresion OpBinarioB Expresion | parenAbierto ExpresionB parenCerrado

Expresion ::= identificador opAsignaciones Expresion

Expresiones ::= Expresiones Expresion |

FuncionPredefinida ::= Input | Print | IntFuncion

IntFuncion ::= intReservado parenAbierto FuncionPredefinida parenCerrado

| intReservado parenAbierto Funcion parenCerrado

| intReservado parenAbierto STRING parenCerrado | errInput

Input ::= iinput parenAbierto Expresion parenCerrado | iinput parenAbierto parenCerrado

| iinput errInput | errInput

Print ::= pprint parenAbierto ElementosLista2 parenCerrado | errPrint

| pprint parenAbierto errPrint parenCerrado

ElementosLista2 ::= Expresion MasElementosLista2

MasElementosLista2 ::= coma Expresion MasElementosLista2 |

OpAritmeticos ::= opSuma | opResta | opMultiplicacion | opDivisionE | opDivision | opModulo | opPotencia

OpUnario ::= opSuma | opResta

OpBinario ::= OpAritmeticos

OpBinarioB ::= opComparadores | OpLogicos

OpLogicos ::= opAnd | opOr | opNot

-----**General**-----

CualquierCosa ::= Expresion | If | ForFact | While | Try | bbreak | ccontinue

| def errMalUbicacion dosPuntos | eexcept errMalUbicacion dosPuntos

| var errMalUbicacion | clas errMalUbicacion puntoComa

| finally errMalUbicacion dosPuntos

| eelse errMalUbicacion dosPuntos

| elif errMalUbicacion dosPuntos;

CodigoPrincipal::= Variables CualquierCosas

CualquierCosas::= CualquierCosas CualquierCosa |

Var ::= var | intReservado

Variable::= Var DefVariablesFact | Var errDecVariables | errDecVariables identificador

Variables ::= Variables Variable |

DefVariablesFact ::= identificador DefVariables | errDecVariables DefVariables

DefVariables::= coma identificador DefVariables | errDecVariables identificador DefVariables |
errDecVariables DefVariables |

-----Errores-----

errHeaderOOP::= error

errDecVariables::= error

errPuntoYComa::= error

errExpresion::= error

errExpresionB::= error

errInput::= error

errPrint::= error

errDecFuncion::= error

errParametro1::= error

errParametro::= error

errRange::= error

errIn::= error

errFor::= error

errElse::= error

errDosPuntos ::= error

errTry ::= error

errExcept ::= error

errExcept2 ::= error

errFinally ::= error

errParen ::= error

errMalUbicacion ::= error

¿Cómo compilar y correr el Parser?

Debido a problemas para enviar el archivo por Gmail, se tuvieron que modificar las extensiones .jar a .jara; por lo tanto se debe eliminar esta “a” extra en los siguientes archivos:

Si se desea correr el ejecutable:

- Compilador-mypy\Ejecutable\Compilador.jara
- Compilador-mypy\Ejecutable\lib\ java-cup-11b.jara
- Compilador-mypy\Ejecutable\lib\ jflex-1.6.1.jara

Si se desea compilar el proyecto:

- Compilador-mypy\CodigoFuente\Tools\ java-cup-11b.jara
- Compilador-mypy\CodigoFuente\Tools\ jflex-1.6.1.jara

Para correr el parser se debe usar el archivo Prueba.mypy en la ubicación:

- Compilador-mypy\Ejecutable\
- Compilador-mypy\CodigoFuente\

Según se ejecute el ejecutable o se quiera recompilar el proyecto, de manera respectiva.

Notas extras: Se desarrolló el proyecto en el IDE Netbeans y se utilizaron las librerías Java Cup y JFlex para su desarrollo. Se implementó una interfaz gráfica para facilitar la ejecución del ejecutable, ya que en consola solo se puede correr desde línea de comando. La gramática quedo de manera extensa debido a problemas con encontrados con la recuperación de errores de la librería Java Cup., por lo que se intentó ser lo más específico posible.