Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Compiladores e Intérpretes

Oscar Víquez Acuña

**Tercer Proyecto Programado**

Estudiantes:

Adrián Hernández Soto.

Andrew Alvarado Valenciano.

Sede Santa Clara, Florencia, San Carlos

10/06/17

**Análisis del problema**

El propósito del proyecto es generar a bytecode a partir de instrucciones necesarias, las cuales serán sintácticas y contextualmente correctas para el lenguaje del Mini Python.

La generación del bytecode será por medio de visitas o “visitors”, de esta manera se va a ir generando el código en mientras se recorre en profundidad un árbol.

En los trabajos anteriores las visitas en el código tienen que cumplir con el orden específico, este orden se toma del Parser, pero para este trabajo no tiene que seguir un orden secuencial, ya que en algunos momentos se hacen diferentes visitas antes y para ello se deberá de cambiar el orden.

La generación del bytecode se deberá guardar en un archivo de texto y luego por medio del editor del programa o del que tiene la computadora se podrá visualizar y verificar.

**Soluciones e implementación**

Para la solución se creó una clase Instruccion, el cual cuenta con atributos que guardan datos necesarios para las instrucciones, como el código, el número de línea, parámetros para las etiquetas y un boolean para diferenciar las etiquetas de código normal.

En el generador de código se crea una lista de Instruccion y ahí se guardarán las líneas que se generen. También se crea una variable global con la función de contador, para llevar control de las líneas. Y una variable global para llevar el conteo de argumentos en las distintas partes de código que se necesiten, como por ejemplo en la creación de listas, o en el llamado de funciones con varios argumentos.

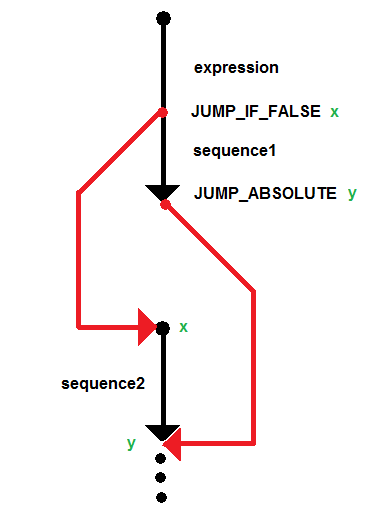
Al final del procedimiento el código generado queda guardado en un archivo .txt en la carpeta del proyecto.

Por medio de visitas a los distintos nodos del árbol se generará el bytecode, en seguida se hablará como se fueron creando las instrucciones de bytecode en algunos de los nodos más representativos.

**ifStatm:**

Para generar el código de un if, se inicia visitando expression para evaluar los argumentos, luego en la lista global llamada listaInstrucciones, se le va a agregar las instrucciones de bytecode, después de visitar los argumentos del if, en este caso se hace un JUMP\_IF\_FALSE, y se guarda la posición de esa instrucción para poder agregarle más adelante a donde va a tener que saltar. Cuando se termina la instrucción, se guarda la posición de la instrucción JUMP para más adelante poder modificarla y definir dónde debe de hacer el jump. Luego se visita el sequence, después de hacer esta visita es donde se modifica a la instrucción JUMP\_IF\_FALSE ya que se sabe cuantas lineas tiene que saltar. Después se crea un JUMP\_ABSOLUTE para el else, se guarda la posición de la instrucción y se repite el proceso del sequence y modificar el JUMP.

En la plantilla se vería algo como lo siguiente:

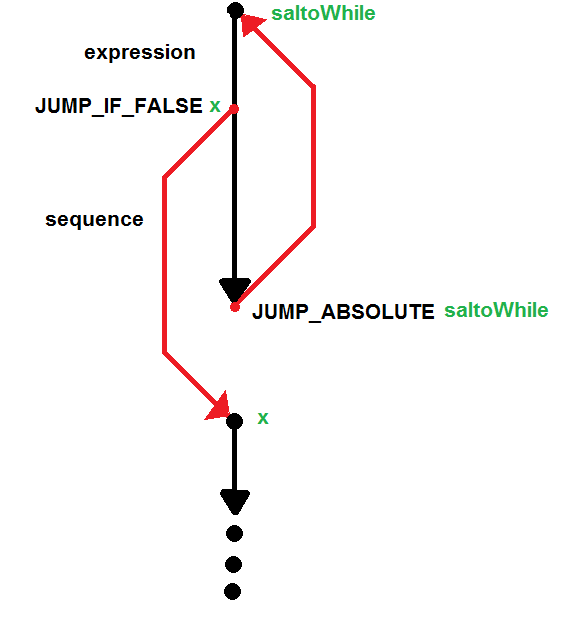


**whileStatm:**

Para hacer la visita del while, primero se crea una variable local (saltoWhile) y esta se iguala a línea, esto se hace para tener la posición inicial del while. Luego se visita a expression para evaluar los argumentos, al finalizar se agrega la instrucción de bytecode a listaInstrucciones, en este caso se utilizará JUMP\_IF\_FALSE. Al igual que como se hizo con el if, se guarda la posición de la instrucción para modificarla después.

Se visita sequence y cuando se termina se agrega a listaInstruciones un JUMP\_ABSOLUTE y como parámetro la variable antes declarada, para volver al inicio del while.

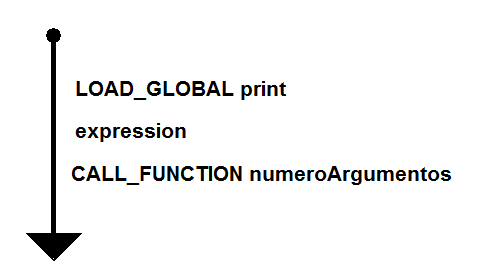
La plantilla del while es la siguiente:



**printStatm:**

Cuando se hace la visita del print lo primero que se hace es ingresar a la listaInstruciones, se le agrega la instrucción de bytecode LOAD\_GLOBAL y su respectiva número de línea, después se hace la visita a expression para leer los argumentos necesarios, una vez leídos, se agrega a la listaInstruciones CALL\_FUNCTION y aquí se le dice la cantidad de parámetros que lleva y se le manda de igual forma el número de línea.

Su plantilla es la siguiente:



**defStatm:**

Esta visita es utilizada en la declaración de funciones. El código que se genera de esta visita son las etiquetas de funciones. Pero antes de generar la etiqueta es necesario conocer de antemano los argumentos o parámetros que lleva la función, para eso lo primero que se realiza es crear una variable tipo String que se igual a la visita de ArgList, el cual se detalla adelante:

**argList:**

Es el encargado de retornar los parámetros de la declaración de funciones. En la visita de este se declara una variable la cual será un String, esto por si la función tiene más de un argumento, está variable se va a igualar a la visita moreArgs. Luego creamos otra variable estática String y se iguala al IDENTIFIER y se le concatena la variable anterior definida. En caso de no tener argumentos se retorna un String vacío.

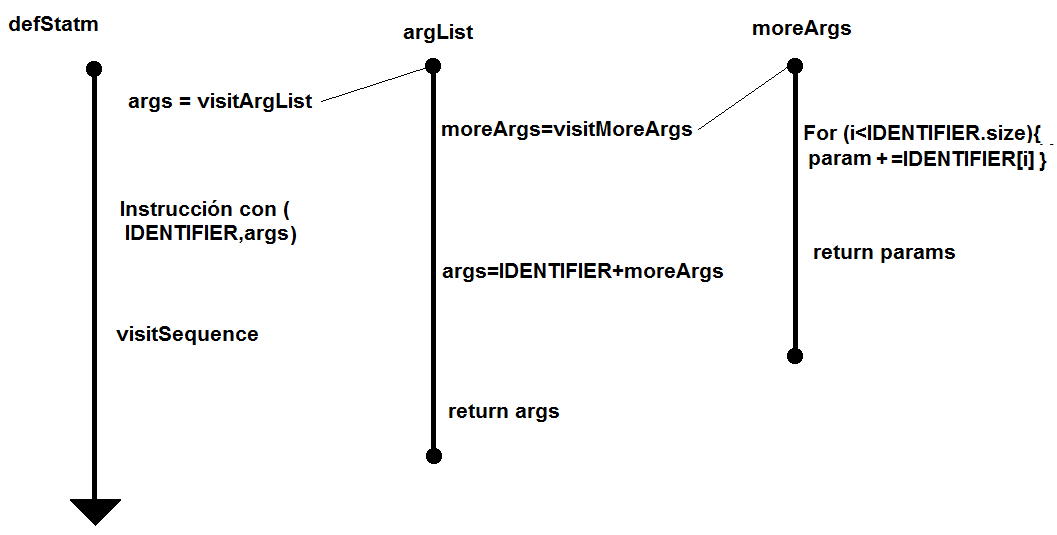
**moreArgs:**

Este es el encargado de sacar todos los parámetros que tenga la función, menos el primero, ya que de éste se encarga el argList.

En la visita de este método, se hace un ciclo el cual crea un String con todos los IDENTIFIER que tiene, luego los retorna.

Al finalizar esas llamadas se puede agregar a las lista de instrucciones la etiqueta requerida con el nombre de la función y sus parámetros. Y finalmente visita sequence para continuar con la función.

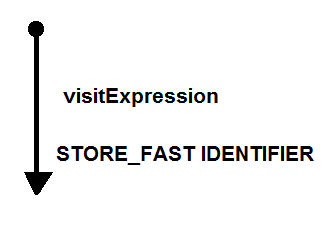
Entonces todo el proceso anterior se ve representado con la siguiente plantilla:



**AssignStatement:**

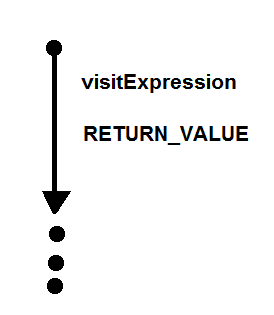
Se inicia con la visita de expression, para leer los argumentos necesarios ya que tienen que ir en ese orden específico, después agregamos en la listaInstruciones la instrucción del bytecode, la cual va a hacer STORE\_FAST y se le concatena el IDENTIFIER.

La plantilla sería:



**ReturnStatement:**

Se inicia visitando expression para leer los argumentos, al finalizar de leer agregamos en la listaInstruciones la instrucción del bytecode, la cual va a hacer RETURN\_VALUE y la línea que llevará. Similar a AssignStatement.



**Resultados obtenidos**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tarea** | **Implementado**  **completamente** | **No implementado**  **o implementado parcialmente** | **Comentarios** |
| Etiquetas que indiquen el inicio de los métodos | X |  |  |
| Consecutivo incremental para cada una de las instrucciones comenzando en 0 | X |  |  |
| Cada instrucción tendrá o no solo un parámetro | X |  |  |
| Posibilidad de separar con TABS o con espacios en blanco. | X |  |  |
| Para que el programa corra, deberá existir un método al final del archivo que se llame Main() y que llevará el último statement del ejemplo como corrida de algún código específico | X |  |  |
| No se permitirá declaración de variables globales, solo métodos | X |  |  |

**Conclusiones**

Para concluir se puede decir que el proyecto en sí ha logrado sus objetivos. Se ha logrado crear un generador de bytecode que funciona correctamente y que realiza las instrucciones necesarias para que el intérprete las utilice.

La técnica que se utilizó fue de mucha utilidad, ya que con los visitors en en el árbol se puede ir creando el código de manera relativamente simple, solo hay que tener cautela con el orden de algunas visitas, puesto que a la hora de generar el código algunas instrucciones van primero que otras.

Con este proyecto se han aprendido muchas cosas, tales como aprender más de cómo se recorre el árbol, cómo funciona el generador de código, y se han incluso comprendido cosas que en el proyecto anterior no se comprendieron. Ha sido de gran utilidad para entender el funcionamiento de un compilador y como este genera el código para que sea interpretado.

**Manual de pruebas**

Puesto que para hacer las pruebas el Análisis Contextual que se había hecho interfería por no estar 100% correcto, se ha desactivado y se crean pruebas suponiendo que el código está correcto en ese aspecto, lo que interesa acá es una correcta generación del bytecode.

**Código probando un while:**

**Código:**

def codWhile():

contador = 0

while (contador < 5):

print ("El contador es : ")

print(contador)

contador = contador + 1

print ("Fin del programa")

def Main():

codWhile()

**ByteCode:**

codWhile()

0 LOAD\_CONST 0

1 STORE\_FAST contador

2 LOAD\_FAST contador

3 LOAD\_CONST 5

4 COMPARE\_OP <

5 JUMP\_IF\_FALSE 17

6 LOAD\_GLOBAL print

7 LOAD\_CONST "El contador es : "

8 CALL\_FUNCTION 1

9 LOAD\_GLOBAL print

10 LOAD\_FAST contador

11 CALL\_FUNCTION 1

12 LOAD\_FAST contador

13 LOAD\_CONST 1

14 BINARY\_ADD

15 STORE\_FAST contador

16 JUMP\_ABSOLUTE 2

17 LOAD\_GLOBAL print

18 LOAD\_CONST "Fin del programa"

19 CALL\_FUNCTION 1

Main()

20 LOAD\_GLOBAL codWhile

21 CALL\_FUNCTION 0

**Código probando un if:**

**Código:**

def edad(ana):

if ana >= 18:

print ("Ana es mayor de edad")

else:

print ("Ana es menor de edad")

def main():

edad(18)

**ByteCode:**

edad(ana)

0 LOAD\_FAST ana

1 LOAD\_CONST 18

2 COMPARE\_OP >=

3 JUMP\_IF\_FALSE 8

4 LOAD\_GLOBAL print

5 LOAD\_CONST "Ana es mayor de edad"

6 CALL\_FUNCTION 1

7 JUMP\_ABSOLUTE 11

8 LOAD\_GLOBAL print

9 LOAD\_CONST "Ana es menor de edad"

10 CALL\_FUNCTION 1

main()

11 LOAD\_GLOBAL edad

12 LOAD\_CONST 18

13 CALL\_FUNCTION 1

**Código probando un while con un if**

**Código:**

def buscar(x,l):

i=0

while( i < len(l) ):

if(x==l[i]):

print("Coincide")

else:

print("No coincide")

i=i+1

def Main():

buscar(2,[1,2,3])

**ByteCode:**

buscar(x, l)

0 LOAD\_CONST 0

1 STORE\_FAST i

2 LOAD\_FAST i

3 LOAD\_GLOBAL len

4 LOAD\_FAST l

5 CALL\_FUNCTION 1

6 COMPARE\_OP <

7 JUMP\_IF\_FALSE 26

8 LOAD\_FAST x

9 LOAD\_FAST l

10 LOAD\_FAST i

11 BINARY\_SUBSCR

12 COMPARE\_OP ==

13 JUMP\_IF\_FALSE 18

14 LOAD\_GLOBAL print

15 LOAD\_CONST "Coincide"

16 CALL\_FUNCTION 1

17 JUMP\_ABSOLUTE 21

18 LOAD\_GLOBAL print

19 LOAD\_CONST "No coincide"

20 CALL\_FUNCTION 1

21 LOAD\_FAST i

22 LOAD\_CONST 1

23 BINARY\_ADD

24 STORE\_FAST i

25 JUMP\_ABSOLUTE 2

Main()

26 LOAD\_GLOBAL buscar

27 LOAD\_CONST 2

28 LOAD\_CONST 1

29 LOAD\_CONST 2

30 LOAD\_CONST 3

31 BUILD\_LIST 3

32 CALL\_FUNCTION 2

**Código probando listas**

**Código**:

def lista():

nombre = "Pepe"

edad = 25

lista = [nombre, edad]

print(lista)

def Main():

lista()

**ByteCode:**

lista()

0 LOAD\_CONST "Pepe"

1 STORE\_FAST nombre

2 LOAD\_CONST 25

3 STORE\_FAST edad

4 LOAD\_FAST nombre

5 LOAD\_FAST edad

6 BUILD\_LIST 2

7 STORE\_FAST lista

8 LOAD\_GLOBAL print

9 LOAD\_FAST lista

10 CALL\_FUNCTION 1

Main()

11 LOAD\_GLOBAL lista

12 CALL\_FUNCTION 0

**Código probando llamada a función**

**Código:**

def sumar(number1, number2):

print (number1)

print (number2)

print (number1 + number2)

def Main():

sumar(10, 15)

**ByteCode:**

sumar(number1, number2)

0 LOAD\_GLOBAL print

1 LOAD\_FAST number1

2 CALL\_FUNCTION 1

3 LOAD\_GLOBAL print

4 LOAD\_FAST number2

5 CALL\_FUNCTION 1

6 LOAD\_GLOBAL print

7 LOAD\_FAST number1

8 LOAD\_FAST number2

9 BINARY\_ADD

10 CALL\_FUNCTION 1

Main()

11 LOAD\_GLOBAL sumar

12 LOAD\_CONST 10

13 LOAD\_CONST 15

14 CALL\_FUNCTION 2

**Código probando recursividad**

**Código:**

def contar(inicio,final):

if(inicio == final):

return final

else:

print(inicio)

inicio = inicio +1

contar(inicio,final)

def Main():

contar(1,10)

**ByteCode:**

contar(inicio, final)

0 LOAD\_FAST inicio

1 LOAD\_FAST final

2 COMPARE\_OP ==

3 JUMP\_IF\_FALSE 7

4 LOAD\_FAST final

5 RETURN\_VALUE

6 JUMP\_ABSOLUTE 18

7 LOAD\_GLOBAL print

8 LOAD\_FAST inicio

9 CALL\_FUNCTION 1

10 LOAD\_FAST inicio

11 LOAD\_CONST 1

12 BINARY\_ADD

13 STORE\_FAST inicio

14 LOAD\_GLOBAL contar

15 LOAD\_FAST inicio

16 LOAD\_FAST final

17 CALL\_FUNCTION 2

Main()

18 LOAD\_GLOBAL contar

19 LOAD\_CONST 1

20 LOAD\_CONST 10

21 CALL\_FUNCTION 2

**Probando el código dado en el enunciado**

**Código:**

def calcularpromedioventas():

ventas = [100,200,300,400,500]

prom = promedio(ventas)

print("El promedio de las ventas es: ")

print(prom)

#promedio de elementos

def promedio(cualquier\_arreglo):

tam = len(cualquier\_arreglo)

resultado=0

#ciclo para recorrer arreglo

cont = 0

sumatoria = 0

while (cont <= tam-1):

sumatoria = sumatoria + cualquier\_arreglo[cont]

cont = cont + 1

if tam > 0:

resultado = sumatoria / tam

return resultado

else:

resultado = sumatoria\*tam

return resultado

def Main():

calcularpromedioventas()

**Bytecode:**

calcularpromedioventas()

0 LOAD\_CONST 100

1 LOAD\_CONST 200

2 LOAD\_CONST 300

3 LOAD\_CONST 400

4 LOAD\_CONST 500

5 BUILD\_LIST 5

6 STORE\_FAST ventas

7 LOAD\_GLOBAL promedio

8 LOAD\_FAST ventas

9 CALL\_FUNCTION 1

10 STORE\_FAST prom

11 LOAD\_GLOBAL print

12 LOAD\_CONST "El promedio de las ventas es: "

13 CALL\_FUNCTION 1

14 LOAD\_GLOBAL print

15 LOAD\_FAST prom

16 CALL\_FUNCTION 1

promedio(cualquier\_arreglo)

17 LOAD\_GLOBAL len

18 LOAD\_FAST cualquier\_arreglo

19 CALL\_FUNCTION 1

20 STORE\_FAST tam

21 LOAD\_CONST 0

22 STORE\_FAST resultado

23 LOAD\_CONST 0

24 STORE\_FAST cont

25 LOAD\_CONST 0

26 STORE\_FAST sumatoria

27 LOAD\_FAST cont

28 LOAD\_FAST tam

29 LOAD\_CONST 1

30 BINARY\_SUBSTRACT

31 COMPARE\_OP <=

32 JUMP\_IF\_FALSE 44

33 LOAD\_FAST sumatoria

34 LOAD\_FAST cualquier\_arreglo

35 LOAD\_FAST cont

36 BINARY\_SUBSCR

37 BINARY\_ADD

38 STORE\_FAST sumatoria

39 LOAD\_FAST cont

40 LOAD\_CONST 1

41 BINARY\_ADD

42 STORE\_FAST cont

43 JUMP\_ABSOLUTE 27

44 LOAD\_FAST tam

45 LOAD\_CONST 0

46 COMPARE\_OP >

47 JUMP\_IF\_FALSE 55

48 LOAD\_FAST sumatoria

49 LOAD\_FAST tam

50 BINARY\_DIVIDE

51 STORE\_FAST resultado

52 LOAD\_FAST resultado

53 RETURN\_VALUE

54 JUMP\_ABSOLUTE 61

55 LOAD\_FAST sumatoria

56 LOAD\_FAST tam

57 BINARY\_MULTIPLY

58 STORE\_FAST resultado

59 LOAD\_FAST resultado

60 RETURN\_VALUE

Main()

61 LOAD\_GLOBAL calcularpromedioventas

62 CALL\_FUNCTION 0

**Bibliografía**

*Stackoverflow*. (n.d.). Retrieved from create a file and write to it in Java: https://stackoverflow.com/questions/2885173/how-do-i-create-a-file-and-write-to-it-in-java

*Stackoverflow*. (n.d.). Retrieved from ArrayList or List declaration in Java: https://stackoverflow.com/questions/12321177/arraylist-or-list-declaration-in-java

*Stackoverflow*. (n.d.). Retrieved from Print ArrayList: https://stackoverflow.com/questions/9265719/print-arraylist