



Práctica 3: Semántica y generación de código. El compilador completo.

591000 - Compiladores

Prof. Marçal Mora Cantallops, Universidad de Alcalá

11/11/2022

Objetivos de la práctica

- Construir un sistema completo de compilación, desde el análisis léxico hasta la ejecución del código.
- Ser capaz de generar un analizador léxico y sintáctico que reconozca un lenguaje en concreto.
- Ser capaz de generar código intermedio que transforme el lenguaje dado.
- Ser capaz de añadir mejoras y funciones adicionales al proceso.
- Profundizar en el concepto de la tabla de símbolos y la gestión de errores.

Consideraciones generales

- La práctica se realizará en grupos de hasta tres personas, siguiendo lo planteado en la PL2.
- Para la realización de la práctica se deberá usar ANTLR. Está permitido el uso de plugins para IntelliJ, NetBeans, Eclipse, Visual Studio Code, Visual Studio IDE, y jEdit.

- Para el tercer caso, se usará Jasmin como assembler del código intermedio generado (que, por lo tanto, deberá seguir su sintaxis).
- El entregable será tanto el código generado como una memoria explicativa del proceso seguido; dicha memoria será el principal factor en la valoración final (previa comprobación de que el código funciona adecuadamente).

Enunciado

Esta práctica consta de cuatro partes separadas.

Primera parte: del CSV al JSON (2 puntos)

Implemente, mediante el uso de visitors, un programa que use la gramática generada para reconocer CSV y traduzca su contenido a formato JSON.

Por ejemplo, en el ejemplo de la práctica anterior:

Nombre; Frase; Tipo; Lanzamiento
Aatrox; the Darkin Blade; Juggernaut; 2013-06-13
Ahri; the Nine-Tailed Fox; Burst; 2011-12-14
Akali; the Rogue Assassin; Assassin; 2010-05-11

Debería convertirse en salida en:

```
[ { Nombre: 'Aatrox',  
  Frase: 'the Darkin Blade',  
  Tipo: 'Juggernaut',  
  Lanzamiento: '2013-06-13' },  
  { Nombre: 'Ahri',  
    Frase: 'the Nine-Tailed Fox',  
    Tipo: 'Burst',  
    Lanzamiento: '2011-12-14' },  
  { Nombre: 'Akali',  
    Frase: 'the Rogue Assassin',  
    Tipo: 'Assassin',  
    Lanzamiento: '2010-05-11' } ]
```

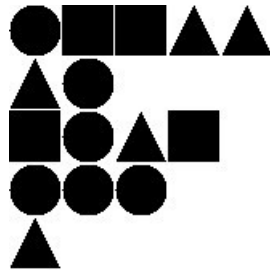
Segunda parte: dibujando las formas (2 puntos)

Implemente, mediante el uso de visitors, un programa que use la gramática generada para reconocer el conjunto de formas (lenguaje *formitas*) y genere un código en cualquier lenguaje (a vuestra elección) que pueda ejecutarse para dibujar las formas en pantalla. Podéis usar `BufferedImage` en Java, por ejemplo, pero no es ni la única opción ni probablemente la más sencilla.

Así, el código de ejemplo de la práctica anterior:

```
imgdim:180,shpdim:32
>>>circulo,cuadrado,cuadrado,triangulo,triangulo| triangulo,circulo|
cuadrado,circulo,triangulo,cuadrado| circulo,circulo,circulo| triangulo<<<
```

Deberá producir una salida que al ejecutar el código generado produzca una imagen similar a la siguiente (podéis cambiar de colores, estilo, etc.):



Tercera parte: generando código intermedio en Jasmin (1,5 puntos)

Esta primera parte es independiente de ANTLR y del lenguaje dado, aunque lo podéis usar de referencia.

Se proporciona una lista de ejemplos y se pide, para cada uno de ellos, realizar las siguientes tareas:

- Entender el ejemplo a ejecutar y proponer un código que cumpla con lo pedido.
- Traducir el código propuesto ("manualmente") a código intermedio Jasmin.

Mostrar el resultado de la ejecución del código intermedio generado.

Los ejemplos a realizar son los siguientes:

1. Ser capaz de generar un programa principal vacío.
2. Ser capaz de generar un programa principal que muestre por pantalla una cadena de texto.
3. Ser capaz de generar un programa principal que multiplique dos números enteros y lo muestre por pantalla.
4. Ser capaz de generar un programa principal que muestre por pantalla el resultado de una operación lógica.
5. Ser capaz de generar un programa principal que muestre por pantalla la concatenación de una cadena de texto y un número.
6. Ser capaz de generar un programa principal que realice uno o varios "if" anidados, mostrando el resultado de cada uno.
7. Ser capaz de generar un bucle for.
8. Ser capaz de generar un bucle while.

Los dos primeros ejemplos se proporcionan. El resto se valorarán a 0,25 por ejemplo correcto.

Cuarta parte: compilando E++ (4,5 puntos)

Esta parte es una extensión sobre el lenguaje analizado en la parte 3 de la práctica 2 y comprende la ejecución de las siguientes tareas:

Nivel básico (1,5 puntos):

- Extender el lenguaje para que pueda:
 - Realizar el resto de operaciones aritméticas comunes.
 - Tener bucles tipo for. Podéis llamarlos como queráis.
 - Tener bucles tipo while. Idem.
 - Admitir el tipo booleano, y, por lo tanto, sus operaciones.
- Adaptad lo que sea necesario en el lenguaje y justificad las decisiones de diseño tomadas.

Nivel intermedio (3 puntos):

- Implementar una estructura de tabla de símbolos, que mantenga las variables encontradas y sus valores/tipos cuando corresponda.
- Implementar la gestión de errores (incluyendo errores semánticos) para los errores que consideréis necesarios.
- Generación de código intermedio Jasmin para al menos un ejemplo de cada una de las funciones del lenguaje (es decir, para una serie de ejemplos representativos a modo de conjunto de pruebas). Estos ejemplos deben incluir también posibles casos de error.
- Ejecución del código intermedio generado para comprobar su correcto funcionamiento.

Para todos ellos debéis asumir que el programa entero estará en un solo fichero (es decir, no hay *includes* o similar).

Secuencia recomendada

Seguir el orden establecido en la propia práctica.

Defensa

En la defensa de la práctica se deberá exponer y explicar los distintos componentes que formen el sistema programado por el alumno, respondiendo a las preguntas del profesor, si corresponde. Adicionalmente, se probarán ficheros de entrada en formato texto que se deberán poder ejecutar (nota: esos ficheros, en caso de incluir funcionalidades extendidas, se adaptarán a la sintaxis especificada por los alumnos).

Ejemplos

Se encuentran como materiales anexos.

Material Adicional

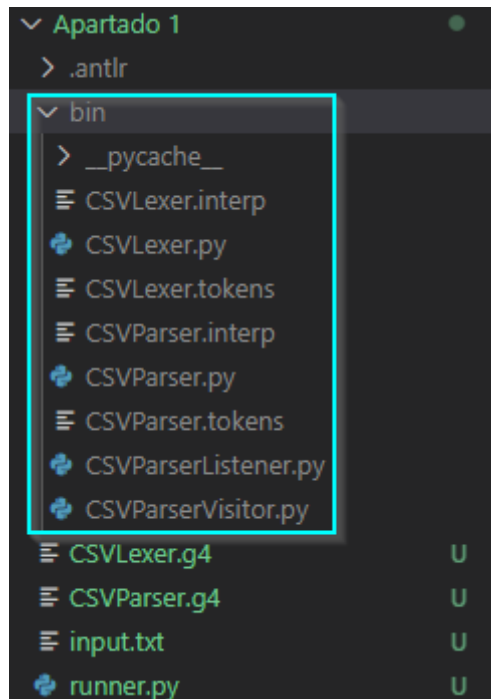
Tenéis también varios vídeos explicativos del proceso completo en la plataforma.

Contenido

Objetivos de la práctica.....	1
Consideraciones generales.....	1
Enunciado.....	1
Primera parte: del CSV al JSON (2 puntos).....	2
Segunda parte: dibujando las formas (2 puntos).....	3
Tercera parte: generando código intermedio en Jasmin (1,5 puntos).....	4
Cuarta parte: compilando E++ (4,5 puntos).....	5
Defensa.....	5
Material Adicional.....	6
Apartado 1.....	8
Apartado 2.....	11
Apartado 3.....	16
Ser capaz de generar un programa principal que multiplique dos números enteros y lo muestre por pantalla.....	17
Ser capaz de generar un programa principal que muestre por pantalla el resultado de una operación lógica.....	18
Ser capaz de generar un programa principal que muestre por pantalla la concatenación de una cadena de texto y un número.....	20
Ser capaz de generar un programa principal que realice uno o varios "if" anidados, mostrando el resultado de cada uno.....	21
Ser capaz de generar un bucle for.....	22
Ser capaz de generar un bucle while.....	23
Apartado 4.....	24
Características y flujo de datos de las expresiones.....	24
Cosas que me gustaría haber hecho.....	25

Apartado 1

Se ha realizado la siguiente estructura de ficheros:



A continuación procederemos a explicar por orden la generación de los ficheros y cómo se ha llegado a esa estructura.

1. Nos generamos el CSVLexer.g4 y el CSVParser.g4 y lo compilamos.
 - Explicado en la anterior práctica.
 - En el CSVParser.g4 se ha añadido dos etiquetas para mejorar la legibilidad.
 - Se ha de insertar la sentencia: “-visitor”

```
1  parser grammar CSVParser;
2
3  options{
4      tokenVocab=CSVLexer;
5  }
6
7  file      : header (line)* EOF;
8
9  header    : cell* NEWLINE;
10
11 line      : cell* NEWLINE;
12 cell      : TEXT_BLOCK # Text_block
13           | CONTENT    # Content
14           ;
15
```

```
~/mnt/e/Onedrive/Datos/Universidad/Cursos 3º/1º Cuatrimestre/compiladores/Laboratorio/4.0 - Práctica de Laboratorio 3/2.1 - Práctica/CompiladoresPL3/Apartado 1$ antlr4 -Dlanguage=Python3 -visitor -o bin CSVLexer.g4 CSVParser.g4
```


2. Se ha creado una nueva clase llamada "Visitor" dentro del fichero "runner.py"
- Se han modificado los métodos heredados de la clase CSVParserVisitor para recorrer los distintos elementos y retornarlos como str.
 - Cabe destacar que se ha utilizado una tabla de simbolos de tipo Lista en la que se añaden las cabeceras del fichero de entrada (la primera línea), que posteriormente se usará para proporcionar una salida correctamente formateada con dichas cabeceras.

```
from bin.CSVParserVisitor import CSVParserVisitor

class Visitor(CSVParserVisitor):
    tabla_simbolos:list = []

    def visitFile(self, ctx: Parser.FileContext):
        self.tabla_simbolos = self.visit(ctx.header())
        ret:str = ''
        for i, line in enumerate(ctx.line()):
            ret += self.visit(line)
            if not (i == len(ctx.line())-1):
                ret += ','
        return '[' + ret + ']'

    def visitHeader(self, ctx: Parser.HeaderContext):
        return [self.visit(cell) for cell in ctx.cell()]

    def visitLine(self, ctx: Parser.LineContext):
        ret:str = ''
        for i, cell in enumerate(ctx.cell()):
            ret += ' ' + self.tabla_simbolos[i] + ': '
            ret += '\"' + self.visit(cell) + '\"'
            if not (i == len(ctx.cell())-1):
                ret += ','
        return '{' + ret + '}'

    def visitContent(self, ctx: Parser.ContentContext):
        return str(ctx.CONTENT())

    def visitText_block(self, ctx: Parser.Text_blockContext):
        return str(ctx.TEXT_BLOCK())
```

```
# Visitor
v:Visitor = Visitor()
resultado:str = v.visit(tree)
print(resultado)
```

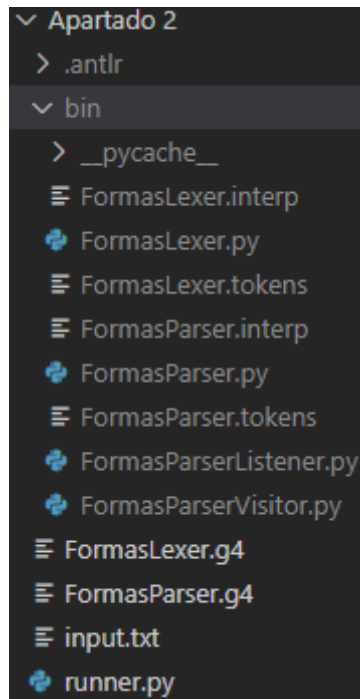
3. Al ejecutarlo retorna lo siguiente:

```
python -u "E:\OneDrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\4.0 - Práctica de Laboratorio 3\2.1 - Práctica\CompiladoresPL3\ejemplos\datos\com.csv"
(file (header (cell Type) (cell Fast Moves) (cell DPS) (cell Power-PvP) (cell Energy-PvP) (cell Cast Time) (cell Turns) (cell DPT) (cell EPT) \r\n) (line (cell Ste) (cell Steel Wing) (cell 13,8) (cell 11-7) (cell 6-5) (cell 0,8s) (cell 2) (cell 3,5) (cell 2,5) \r\n) (line (cell Dra) (cell Dragon Tail) (cell 13,6) (cell 15-9) (cell 9-10) (cell 1,1s) (cell 3) (cell 3,0) (cell 3,3) \r\n) <EOF>)
[ { Type: 'Ste', Fast Moves: 'Steel Wing', DPS: '13,8', Power-PvP: '11-7', Energy-PvP: '6-5', Cast Time: '0,8s', Turns: '2', DPT: '3,5', EPT: '2,5' }, { Type: 'Dra', Fast Moves: 'Dragon Tail', DPS: '13,6', Power-PvP: '15-9', Energy-PvP: '9-10', Cast Time: '1,1s', Turns: '3', DPT: '3,0', EPT: '3,3' } ]
```

[{ Type: 'Ste', Fast Moves: 'Steel Wing', DPS: '13,8', Power-PvP: '11-7', Energy-PvP: '6-5', Cast Time: '0,8s', Turns: '2', DPT: '3,5', EPT: '2,5' }, { Type: 'Dra', Fast Moves: 'Dragon Tail', DPS: '13,6', Power-PvP: '15-9', Energy-PvP: '9-10', Cast Time: '1,1s', Turns: '3', DPT: '3,0', EPT: '3,3' }]

Apartado 2

Se ha realizado la siguiente estructura de ficheros:



A continuación procederemos a explicar por orden la generación de los ficheros y cómo se ha llegado a esa estructura.

1. Nos generamos el FormasLexer.g4 y el FormasParser.g4 y lo compilamos.
 1. Explicado en la anterior práctica.
 2. Se ha de insertar la sentencia: “-visitor”

```
antlr4 -Dlanguage=Python3 -visitor -o bin FormasLexer.g4 FormasParser.g4
```

```
~/mnt/e/OneDrive/Datos/Universidad/Cursos 3º/1º Cuatrimestre/Compiladores/Laboratorio/4.0 - Práctica de Laboratorio 3/2.1 - Práctica/CompiladoresPL3/Apartado 2$ antlr4 -Dlanguage=Python3 -visitor -o bin FormasLexer.g4 FormasParser.g4
```

2. Se ha creado una nueva clase llamada "Visitor" dentro del fichero "runner.py"
 - Se han modificado los métodos heredados de la clase FormasParserVisitor para recorrer los distintos elementos y retornarlos o guardarlos en los atributos.
 - Esta clase hereda de la clase FormasParserVisitor generada con Antlr.
1. Se ha creado dos subclases: una llamada FigurasMeta que se trata de una Metaclass que utilizarán todas las enumeraciones que utilicemos en la clase Visitor. Otra clase llamada Figuras que se trata de una enumeración. Se han creado estas clases, debido a que permitir abstraer mejor los datos que se quieren representar/interpretar. Si queremos añadir un nuevo dato a interpretar, solamente debemos añadir un nuevo apartado en la enumeración.
2. Se han creado esos atributos para guardar la información interpretada por los métodos de la clase.

```
class Visitor(FormasParserVisitor):
    class FigurasMeta(EnumMeta):
        def __contains__(self, obj: str) -> bool:
            return str(obj) in [str(v.value) for v in self.__members__.values()]

    class Figuras(Enum, metaclass = FigurasMeta):
        CIRCULO = "circulo"
        CUADRADO = "cuadrado"
        TRIANGULO = "triangulo"

        def __eq__(self, __o: object) -> bool:
            return True if (str(self.value) == str(__o)) else False

# Atributos elementos Parser
imgdim:int = -1
shpdim:int = -1
filas:list = []
# Atributos para la GUI
window:Tk = None
canvas:Canvas = None

def __init__(self, tree:Parser.GeoContext) -> None:
    self.visit(tree=tree)
```

Los métodos para interpretar los datos de entrada serían:

```
def visitGeom(self, ctx: Parser.GeoContext):
    self.visit(ctx.dimensions())
    self.visit(ctx.shapes())
    return

def visitShapes(self, ctx: Parser.ShapesContext):
    for row in ctx.row():
        self.filas.append(self.visit(row))
    return

def visitRow(self, ctx: Parser.RowContext):
    ret: list = []
    for index_shape in range(len(ctx.SHAPE())):
        shape: str = str(ctx.SHAPE(index_shape))
        if shape in Visitor.Figuras:
            ret.append(shape)
    return ret

def visitDimensions(self, ctx: Parser.DimensionsContext):
    self.imgdim = self.visit(ctx.imgdim())
    self.shpdim = self.visit(ctx.shpdim())
    return

def visitImgdim(self, ctx: Parser.ImgdimContext):
    return int(ctx.INT().__str__())

def visitShpdim(self, ctx: Parser.ShpdimContext):
    return int(ctx.INT().__str__())

def __str__(self) -> str:
    return "imgdim: " + str(self.imgdim) + " - shpdim: " + str(self.shpdim) + " - filas: " + str(self.filas)
```

- Como se puede observar, se van guardando los datos que se interpretan en los atributos de la clase.
- Hasta ahora, nuestro código haría lo siguiente:

```
PS E:\OneDrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\4.0 - Práctica de Laboratorio 3\2.1 - Práctica\CompiladoresPL3> python -u "e:\OneDrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\4.0 - Práctica de Laboratorio 3\2.1 - Práctica\CompiladoresPL3\Apartado 2\runner.py"
(gem (dimensions (imgdim imgdim: 180) , (shpdim shpdim: 32)) (shapes >>> (row circulo , cuadrado , cuadrado , triangulo , triangulo) | (row triangulo , circulo) | (row cuadrado , circulo , triangulo , cuadrado) | (row circulo , circulo , circulo) | (row triangulo) <<<) <EOF>)
imgdim: 180 - shpdim: 32 - filas: [['circulo', 'cuadrado', 'cuadrado', 'triangulo', 'triangulo'], ['triangulo', 'circulo'], ['cuadrado', 'circulo', 'triangulo', 'cuadrado'], ['circulo', 'circulo', 'circulo'], ['triangulo']]
Enter para salir
```

imgdim: 180 - shpdim: 32 - filas: [['circulo', 'cuadrado', 'cuadrado', 'triangulo',
'triangulo'], ['triangulo', 'circulo'], ['cuadrado', 'circulo', 'triangulo', 'cuadrado'], ['circulo',
'circulo', 'circulo'], ['triangulo']]

- Le programamos una interfaz gráfica que muestre los datos guardados en los atributos de esta forma: (Al utilizar una enumeración es más fácil distinguir los diferentes tipos de datos)

```
def dibujar(self):
    # Dibujar ventana
    self.window = Tk()
    self.window.configure(background = "white")
    self.window.title("Formas")
    # Dibujar canvas
    self.canvas = Canvas(self.window, width=self.imgdim, height = self.imgdim, bg="gray")
    self.canvas.pack()
    # Dibujar elementos
    color:str = "pink"
    for index_fila, fila in enumerate(self.filas):
        for index_elemento, elemento in enumerate(fila):
            if (elemento == Visitor.Figuras.CIRCULO):
                self.canvas.create_oval((index_elemento*self.shpdim), (index_fila*self.shpdim),\
                    ((index_elemento+1)*self.shpdim), ((index_fila+1)*self.shpdim), fill = color)
                next
            if (elemento == Visitor.Figuras.CUADRADO):
                self.canvas.create_rectangle((index_elemento*self.shpdim), (index_fila*self.shpdim),\
                    ((index_elemento+1)*self.shpdim), ((index_fila+1)*self.shpdim), fill =color)
                next
            if (elemento == Visitor.Figuras.TRIANGULO):
                self.canvas.create_polygon((index_elemento*self.shpdim), (index_fila*self.shpdim),\
                    ((index_elemento+1)*self.shpdim), (index_fila*self.shpdim),\
                    int(((index_elemento*self.shpdim)+((index_elemento+1)*self.shpdim))/2), ((index_fila+1)*self.shpdim), fill = color)
                next
```

- En la clase principal (Main) nos creamos un objeto de la clase anterior y le pasamos el arbol a interpretar.

```
# Visitor
v:Visitor = Visitor(tree=tree)
print(v)
v.dibujar()
input("Enter para salir")
```

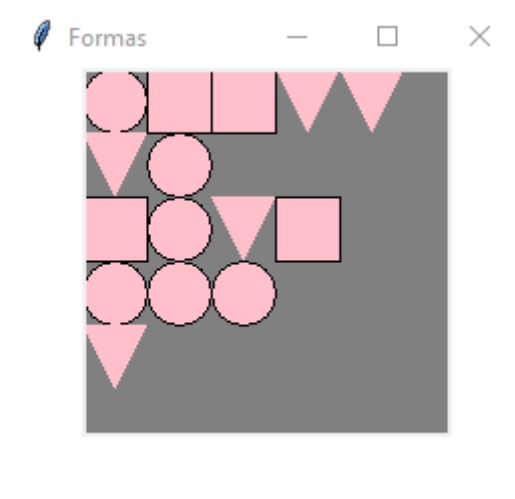
3. Al ejecutarlo retorna lo siguiente:

```
PS E:\OneDrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\4.0 - Práctica de Laboratorio 3\2.1 - Práctica\CompiladoresPL3> python -u "e:\OneDrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\4.0 - Práctica de Laboratorio 3\2.1 - Práctica\CompiladoresPL3\Apartado 2\runner.py"
(gom (dimensions (imgdim imgdim: 180) , (shpdim shpdim: 32)) (shapes >>> (row circulo , cuadrado , cuadrado , triangulo , triangulo) | (row triangulo , circulo) | (row cuadrado , circulo , triangulo , cuadrado) | (row circulo , circulo , circulo) | (row triangulo) <<<) <EOF>)
imgdim: 180 - shpdim: 32 - filas: [['circulo', 'cuadrado', 'cuadrado', 'triangulo', 'triangulo'], ['triangulo', 'circulo'], ['cuadrado', 'circulo', 'triangulo', 'cuadrado'], ['circulo', 'circulo', 'circulo'], ['triangulo']]
Enter para salir
```

(gom (dimensions (imgdim imgdim: 180) , (shpdim shpdim: 32)) (shapes >>> (row circulo , cuadrado , cuadrado , triangulo , triangulo) | (row triangulo , circulo) | (row cuadrado , circulo , triangulo , cuadrado) | (row circulo , circulo , circulo) | (row triangulo) <<<) <EOF>)

imgdim: 180 - shpdim: 32 - filas: [['circulo', 'cuadrado', 'cuadrado', 'triangulo', 'triangulo'], ['triangulo', 'circulo'], ['cuadrado', 'circulo', 'triangulo', 'cuadrado'], ['circulo', 'circulo', 'circulo'], ['triangulo']]

Enter para salir



- Como se puede observar interpreta correctamente los datos, y los tamaños.
 - o El canvas (correspondiente por el recuadro gris) tiene el tamaño especificado en el atributo imgdim. - imgdim: 180
 - o Cada una de las figuras tienen el tamaño especificado en el atributo shpdim. - shpdim: 32
 - o Los datos se distinguen y se dibujan según la interpretación realizada anteriormente.

Apartado 3

Los ejemplos a realizar son los siguientes:

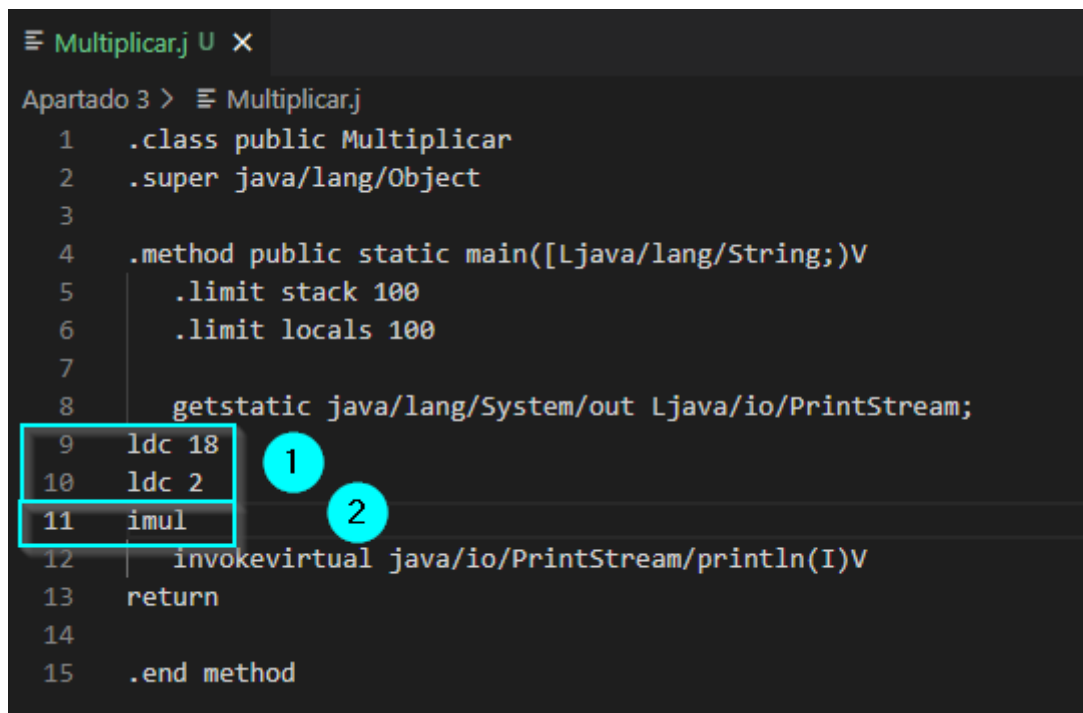
1. Ser capaz de generar un programa principal vacío. **PROPORCIONADO POR EL PROFESOR**
2. Ser capaz de generar un programa principal que muestre por pantalla una cadena de texto. **PROPORCIONADO POR EL PROFESOR**

Ser capaz de generar un programa principal que multiplique dos números enteros y lo muestre por pantalla.

- o En este apartado se pide multiplicar dos números como se muestra en la imagen.

```
>>> a = 18
>>> b = 2
>>> print(a*b)
36
>>>
```

- o Para ello, vamos a ir almacenando los números en la pila y se hará la multiplicación de la siguiente forma:



```
Multiplicar.j U X
Apartado 3 > Multiplicar.j
1 .class public Multiplicar
2 .super java/lang/Object
3
4 .method public static main([Ljava/lang/String;)V
5     .limit stack 100
6     .limit locals 100
7
8     getstatic java/lang/System/out Ljava/io/PrintStream;
9     ldc 18
10    ldc 2
11    imul
12    invokevirtual java/io/PrintStream/println(I)V
13    return
14
15 .end method
```

- o Al ejecutarlo muestra lo siguiente:

```
PS E:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio4.0 - Práctica de Laboratorio 3\2.1 - Práctica\CompiladoresPL3\Apartado 3> java -jar .\Jasmin.jar *.j
Generated: HolaMundo.class
Generated: Multiplicar.class
Generated: Sumar.class
Generated: Vacio.class
PS E:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio4.0 - Práctica de Laboratorio 3\2.1 - Práctica\CompiladoresPL3\Apartado 3> java Multiplicar
36
```

Ser capaz de generar un programa principal que muestre por pantalla el resultado de una operación lógica.

- o Realizar una operación lógica como se muestra en la imagen.

```
>>> a:bool = True
>>> b:bool = False
>>> print(a and b)
False
>>> print(a or b)
True
```

- o Jasmin no cuenta con tipos Booleanos, sino con tipos enteros (Int) donde el 1 es TRUE y el 0 es FALSE.
- o Para ello, vamos a ir almacenando los elementos booleanos (con valores 1 para TRUE y 0 para FALSE) en la pila y se hará las operaciones lógicas de la siguiente forma:

```
Apartado 3 > ≡ Logica.j
1  .class public Logica
2  .super java/lang/Object
3
4  .method public static main([Ljava/lang/String;)V
5      .limit stack 100
6      .limit locals 100
7
8      getstatic java/lang/System/out Ljava/io/PrintStream;
9      ldc 1
10     ldc 0
11     iand
12     invokevirtual java/io/PrintStream/println(I)V
13     return
14
15     .end method
```

```
Apartado 3 > ≡ Logica.j
1  .class public Logica
2  .super java/lang/Object
3
4  .method public static main([Ljava/lang/String;)V
5      .limit stack 100
6      .limit locals 100
7
8      getstatic java/lang/System/out Ljava/io/PrintStream;
9      ldc 1
10     ldc 0
11     ior
12     invokevirtual java/io/PrintStream/println(I)V
13     return
14
15     .end method
```

- o Al ejecutarlo muestra lo siguiente:

```
PS E:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\4.0 - Práctica de Laboratorio 3\2.1 - Práctica\CompiladoresPL3\Apartado 3> java -jar .\jasmin.jar *.j
Generated: HolaMundo.class
Generated: Logica.class
Generated: Multiplicar.class
Generated: Sumar.class
Generated: Vacio.class
PS E:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\4.0 - Práctica de Laboratorio 3\2.1 - Práctica\CompiladoresPL3\Apartado 3> java Logica
0
```

Operación True and False = 0 (False)

```
PS E:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\4.0 - Práctica de Laboratorio 3\2.1 - Práctica\CompiladoresPL3\Apartado 3> java -jar .\jasmin.jar *.j
Generated: HolaMundo.class
Generated: Logica.class
Generated: Multiplicar.class
Generated: Sumar.class
Generated: Vacio.class
PS E:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\4.0 - Práctica de Laboratorio 3\2.1 - Práctica\CompiladoresPL3\Apartado 3> java Logica
1
```

Operación True or False = 1 (True)

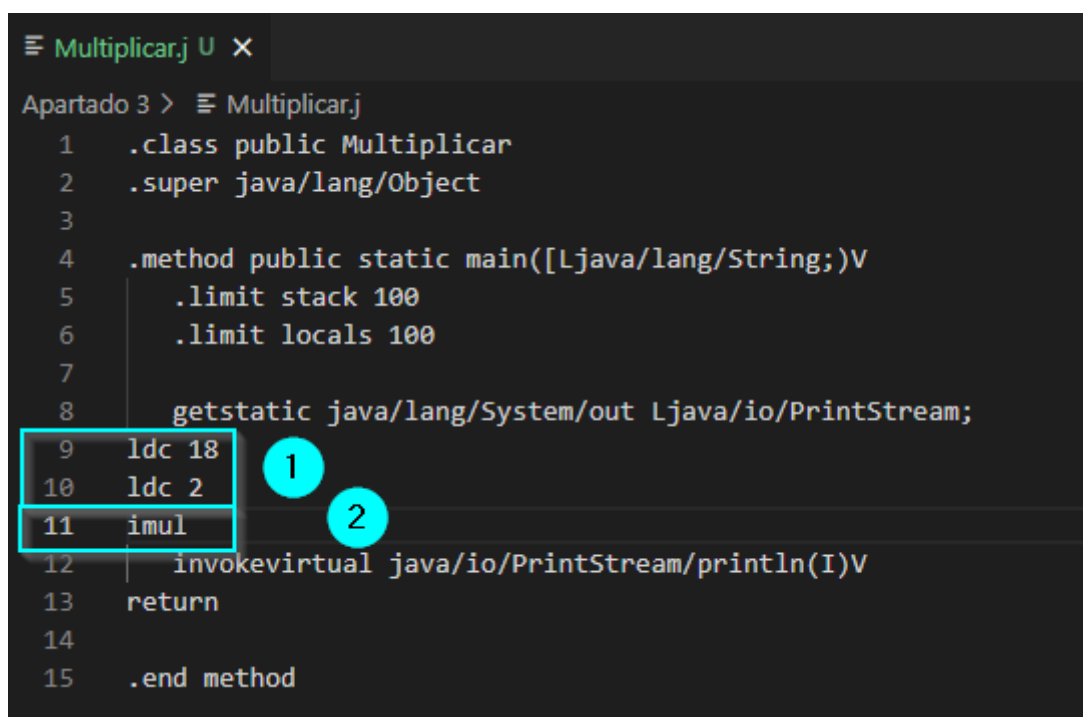
Ser capaz de generar un programa principal que muestre por pantalla la concatenación de una cadena de texto y un número.

- o En este apartado se pide mostrar un texto y un número como se muestra en la imagen.

```
>>> a = 18
>>> b = 2
>>> print(a*b)
36
>>>
```

<http://www2.cs.uidaho.edu/~jeffery/courses/445/code-jasmin.html>

- o Para ello, vamos a ir almacenando los números en la pila y se hará la concatenación de la siguiente forma:



```

Multiplicar.j U X
Apartado 3 > Multiplicar.j
1  .class public Multiplicar
2  .super java/lang/Object
3
4  .method public static main([Ljava/lang/String;)V
5      .limit stack 100
6      .limit locals 100
7
8      getstatic java/lang/System/out Ljava/io/PrintStream;
9      ldc 18      1
10     ldc 2
11     imul      2
12     invokevirtual java/io/PrintStream/println(I)V
13     return
14
15     .end method
```

- o Al ejecutarlo muestra lo siguiente:

```
PS E:\OneDrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio4.0 - Práctica de Laboratorio 3\2.1 - Práctica\CompiladoresPL3\Apartado 3> java -jar .\jasmin.jar *.j
Generated: HolaMundo.class
Generated: Multiplicar.class
Generated: Sumar.class
Generated: Vacio.class
PS E:\OneDrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio4.0 - Práctica de Laboratorio 3\2.1 - Práctica\CompiladoresPL3\Apartado 3> java Multiplicar
36
```

Ser capaz de generar un programa principal que realice uno o varios “if” anidados, mostrando el resultado de cada uno.

Ser capaz de generar un bucle for.

Ser capaz de generar un bucle while.

Apartado 4

No se ha podido terminar este apartado por falta de tiempo. Hemos tenido muchas cosas que hacer y no nos hemos podido poner antes con la práctica. Pero lo que tenemos es lo siguiente.

Se creó la gramática del lenguaje, facilitando el procesado del parse tree en python mediante el uso de tags para las distintas variantes de las reglas del parser.

No se ha hecho ningún tipo de análisis semántico dentro de la gramática para desacoplarla del lenguaje de implementación del compilador.

Debido a la falta de tiempo solo se ha podido hacer el parsing de expresiones de forma incompleta. Aunque al haberse codificado una infraestructura para generalizar la creación de operaciones y conversiones, se simplifica mucho la traducción a jasmin.

Características y flujo de datos de las expresiones

Las expresiones se convierten a notación polaca inversa mediante visitors, usando tipos enumerados y dataclasses de python para organizar y añadir información sobre los tipos de datos, entre otras cosas.

```
[Leaf(type=<DataType.INTEGER: (0,)>, value=5), Leaf(type=<DataType.INTEGER: (0,)>, value=2), Operation(lhs=<DataType.INTEGER: (0,)>, rhs=<DataType.INTEGER: (0,)>, ret=<DataType.INTEGER: (0,)>, inter='isub')]
```

Además, se comprueba que las operaciones que se quieren realizar existan dentro del lenguaje. Para ello se han definido una serie de estructuras de datos dentro del archivo compiler/expression.py que permiten especificar las operaciones soportadas por el lenguaje, así como las instrucciones de jasmin a las que se traduce. En caso de que los tipos de una expresión no coincidan con los que están definidos, se intentan aplicar conversiones de tipos de datos (casting) a los operandos para encontrar una combinación que funcione. El método utilizado actualmente es muy primitivo y se podría hacer mucho mejor. De nuevo, con más tiempo.

Una vez se ha realizado este proceso, una expresión quedaría así (asignar b = "Hola mundo" + a;P):

```
[Leaf(type=<DataType.STRING: (3,)>, value='Hola mundo'), Leaf(type=<LeafType.SYMBOL: (0,)>, value=Symbol(name='a', type=<DataType.FLOAT: (1,)>, index=0)), CastInformation(from_type=<DataType.FLOAT: (1,)>, to_type=<DataType.STRING: (3,)>, inter_action='\n{load}\ninvokestatic java/
```

```

lang/Float/toString(F)Ljava/lang/String;'), Operation(lhs=<DataType.STRING: (3,)>,
rhs=<DataType.STRING: (3,)>, ret=<DataType.STRING: (3,)>,
inter="\nnew java/lang/StringBuilder\ndup\nninvokespecial
java/lang/StringBuilder/<init>()V\n{lhs}\ninvokevirtual java/lang/StringBuilder/ap
pend(Ljava/lang/String;)Ljava/lang/StringBuilder;\n{rhs}\ninvokevirtual
java/lang/StringBuilder/append(Ljava/lang/String;)Ljava/lang/StringB
uilder;\ninvokevirtual java/lang/StringBuilder/toString()Ljava/lang/String;')]]

```

Teniendo ya este array se puede generar el código de Jasmin fácilmente:

```

new java/lang/StringBuilder
dup
invokespecial java/lang/StringBuilder/<init>()V
ldc "Hola mundo"
invokevirtual
java/lang/StringBuilder/append(Ljava/lang/String;)Ljava/lang/StringBuilder;

fload_0
invokestatic java/lang/Float/toString(F)Ljava/lang/String;
invokevirtual
java/lang/StringBuilder/append(Ljava/lang/String;)Ljava/lang/StringBuilder;
invokevirtual java/lang/StringBuilder/toString()Ljava/lang/String;

```

Los índices de las variables así como las labels usadas en conversiones y operaciones se generan automáticamente de forma única.

Cosas que me gustaría haber hecho

Obviamente terminar lo básico de la práctica. Pero con más tiempo me hubiese gustado aplicar optimizaciones al sistema de búsqueda de operadores válidos, métodos (se podrían añadir fácilmente con la gramática que tenemos ahora).

- Optimizaciones al sistema de búsqueda de operadores válidos
- Métodos (se podrían añadir fácilmente con la gramática que se tiene ahora).
- Scope a la tabla de símbolos

Me gustaría ir a la defensa para explicar todas estas ideas y como habría hecho el resto de la práctica.