MANUAL TÉCNICO

Clase Controller

Método “readFileGLC”

Se encarga de leer el archivo de gramáticas libres del contexto, recibiendo como parámetro la ruta del archivo.

```
def readFileGLC(self,route):
    self.inputFile = open(route,encoding='utf-8').read()
```

Método “grammarRecognition”

Se encarga de hacer un Split de salto de línea al texto obtenido del archivo cargado previamente, creando una lista en la que en cada posición hay una línea del archivo de entrada.

```
def grammarRecognition(self):
    self.inputFile = self.inputFile.split('\n')
    self.identifyElementsGLC()
```

Método “popLine”

Se encarga de eliminar la posición 0 de la lista creada previamente a partir del archivo de entrada.

```
def popLine(self):
    try:
        return self.inputFile.pop(0)
    except:
        return None
```

Método “viewLine”

Se encarga de verificar que aun existan elementos dentro de la lista del archivo de entrada.

```
def viewLine(self):
    try:
        return self.inputFile[0]
    except:
        return None
```

Método “identifyElementsGLC”

Se encarga de agregar línea por línea los elementos del archivo de entrada a los atributos correspondientes del objeto de gramáticas, verificando que la gramática ingresada cumpla con ser una gramática libre del contexto para poder ser cargado al sistema. Al momento de que el método **viewLine** retorne un símbolo de porcentaje (%) la gramática será agregada a la lista de gramáticas, mientras el método siga retornando elementos el método **identifyElementsGLC** hará una llamada recursiva.

```
def identifyElementsGLC(self):
    if self.line == 0:
        self.grammar = GLC()
        self.grammar.name = self.popLine()
    elif self.line == 1:
        self.grammar.nonTerminals = self.popLine().split(',')
        for nonTerminal in self.grammar.nonTerminals:
            self.grammar.path[nonTerminal] = {}
    elif self.line == 2:
        self.grammar.terminals = self.popLine().split(',')
    elif self.line == 3:
        self.grammar.initialNonTerminal = self.popLine()
    elif self.line == 4:
        self.productions = []
        self.isValid = True
        self.isGLC = False
    if self.line >= 4:
        production = self.popLine().split('>')
        destinationInput = production[1].split(' ')

        production[1] = [s for s in destinationInput if s]
        production[0] = production[0].replace(' ','')
        if not production[0] in self.grammar.nonTerminals:
            self.isValid = False
        if len(production[1]) == 1:
            if production[1][0] in self.grammar.nonTerminals:
                self.productions.append(Production(production[0],destiny=production[1][0]))


            elif production[1][0] in self.grammar.terminals:
                self.productions.append(Production(production[0],input1=production[1][0]))


            else: self.isValid = False
        elif len(production[1]) == 2:
            if production[1][0] in self.grammar.terminals and production[1][1] in self.grammar.nonTerminals:
                self.productions.append(Production(production[0],production[1][0],production[1][1]))


            else: self.isValid = False
        elif len(production[1]) == 3:
            if production[1][0] in self.grammar.terminals and production[1][1] in self.grammar.nonTerminals and product
                self.productions.append(Production(production[0],production[1][0],production[1][1],production[1][2]))
                self.isGLC = True


            else: self.isValid = False

    self.line += 1
    if self.viewLine() == '%':
        if self.isValid and self.isGLC:
            self.grammar.productions = self.productions
            self.grammar.path = self.getPathGLC()
            self.grammars.append(self.grammar)
        self.popLine()
        self.count = 0
        self.line = 0
    if self.viewLine():
        self.identifyElementsGLC()
```

Clase Stack

Método “checkPop”

Se encarga de verificar que lo que se saque de la pila esté dentro de la pila y sea el último elemento agregado el pop es válido. Si no hay nada en la pila o hay algo y no se saca nada de la pila el pop es válido.

Método “popStack”

Se encarga de verificar que lo que se saque de la pila sea el último elemento agregado de lo contrario si no se saca nada no se modifica la pila.

Método “addStack”

Se encarga de verificar que el elemento a agregar no sea el signo de dólar ($) de lo contrario no agrega nada.

```python
class Stack(list):
    def checkPop(self,pop):
        if len(self) == 0 and pop == '$':
            return True
        if len(self) > 0 and pop != '$':
            if pop == self[len(self) - 1]:
                return True
        if len(self) > 0 and pop == '$':
            return True
        return False

    def popStack(self,pop):
        if pop != '$':
            if pop == self[len(self) - 1]:
                self.pop()

    def addStack(self,add):
        if add != '$':
            self.append(add)
```