Manual Tecnico

Autor: James Osmin Gramajo Cárcamo DPI: 3517 27817 0922
Estudiante Externo Centro Universitario de Occidente (CUNOC) Quetzaltenango

Requerimientos Mínimos:

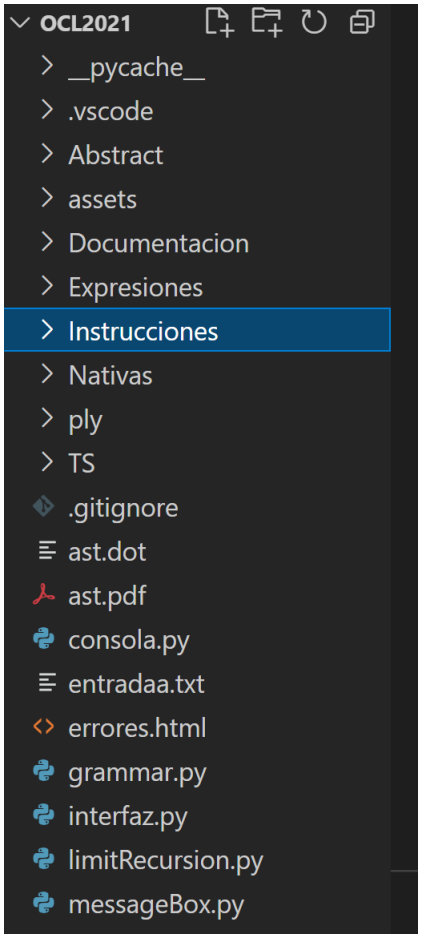
- Procesador: 2 GHz compatible con PAE, NX y SSE2.
- RAM: 1 GB (32 bits) o 2 GB (64 bits).
- Espacio en disco duro: 512 Mb (32 bits) o 1 GB (64 bits).
- Tarjeta gráfica: Dispositivo gráfico Microsoft DirectX 9 con
- controlador
- WDDM
- Resolución de pantalla de al menos 1366 x 768 píxeles.
- Es posible que algunos juegos y programas requieran tarjetas
- gráficas compatibles con DirectX 10 o superior para un rendimiento óptimo.
- Python 3.8.1 o posterior
- Maquina virtual(Virtual machine) o cualquier otro sistema operativo.
- Windows o Linux.
- Editor de Codigo para su ejecución local (Visual Studio) o cualquier otro compatible.

APLICACIÓN: JPR Compilador 2021 (Python)
Funcionalizades:
Analizar y ejecutar código JPR Language 1.0 analizar código de manera
Lexica sintáctica y semántica,
elaborar reporte de errores (Lexico, sintactico y semánticos)
Gestor de Archivos(Abir, guardar, guardar como)
Generar AST
Tabla de símbolos.

Codigo fuente y explicaciones técnicas generales

El compilador fue elaborado con el lenguaje de programación Pytthon y la herramienta PLY para el reconocimiento de la gramática(lenguaje)

A continuación se muestra las siguientes carpetas contenedoras del código fuente:



CARPETA Abstrac:

Esta carpeta contiene dos clases

Instrucción.py

(esta clase @abstractmethod encargada de las ejecuciones de todas las instrucciones del código que se definirán mas adelante.

NodoAST.py

Esta clase es la encargada de generar dichos nodos y sus derivados para la graficacion del Arbol AST.

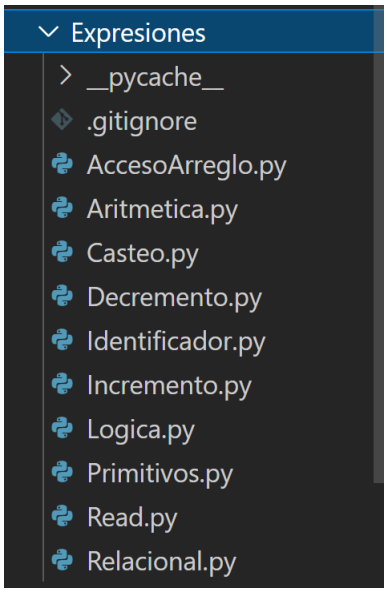
CARPETA Assets:

Esta carpeta contiene el formato .html [ara la exportación de errores asi también los archivos .js y .css para la generarcion de dicho archivo, este es utilizado para la exportación de reporte de errores que contiene el código ingresado por el usuario.

CARPETA Documentacion:

Esta carpeta contiene la documentación necesaria para el programa:
Manual de Usuario.pdf
Manual Tecnico y la Gramatica utilizada para el reconocimiento del lenguaje.

CARPETA Expresiones:



Esta carpeta tiene contiene las clases

AccesoArreglo.py :

encargada de acceder a los arreglos mediante y sus posiciones en memoria que almacenan en cada dimensión.

Aritmetica.py:

Dicha clase es la encargada de validar y ejecutar las operaciones aritméticas que admite el lenguajes.

```
if self.operador == OperadorAritmetico.MAS: #SUMA
```

esta analiza si el operador aritmético es una suma

luego verifica los dos lados de la operación tanto el izquierdo como el derecho

```
if self.OperacionIzq.tipo == TIPO.ENTERO and self.OperacionDer.tipo == TIPO.ENTER
O:
                    self.tipo = TIPO.ENTERO
                    return self.obtenerVal(self.OperacionIzq.tipo, izq) + self.ob
tenerVal(self.OperacionDer.tipo, der)
```

si los dos son ENTEROS ejecuta la suma y devuelve el valor tipo entero y asi ejecuta cada una de las operaciones tanto en SUMA, RESTA, MULTIPLICACION, DIVISION, MODULO, POTENCIA.

Casteo.py

Esta clase es la encargada castear un dato o variable y convertirla a su equivalente en el formato deseado.

```
if self.tipo == TIPO.DECIMAL:
            # (DOUBLE) ENTERO
            if self.expresion.tipo == TIPO.ENTERO:
                try:
                    return float(self.obtenerVal(self.expresion.tipo, val))
                except:
                    return Excepcion("Semantico", "No se puede castear para (DOUB
LE) INT.", self.fila, self.columna)
```

verifica si el tipo de dato a convertir es Decimal(DOuble) entero

verifica si el tipo de dato de la expresión es entero y retorna el numero entero en formato decimal. Y así ejecuta todas las posibilidades de casteos permitida en el Compilador JPR

Incremento.py

Esta función le suma una unidad (+1) a la variable indicada de la siguiente forma

```
if simbolo.tipo == TIPO.DECIMAL or simbolo.tipo == TIPO.ENTERO:
            self.tipo = simbolo.tipo
            simbolo.valor = simbolo.valor-1
            table.actualizarTabla(simbolo)
            return simbolo.valor
```

si la variable es de tipo entero o decimal esta le suma una unidad y retorna su nuevo valor.

Decremento.py

Esta función le resta una unidad (-1) a la variable indicada de la siguiente forma

```
if simbolo.tipo == TIPO.DECIMAL or simbolo.tipo == TIPO.ENTERO:
            self.tipo = simbolo.tipo
            simbolo.valor = simbolo.valor-1
            table.actualizarTabla(simbolo)
            return simbolo.valor
```

si la variable es de tipo entero o decimal esta le suma una unidad y retorna su nuevo valor

Identificador.py:

Esta clase es la que registra el ID de una variable en la tabla de símbolos además para que el programa pueda ser sensitive case los identificadores o ID sosn almacenados en la tabla de símbolos en minúscula.

```
 simbolo = table.getTabla(self.identificador.lower())
            self.tipo = simbolo.getTipo()
            return simbolo.getValor()
```

Logica.py

Esta clase trabaja igualmente que la clase lógica compara los dos lados de una operación y ejecuta la operación según sea la condición indicada

```
def interpretar(self, tree, table):
        izq = self.OperacionIzq.interpretar(tree, table)
        if isinstance(izq, Excepcion): return izq
        if self.OperacionDer != None:
            der = self.OperacionDer.interpretar(tree, table)
            if isinstance(der, Excepcion): return der

        if self.operador == OperadorLogico.AND:
            if self.OperacionIzq.tipo == TIPO.BOOLEANO and self.OperacionDer.tipo
 == TIPO.BOOLEANO:
                return self.obtenerVal(self.OperacionIzq.tipo, izq) and self.obte
nerVal(self.OperacionDer.tipo, der)
            return Excepcion("Semantico", "Tipo Erroneo de operacion para &&.", s
elf.fila, self.columna)
        elif self.operador == OperadorLogico.OR:
```

```
            if self.OperacionIzq.tipo == TIPO.BOOLEANO and self.OperacionDer.tipo
 == TIPO.BOOLEANO:
                return self.obtenerVal(self.OperacionIzq.tipo, izq) or self.obten
erVal(self.OperacionDer.tipo, der)
            return Excepcion("Semantico", "Tipo Erroneo de operacion para ||.", s
elf.fila, self.columna)
        elif self.operador == OperadorLogico.NOT:
            if self.OperacionIzq.tipo == TIPO.BOOLEANO:
                return not self.obtenerVal(self.OperacionIzq.tipo, izq)
            return Excepcion("Semantico", "Tipo Erroneo de operacion para !.", se
lf.fila, self.columna)
        return Excepcion("Semantico", "Tipo de Operacion logica no Especificado."
, self.fila, self.columna)
```

verifica si son datos booleanos busca excepciones u errores retorna un valor.

Primitivos.py

Encargada de todos los datos almacenador en una variables y ejecuta el método abstracto de una instrucción dada.

Read.py

Esta clase es la encargada de leer datos del usuario mas bien es un método de entrada de datos para la ejecución de una aplicación,

```
class Read(Instruccion):
    def __init__(self, fila, columna):
        self.fila = fila
        self.columna = columna
        self.tipo = TIPO.CADENA

    def interpretar(self, tree, table):

        LECTURA_VAL =MiniConsola()
        valorIngresado= LECTURA_VAL.getValor()
        return valorIngresado



    def getNodo(self):
        nodo = NodoAST("READ")
        return nodo
```

al ejecutar dicha instrucción Read esta hace una nueva instancia a la clase MiniConsola que es la encargada de mostar una ventana emergente con un input para la entrada de datos al sistema y los datos los almacena como TIPO CADENA o STRING y mediante el método getValor() se obtienen el valor ingresado en el input.

MiniConsola.py

Esta clase muestra una ventana visual con un input para la entrada de datos al sistema y captura los datos y los almacena como TIPO CADENA o STRING y mediante el método getValor() se obtienen el valor ingresado en el input

```
from Abstract.NodoAST import NodoAST
from Abstract.Instruccion import Instruccion
from TS.Tipo import TIPO
import tkinter as tk
from tkinter import simpledialog

class MiniConsola():
    # the input dialog
    def __init__(self):
        pass

    def getValor(self):
        ROOT = tk.Tk()
        ROOT.withdraw()
        self.USER_INP = simpledialog.askstring(title="Consola JPR",prompt="Ingres
a en consola:")
        print(self.USER_INP)
        return self.USER_INP

    def close_window (root):
        root.destroy()
```

Relacional.py

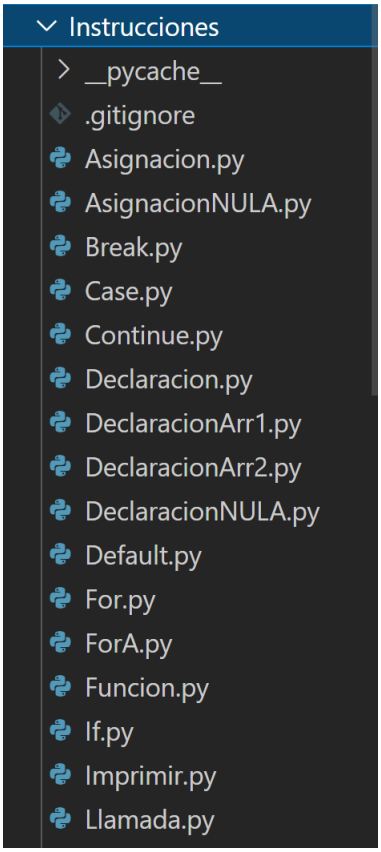
Esta clase trabaja de igual forma que Artimetica.py y Logica.py

Esta es la encargada de ejecutar y/u ejecutar las operaciones relacionales mediante los símbolos > , >= <, <=, == ¡ entre otras.

```
#-----------------------------------------MENOR QUE  < --------------------
----------------------------
        if self.operador == OperadorRelacional.MENORQUE:
            if self.OperacionIzq.tipo == TIPO.ENTERO and self.OperacionDer.tipo =
= TIPO.ENTERO:
                return self.obtenerVal(self.OperacionIzq.tipo, izq) < self.obtene
rVal(self.OperacionDer.tipo, der)
            elif self.OperacionIzq.tipo == TIPO.ENTERO and self.OperacionDer.tipo
 == TIPO.DECIMAL:
                return self.obtenerVal(self.OperacionIzq.tipo, izq) < self.obtene
rVal(self.OperacionDer.tipo, der)
            elif self.OperacionIzq.tipo == TIPO.DECIMAL and self.OperacionDer.tip
o == TIPO.ENTERO:
                return self.obtenerVal(self.OperacionIzq.tipo, izq) < self.obtene
rVal(self.OperacionDer.tipo, der)
            elif self.OperacionIzq.tipo == TIPO.DECIMAL and self.OperacionDer.tip
o == TIPO.DECIMAL:
                return self.obtenerVal(self.OperacionIzq.tipo, izq) < self.obtene
rVal(self.OperacionDer.tipo, der)
            elif self.OperacionIzq.tipo == TIPO.BOOLEANO and self.OperacionDer.ti
po == TIPO.BOOLEANO:
                return self.obtenerVal(self.OperacionIzq.tipo, izq) < self.obtene
rVal(self.OperacionDer.tipo, der)
            return Excepcion("Semantico", "Tipo Erroneo de operacion para <.", se
lf.fila, self.columna)
```

y verifica ambos lados de la operación verifica que sea la compinacion entero entero o decimal entero etc. y valida las operación y retorna el valor en Tipo Booleano.

CARPETA Instrucción



Asignacion.py

Esta clase realiza la asignación de valor a una variable dereclara y crear el símbolo con sus atributos y lo almcena en la tabla de símbolos.

```
class Asignacion(Instruccion):
    def __init__(self, identificador, expresion, fila, columna):
        self.identificador = identificador
        self.expresion = expresion
        self.fila = fila
        self.columna = columna
        self.arreglo = False

    def interpretar(self, tree, table):
        value = self.expresion.interpretar(tree, table) # Valor a asignar a la va
riable
        if isinstance(value, Excepcion):
            tree.getExcepciones().append(value)
```

```
            tree.updateConsolaError(value.toString())
            return value

        simbolo = Simbolo(self.identificador, self.expresion.tipo, self.arreglo,
self.fila, self.columna, value)

        result = table.actualizarTabla(simbolo)

        if isinstance(result, Excepcion):
            tree.getExcepciones().append(result)
            tree.updateConsolaError(result.toString())
            return result
        return None

    def getNodo(self):
        nodo = NodoAST("ASIGNACION")
        nodo.agregarHijo(str(self.identificador))
        nodo.agregarHijoNodo(self.expresion.getNodo())
        return nodo
```

AsignacionNula.py

Esta clase realiza la asignación de Null a una variable dereclara y crear el símbolo con sus atributos y lo almcena en la tabla de símbolos.

```
class AsignacionNULA(Instruccion):
    def __init__(self, identificador, expresion, fila, columna):
        self.identificador = identificador
        self.expresion = expresion
        self.fila = fila
        self.columna = columna
        self.arreglo = False

    def interpretar(self, tree, table):
        value = "None"
        if isinstance(value, Excepcion):
            return value

        simbolo = Simbolo(self.identificador, TIPO.NULO, self.arreglo, self.fila,
 self.columna, value)
```

```
        result = table.actualizarTabla(simbolo)

        if isinstance(result, Excepcion):
            tree.getExcepciones().append(result)
            tree.updateConsolaError(result.toString())
            return result
        return None

    def getNodo(self):
        nodo = NodoAST("ASIGNACION")
        nodo.agregarHijo(str(self.identificador))
        nodo.agregarHijoNodo(self.expresion.getNodo())
        return nodo
```

Brake.py

Esta clase es la encargada de reconocer el Brake en el código fuente y para sus ejecución según sea el caso la lógica del brake se trabaja en las instrucciones swirch entre otras.

```
class Break(Instruccion):
    def __init__(self, fila, columna):

        self.fila = fila
        self.columna = columna

    def interpretar(self, tree, table):
        return self

    def getNodo(self):
        nodo = NodoAST("BREAK")
        return nodo
```

Case.py

Esta clase ejecuta si la condición de un switch es la misma con el case

```
class Case(Instruccion):
    def __init__(self, expresion, instrucciones, fila, columna):
        self.expresion         = expresion
        self.instrucciones     = instrucciones
        self.fila              = fila
```

```
        self.columna           = columna



    def interpretar(self, tree, table):

        nuevaTabla = TablaSimbolos(table)  # NUEVO ENTORNO

        for instruccion in self.instrucciones:
            result = instruccion.interpretar(tree, nuevaTabla)  # EJECUTA INSTRUC
CION ADENTRO DEL CASE
            if isinstance(result, Excepcion):
                tree.getExcepciones().append(result)
                tree.updateConsolaError(result.toString())

            if isinstance(result, Break): return True
            if isinstance(result, Return): return result



    def getNodo(self):
        nodo = NodoAST("CASE")

        instrucciones = NodoAST("INSTRUCCIONES")
        for instr in self.instrucciones:
            instrucciones.agregarHijoNodo(instr.getNodo())
        nodo.agregarHijoNodo(instrucciones)
        return nodo
```

Continue.py

Esta clase detiene la ejecución de un ciclo y opera la siguiente iteración

```
class Continue(Instruccion):
    def __init__(self, fila, columna):

        self.fila = fila
        self.columna = columna

    def interpretar(self, tree, table):
        return self

    def getNodo(self):
        nodo = NodoAST("CONTINUE")
        return nodo
```