¿Mio?

Mio es un lenguaje de programación diseñado como práctica complementaria para las actividades de la asignatura de teoria de la computación, este lenguaje solo llega a abarcar las partes de un compilador que corresponden a el analizador léxico y el analizador sintactico. Como tal todo esta construido con python sin uso de ninguna libreria externa.

Partes del proyecto

- **pseudoCLI.** Se busca crear una pequeña interfaz de linea comandos para darle más dinamismo a la ejecucion de MIO.
- **Analizador Léxico.** Se busca crear un analizador léxico que permita la identificación de tokens de distinto tipo de un arhivo fuente .mio, como tal se solicita que recopile dicha información en dos archivos, un .lex que contendra como tal un stack para el analizador sintáctico y un .sim que contrendra la tabla de identificadores de los distintos tipos de datos de MIO.
- **Analizador Sintáctico.** Por último, se debe de tener en cuenta que el programa pueda identificar errores al momento de compilar, actualmente esta parte esta bajo testeo, pero por ahora parece que es funcinal.

Requerimientos

Como se mencionó anteriormente, Mio esta construido con python, asi que se necesita tener en la computadora python 3, no se requiere que se utilice un sistema operativo en particular.

Quick start

Para probar el proyecto actual de MIO, basta con que tengas la carpeta principal del proyecto, ya sea descargandola desde el repositorio del proyecto o mediante la carpeta misma.

Lo primero que tenemos que hacer es ejecutar la pseudoCLI de MIO: **MiCLI**, para ello ejecutamos el comando `python runMIO,py` .

Una vez ejecutado el comando, ya estaras en la pseudoCLI del proyecto, como tal el CLI se configura para agarrar el directorio donde se ejecuta, por lo que puedes compilar los archivos de MIO utilizando el comando `analex (nombre del archivo)` aunque recuerda que debe de tener la terminación `.mio`, si no el CLI lo rechazará.

Cuando se compile un archivo, se generarán dos los dos archivos que se mencionaron antes.

Ahora mismo el Analizador sintactico no funciona, por lo que no es posible la evaluación de si el código funciona o no, pero muy pronto estará disponible :D

Si quieres dejar de usar MIiCLI, solo tienes que usar el comando `\q` y con ello volverás a tu shell normal.

También puedes usar el comando `\h` para recibir más información sobre lo que se puede hacer con MiCLI (actualmente no mucho :c).

Documentación.

En caso de que quieras comprobar las pruebas que se han hecho, en la carpeta docs del proyecto hay Jupyter Nootebooks con pruebas individuales para cada una de las partes del proyecto, aunque si quieres ver el código en su máximo esplendor siempre puedes ir a la carpeta src, que es donde esta el código principal hecho en python y comentado.

Próximos aditamentos.

- Se espera que pronto MiCLI pueda hacer operaciones de movimiento entre carpetas-
- Se espera que pronto se puedan ejecutar comandos MIO en la propia CLI.
- Se incluirá la opción para crear archivos dentro de MiCLI.
- Se espera poder hacer una especie de debug, aunque sea solo simulación por que como tal no tenemos analizador semántico, todo por el bien de la práctica académica.

Casos de Uso.

Caso 1. Prueba de testeo general Código fuente:

```
 # Programa de testeo general
PROGRAMA testeoGeneral
IMPRIME Apoquinto
TAL = 3 + 2
REPITE 3 VECES
Apo = 3 + 2
FINREP
SI Apos > 5 ENTONCES
SEC = 3 + 2
JOSE = 3 + 2
FINSI
FINPROG
```

Archivo `.lex`

```
 PROGRAMA
[id]ID0
IMPRIME
[id]ID1
[id]ID2
=
[val]
[op_ar]
[val]
REPITE
[val]
VECES
[id]ID3
=
[val]
[op_ar]
[val]
FINREP
SI
[id]ID4
[op_rel]
[val]
ENTONCES
[id]ID5
=
[val]
[op_ar]
[val]
[id]ID6
=
[val]
[op_ar]
[val]
FINSI
FINPROG
```

Archivo `.sim`

```
 IDS
testeoGeneral,  [id]ID0
Apoquinto,  [id]ID1
TAL,    [id]ID2
Apo,    [id]ID3
Apos,   [id]ID4
SEC,    [id]ID5
JOSE,   [id]ID6

TXT

VAL
3,  3
2,  2
3,  3
3,  3
2,  2
5,  5
3,  3
2,  2
3,  3
2,  2
```

Output:

```
[MIO]~ analex test.mio
Ejecutando Analex...
Ejecutando Anasin...
Iniciando compilación...
Compilación exitosa :D.
```

Caso 2. Testeo general pero sin los FIN

```
# Programa de testeo general pero sin los FIN
PROGRAMA ProgramaTesteoMal
IMPRIME Apoquinto
TAL = 3 + 2
REPITE 3 VECES
Apo = 3 + 2
SI Apos > 5 ENTONCES
SEC = 3 + 2
JOSE = 3 + 2
FINPROG
```

Archivo `.lex`

```
PROGRAMA
[id]ID0
IMPRIME
[id]ID1
[id]ID2
=
[val]
[op_ar]
[val]
REPITE
[val]
VECES
[id]ID3
=
[val]
[op_ar]
[val]
SI
[id]ID4
[op_rel]
[val]
ENTONCES
[id]ID5
=
[val]
[op_ar]
[val]
[id]ID6
=
[val]
[op_ar]
[val]
FINPROG
```

Archivo `.sim`

```
 IDS
ProgramaTesteoMal,  [id]ID0
Apoquinto,  [id]ID1
TAL,    [id]ID2
Apo,    [id]ID3
Apos,   [id]ID4
SEC,    [id]ID5
JOSE,   [id]ID6

TXT

VAL
3,  3
2,  2
3,  3
3,  3
2,  2
5,  5
3,  3
2,  2
3,  3
2,  2
```

Output:

```
 [MIO]~ analex Caso2.mio
Ejecutando Analex...
Ejecutando Anasin...
Iniciando compilación...
Error: SI no finalizado, favor de poner un FINSI. [Linea 28]
Error: REPITE VECES no finalizado, favor de poner un FINREP. [Linea 28]
Compilación fallida.
```

Caso 3. Factorial.mio Código fuente:

```
 # Programa que calcula el factorial de un número
PROGRAMA factorial
# VarX acumula los productos por iteración
VarX = 1
# VarY contiene el iterador del factor
VarY = 0
LEE Num
# Aplica Num! = 1 * 2 * 3 * ... * Num
REPITE Num VECES
VarY = VarY + 1
VarX = VarX * VarY
FINREP
IMPRIME "Factorial de "
IMPRIME Num
IMPRIME " es "
IMPRIME VarX
FINPROG
```

Archivo `.lex`

```
 PROGRAMA
[id]ID0
[id]ID1
=
[val]
[id]ID2
=
[val]
LEE
[id]ID3
REPITE
[id]ID4
VECES
[id]ID5
=
[id]ID6
[op_ar]
[val]
[id]ID7
=
[id]ID8
[op_ar]
[id]ID9
FINREP
IMPRIME
[txt]TXT0
IMPRIME
[id]ID10
IMPRIME
[txt]TXT1
IMPRIME
[id]ID11
FINPROG
```

Archivo `.sim`

```
 IDS
factorial,  [id]ID0
VarX,   [id]ID1
VarY,   [id]ID2
Num,    [id]ID3
Num,    [id]ID4
VarY,   [id]ID5
VarY,   [id]ID6
VarX,   [id]ID7
VarX,   [id]ID8
VarY,   [id]ID9
Num,    [id]ID10
VarX,   [id]ID11

TXT
"Factorial de ",    [txt]TXT0
" es ", [txt]TXT1

VAL
1,  1
0,  0
1,  1
```

Caso 4. Programa para probar el SI Codigo Fuente

```
 # Programa para probar el SI
PROGRAMA siTest
SI Apo < 4 ENTONCES
REPITE 4 VECES
IMPRIME "yei"
FINREP
FINSI
FINPROG
```

Archivo `.lex`

```
 PROGRAMA
[id]ID0
SI
[id]ID1
[op_rel]
[val]
ENTONCES
REPITE
[val]
VECES
IMPRIME
[txt]TXT0
FINREP
FINSI
FINPROG
```

Archivo `.sim`

```
 IDS
siTest, [id]ID0
Apo,    [id]ID1

TXT
"yei",  [txt]TXT0

VAL
4,  4
4,  4
```

Output

```
 [MIO]~ analex Caso4.mio
Ejecutando Analex...
Ejecutando Anasin...
Iniciando compilación...
Compilación exitosa :D.
```