PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

1INF06: ESTRUCTURA DE DATOS Y PROGRAMACIÓN METÓDICA

Docente:

BELLO RUIZ, ALEJANDRO TORIBIO

H – 0582

Alumno:

Campana Yauri, Dan-el Mauricio	20125121
Palacios Coronado, Antony	20141735
Perez Cabrera, Estephany Elizabeth	20142604

Tema:

INFORME 3: Desarrollo de programas en Linux Mint 18.3 Cinnamon usando los lenguajes de programación Python 3 y SWI-Prolog.

Abril del 2018

ÍNDICE:

Introducción…..... 3

Implementación TAD a Prolog.....…..... 3

Implementación en interfaz de Python con Prolog SWI.....…..... 10

Parte trabajada por los integrantes..... 11

Bibliografías.....…..... 12

Anexos.....……..... 13

Implentación de pilas:

La implementación de listas en pilas es una estructura de datos lineal cuya característica principal es que el acceso a elementos se realiza en orden inverso al de su almacenamiento. Esta estructura es llamada LIFO (Last In, First Out – último entrar, primero en salir). En esta ocasión vamos a implementar pilas en Tipos de Datos Abstractos.
Los tipos de datos LIFO son muy usados en los diseños de algoritmos, ya que es muy común, por ejemplo, almacenar y evaluar los datos en la operación de una calculadora, en los editores de texto (cursor), etc.

SWI-PROLOG:

especificación PILAS[ELEM]
usa BOOLEANOS
tipos pila
operaciones

pila-vacia	:	→ pila {constructora}
apilar	: elemento pila	→ pila {constructora}
desapilar	: pila	$\rightarrow_p$ pila
cima	: pila	$\rightarrow_p$ elemento
es-pila- vacia?	: pila	→ bool

variables
e : elemento
p : pila

* Para el caso de pila-vacía, se coloca una lista vacía en prolog: “[ ]”, así es como definimos una constructora.

```
pila_vacia([]).
```

```
?- pila_vacia([]).
true.
?- pila_vacia([1]).
false.
```

* Para definir apilar simplemente agregaremos el elemento en el **head** de la lista de prolog. También tomaremos un caso en que agreguemos un elemento a una pila vacía.

```
apilar(E,[],[E]).
apilar(E,Xs,[E|Xs]).
```

```
?- apilar(3,[],R).
R = [3].
?- apilar(3,[4,5],R).
R = [3, 4, 5].
```

* En desapilar, primero colocaremos nuestro caso error en el cuál nos encontramos con una pila-vacía. Para el caso general, retirares el **head** de la lista de prolog, mostrando solo el resto.

```
desapilar(_,[]):-false.
```

```
desapilar([_|Xs],Xs).

?- desapilar([],R).
false.
?- desapilar([3,4,5],R).
R = [4, 5].
```

* Para realizar cima, devolveremos un error, en este caso **false**, si no encontramos ningún elemento en nuestra pila, lista de prolog. Para el caso general, solo devolveremos el **head** de nuestra pila.

```
cima([],_):-false.
cima([X|_],X).

?- cima([],R).
false.
?- cima([5,6,7],R).
R = 5.
```

* Nos definirá si una pila es vacía o no.

```
es_pila_vacia([]):-true.

?- es_pila_vacia([]).
true.
?- es_pila_vacia([1,2,3]).
false.
```

Extendiendo Pilas:

especificación PILAS+(ELEM)
usa PILAS(ELEM), NATURALES
operaciones
    profundidad : pila → nat
    fondo : pila →p elemento
    inversa : pila → pila
    duplicar : pila → pila
    concatenar : pila pila → pila
    entremezclar : pila pila → pila
operaciones privadas
    apilar-inversa : pila pila → pila
variables
    e, f : elemento
    p, q : pila

* Para definir profundidad necesitaremos nuestro caso base, el cuál será cuando la pila esté vacía. Esto nos dará como resultado cero. En el caso general lo haremos recursivamente, quitando el primer elemento de la pila hasta llegar al caso base y sumando 1 por cada elemento retirado de la pila.

```
profundidad([],X):- X is 0.
profundidad([_|Xs],L):- profundidad(Xs,R), L is R + 1.

?- profundidad([],R).
R = 0.
?- profundidad([5,67,7],R).
```

```
R = 3.
```

* En fondo primero definiremos nuestro caso error, el cual será cuando tengamos nuestra pila vacía. Nuestro caso base será cuando sea el único elemento de la pila. Para el caso general, aplicaremos de modo recursivo la misma función retirando cada uno de los elementos hasta llegar al caso base.

```
fondo([],_):-false.
fondo([X],X):-!.
fondo([_|Xs],E):- fondo(Xs,E).

?- fondo([],R).
false.
?- fondo([4,6,6,71,6],R).
R = 6.
```

*Para definir inversa implementaremos antes apilar_inversa, el cuál recibirá 2 listas e irá colocando los elementos de la primera sobre la segunda. Entonces, para inversa, colocaremos la primera pila y la segunda será una pila vacía.

```
apilar_inversa([],Xs,Xs).
apilar_inversa([X|Xs],Ys,Zs):- apilar(X,Ys,R), apilar_inversa(Xs,R,Zs).

inversa([],[]).
inversa(Xs,R):- apilar_inversa(Xs,[],R),!.

?- inversa([5,6,7,8],R).
R = [8, 7, 6, 5].

*En duplicar realizaremos el caso base, el cual recibe una pila vacía. Para
el caso general tomará elemento por elemento y lo apilará dos veces, luego
irá de forma recursiva hasta llegar al caso base.

duplicar([],[]):-!.
duplicar([X|Xs],Ys):- duplicar(Xs,R), apilar(X,[X|R],Ys).

?- duplicar([1,2,3,4],R).
R = [1, 1, 2, 2, 3, 3, 4, 4].
```

* Para concatenar dos pilas crearemos un caso base, en el cual al tener 1 pila y otra vacía, nos devuelve la misma pila. Para el caso general colocaremos los elementos de la primera sobre la segunda de forma recursiva apilando los elementos de la segunda sobre la primera.

```
concatenar(Xs,[],Xs):-!.
concatenar(Xs,[Y|Ys],Zs):- concatenar(Xs,Ys,R), apilar(Y,R,Zs).

?- concatenar([1,2,3],[6,7,8],R).
R = [6, 7, 8, 1, 2, 3].
```

* En entremezclar iremos intercambiando entre consultas, ya que se irá quitando elementos y colocando en otra pila, dependiendo de la profundidad de la pila.

```
entremezclar(Ys,[],Ys):-!.
entremezclar([],Ys,Ys):-!.
entremezclar([X|Xs],[Y|Ys],[X,Y|L]):- entremezclar(Xs,Ys,L),
profundidad(Xs,A), profundidad(Ys,B), A >= B, !.
```

```
entremezclar([X|Xs],[Y|Ys],[Y,X|L]):- entremezclar(Xs,Ys,L),
profundidad(Xs,A), profundidad(Ys,B), A < B, !.

?- entremezclar([1,3,5,7,9],[2,4,6,8],R).
R = [1, 2, 3, 4, 5, 6, 7, 8, 9].
?- entremezclar([1,3,5,7,9],[2,4,6],R).
R = [1, 2, 3, 4, 5, 6, 7, 9].
```

Secuencia de pilas:

especificación SECUENCIAS[ELEM]
usa PILAS[ELEM], BOOLEANOS
tipos secuencia
operaciones

crear : ← secuencia
insertar : secuencia elemento ← secuencia
eliminar : secuencia ←p secuencia
actual : secuencia ←p secuencia
avanzar : secuencia ←p secuencia
reiniciar : secuencia ←p secuencia
fin? : secuencia ←p bool
es-sec- vacía? : secuencia ←p bool

operaciones privadas
(_, _) : pila pila ← secuencia { constructora }

Variables
e : elemento
s : secuencia
iz, dr : pila

* Para el caso de crear una secuencia daremos como entrada dos pilas, pero en este caso la cima de la pila izquierda estará al lado de la cima del de la derecha.

```
crear([],Xs,Xs):-!.
crear([X|Xs],Ys,R):- apilar(X,Ys,L), crear(Xs,L,R).

?- crear([1,2,3],[4,5,6],R).
R = [3, 2, 1, 4, 5, 6].
```

* Para insertar un elemento delante del punto de interés, agregaremos el elemento a la pila de la izquierda y ejecutaremos crear.

```
insertar(Xs,Ys,E,R):- apilar(E,Xs,L), crear(L,Ys,R).

?- insertar([1,2,3],[4,5,6],0,R).
R = [3, 2, 1, 0, 4, 5, 6].
```

* Para eliminar el elemento en el punto de interés, crearemos la secuencia sin el elemento en el punto de interés, es decir, quitando el head de la pila de la derecha y ejecutando crear con el resto. Además, se tomará en un caso error en el cuál no hay ningún elemento en la pila derecha.

```
eliminar(_,[],false):-!.
eliminar(Xs,[_|Ys],R):-  crear(Xs,Ys,R).
```

```
?- eliminar([1,2,3],[],R).
R = false.
?- eliminar([1,2,3],[5,4,7],R).
R = [3, 2, 1, 4, 7].
```

* Para el caso actual, nos devolverá el head de la pila derecha. Si no hay elemento en la pila derecha nos retornará **false**.

```
actual(_,[],false):-!.
actual(_,[Y|_],Y).
```

```
?- actual([1,2,3],[],R).
R = false.
?- actual([1,2,3],[7,8,9,0],R).
R = 7.
```

* En el caso de avanzar mandaremos el elemento en el punto de interés de la pila derecha a la pila izquierda y luego crearemos la secuencia.

```
avanzar(_,[],false):-!.
avanzar(Ys,[X|Xs],R):- crear([X|Ys],Xs,R).
```

```
?- avanzar([1,2,3],[],R).
R = false.
?- avanzar([1,2,3],[0,7,6,4,9],R).
R = [3, 2, 1, 0, 7, 6, 4, 9].
```

* Para reiniciar mandaremos cada elemento del head de la pila izquierda a la pila derecha implementando una recursiva hasta llegar al caso base el cual la pila izquierda es vacía.

```
reiniciar([],Ys,R):- crear([],Ys,R).
reiniciar([X|Xs],Ys,R):- apilar(X,Ys,L), reiniciar(Xs,L,R).
```

```
?- reiniciar([],[1,2,3],R).
R = [1, 2, 3].
?- reiniciar([5,6,7,8,0],[1,2,3],R).
R = [0, 8, 7, 6, 5, 1, 2, 3].
```

* Para definir fin nos mostrará si la pila derecha está vacía o no.

```
fin(_,[],true).
```

```
?- fin([1,2,3],[],R).
R = true.
?- fin([1,2,3],[1],R).
false.
```

* Finalmente, tendremos una secuencia vacía si ambas pilas, izquierda y derecha, están vacías.

```
es_sec_vacia([],[],true).
```

```
?- es_sec_vacia([],[],R).
R = true.
?- es_sec_vacia([1,2],[],R).
false.
?- es_sec_vacia([1,2],[4,5],R).
```

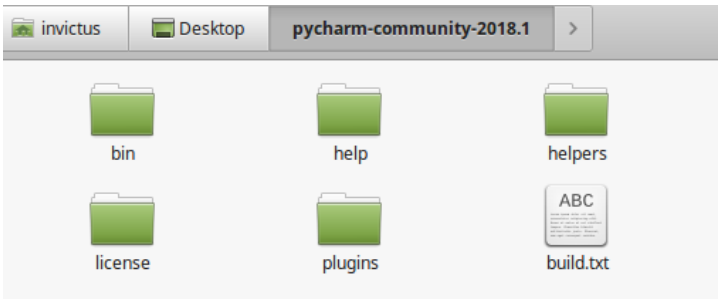
```
false.
?- es_sec_vacia([],[4,5],R).
false.
```

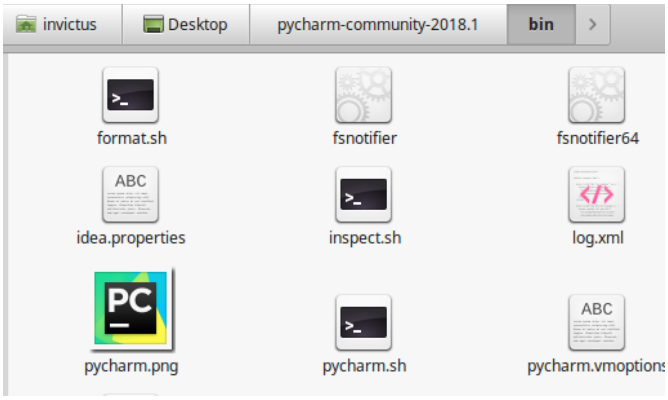
PYTHON:

Para la realización de la interfa grafica usaremos el IDE pycharm y se necesitara el paquete “pyswip” con el cual haremos la conexión con Prolog desde Python.

Instalación de Pycharm:
Desde la página web de Pycharm descargamos de la versión Pycharm Community Edition el archivo “.tar.gz”.

https://www.jetbrains.com/pycharm/download/#section=linux

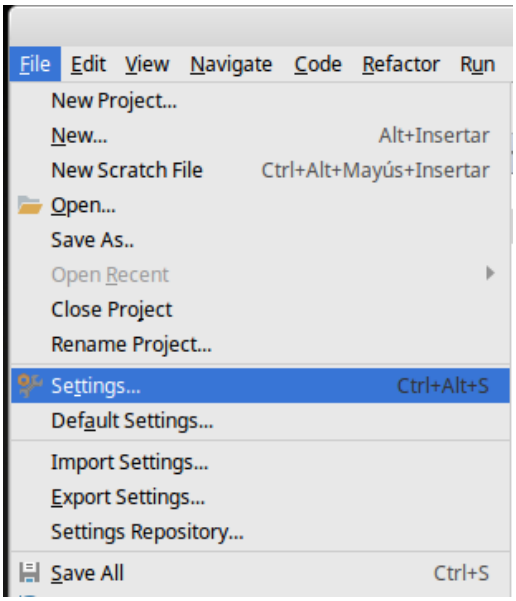




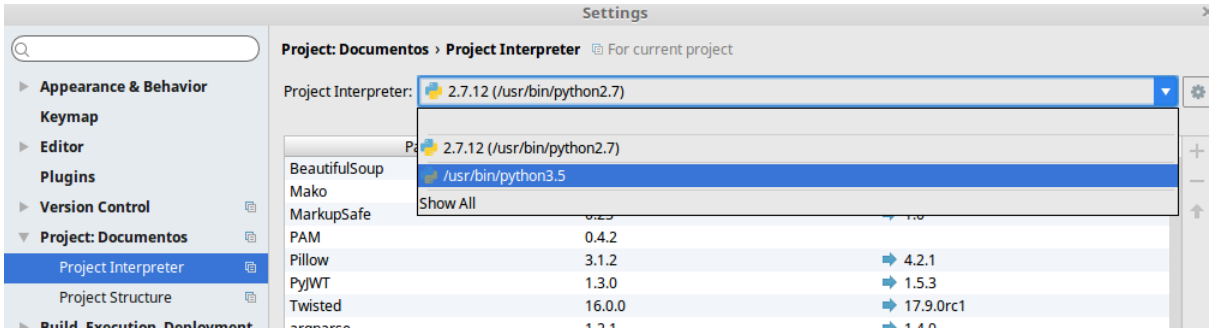
Una vez descargado el archivo, lo descomprimimos y extraemos la carpeta. Dentro de la carpeta nos dirigimos a la carpeta *bin* y ejecutamos el archivo *pycharm.sh*.

Instalación del paquete pyswip:

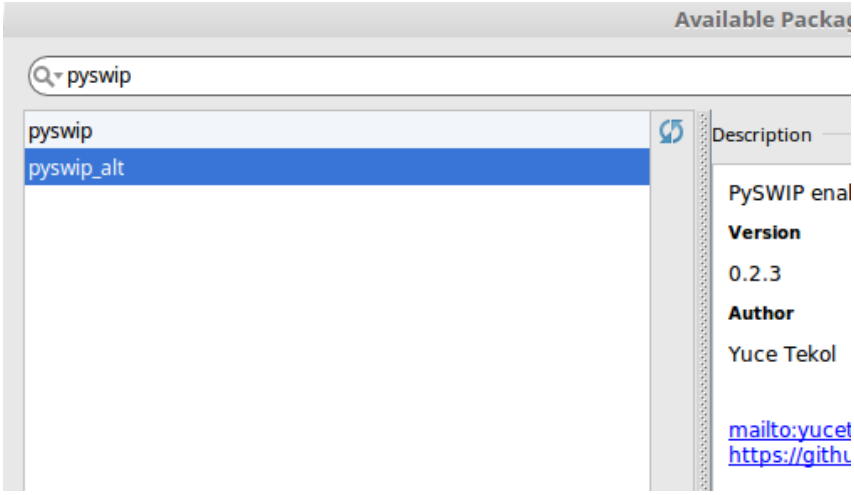
Una vez ejecutado el pycharm, nos dirigimos a la opción *Files -> Settings* en la barra de opciones.



Luego ejegimos la opción *Project -> Project interpreter* donde elegimos el interprete Python 3.5.

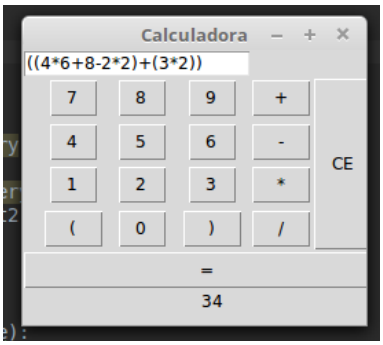


Finalmente accedemos a la opción para la instalación de paquetes, buscamos el paquete *pyswip_alt* y seleccionamos la opción *Install Package.*



IMPLEMENTACIÓN EN PYTHON USANDO PYSWIP:

Ventana de librería Tkinter



El programa desarrollado es una calculadora infija que usa pilas para separar los números y los símbolos.

Para la operación se seguirá el siguiente algoritmo:

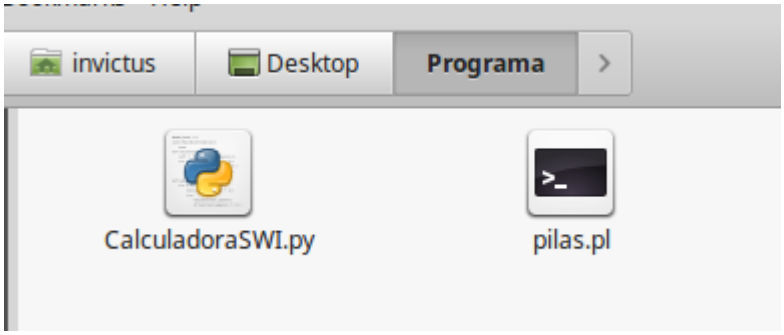
1. Read an input character
2. Actions that will be performed at the end of each input

Opening brackets	(2.1) Push it into stack and then Go to step (1)
Digit	(2.2) Push into stack, Go to step (1)
Operator	(2.3) Do the comparative priority check
	(2.3.1) if the character stack's top contains an operator with equal or higher priority, then pop it into op
	Pop a number from integer stack into op2
	Pop another number from integer stack into op1
	Calculate op1 op op2 and push the result into the integer stack
Closing brackets	(2.4) Pop from the character stack
	(2.4.1) if it is an opening bracket, then discard it and Go to step (1)
	(2.4.2) To op, assign the popped element
	Pop a number from integer stack and assign it op2
	Pop another number from integer stack and assign it to op1
	Calculate op1 op op2 and push the result into the integer stack
	Convert into character and push into stack
	Go to the step (2.4)
New line character	(2.5) Print the result after popping from the STOP

Véase código de python en el Anexo

Para ejecutar el programa primero se debe tener en una sola carpeta el archivo de prolog con extensión ***.pl*** y el programa en python con extensión ***.py*** .

Ejemplo:



Partes trabajadas por los integrantes:

Todos trabajaron por igual.
Al momento de crear las funciones en prolog.
También al momento de escribir el informe, se hizo mediante Documento de Google, se trabajó en simultáneo.
Para los métodos de python se tuvo que cambiar algunas operaciones para poder conectar, enviar y extraer datos.
Para la interfaz gráfica, se usó la misma interfaz que la exposición, pero cambiando las funciones para poder extraer y enviar los datos al tkinter con pyswip.

Prolog:

Campana Yauri, Dan-el Mauricio	**20125121**
Palacios Coronado, Antony	**20141735**
Perez Cabrera, Estephany Elizabeth	**20142604**

Informe:

Campana Yauri, Dan-el Mauricio	**20125121**
Palacios Coronado, Antony	**20141735**
Perez Cabrera, Estephany Elizabeth	**20142604**

Métodos Python:

Campana Yauri, Dan-el Mauricio	**20125121**
Palacios Coronado, Antony	**20141735**
Perez Cabrera, Estephany Elizabeth	**20142604**

Interfaz:

Campana Yauri, Dan-el Mauricio	**20125121**
Palacios Coronado, Antony	**20141735**
Perez Cabrera, Estephany Elizabeth	**20142604**

Bibliografías:

Marti Ortega, Verdejo
Estructura de datos y métodos algorítmicos (EDMA). Consulta 1 de Abril de 2018

Rance D. Necaise
Data Structures and Algorithms Using Python (DSAUP). Consulta 3 de Abril de 2018

Jet Brains
Download PyCharm. Consulta 3 de Abril de 2018
https://www.jetbrains.com/pycharm/download/#section=linux

WIKIBOOKS
Data Structures/Stacks and Queues. Consulta 3 de Abril de 2018
https://en.wikibooks.org/wiki/Data_Structures/Stacks_and_Queues

Anexos:

Prolog:

```
%------PILAS------%

pila_vacia([]).

apilar(E,[],[E]):-!.
apilar(E,Xs,[E|Xs]).

desapilar(_,[ ]):-false.
desapilar([_|Xs],Xs).

cima([ ],_):-false.
cima([X|_],X).

es_pila_vacia([ ]):-true.

Profundidad([ ],X):- X is 0.
profundidad([_|Xs],L):- profundidad(Xs,R), L is R + 1.

fondo([],_):-false.
fondo([X],X):-!.
fondo([_|Xs],E):- fondo(Xs,E).


apilar_inversa([],Xs,Xs).
apilar_inversa([X|Xs],Ys,Zs):- apilar(X,Ys,R), apilar_inversa(Xs,R,Zs).


inversa([],[]):-!.
inversa(Xs,R):- apilar_inversa(Xs,[ ],R),!.

duplicar([],[]):-!.
duplicar([X|Xs],Ys):- duplicar(Xs,R), apilar(X,[X|R],Ys).

concatenar(Xs,[],Xs):-!.
concatenar(Xs,[Y|Ys],Zs):- concatenar(Xs,Ys,R), apilar(Y,R,Zs).

entremezclar(Ys,[],Ys):-!.
entremezclar([],Ys,Ys):-!.
entremezclar([X|Xs],[Y|Ys],[X,Y|L]):- entremezclar(Xs,Ys,L), profundidad(Xs,A),
profundidad(Ys,B), A >= B, !.
entremezclar([X|Xs],[Y|Ys],[Y,X|L]):- entremezclar(Xs,Ys,L), profundidad(Xs,A),
profundidad(Ys,B), A < B, !.

%--------SECUENCIAS----------
crear([],Xs,Xs):-!.
crear([X|Xs],Ys,R):- apilar(X,Ys,L), crear(Xs,L,R).

insertar(Xs,Ys,E,R):- apilar(E,Xs,L), crear(L,Ys,R).

eliminar(_,[],false):-!.
eliminar(Xs,[_|Ys],R):-  crear(Xs,Ys,R).

actual(_,[],false):-!.
actual(_,[Y|_],Y).
```

```
avanzar(_,[],false):-!.
avanzar(Ys,[X|Xs],R):- crear([X|Ys],Xs,R).

reiniciar([],Ys,R):- crear([],Ys,R).
reiniciar([X|Xs],Ys,R):- apilar(X,Ys,L), reiniciar(Xs,L,R).

fin(_,[],true).

es_sec_vacia([],[],true).
```

Python:

```
from tkinter import *
from pyswip import *
############---------------CALCULADORA---------------################
class CalculadoraPL():
    def __init__(self):
        result = list(prolog.query("pila_vacia(R)"))
        result = result[0]['R']
        self.symbol = result[:]
        self.numero = result[:]
        self.resultado = None
    def estaVacia(self):
        cad = str(self.numero)
        result = list(prolog.query("pila_vacia(%s)" % (cad)))
        cad = str(self.symbol)
        result2 = list(prolog.query("pila_vacia(%s)" % (cad)))
        if result == [] or result2 == []:
            return False
        else:
            return True
    def apilar(self, item, nombre):
        if nombre == "N":
            cad = str(self.numero)
            result = list(prolog.query("apilar(%s, %s, R)" % (item, cad)))
            self.numero = result[0]['R']
            return self.numero
        if nombre == "S":
            self.symbol.insert(0,item)
            return self.symbol
    def desapilar(self, nombre):            #En este caso usaremos el
apilar para devolvernos el valor que hemos quitado
        assert  not self.estaVacia(), "NO HAY ELEMENTOS A OPERAR"
        if nombre == "N":
            cad = str(self.numero)
            item = self.numero[0]
            result = (list(prolog.query("desapilar(%s,R)" % (cad))))
            self.numero = result[0]['R']
            #print(self.numero)
            return item
        if nombre == "S":
            return self.symbol.pop(0)
    def cima(self):
        assert not self.estaVacia(), "NO HAY ELEMENTOS A OPERAR"
        cad = str(self.symbol)
```

```
        result = (list(prolog.query("cima(%s,R)" % (cad))))
        return result[0]['R']
    def calcular(self, cadena):
        for i in range(0, len(cadena)):
            c = self.convertir(cadena[i])
            if(c[1]==0):                            #Abre paréntesis
                self.apilar("(","S")
                #self.symbol.push(c[0])
                continue
            elif(c[1]==2):                          #Si es un dígito
                n = cadena[i-1]
                if(48<= ord(n)  <= 57):
                    n = self.desapilar("N")* 10 + c[0]
                    #n = self.numero.pop()*10 + c[0]
                    self.apilar(n,"N")
                    #self.numero.push(n)
                else:
                    self.apilar(c[0], "N")
                    #self.numero.push(c[0])
                continue
            elif(c[1]==3):                          #Si es un operador
                #self.symbol.push(c[0])
                if (self.cima() == "("):
                #if(self.symbol.peek() == "("):
                    self.apilar(c[0],"S")
                    #self.symbol.push(c[0])
                elif (self.prioridad(self.cima(), c[0]) == True):
                #elif(self.prioridad(self.symbol.peek(),c[0]) == True):
                    op2 = self.desapilar("N")
                    op = self.desapilar("S")
                    op1 = self.desapilar("N")
                    self.resultado = self.operar(op1, op, op2)
                    self.apilar(self.resultado,"N")
                    #self.numero.push(self.resultado)
                    self.apilar(c[0],"S")
                    #self.symbol.push(c[0])
                else:
                    self.apilar(c[0],"S")
                    #self.symbol.push(c[0])
            elif(c[1]==1):                              #Cierra paréntesis
                while(1):
                    op = self.desapilar("S")
                    #op = self.symbol.pop()
                    if(op == "("):
                        break
                    else:
                        op2 = self.desapilar("N")
                        op1 = self.desapilar("N")
                        self.resultado = self.operar(op1, op, op2)
                        self.apilar(self.resultado,"N")
                        #self.numero.push(self.resultado)
        #print(self.numero.pop())
        return self.desapilar("N")
        #return self.numero.pop()
    def prioridad(self, p1, p2):                    #Si la prioridad es
mayor o igual
        if((p1=="*" or p1=="/") and (p2=="-" or p2=="+")) or (p1==p2):
            return True
```

```
        elif((p1=="*" and p2=="/") or (p1=="/" and p2=="*")):
            return True
        return False
    def operar(self,op1,op,op2):
        if (op=="*"):
            return op1*op2
        if (op == "/"):
            return op1 // op2
        if (op == "+"):
            return op1 + op2
        if (op == "-"):
            return op1 - op2
    def convertir(self, item):
        if(item == "("):
            return item, 0                  #Abre paréntesis
        elif(item == ")"):
            return item, 1                  #Cierra paréntesis
        elif(48<= ord(item)  <= 57):
            return ord(item)-48, 2          #Si es un dígito
        else:
            return item, 3                  #Si es un operador

##########----------------INTERFACE-------------------###############
def set_text(num):
    if(num=="CE"):
        e.delete(0, END)
    elif(num=="EQUAL"):
        datos = e.get()
        C = CalculadoraPL()
        C.symbol = []
        C.numero = []
        resultado = C.calcular("(" + datos + ")")
        mLabel.config(text = resultado)
    else:
        e.insert(END, num)
    return
window = Tk()
window.geometry("250x200")
window.title("Calculadora")
e = Entry(window)
e.grid(row = 1, columnspan=4)
datos = StringVar()
b_0 = Button(window, text = "0", anchor="w", width=1, command =
lambda:set_text("0"))
b_0.grid(row=7, column=2)
b_1 = Button(window, text = "1", anchor="w", width=1, command =
lambda:set_text("1"))
b_1.grid(row=6, column=1)
b_2 = Button(window, text = "2", anchor="w", width=1, command =
lambda:set_text("2"))
b_2.grid(row=6, column=2)
b_3 = Button(window, text = "3", anchor="w", width=1, command =
lambda:set_text("3"))
b_3.grid(row=6, column=3)
b_4 = Button(window, text = "4", anchor="w", width=1, command =
lambda:set_text("4"))
b_4.grid(row=5, column=1)
b_5 = Button(window, text = "5", anchor="w", width=1, command =
lambda:set_text("5"))
```

```
b_5.grid(row=5, column=2)
b_6 = Button(window, text = "6", anchor="w", width=1, command =
lambda:set_text("6"))
b_6.grid(row=5, column=3)
b_7 = Button(window, text = "7", anchor="w", width=1, command =
lambda:set_text("7"))
b_7.grid(row=4, column=1)
b_8 = Button(window, text = "8", anchor="w", width=1, command =
lambda:set_text("8"))
b_8.grid(row=4, column=2)
b_9 = Button(window, text = "9", anchor="w", width=1, command =
lambda:set_text("9"))
b_9.grid(row=4, column=3)
b_clear = Button(window, text = "CE", anchor=CENTER, width=2,height=8,
command = lambda:set_text("CE"))
b_clear.grid(row=4, column=5,rowspan=4)
b_mas = Button(window, text = "+", anchor=CENTER, width=2, command =
lambda:set_text("+"))
b_mas.grid(row=4, column=4)
b_menos = Button(window, text = "-", anchor=CENTER, width=2, command =
lambda:set_text("-"))
b_menos.grid(row=5, column=4)
b_multi = Button(window, text = "*", anchor=CENTER, width=2, command =
lambda:set_text("*"))
b_multi.grid(row=6, column=4)
b_divi = Button(window, text = "/", anchor=CENTER, width=2, command =
lambda:set_text("/"))
b_divi.grid(row=7, column=4)
b_oBra = Button(window, text = "(", anchor=CENTER, width=2, command =
lambda:set_text("("))
b_oBra.grid(row=7, column=1)
b_cBra = Button(window, text = ")", anchor=CENTER, width=2, command =
lambda:set_text(")"))
b_cBra.grid(row=7, column=3)
b_igual = Button(window, text = "=", anchor=CENTER, width=30, command =
lambda:set_text("EQUAL"))
b_igual.grid(row=8, columnspan=15)
mLabel = Label(window, anchor = CENTER)
mLabel.grid(row = 9, column = 3)


if __name__ == "__main__":
    prolog= Prolog()
    prolog.consult('pilas.pl')
    #pila = CalculadoraPL()
    window.mainloop()
```