


UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
LENGUAJES FORMALES Y DE PROGRAMACION

# PROYECTO 1

## MANUAL TECNICO

Walter Gustavo Cotí Xalin
201700522

**Guatemala, Octubre del 2020**

# Contenido

**Inicio**

El método main tiene un ciclo while este es para recibir comandos en consola.

```
def main():
    while not salir:
        entrada = input(">>")
        cadena_Entrada(entrada)
```

El método cadena_Entrada() es una combinación de autómatas donde se analiza carácter por carácter.

Para saber que comando es al que se refiere se concatena el primer grupo de caracteres de cada cadena ingresada y dependiendo del resultado se cambia el valor del estado.

```
def cadena_Entrada(cadenita):
    condicion_dos = False
    estado = 0
    texto = ""
    for pos in range(len(cadenita)):
      try:
        if estado == 0:
            if cadenita[pos].isspace():
                if texto.upper() == "CREATE":
                    estado = 1                      #terminado falta validar duplicados
                    var_global.lst_tokens.append(texto)
                    texto = ""
                elif texto.upper() == "LOAD":       #terminado ajustes, validaciones
                    estado = 3
                    var_global.lst_tokens.append(texto)
                    texto = ""
                elif texto.upper() =="USE":         #terminado
                    estado = 7
                    var_global.lst_tokens.append(texto)
                    texto = ""
                elif texto.upper() == "SELECT":
                    estado = 20
                    var_global.lst_tokens.append(texto)
                    texto = ""                      #falta aun
                elif texto.upper() == "LIST":
                    estado = 9
                    var_global.lst_tokens.append(texto)
                    texto = ""
                elif texto.upper() == "PRINT":      #terminado
                    estado = 10
                    var_global.lst_tokens.append(texto)
                    texto = ""
                elif texto.upper() == "MAX":    #terminado
                    estado = 12
                    var_global.lst_tokens.append(texto)
                    texto = ""
                elif texto.upper() == "MIN":    #terminado
```

Si llegara a ingresarse un  comando que no es reconocido puede que se ignore o imprima un mensaje de error en consola.

```
            elif texto.upper() == "SUM":    #terminado
                estado = 15
                var_global.lst_tokens.append(texto)
                texto = ""
            elif texto.upper() == "COUNT":
                estado = 16
                var_global.lst_tokens.append(texto)
                texto = ""
            elif texto.upper() == "REPORT":
                estado = 17
                var_global.lst_tokens.append(texto)
                texto = ""
            elif texto.upper() == "SCRIPT": #terminado
                estado = 14
                texto = ""
            else:
                print("ERROR, " + texto + " no es un comando valido")
        else:
            texto = texto + cadenita[pos]
```

## CREATE SET

Para crear un set de memoria se analiza la cadena ingresada y automáticamente la palabra que se encuentre después de “SET” se toma como identificador del set de memoria.

```
#---------------------------------------COMANDO CREATE SET-----------------------------------------
    elif estado == 1:
        if  cadenita[pos].isspace():
            if texto.upper() == "SET":
                estado = 2
                var_global.lst_tokens.append(texto)
                texto = ""
            else:
                texto = ""
                continue
        else:
            texto = texto + cadenita[pos]

    elif estado == 2:
        num = len(cadenita)
        if pos == (len(cadenita)-1):
            texto = texto + cadenita[pos]
            nuevoSet = obj_set.Objeto_Set(texto)        #corregir guardadado
            var_global.arregloSEts.append(nuevoSet)     #corregir guardado
            print("Set -> "+ texto +" <-creado con exito")
            print("-----------------------------------------------------")
        else:
            texto = texto + cadenita[pos]
```

## LOAD INTO

Después de la palabra “INTO” se toma el set de memoria al cual se le agregara la información contenida en los archivos .aon, estos están justo después de la palabra clave “FILES” pudiendo ser mas de uno separados por una coma (,).

```
#-------------------------------------COMANDO LOAD INTO ---------------------------------
        elif estado == 3:
            if  cadenita[pos].isspace():
                if texto.upper() == "INTO":
                    estado = 4
                    var_global.lst_tokens.append(texto)
                    texto = ""
                else:
                    texto = ""
                    continue
            else:
                texto = texto + cadenita[pos]
        elif estado == 4:
            if  cadenita[pos].isspace():
                set_work = texto
                estado = 5
                texto = ""
            else:
                texto = texto + cadenita[pos]
        elif estado == 5:
            if  cadenita[pos].isspace():
                if texto.upper() == "FILES":
                    estado = 6
                    texto = ""
                else:
                    texto = ""
                    continue
            else:
                texto = texto + cadenita[pos]
        elif estado == 6:
            if cadenita[pos].isspace():
                continue
            else:
                if cadenita[pos] == ",":
                    var_global.lst_tokens.append(",")
                    verqeupaso = readFile.openFile(texto)
                    leeraon.leercontenido(readFile.openFile(texto),set_work)
                    print("Archivo -> " + texto + " <-  cargado a memoria")
```

```
if cadenita[pos] == ",":
    var_global.lst_tokens.append(",")
    verqeupaso = readFile.openFile(texto)
    leeraon.leercontenido(readFile.openFile(texto),set_work)
    print("Archivo -> " + texto + " <-  cargado a memoria")
    texto = ""
elif pos == (len(cadenita)-1):
    texto = texto + cadenita[pos]
    verpaso = readFile.openFile(texto)
    leeraon.leercontenido(readFile.openFile(texto),set_work)
    print("Archivo -> " + texto + " <-  cargado a memoria")
    print("-------------------------------------------------")
    texto = ""
else:
    texto = texto + cadenita[pos]
```

## USE SET

Nuevamente el set de memoria con el que se trabajará será automáticamente el identificador que se encuentre después de la palabra clave “SET”.

Se valida que el identificador no se repita en el arreglo de set de memoria.

```
#---------------------------------------COMANDO USE SET-----------------------------------------
        elif estado == 7:
            if  cadenita[pos].isspace():
                if texto.upper() == "SET":
                    estado = 8
                    var_global.lst_tokens.append(texto)
                    texto = ""
                else:
                    texto = ""
                    continue
            else:
                texto = texto + cadenita[pos]
        elif estado == 8:
            datoExiste = False
            if  pos == (len(cadenita)-1):
                texto = texto + cadenita[pos]
                for i in range(len(var_global.arregloSEts)): #crear metodos que retornen boleano
                    com_name = var_global.arregloSEts[i]
                    if com_name.getnombre() == texto:
                        datoExiste = True
                        break
                    else:
                        continue
                if datoExiste:
                    var_global.trabajar_set = texto
                    print("El set de memoria a utilizar ahora es -> " + var_global.trabajar_set)
                    print("-------------------------------------------------")
                else:
                    print("Set " + var_global.trabajar_set + " no existe")
            else:
                texto = texto + cadenita[pos]
```

## SELECT

Para este comando se considera la posibilidad de que se seleccionen uno o todos los atributos, primero se analiza la cadena hasta encontrar una coma tomando lo anterior como un atributo, si se llega a un espacio este algoritmo verifica si el carácter anterior es una coma o no.

También otro carácter valido dentro es el símbolo “ * ” que indica que se toma la totalidad de atributos del set elegido.

```
#---------------------------------------COMANDO SELECT-----------------------------------
        elif estado == 20:
            if  pos == (len(cadenita)-1):
                if cadenita[pos] == "*":
                    select_all_atrib()
            elif cadenita[pos].isspace():
                if cadenita[pos-1] == ",":
                    continue
                elif texto == "*":
                    select_all_atrib()
                    texto = ""
                    estado = 21
                else:
                    var_global.lst_atributos.append(texto)
                    texto = ""
                    estado = 21
            else:
                if cadenita[pos] == ",":
                    var_global.lst_atributos.append(texto)
                    var_global.lst_tokens.append(",")
                    texto = ""
                else:
                    texto = texto + cadenita[pos]
        elif estado == 21:
            if cadenita[pos].isspace():
                if texto.upper() == "WHERE":
                    estado = 22
                    var_global.lst_tokens.append(texto.upper())
                    texto = ""
            else:
                texto = texto + cadenita[pos]
```

Nuevamente se llega a una palabra reservada, esta vez es “WHERE” que indica que todo lo que prosigue es o son condiciones que deben cumplirse para tomar en cuenta la selección.

Primero se verifica si lo que se encuentra es un símbolo “ * “ o es una palabra, de ser un asterisco se toman todos los registros del set de memoria seleccionado previamente.

De ser una palabra se guarda como el nombre del atributo que debe de cumplir la condición siguiente.

Siguiendo con el recorrido se agrupan los caracteres hasta encontrar un espacio, este carácter o pareja de caracteres pueden ser: <,>,<=,>=,=,!=, sabiendo que condición es se almacena en una variable y se prosigue con el análisis de la cadena.

```
        elif estado == 22:
            if  pos == (len(cadenita)-1):
                if cadenita[pos] == "*":
                    for set_memoria in var_global.arregloSEts:
                        if set_memoria.getnombre() == var_global.trabajar_set:
                            for elemento in set_memoria.getlist():
                                var_global.lista_res_final.append(elemento)
                                                    #si es * son todos los elementos del set
                    seleccionar.imprimir_resultado()
                elif cadenita[pos].isspace():
                    var_global.key_atributo.append(texto)
                    estado = 23
                    texto =""
                else:
                    texto = texto + cadenita[pos]
            elif estado == 23:
                if cadenita[pos].isspace():
                    estado = 24
                    var_global.operator_cond.append(texto)
                    var_global.lst_tokens.append(texto.upper())
                    texto = ""
                else:
                    texto = texto + cadenita[pos]
```

Llegado al estado “24” se analiza si la palabra se encuentra al final de la cadena o se tienen mas caracteres si se da el cazo de no ser el final se cambia de estado y se continua el análisis.

```
elif estado == 24:
    if  pos == (len(cadenita)-1):
        texto = texto + cadenita[pos]
        var_global.value_atributo.append(texto)
        if condicion_dos:
            seleccionar.buscarCondicion(var_global.key_atributo[0],var_global.operator_cond[0],var_global.value_atributo[0],var_global.lst_primeraCond)
            seleccionar.buscarCondicion(var_global.key_atributo[1],var_global.operator_cond[1],var_global.value_atributo[1],var_global.lst_segundaCond)
            seleccionar.buscarOperador();
        else:
            seleccionar.buscarCondicion(var_global.key_atributo[0],var_global.operator_cond[0],var_global.value_atributo[0],var_global.lst_primeraCond)
            seleccionar.buscarOperador();
        texto =""
    elif cadenita[pos].isspace():
        var_global.value_atributo.append(texto)
        condicion_dos = True
        texto =""
        estado = 27
    elif cadenita[pos]== "\"":
        var_global.lst_tokens.append("\"")
        estado = 25
    else:
        texto = texto + cadenita[pos]
```

Para el valor de la condición a buscar se tienen dos reglas:

Esta entre comillas, o es un valor numérico o booleano.

De ser un valor entre comillas se cambia de estado y se concatena todo lo que se encuentre entre las comillas.

De ser un valor numérico o booleano se concatena todo hasta encontrar un espacio. Seguido de esto se guarda el operador que une las dos condiciones y

posteriormente se repite el procedimiento para obtener el nombre del atributo y el valor de la condición.

```
elif estado == 25:
    if  pos == (len(cadenita)-1):
        X = cadenita[pos]
        if  cadenita[pos]=="\"":
            var_global.value_atributo.append(texto)
            var_global.lst_tokens.append("\"")
            texto =""
            if condicion_dos:
                seleccionar.buscarCondicion(var_global.key_atributo[0],var_global.operator_cond[0],var_global.value_atributo[0],var_global.lst_primeraCon
                seleccionar.buscarCondicion(var_global.key_atributo[1],var_global.operator_cond[1],var_global.value_atributo[1],var_global.lst_segundaCon
                seleccionar.buscarOperador();
            else:
                seleccionar.buscarCondicion(var_global.key_atributo[0],var_global.operator_cond[0],var_global.value_atributo[0],var_global.lst_primeraCon
                seleccionar.buscarOperador();
    elif cadenita[pos]=="\"":
            var_global.value_atributo.append(texto)
            var_global.lst_tokens.append("\"")
            texto =""
            condicion_dos = True
            estado = 26
    else:
        texto = texto + cadenita[pos]
elif estado == 26:
    if cadenita[pos].isspace():
        estado = 27
elif estado == 27:
    if cadenita[pos].isspace():
        var_global.operator = texto
        var_global.lst_tokens.append(texto)
        texto=""
        estado = 22
    else:
        texto = texto + cadenita[pos]
```

## LIST ATTRIBUTES

Recorre el arreglo de set's de memoria, para buscar el set que se eligió con el comando "USE SET" , teniendo el set correcto se procede a recorrer el arreglo de elementos guardados en el e imprimir los atributos que se tengan.

```
#-----------------------------------------COMANDO LIST ATTRIBUTES ---------------------------------------
        elif estado == 9:
            if  pos == (len(cadenita)-1):
                texto = texto + cadenita[pos]
                if texto.upper() == "ATTRIBUTES":
                    var_global.lst_tokens.append(texto)
                    for i in range(len(var_global.arregloSEts)):
                        objeto_set = var_global.arregloSEts[i]
                        if objeto_set.getnombre() == var_global.trabajar_set:
                            lista_elementos = objeto_set.getlist()
                            elemento_ver = lista_elementos[0]
                            for atrib in elemento_ver.keys():
                                print("- " + atrib)
                            print("-----------------------------------------------------")
                        else:
                            continue
                else:
                    texto = ""
                    continue
            else:
                texto = texto + cadenita[pos]
```

**PRINT IN**

Únicamente al ingresar cualquiera de los colores se ejecuta la acción que imprime el texto posterior en el color seleccionado.

```
#----------------------------------------COMANDO PRINT IN COLOR--------------------
        elif estado == 10:
            if  cadenita[pos].isspace():
                if texto.upper() == "IN":
                    estado = 11
                    var_global.lst_tokens.append(texto)
                    texto = ""
                else:
                    texto = ""
                    continue
            else:
                texto = texto + cadenita[pos]

        elif estado == 11:
            if  pos == (len(cadenita)-1):
                texto = texto + cadenita[pos]
                if texto.upper() ==  "BLUE":
                    var_global.lst_tokens.append(texto.upper())
                    print(Fore.BLUE)
                elif texto.upper() ==  "RED":
                    var_global.lst_tokens.append(texto.upper())
                    print(Fore.RED)
                elif texto.upper() ==  "GREEN":
                    var_global.lst_tokens.append(texto.upper())
                    print(Fore.GREEN)
                elif texto.upper() ==  "YELLOW":
                    var_global.lst_tokens.append(texto.upper())
                    print(Fore.YELLOW)
                elif texto.upper() ==  "ORANGE":
                    var_global.lst_tokens.append(texto.upper())
                    print(Fore.RED)
                elif texto.upper() ==  "PINK":
                    var_global.lst_tokens.append(texto.upper())
                    print(Fore.MAGENTA)
                else:
                    print("El color que se a seleccionado no existe")
            else:
                texto = texto + cadenita[pos]
```

**MAX**

Se considera el atributo a buscar un máximo a la palabra que se encuentre posterior a este comando.

Comienza recorriendo el arreglo de set de memoria, ubicando el set correcto y posteriormente obtener la lista de elementos almacenados en él.

Se agregan los valores a una lista para poder trabajar con la librería “max” , dependiendo del tipo de lista que se obtenga podría ser de tipo float o string.

```
#----------------------------------------COMANDO MAX ATRIBUTO----------------------------------------

        elif estado == 12:
            if  pos == (len(cadenita)-1):
                texto = texto + cadenita[pos]
                for i in range(len(var_global.arregloSEts)):
                        objeto_set = var_global.arregloSEts[i]
                        if objeto_set.getnombre() == var_global.trabajar_set:
                            lista_elementos = objeto_set.getlist()
                            for cel in range(len(lista_elementos)):
                                datos = lista_elementos[cel]
                                prueba_result = datos.get(texto)
                                try:
                                    if float(prueba_result).as_integer_ratio():
                                        var_global.lst_max.append(prueba_result)

                                except:
                                    var_global.lst_max.append(prueba_result)

                            try:
                                print(texto + " maximo es: " + str(max(var_global.lst_max,key=float)))
                                var_global.lst_max.clear()
                            except:
                                print(texto + " maximo es: " + str(max(var_global.lst_max,key=ascii)))
                                var_global.lst_max.clear()
                print("-------------------------------------------------")

            else:
                texto = texto + cadenita[pos]
```

**MIN**

Se considera el atributo a buscar un mínimo a la palabra que se encuentre posterior a este comando.

Comienza recorriendo el arreglo de set de memoria, ubicando el set correcto y posteriormente obtener la lista de elementos almacenados en él.

Se agregan los valores a una lista para poder trabajar con la librería “min” , dependiendo del tipo de lista que se obtenga podría ser de tipo float o string.

Se imprime el resultado.

```
#---------------------------------------COMANDO MIN ATRIBUTO---------------------------------------
        elif estado == 13:
            if  pos == (len(cadenita)-1):
                texto = texto + cadenita[pos]
                for i in range(len(var_global.arregloSEts)):
                        objeto_set = var_global.arregloSEts[i]
                        if objeto_set.getnombre() == var_global.trabajar_set:
                            lista_elementos = objeto_set.getlist()
                            for cel in range(len(lista_elementos)):
                                datos = lista_elementos[cel]
                                prueba_result = datos.get(texto)
                                try:
                                    if float(prueba_result).as_integer_ratio():
                                        var_global.lst_min.append(prueba_result)
                                except:
                                    var_global.lst_min.append(prueba_result)
                            try:
                                print(texto + " minimo es: " + str(min(var_global.lst_min,key=float)))
                                var_global.lst_min.clear()
                            except:
                                print(texto + " minimo es: " + str(min(var_global.lst_min,key=ascii)))
                                var_global.lst_min.clear()
                print("-------------------------------------------------")
            else:
                texto = texto + cadenita[pos]
```

**SUM**

Se considera el atributo para sumar a la palabra que se encuentre posterior a este comando.

Comienza recorriendo el arreglo de set de memoria, ubicando el set correcto y posteriormente obtener la lista de elementos almacenados en él.

Se agregan los valores a una lista esto solamente si es de tipo numérico para poder trabajar con la librería “sum”.

```
#----------------------------------------COMANDO SUM-----------------------------
        elif estado == 15:
            if cadenita[pos].isspace():
                continue
            else:
                if cadenita[pos] == ",":
                    var_global.lst_tokens.append(",")
                    var_global.lst_atributos.append(texto)
                    texto = ""
                elif cadenita[pos] == "*":
                    for i in range(len(var_global.arregloSEts)):
                        objeto_set = var_global.arregloSEts[i]
                        if objeto_set.getnombre() == var_global.trabajar_set:
                            lista_elementos = objeto_set.getlist()
                            elemento_ver = lista_elementos[0]
                            for atrib in elemento_ver.keys():
                                var_global.lst_atributos.append(atrib)
                        else:
                            continue
                    suma()

                elif pos == (len(cadenita)-1):
                    texto = texto + cadenita[pos]
                    var_global.lst_atributos.append(texto)
                    texto = ""
                    suma()
                else:
                    texto = texto + cadenita[pos]
```

```
def suma():
    for atributo_bus in var_global.lst_atributos:
        value_atrib = ""
        for obj in var_global.arregloSEts:
            if obj.getnombre() == var_global.trabajar_set:
                lista_datos = obj.getlist()
                for datos_element in lista_datos:
                    value_atrib = datos_element.get(atributo_bus)
                    try:

                            var_global.lst_sum.append(value_atrib)
                    except:
                        continue
        try:
            if var_global.lst_sum:
                suma_Fin = list(map(float, var_global.lst_sum))
                print( "Suma total de " + atributo_bus + " = " + str(sum(suma_Fin)))

                var_global.lst_sum.clear()

            else:
                print("Atributo "+ atributo_bus+ " de tipo string, no puede sumarse")
        except:
            print("Atributo de tipo string")
    print("------------------------------------------------")
    var_global.lst_atributos.clear()
```

## COUNT

Este comando puede contar mas de un atributo, al ser de esta forma almaceno cada nombre de atributo en un listado igualmente se puede simplemente colocar un “ * “ que denota la totalidad de atributos.

Teniendo la lista de atributos utilizando la propiedad count() de las listas se imprime cada atributo con su respectivo conteo.

```
#----------------------------------------COMANDO COUNT-------------------------------
        elif estado == 16:
            if cadenita[pos].isspace():
                continue
            else:
                if cadenita[pos] == ",":
                    var_global.lst_tokens.append(",")
                    var_global.lst_atributos.append(texto)
                    texto = ""
                elif cadenita[pos] == "*":
                    for i in range(len(var_global.arregloSEts)):
                        objeto_set = var_global.arregloSEts[i]
                        if objeto_set.getnombre() == var_global.trabajar_set:
                            lista_elementos = objeto_set.getlist()
                            elemento_ver = lista_elementos[0]
                            for atrib in elemento_ver.keys():
                                var_global.lst_atributos.append(atrib)
                        else:
                            continue
                    contar()

                elif pos == (len(cadenita)-1):
                    texto = texto + cadenita[pos]
                    var_global.lst_atributos.append(texto)
                    texto = ""
                    contar()
                else:
                    texto = texto + cadenita[pos]
```

```
def contar():
    for atributo_bus in var_global.lst_atributos:
        value_atrib = ""
        for obj in var_global.arregloSEts:
            if obj.getnombre() == var_global.trabajar_set:
                lista_datos = obj.getlist()
                for datos_element in lista_datos:
                    value_atrib = datos_element.get(atributo_bus)
                    try:
                        var_global.lst_count.append(value_atrib)
                    except:
                        continue
        try:
            if var_global.lst_count:
                print("Datos registrados "+ atributo_bus + " = " + str(len(var_global.lst_count)))
                var_global.lst_count.clear()
            else:
                pass
        except:
            print("Atributo de tipo string")
    print("-------------------------------------------------")
    var_global.lst_atributos.clear()
```

## SCRIPT

Para podre cargar archivos de tipo siql que contengan los comandos a realizarse primero se recorre el comando y se separan los archivos por dirección o nombre esto por cada coma.

```
#-----------------------------------------COMANDO SCRIPT ----------------------------------------
        elif estado == 14:
            if cadenita[pos].isspace():
                continue
            else:
                if cadenita[pos] == ",":
                    script_go(texto)
                    texto = ""
                    #set_work = ""
                elif pos == (len(cadenita)-1):
                    texto = texto + cadenita[pos]
                    script_go(texto)
                    texto = ""
                else:
                    texto = texto + cadenita[pos]
```

La función “script_go()” lee y retorna el contenido del script, enviando fila por fila cada instrucción al método “cadena_Entrada()”.

```
def script_go(direccion):
    script_comando = ""
    scrit_contenido = readFile.openFile(direccion)
    try:
        for po in range(len(scrit_contenido)):
            if scrit_contenido[po] == "\n":
                continue
            else:
                if scrit_contenido[po] == ";":
                    cadena_Entrada(script_comando)
                    var_global.lst_tokens.append(";")
                    script_comando = ""
                else:
                    script_comando = script_comando + scrit_contenido[po]
    except:
        print("Error al intentar leer el archivo")
```

## REPORT TOKENS / TO

Se verifica que tipo de reporte se desea hacer, de ser un reporte de tokens este generará un archivo xml con el nombre del set de memoria utilizado.

```
#---------------------------------------COMANDO REPORT TOKENS/TO---------------------------------
        elif estado == 17:
            if  pos == (len(cadenita)-1):
                texto = texto + cadenita[pos]
                if texto.upper() == "TOKENS":
                    var_global.lst_tokens.append(texto)
                    report.reportTokens(var_global.trabajar_set)
                    texto = ""
            elif cadenita[pos].isspace():
                if texto.upper() == "TO":
                    var_global.lst_tokens.append(texto)
                    estado = 18
                    texto = ""
            else:
                texto = texto + cadenita[pos]

        elif estado == 18:
            if  cadenita[pos].isspace():
                var_global.trabajar_set = texto
                estado = 19
                texto = ""
            else:
                texto = texto + cadenita[pos]
        elif estado == 19:
            if  pos == (len(cadenita)-1):
                texto = texto + cadenita[pos]
                cadena_Entrada(texto)
                texto = ""
            else:
                texto = texto + cadenita[pos]
```

# Diagrama de flujo SimpleQL CLI

WALTER COTÍ