

# Práctica 2 T. Computación

## Aspectos a tener en cuenta:

- En el jflap manejamos los espacios con el caracter "ç" solo por notacion.
- en el jflap mandamos los estados de (;) al inicio para volver a verificar, pero en el programa vamos a generar una lista de tokens eliminando los (;).
- en el programa tambien vamos a eliminar (,) con el objetivo de prevenir falsos negativos en lineas como: `int a = 100, b = 34, ter = 23;`  
No vamos a considerar casos en los que nos den asignaciones o comparaciones como `id=12`, siempre consideramos que tendran espacios, por ejemplo: `id = 12;`
- Si en algun estado no he puesto flechas pero le llega un valor no esperado es obvio que se indetermina, por ejemplo. En el estado q10 no he puesto el caso hipotetico de que a el numero +12.875 le llegaria una letra "a" sin embargo, obviamente se indetermina.
- no se consideran casos como: `10;.000;` que a pesar de ser no ser erroneos, no los tomaremos en cuenta.

## Descripcion de los estados:

### Q0, estado vacio

- se considera valido porque el compilador no detecta error alguno.

- una linea totalmente vacia NO marca error alguno, por ende no hay caso de levantar la bandera de que hay un error.
- si se reciben multiples combinaciones de espacios (representados con el caracter ç) o de ;(punto y comas).

## Q1, se recibe un signo + o -

- Puede tener muchos espacios
- si recibe un ; se va a la basura
- si recibe un cero **se considera decimal**, pero tambien es un paso a ser octal o hexadecimal.

## Q3, 0 (decimal si esta solo) o paso a octal o hexadecimal

- Estado valido por ser cero decimal
- si llega otro numero entre 1 a 7 pasa a octal
- si llega el caracter "x" pasa a previo hexa
- si se recibe (;) vuelve a iniciar otra verificacion
- Si recibe un (.) Pasa a q10 que sería 0.

## Q4, blanco con espacios (ya no puede ser octal ni hexa) pero sigue siendo decimal

- Valido ya que en q3 es valido, solo que tiene blancos.
- inserto este caso ya que si se detecta una coma
- ahora si detecta una coma o un punto y coma ya reinicia
- para evitar casos como: - 0 ; +0, mejor solo tendria (+ 0 ) y si recibe un (;) se inicia otra verificacion.

## Q5, numero valido con coma(,)

- Es valido, ya que lo es en su anterior estado
- espera por otro identificador
- si recibe ; o un numero se va a la basura

## Q6, octal valido

- se mantiene si recibe numeros del 1 al 7, despues del primer 0n (donde n es distinto de 0) ya pueden ser del 0 al 7.
- si recibe (,) espera otro indentificador (va a q5)
- si recibe (;) se va al incio (q0) ya que la asignacion es correcta

## Q7, paso a hexadecimal valido (0x | 0X | +0X | +0x | -0X | -0x)

- si recibe un numero entre 0 y f se va al estado q9
- 

## Q9 HEXADECIMAL valido

- se mantiene mientras le llegan caracteres del 0 al 9, y de la a-f o A-F
- si recibe (,) espera otro indentificador (va a q5)
- si recibe (;) se va al incio (q0) ya que la asignacion es correcta

## Q8 DECIMAL ENTERO valido

- Una vez que se recibio un numero del 1-9 ya puede recibir del 0-9
- si recibe (,) espera otro indentificador (va a q5)
- si recibe (;) se va al incio (q0) ya que la asignacion es correcta
- tambien si se detecta automaticamente un solo (.) es valido y espera por datos decimales, si no recibe se toma como 0.0

## Q10 decimal con punto decimal

- En este punto se tiene algo como +12. o - 34.
- si se recibe ; inmediatamente despues del punto se toma como .0, entonces es valido y volvemos a q0 esperando otro analisis.
- Es valido ya que en el analizador lexico no importa si solo se tiene 10. o 9875. estos valores se toman como 10.0 y 9875.0 respectivamente.
- En este punto podemos entonces recibir otros numeros decimales para crear la parte de punto decimal.

## Q11 (posible paso) decimal con punto decimal

- Dado que un numero como float pou = .09; se considera valido y se toma como 0.09 debemos buscar el caso en el que al recibir (.) sea un posible candidato a ser un decimal.
- si se recibe al menos un numero pasa a decimal valido (q10).
- si se recibe inmediato despues del punto un (;) se invalida y se va a la basura. Al igual si se recibe (,) o (ç) (caso no mostrado en el jflap).

A PARTIR DE AQUI (Q13) EN EL JFLAP NO PONDRE LOS ESTADOS QUE INDETERMINAN, PERO AQUI SI LOS MENCIONO

## Q12 puente a crear un decimal con exponente (con punto o entero)

- Se llega a este punto si de q10 se recibe una letra ya sea "e" o "E"  
entonces es candidato a ser decimal con punto y exponente
- Si de q8, decimal entero, se recibe "e" o "E"  
entonces es candidato a convertirse en un entero exponentia.

- obvio si llega una letra, un espacio, un (;) o (,) se indetermina y va a la basura.
- no puede haber ni letras ni espacios ni nada que no sean numeros del 0 al 9 despues de la e o E
- 

## **Q13 exponencial valido, ya sea entero o con punto**

### **Q14 como el exponente (e) puede tener signo, llega aqui.**

- Se llega solo desde q12 si es que se detecta un solo signo, ya sea + o -
- si llega un numero entre el 0 y 9 pasa a q13 y es un exponencial valido.
- obvio si llega una letra, un espacio, un (;) o (,) se indetermina y va a la basura.
- no puede haber ni letras ni espacios ni nada que no sean numeros del 0 al 9 despues de la e o E

## **Q2, solo basura (se levanta la bandera de error)**

### **consideraciones del codigo:**

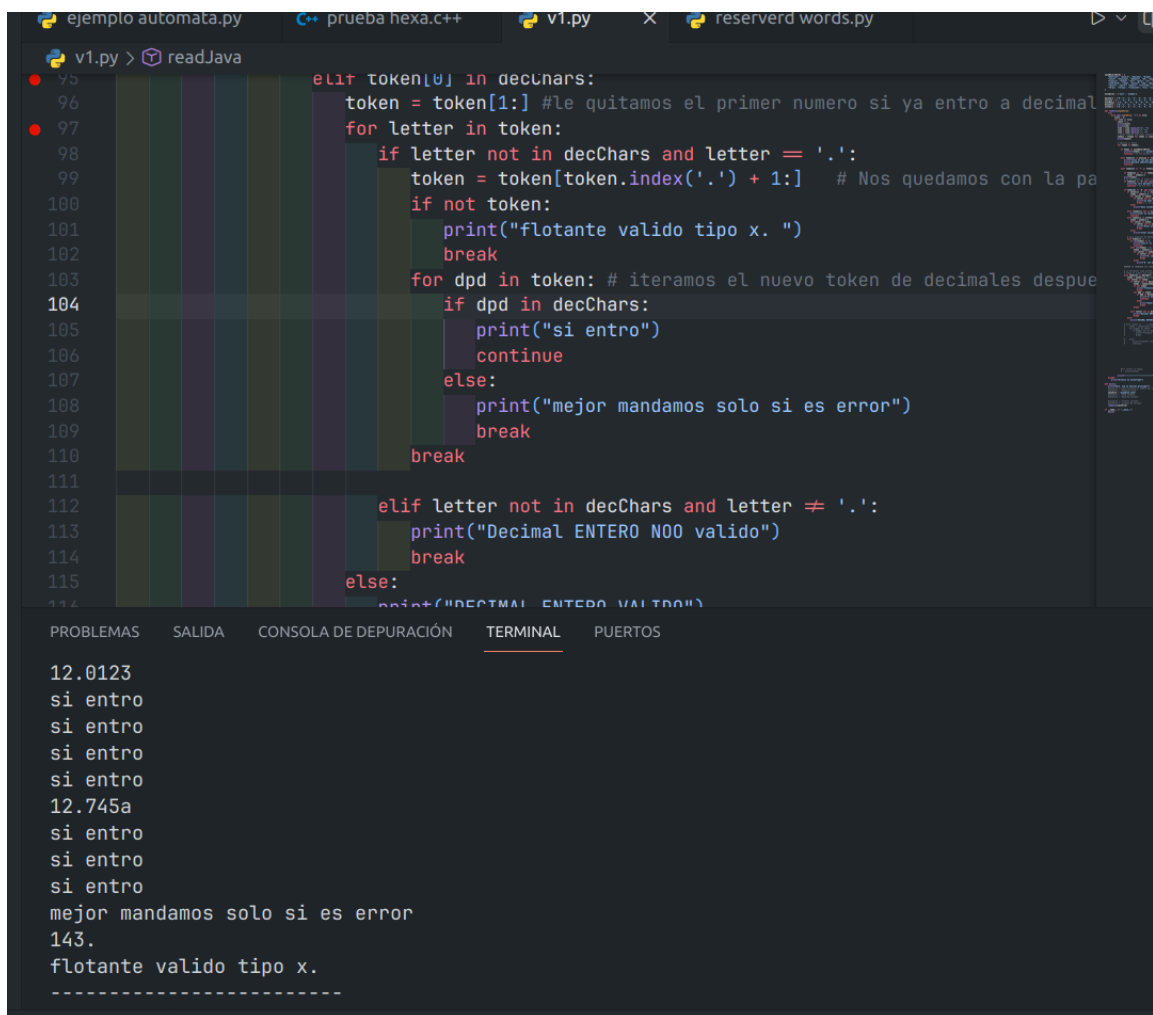
- Me tope con que en los flotantes al depurar un flotante por ejemplo: "12.23a" daba como salida:

flotante no valido  
decimal valido

y demas salidas.

En consecuencia solo vamos a guardar en el array de advertencias los errores a los cuales se ingresa solo en los else, cuando pase no se manda error alguno, en teoria el ejemplo anterior si pasaria aun por los guardados positivos pero como no guardan mensaje alguno solo se guardaria en el array:

linea x      |      mensaje que se genero al pasar por "flotante no valido"



```
ejemplo automata.py  prueba hexa.c++  v1.py  X  reservard words.py
v1.py > readJava
95 elif token[0] in decChars:
96     token = token[1:] #le quitamos el primer numero si ya entro a decimal
97     for letter in token:
98         if letter not in decChars and letter == '.':
99             token = token[token.index('.') + 1:] # Nos quedamos con la pa
100         if not token:
101             print("flotante valido tipo x. ")
102             break
103         for dpd in token: # iteramos el nuevo token de decimales despue
104             if dpd in decChars:
105                 print("si entro")
106                 continue
107             else:
108                 print("mejor mandamos solo si es error")
109                 break
110         break
111
112 elif letter not in decChars and letter != '.':
113     print("Decimal ENTERO NOO valido")
114     break
115 else:
116     print("DECIMAL ENTERO VALIDO")
117
```

PROBLEMAS    SALIDA    CONSOLA DE DEPURACIÓN    TERMINAL    PUERTOS

```
12.0123
si entro
si entro
si entro
si entro
12.745a
si entro
si entro
si entro
mejor mandamos solo si es error
143.
flotante valido tipo x.
-----
```

The screenshot shows an IDE with a Python file named `v1.py` open. The script processes tokens from a list `decChars`. It iterates through each token, and if it contains a decimal point, it splits it into integer and decimal parts. If the decimal part is non-empty, it prints a message indicating a valid float. Otherwise, it prints an error message. The terminal output shows the execution of the script on the input `12. 12.. 12.a 12.d.`, resulting in the following output:

```
3
12. 12.. 12.a 12.d.

['12.', '12..', '12.a', '12.d.']
12.
flotante valido tipo x.
12..
mejor mandamos solo si es error
12.a
mejor mandamos solo si es error
12.d.
mejor mandamos solo si es error
-----
```

## version 1 del codigo al 13 noviembre

```
javaReservWords = [  
    "abstract", "assert", "boolean", "break", "byte", "case",  
    "const", "continue", "default", "do", "double", "else",  
    "finally", "float", "for", "goto", "if", "implements",  
    "interface", "long", "native", "new", "package", "private",  
    "return", "short", "static", "strictfp", "super", "switch",  
    "throw", "throws", "transient", "try", "void", "volatile"  
]  
  
exception = ['null', 'window']
```

```

decChars = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
atozLow = ['_', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
atozMayus = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
hexChars = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f']
octChars = ['0', '1', '2', '3', '4', '5', '6', '7']

```

```

def readJava(JavaFile):
    try:
        with open (JavaFile, 'r') as file:
            count = 0;
            for line in file:
                count += 1
                print(count)
                print(line)
                line = line.replace("\n", "")
                line = line.replace(";", "")
                line = line.replace(",", "")
                # quiza tambien podriamos eliminar los "{" "}"
                tokens = [token for token in line.split(" ") if token]
                print(tokens)

            # Empieza lo bueno
            for token in tokens:

                if token in javaReservWords: # valida pala
                    print(f"{token} :::: palabra reservada")
                    continue # ya fue palabra reservada, sal

                elif token[0] in atozLow or token[0] in atozMayus:
                    print("casos en los que desde el inicio")
                    print("posible identificador valido")
                    continue

                elif token[0] == '+' or token[0] == '-' or token[0] == '*':
                    if token[0] == '+' or token[0] == '-':
                        token = token[1:]
                    print(token)

```



```

# en este punto el token actual es numero
if token[0] == '0' and len(token) == 1:
    print("es cero decimal")
    continue # ya fue cero decimal

if token[0] == '0' and len(token) > 1:
    if token[1] == 'x' or token[1] == 'X':
        token = token[2:] # quitamos el 0x
        for letter in token:
            if letter not in hexChars:
                print("ya mamo el hexa") # se
                break # si ya no fue valido
        else:
            print("hexa valido")

    elif (token[1] not in octChars and token[1] != 'x' and token[1] != 'X'):
        print("octal no valido")
        continue
    elif token[1] in octChars:
        token = token[2:]
        for letter in token:
            if letter not in octChars:
                print("Octal no valido")
                break
        else:
            print("octal valido")

# caso especial de despues del 0 (q3)
elif token[1] == '.':
    if len(token) == 2:
        print("caso 0. el cual es valido")
        continue
    elif len(token) > 2: # aqui solo se
        token = token[2:] # recortamos
        for letter in token:
            if letter not in decChars:
                print("0. con decimales no")
                break

```

```

else:
    print("0. con decimales valido")

##### SE TERMINAN LOS CASOS EN LOS QUE

# is decimales time pinche puerka
# llega el numero sin signo
elif token[0] in decChars:
    token = token[1:] #le quitamos el primer caracter
    for letter in token:
        if letter not in decChars and letter != '.':
            token = token[token.index('.') + 1:]
            if not token:
                print("flotante valido tipo entero")
                break
        for dpd in token: # iteramos el token
            if dpd in decChars:
                print("si entro")
                continue
            else:
                print("mejor mandamos solo el numero")
                break
        break

    elif letter not in decChars and letter != '.':
        print("Decimal ENTERO NOO valido")
        break
else:
    print("DECIMAL ENTERO VALIDO")

# elif letter == '.': # detecta punto decimal
#     token = token[token.index('.') + 1:]
#     for digdec in token:
#         if digdec not in decChars or digdec != '.':
#             print("flotante noo valido")
#             break

#     else:

```

```

#         print("flotante valido")
#         continue

#for letter in token:
#    print(letter)

print("-----")
except:
    print("Archivo no encontrado")

def main():
    print("Hola, soy la funcion principal")
    #JavaFile = input("Inserta el nombre del archivo .java")
    #JavaFile = "ejemplo2.java"
    JavaFile = "ejemplo2.java"
    #JavaFile = "hexa validos"
    #JavaFile = "hexa No validos"

    #JavaFile = "octales validos"
    #JavaFile = "octales NO validos"
    readJava(JavaFile)

if __name__ == "__main__":
    main()

```

notas fin de sesion:

- quite los " comillas (") tanto dobles y simples ('). Posiblemente esta condicion la quite pero veremos
  - hice la lista de operadores, quiza cesar la cambie a su modo, solo haria falta modificar ciertas condiciones la cual es un if token in operadores del incio
  - agregue al inicio unos ifs por si es un tipo de parentesis. ESTOOOO SI CORREGIRLO Y METERLOS EN UNA LISTA LLAMADA "delimitadores".  
VAMOS A VERIFICAR QUE LOS DELIMITADORES ESTEN  
BALANCEADOS, SERA TAMBIEN UN ERROR A RETORNAR
- nota: las lineas marcadas al inicio corresponen a estas anotaciones

1. me quede trabajando en las lineas 121 y 129 las cuales van a validar los exponenciales

Sin título

Practica 2 v2.0 c++