ANTHONY ASILO
PLC EXAM 2
UMOJA
FALL 2020

CODE FOR QUESTIONS 1,2,6,8

TABLE OF CONTENTS

QUESTION 1

DRIVER.PY

```
#######################################
```

```
####################################
############ PLC EXAM 2 ############
###### Created by Anthony Asilo #####
########## November 2020 ##########
#### https://github.com/pillared ####
####################################
####################################

"""
DRIVER.PY TAKES A TEXT FILE AS INPUT, AND FOR EACH STRING
WILL CHECK ALL FUNCTIONS TO FIND A VALID TOKEN
"""


from perl import validateToken_Perl
from javastring import validateToken_JavaString
from cint import validateToken_cInt
from cchar import validateToken_cChar
from cfloat import validateToken_cFloat
from ops import validateToken_Ops

import sys

def main():
    # a giant list of test words for function matching
    token={

        'token_A' : 'Perl',
        'token_B' : 'JavaString',
        'token_C' : 'cInt',
        'token_D' : 'cChar',
        'token_E' : 'cFloat',
        'token_F' : 'Ops'

    }
    arr = []
    string = ""
    filex = open("q1.txt", "r")
    string = filex.read()
    arr = string.split( )
    print(arr)
    filex.close()
    ans = None
    xo= None

    for word in arr:
```

```
        # pass to all of the validations
        # if true, let it be known what token type that word is
        print(word)
        for each in token.values():
            try:
                # eval function
                xo = "validateToken_" + each + "(" + word + ")"
                # call each specific function to test same word.
                if(eval(xo)):
                    ans = each
                    print("TOKEN FOUND :", word, "is:", each, ans)
            except SyntaxError:
                pass
        print('\n')


if __name__ == "__main__":
    main()
```

PERL.PY

```
####################################
####################################
############ PLC EXAM 2 ############
###### Created by Anthony Asilo #####
########## November 2020 ##########
#### https://github.com/pillared ####
####################################
####################################

"""
PERL.PY TAKES A WORD AS INPUT, AND FOR EACH LETTER
WILL CHECK ALL OPTIONS TO CHECK A VALID TOKEN
"""

import sys
bits={
    'dollar' : '$',
    'at' : '@',
    'perc' : '%',
    'und' : '_',
    'letA' : 'A',
    'letB' : 'B',
    'letC' : 'C',
    'letD' : 'D',
    'letE' : 'E',
    'letF' : 'F',
    'letG' : 'G',
    'letH' : 'H',
    'letI' : 'I',
    'letJ' : 'J',
    'letK' : 'K',
    'letL' : 'L',
    'letM' : 'M',
    'letN' : 'N',
    'letO' : 'O',
    'letP' : 'P',
    'letQ' : 'Q',
    'letR' : 'R',
    'letS' : 'S',
    'letT' : 'T',
    'letU' : 'U',
    'letV' : 'V',
    'letW' : 'W',
    'letX' : 'X',
    'letY' : 'Y',
    'letZ' : 'Z',
```

```
    'leta' : 'a',
    'letb' : 'b',
    'letc' : 'c',
    'letd' : 'd',
    'lete' : 'e',
    'letf' : 'f',
    'letg' : 'g',
    'leth' : 'h',
    'leti' : 'i',
    'letj' : 'j',
    'letk' : 'k',
    'letl' : 'l',
    'letm' : 'm',
    'letn' : 'n',
    'leto' : 'o',
    'letp' : 'p',
    'letq' : 'q',
    'letr' : 'r',
    'lets' : 's',
    'lett' : 't',
    'letu' : 'u',
    'letv' : 'v',
    'letw' : 'w',
    'letx' : 'x',
    'lety' : 'y',
    'letz' : 'z',
    'num0' : '0',
    'num1' : '1',
    'num2' : '2',
    'num3' : '3',
    'num4' : '4',
    'num5' : '5',
    'num6' : '6',
    'num7' : '7',
    'num8' : '8',
    'num9' : '9',
    'BEGIN' : 1,
    'NEXT' : None
}
def split(word):
    return [char for char in word]

def validateToken_Perl(arr):
    identifier = None
    for char in arr:
        # and isnumeric() for numbers, otherwise check hard code wise
```

```
        if(char in bits.values()):
            # Make sure that identifier only happens once!!!! otherwise FAIl
            if (identifier == None):
                if( (char == "$" or char == "%" or char == "@")):
                    identifier = char
                    #first letter, make sure doesnt happen again in word...
                    #make sure from now on letters, underscore, numbers,...
                    #print(char)
                else:
                    return False
            else:
                if( char.isalnum() or char == "_"):
                    pass
                else:
                    return False
        else:
            return False
    return True

def main():
    print("CORRECT PERL WORDS INCLUDE")
    print("$nwi_nw")
    print("@nwif13")
    print("%nwfm_n2ei\n")

    print("INCORRECT PERL WORDS INCLUDE")
    print("$nwi_#nw")
    print("@nwif%13")
    print("%nw@m_n2ei\n")

    print("OUR PERLWORD")
    perlword = '$nq93b'
    arr = split(perlword)
    print(perlword)
    print(arr)

    if( validateToken_Perl(arr) ):
        print("Arr is a PERL!")
    else:
        sys.exit("ERROR FAILED PARSING OF TOKEN ")

    """for key, value in bits.items():
        print(key, ' : ', value)"""


if __name__ == "__main__":
```

```
    main()
```

JAVASTRING.PY

```
#####################################
#####################################
############ PLC EXAM 2 ############
###### Created by Anthony Asilo #####
########## November 2020 ##########
#### https://github.com/pillared ####
#####################################
#####################################

"""
JAVASTRING.PY TAKES A WORD AS INPUT, AND FOR EACH LETTER
WILL CHECK ALL OPTIONS TO CHECK A VALID TOKEN
"""

import sys


"""
VALID
String s = "a dog jumped over the fucking moon!!";
String s = "N@I)INR@ )(#B ";
String s = "OI #OR\"NJ O#";
String s = "n210h_8";
String s = "";
String

Carriage return and newline: "\r" and "\n"
Backslash: "\\\\"
Single quote: "\'"
Horizontal tab and form feed: "\t" and "\f"

System.out.println('a'); //a
"""

bits={
    'num0' : '0',
    'num1' : '1',
    'num2' : '2',
```

```
'num3' : '3',
'num4' : '4',
'num5' : '5',
'num6' : '6',
'num7' : '7',
'num8' : '8',
'num9' : '9',
'letA' : 'A',
'letB' : 'B',
'letC' : 'C',
'letD' : 'D',
'letE' : 'E',
'letF' : 'F',
'letG' : 'G',
'letH' : 'H',
'letI' : 'I',
'letJ' : 'J',
'letK' : 'K',
'letL' : 'L',
'letM' : 'M',
'letN' : 'N',
'letO' : 'O',
'letP' : 'P',
'letQ' : 'Q',
'letR' : 'R',
'letS' : 'S',
'letT' : 'T',
'letU' : 'U',
'letV' : 'V',
'letW' : 'W',
'letX' : 'X',
'letY' : 'Y',
'letZ' : 'Z',
'leta' : 'a',
'letb' : 'b',
'letc' : 'c',
'letd' : 'd',
'lete' : 'e',
'letf' : 'f',
'letg' : 'g',
'leth' : 'h',
'leti' : 'i',
'letj' : 'j',
'letk' : 'k',
'letl' : 'l',
'letm' : 'm',
```

```
'letn' : 'n',
'leto' : 'o',
'letp' : 'p',
'letq' : 'q',
'letr' : 'r',
'lets' : 's',
'lett' : 't',
'letu' : 'u',
'letv' : 'v',
'letw' : 'w',
'letx' : 'x',
'lety' : 'y',
'letz' : 'z',
'symbol0' : '~' ,
'symbol1' : '`' ,
'symbol2' : '!' ,
'symbol3' : '@' ,
'symbol4' : '#' ,
'symbol5' : '$' ,
'symbol6' : '%' ,
'symbol7' : '^' ,
'symbol8' : '&' ,
'symbol9' : '*' ,
'symbol10' : '(' ,
'symbol11' : ')' ,
'symbol12' : '-' ,
'symbol13' : '_' ,
'symbol14' : '+' ,
'symbol15' : '=' ,
'symbol16' : '{' ,
'symbol17' : '[' ,
'symbol18' : '}' ,
'symbol19' : ']' ,
'symbol20' : '|' ,
'symbol21' : ' ' ,
'symbol22' : ':' ,
'symbol23' : ';' ,
'symbol24' : "" ,
'symbol25' : """ ,
'symbol26' : '<',
'symbol27' : ',',
'symbol28' : '>',
'symbol29' : '.',
'symbol30' : '?',
'symbol31' : '/',
'BEGIN' : 1,
```

```
    'NEXT' : None
}
validbits = {
    't':'t',
    'r':'r',
    'n':'n',
    'f':'f',
    "":"",
    """:"""",
    '\\':'\\',

}

def validateToken_JavaString(arr):
    identifier = None
    previous = None
    next = None
    isSlash = False
    isDoubleSlash = False
    first = None
    size = 0
    count = 0

    for char in arr:
        size = len(arr)
        if( count == 0  and first == None):
            if(char == "\""):
                first = char
                previous = char
            else:
                return False
        elif(count == size-1 ):
            if(char == '\"'):
                return True
            else:
                return False
        elif(isSlash == True):
            if(char in validbits.values()):
                pass
            elif(char == "\\"):
                isDoubleSlash = True
                isSlash = False
            else:
                return False
        elif(first != None):
            if(char in bits.values()):
```

```
            pass
        elif(char == "\\"):
            isSlash = True
        else:
            return False
    else:
        return False

    count+=1


def main():
    print('test')

    testStrings = ["'a"', "'string"', "'str ing\\t"', "'string\\"', "'stri\\"s"', "'st \\ \" ri\\"s"',
"valid??@123", "'valisd@@@/.,[][33\\{1!@#$%"']
    for word in testStrings:
        print(word)
        print('\t' + str(validateToken_JavaString(word)))
        print()


if __name__ == '__main__':
    main()
```

CINT.PY

```
####################################
####################################
############# PLC EXAM 2 ############
###### Created by Anthony Asilo #####
########### November 2020 ###########
#### https://github.com/pillared ####
####################################
####################################

"""
CINT.PY TAKES A WORD AS INPUT, AND FOR EACH LETTER
WILL CHECK ALL OPTIONS TO CHECK A VALID TOKEN
"""

import sys

"""
   int               dec_int    = 28;
   unsigned            dec_uint   = 4000000024u;
   long              dec_long   = 2000000022l;
   unsigned long       dec_ulong  = 4000000000ul;
   long long           dec_llong  = 9000000000LL;
   unsigned long long  dec_ullong = 900000000001ull;

   /* Octal Constants */
   int               oct_int    = 024;
   unsigned            oct_uint   = 04000000024u;
   long              oct_long   = 02000000022l;
   unsigned long       oct_ulong  = 04000000000UL;
   long long           oct_llong  = 0440000000000000ll;
   unsigned long long  oct_ullong = 044400000000000001Ull;

   /* Hexadecimal Constants */
   int               hex_int    = 0x2a;
   unsigned            hex_uint   = 0XA0000024u;
   long              hex_long   = 0x20000022l;
   unsigned long       hex_ulong  = 0XA0000021uL;
   long long           hex_llong  = 0x8a000000000000ll;
   unsigned long long  hex_ullong = 0x8A40000000000010uLL;
"""


bits={
   'zero':'0',
   'one':'1',
   'two':'2',
```

```
    'three':'3',
    'four':'4',
    'five':'5',
    'six':'6',
    'seven':'7',
    'eight':'8',
    'nine':'9',
    'a':'a',
    'A':'A',
    'b':'b',
    'B':'B',
    'c':'c',
    'C':'C',
    'd':'d',
    'D':'D',
    'e':'e',
    'E':'E',
    'f':'f',
    'F':'F',
    'prefix_x':'x',
    'prefix_X':'X',
    'unsigned_u':'u',
    'unsigned_U':'U',
    'long_l':'l',
    'long_L':'L',
    'long_ll':'ll',
    'long_LL':'LL',
}
valid_dec = {
    'zero':'0',
    'one':'1',
    'two':'2',
    'three':'3',
    'four':'4',
    'five':'5',
    'six':'6',
    'seven':'7',
    'eight':'8',
    'nine':'9',
}
valid_hex = {
    'zero':'0',
    'one':'1',
    'two':'2',
    'three':'3',
    'four':'4',
```

```
    'five':'5',
    'six':'6',
    'seven':'7',
    'eight':'8',
    'nine':'9',
    'a':'a',
    'A':'A',
    'b':'b',
    'B':'B',
    'c':'c',
    'C':'C',
    'd':'d',
    'D':'D',
    'e':'e',
    'E':'E',
    'f':'f',
    'F':'F',
}

valid_oct={
    'zero':'0',
    'one':'1',
    'two':'2',
    'three':'3',
    'four':'4',
    'five':'5',
    'six':'6',
    'seven':'7',
}

suffix={
    'prefix_x':'x',
    'prefix_X':'X',
    'unsigned_u':'u',
    'unsigned_U':'U',
    'long_l':'l',
    'long_L':'L',

}

def split(word):
    return [char for char in word]

def validateToken_cInt(arr):

    identifier = None
```

```
previous = None
next = None
isHex = False
isDecimal = False
isOctal = False
neverHex = False
neverDecimal = False
neverOctal = False
first = None
second = None
firstSuffix = None
noMoreHexPlease = False
noMoreDecPlease = False
noMoreOctPlease = False
unsigned = False #u
_long = False #l
unsignedlong = False #ul
_longlong = False #ll
unsigned_longlong = False #ull
ucount = 0
lcount = 0

for char in arr:
    if(char in bits.values()):
        if(isOctal):
            if(char in valid_oct.values()):
                pass
            else:
                if(firstSuffix == None):
                    noMoreOctPlease = True
                    if(char == 'u' or char == 'U'):
                        ucount+=1
                        unsigned = True
                    elif(char == 'l'):
                        lcount+=1
                        _long = True
                    else:
                        return False
                    firstSuffix = char
                    previous = firstSuffix
                elif(noMoreOctPlease):
                    if(lcount > 2 or ucount > 1):
                        print(lcount,ucount)
                        return False
                    elif(previous == 'u'):
```

```
            unsigned = True
            if(char is not None):
               return False
         elif(previous == 'l'):
            if(firstSuffix == 'U' and char == 'l'):
               lcount+=1
               unsigned_longlong = True
            elif(char == 'l'):
               lcount+=1
               _longlong = True
            else:
               return False
         elif(previous == 'U'):
            if(char == 'l' or char == 'L'):
               lcount+=1
               unsignedlong = True
            else:
               return False
         else:
            return False
      else:
         return False

elif(isDecimal):
   if(char in valid_dec.values()):
      pass
   else:
      if(firstSuffix == None):
         noMoreDecPlease = True
         if(char == 'u'):
            ucount+=1
            unsigned = True
         elif(char == 'l'):
            lcount+=1
            _long = True
         elif(char == 'L'):
            lcount+=1
            _long = True
         else:
            return False
         firstSuffix = char
         previous = firstSuffix
      elif(noMoreDecPlease):
         print(lcount,ucount)
         if(lcount > 2 or ucount > 1):
            print(lcount,ucount)
```

```
                return False
            elif(previous == 'u'):
                if(char == 'l'):
                    lcount+=1
                    unsignedlong = True
                    #print('unsignedlong = true')
                elif(char != 'l'):
                    return False
            elif(previous == 'l'):
                if(firstSuffix == 'u' and char == 'l'):
                    lcount+=1
                    unsigned_longlong = True
                    #print('longlong = true')
                elif(char == 'l'):
                    lcount+=1
                    _longlong = True
                elif(char != 'l'):
                    return False

            elif(previous == 'L'):
                if(char == 'L'):
                    lcount+=1
                    unsigned_longlong = True
                    #print('unsignedlonglong = true')
                else:
                    return False
        else:
            return False
elif(isHex):
    if(char in valid_hex.values()):
        #print("no worries, only a hex value")
        pass
    else:
        if(firstSuffix == None):
            noMoreHexPlease = True
            if(char == 'u'):
                ucount+=1
                unsigned = True
                #print('unsigned = true')
            elif(char == 'l'):
                lcount+=1
                _long = True
                #print('long = true')
            else:
                return False
            firstSuffix = char
```

```
            previous = firstSuffix
        elif(noMoreHexPlease):
            if(lcount > 2 or ucount > 1):
                print(lcount, ucount)
                return False
            elif(previous == 'u'):
                if(char == 'L'):
                    lcount+=1
                    unsignedlong = True
                    #print('unsignedlong = true')
                elif(char != 'L'):
                    return False
            elif(previous == 'l'):
                if(char == 'l'):
                    lcount+=1
                    _longlong = True
                    #print('longlong = true')
                elif(char != 'l'):
                    return False
            elif(previous == 'L'):
                if(char == 'L'):
                    lcount+=1
                    unsigned_longlong = True
                    #print('unsignedlonglong = true')
            else:
                return False
        else:
            return False
    elif(first == None):
        if(char != '0'):
            neverHex = True
            neverOctal = True
            isDecimal = True
            print("is Decimal")
        first = char
    elif(first == "0" and second == None):
        # the next can be a numer or it can be x for hex or b for binary
        second = char
        if(char == 'x' or char == 'X'):
            print('is Hex')
            neverOctal = True
            neverDecimal = True
            isOctal = False
            isHex = True
        elif(char != 0):
            print('is Octal')
```

```
                neverDecimal = True
                isOctal = True
                isHex = False
            else:
                pass #print('wtf')

        else:
            return False

    else:
        return False
    previous = char
    if(lcount > 2 or ucount > 1):
        print(lcount, ucount)
        return False
  return True

def main():

  trueWords =
["28","4000000024u","2000000022l","4000000000ul","9000000000LL","900000000001ull","02
4","04000000024u","02000000022l","04000000000UL",
"044000000000000ll","0444000000000000001Ull", "0x2a","0XA0000024u",
"0x20000022l","0XA0000021uL", "0x8a000000000000ll", "0x8A40000000000010uLL"]
  falseWords = ['28', '4000000024ulll', '2000000022lll', '4000000000uul', '9000000000LLL',
'900000000001ulll', '024', '04000000024uu', '02000000022ll', '04000000000ULL',
'044000000000000uull', '04440000000000001Ulll', '0x2a', '0XA0000024uu',
'0x20000022ll','0XA0000021uLLL','0x8a000000000000lll','0x8A40000000000010uLLL']

  for word in falseWords:
    if( validateToken_cInt(word) ):
      print(str(word) + "  -----VALID------")
    else:
      print(str(word) + "  -----ERROR------")
    print()


if __name__ == "__main__":
  main()
```

CCHAR.PY

```
####################################
####################################
############ PLC EXAM 2 ############
###### Created by Anthony Asilo #####
########## November 2020 ##########
#### https://github.com/pillared ####
####################################
####################################

"""
CCHAR.PY TAKES A WORD AS INPUT, AND FOR EACH LETTER
WILL CHECK ALL OPTIONS TO CHECK A VALID TOKEN
"""

import sys

bits={
    'num0' : '0',
    'num1' : '1',
    'num2' : '2',
    'num3' : '3',
    'num4' : '4',
    'num5' : '5',
    'num6' : '6',
    'num7' : '7',
    'num8' : '8',
    'num9' : '9',
    'letA' : 'A',
    'letB' : 'B',
    'letC' : 'C',
    'letD' : 'D',
    'letE' : 'E',
    'letF' : 'F',
    'letG' : 'G',
    'letH' : 'H',
    'letI' : 'I',
    'letJ' : 'J',
    'letK' : 'K',
    'letL' : 'L',
    'letM' : 'M',
    'letN' : 'N',
    'letO' : 'O',
    'letP' : 'P',
    'letQ' : 'Q',
    'letR' : 'R',
    'letS' : 'S',
```

```
'letT' : 'T',
'letU' : 'U',
'letV' : 'V',
'letW' : 'W',
'letX' : 'X',
'letY' : 'Y',
'letZ' : 'Z',
'leta' : 'a',
'letb' : 'b',
'letc' : 'c',
'letd' : 'd',
'lete' : 'e',
'letf' : 'f',
'letg' : 'g',
'leth' : 'h',
'leti' : 'i',
'letj' : 'j',
'letk' : 'k',
'letl' : 'l',
'letm' : 'm',
'letn' : 'n',
'leto' : 'o',
'letp' : 'p',
'letq' : 'q',
'letr' : 'r',
'lets' : 's',
'lett' : 't',
'letu' : 'u',
'letv' : 'v',
'letw' : 'w',
'letx' : 'x',
'lety' : 'y',
'letz' : 'z',
'symbol0' : '~' ,
'symbol1' : '`' ,
'symbol2' : '!' ,
'symbol3' : '@' ,
'symbol4' : '#' ,
'symbol5' : '$' ,
'symbol6' : '%' ,
'symbol7' : '^' ,
'symbol8' : '&' ,
'symbol9' : '*' ,
'symbol10' : '(' ,
'symbol11' : ')' ,
'symbol12' : '-' ,
```

```
    'symbol13' : '_' ,
    'symbol14' : '+' ,
    'symbol15' : '=' ,
    'symbol16' : '{' ,
    'symbol17' : '[' ,
    'symbol18' : '}' ,
    'symbol19' : ']' ,
    'symbol20' : '|' ,
    'symbol21' : '\\' ,
    'symbol22' : ':' ,
    'symbol23' : ';' ,
    'symbol24' : "" ,
    'symbol25' : """ ,
    'symbol26' : '<',
    'symbol27' : ',',
    'symbol28' : '>',
    'symbol29' : '.',
    'symbol30' : '?',
    'symbol31' : '/',
    'BEGIN' : 1,
    'NEXT' : None
}
```

#\b	Backspace
#\f	Form feed
#\n	New line
#\r	Carriage return
#\t	Horizontal tab
#\”	Double quote
#\’	Single quote
#\\	Backslash
#\v	Vertical tab
#\a	Alert or bell
#\?	Question mark
#\N	Octal constant (N is an octal constant)
#\XN	Hexadecimal constant (N – hex.dcml cnst)

```
def split(word):
    return [char for char in word]

def validateToken_cChar(arr):
    size = len(arr)
```

```
count = 1
identifier = None
first = None
last = None
previous = None
next = []
firstIsQuotation = False
allowBackSlashChar = True
isX = False
noMore = False
doubleSlash = False
for char in arr:

    print(char)
    if (noMore):
        return False
    if(count == size):
        if(char != last):
            return False
    elif(char in bits.values()):
        if (identifier == None):
            if(char == '\"' or char == "\"" ):
                identifier = char
                first = char
                last = char
            else:
                return False
        else:
            if(previous == '\\'):
                if(char == 'b'):
                    pass
                elif(char == 'f'):
                    pass
                elif(char == 'n'):
                    pass
                elif(char == 'r'):
                    pass
                elif(char == 't'):
                    pass
                elif(char == "'"):
                    pass
                elif(char == '"'):
                    pass
                elif(char == '\\'):
                    doubleSlash = True
                    pass
```

```
                elif(char == 'v'):
                    pass
                elif(char == 'a'):
                    pass
                elif(char == '?'):
                    pass
                elif(char == 'N'):
                    pass
                elif(char == 'X'):
                    isX = True
                    pass
                else:
                    return False
            elif(char =="\\"):
                previous = char
            elif(previous == "X" and isX):
                if(char == "N"):
                    pass
                else:
                    return False
            else:
                previous = char

        else:
            return False
        count+=1
    return True

def main():

    trueWords = ["\'2\'", "\'!\'", "\"&\"", "\'\n\'", "\'\?\'", "\'\\\'", "\'\f\'", "\'\XN\'", "\']\'", "\'n\'"]
    falseWords = ['\'x\'' , "a\'c\'",  ""]

    for word in trueWords:
        arr = split(word)
        print(word, validateToken_cChar(arr))
        print('\n')

    for word in falseWords:
        arr = split(word)
        print(word, validateToken_cChar(arr))
        print('\n')

if __name__ == "__main__":
    main()
```

CFLOAT.PY

```
#####################################
#####################################
############ PLC EXAM 2 ############
###### Created by Anthony Asilo #####
########## November 2020 ###########
#### https://github.com/pillared ####
#####################################
#####################################

"""
CFLOAT.PY TAKES A WORD AS INPUT, AND FOR EACH LETTER
WILL CHECK ALL OPTIONS TO CHECK A VALID TOKEN
"""

"""
15.75
1.575E1   /* = 15.75   */
1575e-2   /* = 15.75   */
-2.5e-3   /* = -0.0025 */
25E-4

10.0L  /* Has type long double  */
10.0F  /* Has type float        */

.0075e2
0.075e1
.075e1
75e-2
"""

import sys

bits={
   'num0' : '0',
   'num1' : '1',
   'num2' : '2',
   'num3' : '3',
   'num4' : '4',
   'num5' : '5',
   'num6' : '6',
   'num7' : '7',
   'num8' : '8',
   'num9' : '9',
   'minus' : '-',
```

```
    'plus' : '+',
    'let_e' : 'e',
    'let_E' : 'E',
    'let_l' : 'l',
    'let_L' : 'L',
    'let_f' : 'f',
    'let_F' : 'F',
    'dec' : '.'
}

def split(word):
    return [char for char in word]

def validateToken_cFloat(arr):
    identifier = None
    first = None
    previous = None
    next = []
    firstIsDec = False
    allowDec = True
    allowE = True
    allowL = True
    allowF = True
    allowSign = True
    allowP = True
    noMore = False
    for char in arr:
        if (noMore):
            return False
        elif(char in bits.values()):
            if (identifier == None):
                identifier = char
                first = char
                if(char.isnumeric()):
                    next = ["e", ".", '0-9']
                elif(char == "."):
                    firstIsDec = True
                    allowDec = False
                    next = ['0-9']
                elif(char == "-"):
                    next = ['.', '0-9']
                else:
                    return False
            else:
                if( char.isnumeric() ):
                    previous = char
```

```
            elif(allowDec and char == "."):
                allowDec = False
                if(firstIsDec):
                    return False
                previous = char
                next = ['0-9']
            elif(previous.isnumeric()):
                if(allowE and (char == 'e' or char =='E')):
                    allowE = False
                    allowL = False
                    allowF = False
                    previous = char
                elif(allowL and (char == 'l' or char =='L')):
                    allowE = False
                    allowL = False
                    allowF = False
                    noMore = True
                    previous = char
                elif(allowF and (char == 'f' or char =='F')):
                    allowE = False
                    allowL = False
                    allowF = False
                    noMore = True
                    previous = char
                else:
                    return False
            elif(allowSign and (char == "+" or char == "-")):
                allowSign = False
            elif(allowSign == False):
                return False
            else:
                return False
        else:
            return False
    return True

def main():

    trueWords = ["15.75", "1.575E1", "1575e-2", "-2.5e-3", "25E-4", "10.0L", "10.0F", ".0075e2",
"0.075e1", ".075e1", "75e-2"]
    falseWords = ["15.L75L", "1.57.5E1", "157.5e-+", "+2.5e-3", "25.L-4", "10.0LF", "1x0.0F",
".0075ef2", "0.075ee1", ".075e1f", "75e--2"]

    for word in trueWords:
        arr = split(word)
        print(word, validateToken_cFloat(arr))
```

```
        print('\n')

    for word in falseWords:
        arr = split(word)
        print(word, validateToken_cFloat(arr))
        print('\n')


if __name__ == "__main__":
    main()
```

OPS.PY

```
#####################################
#####################################
############ PLC EXAM 2 ############
###### Created by Anthony Asilo #####
########## November 2020 ###########
#### https://github.com/pillared ####
#####################################
#####################################

"""
OPS.PY TAKES A WORD AS INPUT, AND FOR EACH LETTER
WILL CHECK ALL OPTIONS TO CHECK A VALID TOKEN
"""

# // – Addition --> '+'
# // – Assignment --> '='
# // – Subtraction --> '-'
# // – Multiplication --> '*'
# // – Increment --> '++' | '+='
# // – Decrement --> '--' | '-='
# // – Modulo Operator --> '%'
# // – Logical And --> '&&' | 'and'
# // – Logical Or --> '||' | 'or'
# // – Logical Not --> '!' | 'not'
# // – Open Code Block --> '{'
# // – CloCode Block --> '}'
# // – Open Function parameter – '('
# // - CloFunction parameter --> '):'

def validateToken_Ops(char):
        if (char == '+' ):
                pass
        elif ( char == '-' ): # – Addition --> '+'  1lvl
                pass
        elif ( char == '='): # – Assignment --> '=' 1lvl
                pass
        elif ( char == '-'): # – Subtraction --> '-' 1lvl
                pass
        elif ( char == '/'): # – Division --> '/' 1lvl
                pass
        elif ( char == '*'): # – Multiplication --> '*' 1lvl
                pass
        elif ( char == '%'): # – Modulo Operator --> '%' 1lvl
                pass
```

```
        elif ( char == '++' or char == '+='): # – Increment --> '++' | '+='  2lvl
                pass
        elif ( char == '--' or char == '-='): # – Decrement --> '--' | '-=' 2lvl
                pass
        elif ( char == '&&' or char == 'and'): # – Logical And --> '&&' | 'and' 2lvl
                pass
        elif ( char == '||' or char == 'or'): # – Logical Or --> '||' | 'or' 2lvl
                pass
        elif ( char == '!' or char == 'not'): # – Logical Not --> '!' | 'not' 1lvl
                pass
        elif ( char == '{'): # – Open Code Block --> '{' 1lvl
                pass
        elif ( char == '}'): # – CloCode Block --> '}' 1lvl
                pass
        elif ( char == '('): # – Open Function parameter – '(' 1lvl
                pass
        elif ( char == ')'): # - CloFunction parameter --> '):' 1lvl
                pass
        else:
                return False
        return True

def main():
        trueWords = ['+', '-', '(', ')', 'or', 'and', '*', '/', '%']
        falseWords = ['lk', '!', 'nand', 'xor', 'ur mom', '*=', '==', '()', '[]']
        for word in trueWords:
                print(word)
                print(validateToken_Ops(word))
                print()
        for word in falseWords:
                print(word)
                print(validateToken_Ops(word))
                print()

if __name__ == "__main__":
        main()
```

Q1.TXT – INCLUDES ALL CASES SOME GOOD SOME BAD

```
\"nai\\ow\"'
\"0x82n3\"'
 \"13.4e12\"'
"'nai\\ow"'
\"0x82n3\"'
\"13.4e12\"'
"'@fuckplc"'
"\'2\"'
 "\'!\"'
  "\"&\""'
  "\"\n\""'
  "\"\?\"'
   "\"\\\"'
   "\"\f\"'
   "\"\XN\"'
    "\"\"'
    "\'n\"'
    "\"15.75\""'
    "\"1.575E1\""'
    "\"1575e-2\""'
    "\"-2.5e-3\""'
    "\"25E-4\""'
    "\"10.0L\""'
    "\"10.0F\""'
    "\".0075e2\"" "\"0.075e1\"" "\".075e1\"" "\"75e-2\""'
    "\"28\""'
    "\"4000000024u\""'
    "\"2000000022l\""'
    "\"4000000000ul\""'
    "\"9000000000LL\""'
    "\"900000000001ull\""'
    "\"024\""'
    "\"04000000024u\""'
    "\"02000000022l\""'
    "\"04000000000UL\""'
    "\"0440000000000000ll\""'
    "\"044400000000000001Ull\""'
    "\"0x2a\""'
    "\"0XA0000024u\""'
     "\"0x20000022l\""'
    "\"0XA0000021uL\""'
     "\"0x8a000000000000ll\""'
    "\"0x8A40000000000010uLL\""'
    "'a"'
    "'string"'
     "'str ing\\t"'
```

```
    "'string\\"'
    "'stri\\"s'"
    '\"st \\ \" ri\\"s\"'
     "\"valid??@123\""
      '\"valisd@@@/.33\\{1!@#$%\""\"+\"'
      '\"-\"'
      '\"(\"'
      '\")\"'
      '\"or\"'
       '\"and\"'
        '\"*\"'
        '\"/\"'
        '\"%\"'


  "\'2\'"
   "\'!\'"
  "\"&\""
   "\'\n\'"
  "\'\?\'"
   "\'\\\'"
  "\'\f\'"
  "\'\XN\'"
   "\'\'"
   "\'n\'"


  '\'x\'"
   "a\'c\'"
   ""

 "15.75"
"1.575E1"
 "1575e-2"
 "-2.5e-3"
 "25E-4" "10.0L" "10.0F" ".0075e2" "0.075e1" ".075e1" "75e-2"

  "15.L75L" "1.57.5E1" "157.5e-+" "+2.5e-3" "25.L-4" "10.0LF" "1x0.0F" ".0075ef2"
"0.075ee1" ".075e1f" "75e--2"

   "28" "4000000024u" "2000000022l" "4000000000ul" "9000000000LL" "900000000001ull"
   "024"
   "04000000024u"
   "02000000022l"
   "04000000000UL" "0440000000000000ll"
   "044400000000000001Ull" "0x2a"
```

```
    "0XA0000024u" "0x20000022l"
    "0XA0000021uL" "0x8a000000000000ll" "0x8A40000000000010uLL"

   '28' '4000000024ulll' '2000000022lll' '4000000000uul' '9000000000LLL' '900000000001ulll'
'024' '04000000024uu' '02000000022ll' '04000000000ULL' '044000000000000uull'
'044400000000000001Ulll' '0x2a' '0XA0000024uu'
'0x20000022ll"0XA0000021uLLL"0x8a000000000000lll"0x8A40000000000010uLLL'

 "'a"' "'string"' "'str ing\\t"' "'string\\"' "'stri\\"s"' "'st \\ \" ri\\"s"' "valid??@123"
"'valisd@@@/.[[33\\{1!@#$%"'
'+' '-' '(' ')' 'or' 'and' '*' '/' '%'
'lk' '!' 'nand' 'xor' 'ur mom' '*=' '==' '()' '[]'

   '@plc' '$nowe_245' '%wbnod20983'
```

OUTPUT.LOG

Script started on Sun Nov 15 20:38:56 2020

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
[?1034hbash-3.2$ python3 driver.py
['\'\\"nai\\\\ow\\"\', '\'\\"0x82n3\\"\', '\'\\"13.4e12\\"\', '\"nai\\\\ow"\', '\'\\"0x82n3\\"\',
'\'\\"13.4e12\\"\', '\"@fuckplc"\', '"\\\'2\\\'"', '"\\\'!\\\'"', '"\\"&\\""', '"\\"\\n\\""', '"\\\'\\?\\\'"',
'"\\\'\\\\\\\\"', '"\\\'\\f\\\'"', '"\\\'\\XN\\\'"', '"\\\'\\\'"', '"\\\'n\\\'"', '"\\"15.75\\""', '"\\"1.575E1\\""',
'"\\"1575e-2\\""', '"\\"-2.5e-3\\""', '"\\"25E-4\\""', '"\\"10.0L\\""', '"\\"10.0F\\""', '"\\".0075e2\\""',
'"\\"0.075e1\\""', '"\\".075e1\\""', '"\\"75e-2\\""', '"\\"28\\""', '"\\"4000000024u\\""',
'"\\"2000000022l\\""', '"\\"4000000000ul\\""', '"\\"9000000000LL\\""', '"\\"900000000001ull\\""',
'"\\"024\\""', '"\\"04000000024u\\""', '"\\"02000000022l\\""', '"\\"04000000000UL\\""',
'"\\"044000000000000ll\\""', '"\\"0444000000000000l1ll\\""', '"\\"0x2a\\""',
'"\\"0XA0000024u\\""', '"\\"0x20000022l\\""', '"\\"0XA0000021uL\\""',
'"\\"0x8a000000000000ll\\""', '"\\"0x8A40000000000010uLL\\""', '\'"a"\', '\'"string"\', '\'"str',
'ing\\\\t"\', '\'"string\\\\"\', '\'"stri\\\\"s"\', '\'\\"st', '\\\\', '\\"', 'ri\\\\"s\\"\', '"\\"valid??@123\\""',
'\'\\"valisd@@@/.33\\\\{1!@#$%\\"\'\\"+\\"\', '\'\\"-\\"\', '\'\\"(\\"\', '\'\\")\\"\', '\'\\"or\\"\',
'\'\\"and\\"\', '\'\\"*\\"\', '\'\\"/\\"\', '\'\\"%\\"\', '"\\\'2\\\'"', '"\\\'!\\\'"', '"\\"&\\""', '"\\\'\\n\\\'"',
'"\\\'\\?\\\'"', '"\\\'\\\\\\\\"', '"\\\'\\f\\\'"', '"\\\'\\XN\\\'"', '"\\\'\\\'"', '"\\\'n\\\'"', '\'\\\'x\\"\', "'a\\\'c\\\'"', '"\'"',
"'15.75'", "'1.575E1'", "'1575e-2'", "'-2.5e-3'", "'25E-4'", "'10.0L'", "'10.0F'", "'.0075e2'",
"'0.075e1'", "'.075e1'", "'75e-2'", "'15.L75L'", "'1.57.5E1'", "'157.5e-+'", "'+2.5e-3'", "'25.L-4'",
"'10.0LF'", "'1x0.0F'", "'.0075ef2'", "'0.075ee1'", "'.075e1f'", "'75e--2'", "'28'", "'4000000024u'",
"'2000000022l'", "'4000000000ul'", "'9000000000LL'", "'900000000001ull'", "'024'",
"'04000000024u'", "'02000000022l'", "'04000000000UL'", "'044000000000000ll'",
"'04440000000000001Ull'", "'0x2a'", "'0XA0000024u'", "'0x20000022l'", "'0XA0000021uL'",
"'0x8a000000000000ll'", "'0x8A40000000000010uLL'", "'28'", "'4000000024ulll'",
"'2000000022lll'", "'4000000000uul'", "'9000000000LLL'", "'900000000001ulll'", "'024'",
"'04000000024uu'", "'02000000022ll'", "'04000000000ULL'", "'044000000000000uull'",
"'04440000000000001Ulll'", "'0x2a'", "'0XA0000024uu'",
"'0x20000022ll"0XA0000021uLLL"0x8a000000000000lll"0x8A40000000000010uLLL'",
'\'"a"\', \'"string"\', \'"str', 'ing\\\\t"\', \'"string\\\\"\', \'"stri\\\\"s"\', \'"st', '\\\\', '\\"', 'ri\\\\"s"\',
"'valid??@123"', '\'"valisd@@@/.[[33\\\\{1!@#$%"\', "'+"', "'-"', "'("', "')"', "'or"', "'and"', "'*"',
"'/"', "'%"', "'lk"', "'!"', "'nand"', "'xor"', "'ur"', "mom"', "'*="', "'=="', "'()"', "'[]"', "'@plc"',
"'$nowe_245"', "'%wbnod20983"']
\"nai\\ow\"'
"
n
a
i
\
o

\"0x82n3\"'
TOKEN FOUND : \"0x82n3\"' is: JavaString JavaString

"
0
x
8
2
n
3
"
TOKEN FOUND : '\"0x82n3\"' is: cChar cChar

'\"13.4e12\"'
TOKEN FOUND : '\"13.4e12\"' is: JavaString JavaString
"
1
3
.
4
e
1
2
"
TOKEN FOUND : '\"13.4e12\"' is: cChar cChar

'"nai\\ow"'
"
n
a
i
\
o

'\"0x82n3\"'
TOKEN FOUND : '\"0x82n3\"' is: JavaString JavaString
"
0
x
8
2
n
3
"
TOKEN FOUND : '\"0x82n3\"' is: cChar cChar

'\"13.4e12\"'
TOKEN FOUND : '\"13.4e12\"' is: JavaString JavaString
"
1
3
.
4
e
1
2
"
TOKEN FOUND : '\"13.4e12\"' is: cChar cChar

'"@fuckplc"'
TOKEN FOUND : '"@fuckplc"' is: JavaString JavaString
"
@
f
u
c
k
p
l
c
"
TOKEN FOUND : '"@fuckplc"' is: cChar cChar

"'2'"
'
2
'
TOKEN FOUND : "'2'" is: cChar cChar

"'!'"
'
!
'
TOKEN FOUND : "'!'" is: cChar cChar

"'&'"
TOKEN FOUND : "'&'" is: JavaString JavaString

"
&
"
TOKEN FOUND : "\"&\"" is: cChar cChar

"\"\n\""
"

"\'\?\'"
'
\
?
'
TOKEN FOUND : "\'\?\'" is: cChar cChar

"\'\\\'"
'
\
'
TOKEN FOUND : "\'\\\'" is: cChar cChar

"\'\f\'"
'

"\'\XN\'"
'
\
X
N
'
TOKEN FOUND : "\'\XN\'" is: cChar cChar

"\'\'"
'
'
TOKEN FOUND : "\'\'" is: cChar cChar

"\'n\'"
'
n
'
TOKEN FOUND : "\'n\'" is: cChar cChar

"\"15.75\""
TOKEN FOUND : "\"15.75\"" is: JavaString JavaString
"
1
5
.
7
5
"
TOKEN FOUND : "\"15.75\"" is: cChar cChar

"\"1.575E1\""
TOKEN FOUND : "\"1.575E1\"" is: JavaString JavaString
"
1
.
5
7
5
E
1

"
TOKEN FOUND : "\"1.575E1\"" is: cChar cChar

"\"1575e-2\""
TOKEN FOUND : "\"1575e-2\"" is: JavaString JavaString
"
1
5
7
5
e
-
2
"
TOKEN FOUND : "\"1575e-2\"" is: cChar cChar

"\"-2.5e-3\""
TOKEN FOUND : "\"-2.5e-3\"" is: JavaString JavaString
"
-
2
.
5
e
-
3
"
TOKEN FOUND : "\"-2.5e-3\"" is: cChar cChar

"\"25E-4\""
TOKEN FOUND : "\"25E-4\"" is: JavaString JavaString
"
2
5
E
-
4
"
TOKEN FOUND : "\"25E-4\"" is: cChar cChar

"\"10.0L\""
TOKEN FOUND : "\"10.0L\"" is: JavaString JavaString

"
1
0
.
0
L
"
TOKEN FOUND : "\"10.0L\"" is: cChar cChar

"\"10.0F\""
TOKEN FOUND : "\"10.0F\"" is: JavaString JavaString
"
1
0
.
0
F
"
TOKEN FOUND : "\"10.0F\"" is: cChar cChar

"\".0075e2\""
TOKEN FOUND : "\".0075e2\"" is: JavaString JavaString
"
.
0
0
7
5
e
2
"
TOKEN FOUND : "\".0075e2\"" is: cChar cChar

"\"0.075e1\""
TOKEN FOUND : "\"0.075e1\"" is: JavaString JavaString
"
0
.
0
7
5
e
1

"
TOKEN FOUND : "\"0.075e1\"" is: cChar cChar

"\".075e1\""
TOKEN FOUND : "\".075e1\"" is: JavaString JavaString
"
.
0
7
5
e
1
"
TOKEN FOUND : "\".075e1\"" is: cChar cChar

"\"75e-2\""
TOKEN FOUND : "\"75e-2\"" is: JavaString JavaString
"
7
5
e
-
2
"
TOKEN FOUND : "\"75e-2\"" is: cChar cChar

"\"28\""
TOKEN FOUND : "\"28\"" is: JavaString JavaString
"
2
8
"
TOKEN FOUND : "\"28\"" is: cChar cChar

"\"4000000024u\""
TOKEN FOUND : "\"4000000024u\"" is: JavaString JavaString
"
4
0
0
0
0

0
0
0
2
4
u
"
TOKEN FOUND : "\"4000000024u\"" is: cChar cChar

"\"2000000022l\""
TOKEN FOUND : "\"2000000022l\"" is: JavaString JavaString
"
2
0
0
0
0
0
0
0
2
2
l

"
TOKEN FOUND : "\"2000000022l\"" is: cChar cChar

"\"4000000000ul\""
TOKEN FOUND : "\"4000000000ul\"" is: JavaString JavaString
"
4
0
0
0
0
0
0
0
0
0
u
l
"
TOKEN FOUND : "\"4000000000ul\"" is: cChar cChar

"\"9000000000LL\""
TOKEN FOUND : "\"9000000000LL\"" is: JavaString JavaString
"
9
0
0
0
0
0
0
0
0
0
L
L
"
TOKEN FOUND : "\"9000000000LL\"" is: cChar cChar

"\"900000000001ull\""
TOKEN FOUND : "\"900000000001ull\"" is: JavaString JavaString
"
9
0
0
0
0
0
0
0
0
0
0
1
u
l
l
"
TOKEN FOUND : "\"900000000001ull\"" is: cChar cChar

"\"024\""
TOKEN FOUND : "\"024\"" is: JavaString JavaString
"

0
2
4
"
TOKEN FOUND : "\"024\"" is: cChar cChar

"\"04000000024u\""
TOKEN FOUND : "\"04000000024u\"" is: JavaString JavaString
"
0
4
0
0
0
0
0
0
0
2
4
u
"
TOKEN FOUND : "\"04000000024u\"" is: cChar cChar

"\"02000000022l\""
TOKEN FOUND : "\"02000000022l\"" is: JavaString JavaString
"
0
2
0
0
0
0
0
0
0
2
2
l
"
TOKEN FOUND : "\"02000000022l\"" is: cChar cChar

"\"04000000000UL\""

TOKEN FOUND : "\"04000000000UL\"" is: JavaString JavaString
"
0
4
0
0
0
0
0
0
0
0
0
U
L
"
TOKEN FOUND : "\"04000000000UL\"" is: cChar cChar

"\"0440000000000000ll\""
TOKEN FOUND : "\"0440000000000000ll\"" is: JavaString JavaString
"
0
4
4
0
0
0
0
0
0
0
0
0
0
0
0
l
l
"
TOKEN FOUND : "\"0440000000000000ll\"" is: cChar cChar

"\"044400000000000001Ull\""
TOKEN FOUND : "\"044400000000000001Ull\"" is: JavaString JavaString
"

0
4
4
4
0
0
0
0
0
0
0
0
0
0
0
0
1
U
l
l
"
TOKEN FOUND : "\"04440000000000001Ull\"" is: cChar cChar

"\"0x2a\""
TOKEN FOUND : "\"0x2a\"" is: JavaString JavaString
"
0
x
2
a
"
TOKEN FOUND : "\"0x2a\"" is: cChar cChar

"\"0XA0000024u\""
TOKEN FOUND : "\"0XA0000024u\"" is: JavaString JavaString
"
0
X
A
0
0
0
0

0
2
4
u
"
TOKEN FOUND : "\"0XA0000024u\"" is: cChar cChar

"\"0x20000022l\""
TOKEN FOUND : "\"0x20000022l\"" is: JavaString JavaString
"
0
x
2
0
0
0
0
0
2
2
l
"
TOKEN FOUND : "\"0x20000022l\"" is: cChar cChar

"\"0XA0000021uL\""
TOKEN FOUND : "\"0XA0000021uL\"" is: JavaString JavaString
"
0
X
A
0
0
0
0
0
2
1
u
L
"
TOKEN FOUND : "\"0XA0000021uL\"" is: cChar cChar

"\"0x8a000000000000ll\""

TOKEN FOUND : "\"0x8a000000000000ll\"" is: JavaString JavaString
"
0
x
8
a
0
0
0
0
0
0
0
0
0
0
0
0
0
l
l
"
TOKEN FOUND : "\"0x8a000000000000ll\"" is: cChar cChar

"\"0x8A40000000000010uLL\""
TOKEN FOUND : "\"0x8A40000000000010uLL\"" is: JavaString JavaString
"
0
x
8
A
4
0
0
0
0
0
0
0
0
0
0
0
0
1
0
u

L
L
"
TOKEN FOUND : "\"0x8A40000000000010uLL\"" is: cChar cChar

"'a"'
TOKEN FOUND : "'a"' is: JavaString JavaString
"
a
"
TOKEN FOUND : "'a"' is: cChar cChar

"'string"'
TOKEN FOUND : "'string"' is: JavaString JavaString
"
s
t
r
i
n
g
"
TOKEN FOUND : "'string"' is: cChar cChar

"'str

ing\\t"'

"'string\\"'
TOKEN FOUND : "'string\\"' is: JavaString JavaString
"
s
t
r
i
n
g
\
"
TOKEN FOUND : "'string\\"' is: cChar cChar

"'stri\\"s"'
"
s
t
r
i
\
"
s

'\"st

\\

\"

ri\\"s\"'

"\"valid??@123\""
TOKEN FOUND : "\"valid??@123\"" is: JavaString JavaString
"
v
a
l
i
d
?
?
@
1
2
3
"
TOKEN FOUND : "\"valid??@123\"" is: cChar cChar

'\"valisd@@@/.33\\{1!@#$%\""\"+\"'
"
v
a

l
i
s
d
@
@
@
/
.
3
3
\
{

'\"-\"'
TOKEN FOUND : '\"-\"' is: JavaString JavaString
"
-
"
TOKEN FOUND : '\"-\"' is: cChar cChar

'\"(\"'
TOKEN FOUND : '\"(\"' is: JavaString JavaString
"
(
"
TOKEN FOUND : '\"(\"' is: cChar cChar

'\")\"'
TOKEN FOUND : '\")\"' is: JavaString JavaString
"
)
"
TOKEN FOUND : '\")\"' is: cChar cChar

'\"or\"'
TOKEN FOUND : '\"or\"' is: JavaString JavaString
"
o
r
"
TOKEN FOUND : '\"or\"' is: cChar cChar

'\"and\"'
TOKEN FOUND : '\"and\"' is: JavaString JavaString
"
a
n
d
"
TOKEN FOUND : '\"and\"' is: cChar cChar

'\"*\"'
TOKEN FOUND : '\"*\"' is: JavaString JavaString
"
*
"
TOKEN FOUND : '\"*\"' is: cChar cChar

'\"/\"'
TOKEN FOUND : '\"/\"' is: JavaString JavaString
"
/
"
TOKEN FOUND : '\"/\"' is: cChar cChar

'\"%\"'
TOKEN FOUND : '\"%\"' is: JavaString JavaString
"
%
"
TOKEN FOUND : '\"%\"' is: cChar cChar

"\'2\"'
'
2
'
TOKEN FOUND : "\'2\"' is: cChar cChar

"\'!\"'
'
!

'
TOKEN FOUND : "\'!\'" is: cChar cChar

"\"&\""
TOKEN FOUND : "\"&\"" is: JavaString JavaString
"
&
"
TOKEN FOUND : "\"&\"" is: cChar cChar

"\'\n\'"
'

"\'\?\'"
'
\
?
'
TOKEN FOUND : "\'\?\'" is: cChar cChar

"\'\\\'"
'
\
'
TOKEN FOUND : "\'\\\'" is: cChar cChar

"\'\f\'"
'

"\'\XN\'"
'
\
X
N
'
TOKEN FOUND : "\'\XN\'" is: cChar cChar

"\'\'"
'
'
TOKEN FOUND : "\'\'" is: cChar cChar

"\'n\'"
'
n
'
TOKEN FOUND : "\'n\'" is: cChar cChar

'\'x\'"
'
x
"

"a\'c\'"
is Decimal
a

""
TOKEN FOUND : "" is: Perl Perl
TOKEN FOUND : "" is: cInt cInt
TOKEN FOUND : "" is: cChar cChar
TOKEN FOUND : "" is: cFloat cFloat

"15.75"
is Decimal
1
TOKEN FOUND : "15.75" is: cFloat cFloat

"1.575E1"
is Decimal
1
TOKEN FOUND : "1.575E1" is: cFloat cFloat

"1575e-2"
is Decimal
1
TOKEN FOUND : "1575e-2" is: cFloat cFloat

"-2.5e-3"
-
TOKEN FOUND : "-2.5e-3" is: cFloat cFloat

"25E-4"
is Decimal
2
TOKEN FOUND : "25E-4" is: cFloat cFloat

"10.0L"
is Decimal
1
TOKEN FOUND : "10.0L" is: cFloat cFloat

"10.0F"
is Decimal
1
TOKEN FOUND : "10.0F" is: cFloat cFloat

".0075e2"
.
TOKEN FOUND : ".0075e2" is: cFloat cFloat

"0.075e1"
0
TOKEN FOUND : "0.075e1" is: cFloat cFloat

".075e1"
.
TOKEN FOUND : ".075e1" is: cFloat cFloat

"75e-2"
is Decimal
7
TOKEN FOUND : "75e-2" is: cFloat cFloat

"15.L75L"
is Decimal
1

"1.57.5E1"
is Decimal
1

"157.5e-+"
is Decimal
1

"+2.5e-3"
+

"25.L-4"
is Decimal
2

"10.0LF"
is Decimal
1

"1x0.0F"
is Decimal
1

".0075ef2"
.

"0.075ee1"
0

".075e1f"
.

"75e--2"
is Decimal
7

"28"
is Decimal
TOKEN FOUND : "28" is: cInt cInt
2
TOKEN FOUND : "28" is: cFloat cFloat

"4000000024u"
is Decimal
TOKEN FOUND : "4000000024u" is: cInt cInt
4

"2000000022l"
is Decimal
TOKEN FOUND : "2000000022l" is: cInt cInt
2
TOKEN FOUND : "2000000022l" is: cFloat cFloat

"4000000000ul"
is Decimal
0 1
TOKEN FOUND : "4000000000ul" is: cInt cInt
4

"9000000000LL"
is Decimal

1 0
TOKEN FOUND : "9000000000LL" is: cInt cInt
9

"900000000001ull"
is Decimal
0 1
1 1
TOKEN FOUND : "900000000001ull" is: cInt cInt
9

"024"
is Octal
TOKEN FOUND : "024" is: cInt cInt
0
TOKEN FOUND : "024" is: cFloat cFloat

"04000000024u"
is Octal
TOKEN FOUND : "04000000024u" is: cInt cInt
0

"02000000022l"
is Octal
TOKEN FOUND : "02000000022l" is: cInt cInt
0
TOKEN FOUND : "02000000022l" is: cFloat cFloat

"04000000000UL"
is Octal
TOKEN FOUND : "04000000000UL" is: cInt cInt
0

"0440000000000000ll"
is Octal
TOKEN FOUND : "0440000000000000ll" is: cInt cInt
0

"04440000000000001Ull"

```
is Octal
TOKEN FOUND : "044400000000000001Ull" is: cInt cInt
0


"0x2a"
is Hex
TOKEN FOUND : "0x2a" is: cInt cInt
0


"0XA0000024u"
is Hex
TOKEN FOUND : "0XA0000024u" is: cInt cInt
0


"0x20000022l"
is Hex
TOKEN FOUND : "0x20000022l" is: cInt cInt
0


"0XA0000021uL"
is Hex
TOKEN FOUND : "0XA0000021uL" is: cInt cInt
0


"0x8a000000000000ll"
is Hex
TOKEN FOUND : "0x8a000000000000ll" is: cInt cInt
0


"0x8A40000000000010uLL"
is Hex
TOKEN FOUND : "0x8A40000000000010uLL" is: cInt cInt
0


'28'
is Decimal
TOKEN FOUND : '28' is: cInt cInt
2
TOKEN FOUND : '28' is: cFloat cFloat
```

'4000000024ulll'
is Decimal
0 1
1 1
2 1
3 1
4

'2000000022lll'
is Decimal
1 0
2 0
3 0
2

'4000000000uul'
is Decimal
0 1
4

'9000000000LLL'
is Decimal
1 0
2 0
3 0
9

'900000000001ulll'
is Decimal
0 1
1 1
2 1
3 1
9

'024'
is Octal
TOKEN FOUND : '024' is: cInt cInt
0

TOKEN FOUND : '024' is: cFloat cFloat

'04000000024uu'
is Octal
0

'02000000022ll'
is Octal
TOKEN FOUND : '02000000022ll' is: cInt cInt
0

'04000000000ULL'
is Octal
0

'044000000000000uull'
is Octal
0

'04440000000000001Ulll'
is Octal
3 1
0

'0x2a'
is Hex
TOKEN FOUND : '0x2a' is: cInt cInt
0

'0XA0000024uu'
is Hex
0

'0x20000022ll"0XA0000021uLLL"0x8a000000000000lll"0x8A40000000000010uLLL'
is Hex
0

'"a"'
TOKEN FOUND : '"a"' is: JavaString JavaString
"
a
"
TOKEN FOUND : '"a"' is: cChar cChar

'"string"'
TOKEN FOUND : '"string"' is: JavaString JavaString
"
s
t
r
i
n
g
"
TOKEN FOUND : '"string"' is: cChar cChar

'"str

ing\\t"'

'"string\\"'
TOKEN FOUND : '"string\\"' is: JavaString JavaString
"
s
t
r
i
n
g
\
"
TOKEN FOUND : '"string\\"' is: cChar cChar

'"stri\\"s"'
"
s
t
r

i
\
"
s

"'st

\\

\"

ri\\"s"'

"valid??@123"
v

"'valisd@@@/.[[33\\{1!@#$%"'
"
v
a
l
i
s
d
@
@
@
/
.
[
[
3
3
\
{

'+'
+
TOKEN FOUND : '+' is: Ops Ops

'-'
-
TOKEN FOUND : '-' is: cFloat cFloat
TOKEN FOUND : '-' is: Ops Ops

'('
(
TOKEN FOUND : '(' is: Ops Ops

')'
)
TOKEN FOUND : ')' is: Ops Ops

'or'
o
TOKEN FOUND : 'or' is: Ops Ops

'and'
is Decimal
a
TOKEN FOUND : 'and' is: Ops Ops

'*'
*
TOKEN FOUND : '*' is: Ops Ops

'/'
/
TOKEN FOUND : '/' is: Ops Ops

'%'
TOKEN FOUND : '%' is: Perl Perl
%
TOKEN FOUND : '%' is: Ops Ops

'lk'

is Decimal
l

'!'
!
TOKEN FOUND : '!' is: Ops Ops

'nand'
n

'xor'
is Decimal
x

'ur

mom'

'*='
*

'=='
=

'()'
(

'[]'
[

'@plc'
TOKEN FOUND : '@plc' is: Perl Perl
@

'$nowe_245'

TOKEN FOUND : '$nowe_245' is: Perl Perl
$

'%wbnod20983'
TOKEN FOUND : '%wbnod20983' is: Perl Perl
%

bash-3.2$ exit
exit

Script done on Sun Nov 15 20:39:06 2020

QUESTION 2

```
/*###################################
###################################
############ PLC EXAM 2 ############
###### Created by Anthony Asilo #####
########## November 2020 ##########
#### https://github.com/pillared ####
###################################
###################################*/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <time.h>

#define ARR_SIZE 10
#define loop_SIZE 100000

void declare_staticArr(){
   static int staticArr[ARR_SIZE];
}
void declare_stackArr(){
   int stackArr[ARR_SIZE];
}
void declare_heapArr(){
   int *heapArr = ( int* ) malloc ( ARR_SIZE * sizeof(int) );
   free(heapArr);
}
int main(int argc, char *argv[]){
   int i = 0;
   double time;
   clock_t begin;
   clock_t end;

   begin = clock();
   while(i < loop_SIZE){
      declare_staticArr();
      i++;
   }end = clock();
   time = (double)(end - begin) / CLOCKS_PER_SEC;
   printf("Time for static: %f seconds\n",time);

   i = 0;
   begin = clock();
   while(i < loop_SIZE){
      declare_stackArr();
      i++;
   }end = clock();
   time = (double)(end - begin) / CLOCKS_PER_SEC;
   printf("Time for stack: %f seconds\n",time);

   i = 0;
   begin = clock();
   while(i < loop_SIZE){
      declare_heapArr();
      i++;
   }end = clock();
   time = (double)(end - begin) / CLOCKS_PER_SEC;
   printf("Time for heap: %f seconds\n ",time);
   return 1;
}
```

QUESTION 6

```
/*
WHILE STATMENT

EBNF:
<whilestmt> -> “while” “(” <boolstmt> “)”  “{”   <block>   “}”
<block> -> {<stmt>}
*/
public static boolean whilestmt(String s){
   x = getNextToken(s)
   if(x == whilest){
      x = getNextToken(s);
      if(x == leftparenthasis){
         x = getNextToken(s);
         if(x == boolstmt){
            x = getNextToken(s);
            if(x == rightparenthasis){
               x = getNextToken(s);
               if(x == opencurl){
                  x = getNextToken(s);
                  if(x == opencurl){
                     x = getNextToken(s);
                     if(x == blockstmt)(){
                        x = getNextToken(s);
                        if(x == closecurl){
                           return true;
                        }else{
                           return false;
                        }
                     }else{
                        return false;
                     }
                  }else{
                     return false;
                  }
               }else{
                  return false;
               }
            }else{
               return false;
            }
         }else{
            return false;
         }
      }else{
         return false;
      }
```

```
   }else{
      return false;
   }
}


/*
IF STATEMENT

EBNF:
<ifstmt> -> “if” “(”  <boolstmt>   “)”  [ <stmt> | “{” <block> “}” ]
{ [ “else”  “if” “(”  <boolstmt>   “)”   [ <stmt> | “{” <block> “}” ] ] }
( [ “else” [ <stmt> | “{” <block> “}” ] )
<block> -> {<stmt>}
*/
//assume getNextToken pops string stack based on spaces.
public static boolean ifstmt(String s){
   x = getNextToken(s);
   if(x == ifst){
      x = getNextToken(s);
      if(x == leftparenthasis){
         x = getNextToken(s);
         if(x == boolstmt){
            x = getNextToken(s);
            if(x == rightparenthasis){
               x = getNextToken(s);
               //check if stmt or opencurl
               if(x == stmt || x == opencurl){
                  //if open curl check for close
                  if(x == opencurl){
                     x = getNextToken(s);
                     if(x == block){
                        x = getNextToken(s);
                        if(x == closecurl){
                           if(s != " " || s != ""){
                              check_for_else_or_elseif_stmt(s);
                           }else{ return true; }
                        }else{ return false; }
                     }else{ return false; }
                  }else if(s != " " || s != ""){
                     check_for_else_or_elseif_stmt(s);
                  }else{ return true; }
               }else{ return false; }
            }else{ return false; }
         }else{ return false; }
      }else{ return false; }
```

```
   }else{ return false; }
}
public static boolean check_for_else_or_elseif_stmt(String s){
   x = getNextToken(s);
   if(x == else_stmt){
      x = getNextToken(s);
      if(x == if_stmt){
         //concatenates popped with string and rechecks if statment
         x = x + " " + s;
         return if_stmt(s);
      }else if (x == stmt || x == opencurl){
         x = getNextToken(s);
            if(x == block){
               x = getNextToken(s);
               if(x == closecurl){
                  if(s != " " || s != ""){
                     ifstmt(s);
                  }else{ return true; }
               }else{ return false; }
            }else{ return false; }
         return true;
      }else {return false;}
   }else{ return false;}
}


/*
ASSIGNMENT STATEMENT EBNF:

EBNF:
<assignstmt> -> <var> “=” [ <var>{ “[” <mem> “]” } | <const> | <func> | “null”]
*/
public static boolean assignstmt(String s){
   x = getNextToken(s);
   if(x == var){
      x = getNextToken(s);
      if(x == equalsign){
         x = getNextToken(s);
         if(x == var || x == var[mem || x == const || x == func || x == null]){
            return true;
         }else { return false; }
      }else { return false; }
   }else { return false; }
}
```

```
/*
MATH STATEMENT EBNF:

EBNF:
<stmt> ->  <var> <op> [ <stmt> | <var> ]
<op> -> [ ‘+’ | ‘-’ | ‘*’ |  ‘/’ | ‘%’ ]
<var> -> [ <letter> | <num> ]
<letter> -> [ A-Z | a-z ] { <let> }{ <num> }
<num> -> [ 0-9 ] { <num> }
*/
//assume getNextToken pops string stack based on spaces.
public static boolean mathexpr(String s){
   x = getNextToken(s);
   if(x == var){
      x = getNextToken(s);
      if(x == op){
         x = getNextToken(s);
         if(x == math_expr){
            return math_expr(x + " " + s);
         }else if(x == var){
            return true;
         }else { return false; }
      }else { return false; }
   }else { return false; }
}
```

QUESTION 8

```
$var = 0;
sub returnfunction {
    return $var;
}
sub dynamicscopingfunction{
    local $var = 1;
    return returnfunction();
}
sub staticscopingfunction{
   my $var = 1;
   return returnfunciton()
}
print "Static Scope: " + returnfunction() ."\n";
print "Dynamic Scope: " + dynamicscopingfunction() . “\n”;
print “Another Static Scope: ” + staticscopingfunction() . “\n”;
```