Entry for Rod's Variables (2105) challenge

It's got too big to put it on a forum. Actually, I tried to do a tokenizer so it figures out all things including numbers and strings. But after figuring things out it outputs only things that relevant to this task (variables, subs/user functions, labels).
Uses keyword list etc from Rod's entry.
Tested on biggest program I've ever seen in LB - freeform404.bas ;)

```
'variable challenge
'tsh73, Jan 2015

global t.maxTokens, t.name, t.type, tp.line, tp.stmnt, tp.num
global curTokenNum, curTokNum, nLines, curStmntNum, contLine


[nonUsedLabel] dummy=1:dummy2=dummy 'non-used line with a (:)
unDimmed(3)=1   'example of un-dimmed array

gosub [setLists]

'fname$="vars01.bas"
'fname$="vars02.bas"
'fname$="test.bas"
filedialog "Select file to process";chr$(0);"open", "*.bas", fname$
if fname$="" then print "No file selected - bye": end
print "Processing "; fname$

t.maxTokens=100000
t.name = 0
t.type = 1
dim token$(t.maxTokens, 1)
tp.line=0
tp.stmnt=1
tp.num=2
dim tokenPos(t.maxTokens, 2)
curTokenNum=0   'global
curTokNum=0     'in a statement (that is between (:))

verbose = 0 '1

open fname$ for input as #1
nLines=1
isContinuation=0
print "Reading parsing line by line..."
```

```
t0=time$("ms")
while not(eof(#1))
    nLines=nLines+1
    line input #1, aLine$
    if verbose then print ">"; aLine$
    curStmntNum=1
    curTokNum=0

    if isContinuation then
        if contLine=0 then contLine = nLines-1
        isContinuation=0
    else
        contLine=0
    end if

    while 1
        scan
        'reading a line splitting it to tokens
        '1) skip all starting spaces
        aLine$=remStartSpaces$(aLine$)
        'exit if nothing left
        if aLine$="" then exit while
        'print ":"; aLine$
        '2) check special sequences
        ''comment
        if left$(aLine$,1)="'" then 'skip line as comment
            if verbose then print "   comment skipped"
            exit while
        end if
        '3) recognize / read / cut token
        select case
        '3.1) check special sequences
            '[label]
        case left$(aLine$,1)="["
            label$=upto$(aLine$, "]")+"]"
            aLine$=after$(aLine$, "]")
            if verbose then print "   label: ";label$
            call storeToken label$, "lbl"
        '3.2) check special sequences
            '"string"
        case left$(aLine$,1)=qq$  'string
            aLine$=mid$(aLine$,2)   'cut left (")
            aString$=qq$+upto$(aLine$,qq$)+qq$
            aLine$=after$(aLine$, qq$)
            if verbose then print "   string: "; aString$
            call storeToken aString$, "str"
```

```
        case left$(aLine$,1)="_" and instr(varChars$, mid$(
aLine$,2,1))<>0 'winConst
            token$=left$(aLine$,1)
            aLine$=mid$(aLine$,2)
            while instr(varChars$;"_", left$(aLine$,1))<>0
                token$=token$+left$(aLine$,1)
                aLine$=mid$(aLine$,2)
                if aLine$="" then exit while
            wend
            if verbose then print "   Windows constant: ";token$
            call storeToken token$, "winConst"

        case left$(aLine$,1)="_"
            aLine$=mid$(aLine$,2)   'cut token
            if verbose then print "   continuation char"
            'call storeToken "_", "_" 'do not store?
             isContinuation=1

        case left$(aLine$,1)=":"
            aLine$=mid$(aLine$,2)   'cut token
            if verbose then print "   operator separator"
            call storeToken ":", ":"
            curStmntNum=curStmntNum+1
            curTokNum=0

        case left$(aLine$,1)="|"
            aLine$=mid$(aLine$,2)   'cut token
            if verbose then print "   menu separator"
            call storeToken "|", "|"

        case left$(aLine$,1)=","
            aLine$=mid$(aLine$,2)   'cut token
            if verbose then print "   parameter separator"
            call storeToken ",", ","

        case left$(aLine$,1)=";"
            aLine$=mid$(aLine$,2)   'cut token
            if verbose then print "   concatenation"
            call storeToken ";", ";"

        case left$(aLine$,1)="="
            aLine$=mid$(aLine$,2)   'cut token
            if verbose then print "   assignment or equal"
            call storeToken "=", "="
```

```
        case left$(aLine$,1)="("
            aLine$=mid$(aLine$,2)   'cut token
            if verbose then print "   opening ("
            call storeToken "(", "("
        case left$(aLine$,1)=")"
            aLine$=mid$(aLine$,2)   'cut token
            if verbose then print "   closing )"
            call storeToken ")", ")"

        case left$(aLine$,1)="#"  'handle
            token$=left$(aLine$,1)
            aLine$=mid$(aLine$,2)
            while instr(varChars$, left$(aLine$,1))<>0
                token$=token$+left$(aLine$,1)
                aLine$=mid$(aLine$,2)
                if aLine$="" then exit while
            wend
            if verbose then print "   handle: ";token$
            call storeToken token$, "hndl"

        case instr(firstVarChars$, left$(aLine$,1))<>0
'name (var, arr, sub, func) or keyword
            token$=left$(aLine$,1)
            aLine$=mid$(aLine$,2)
            while instr(varChars$, left$(aLine$,1))<>0 _
                or (left$(aLine$,1)="_" and instr(varChars$, mid$(
aLine$,2,1))<>0 )
                token$=token$+left$(aLine$,1)
                aLine$=mid$(aLine$,2)
                if aLine$="" then exit while
            wend
            select case
            case instr(
comlist$;typlist$;opelist$, " ";lower$(token$);" ")<>0
                if verbose then print "   keyword: ";token$
                call storeToken token$, "kwrd"
                'if REM
                if lower$(token$) = "rem" then 'comment
                    if verbose then print "   comment skipped"
                    exit while
                end if

            case else
                if verbose then print
"name (var, array, sub, or func): ";token$
                call storeToken token$, "name"
```

```
            end select

        case instr(firstNumChars$, left$(aLine$,1))<>0 'number?
        notANumber=0
        select case
            case instr(digits$, left$(aLine$,1))<>0
                token$=left$(aLine$,1)
                aLine$=mid$(aLine$,2)
            case instr("-.", left$(aLine$,1))<>0 and IsNumber(
left$(aLine$,2))<>0
                token$=left$(aLine$,2)
                aLine$=mid$(aLine$,3)
            case left$(aLine$,2)="-." and IsNumber(left$(aLine$,3))<>0
                token$=left$(aLine$,3)
                aLine$=mid$(aLine$,4)
            case else
                notANumber=1
            end select
            if notANumber=0 then
            'read rest of a number
                while IsNumber(token$)<>0
                    token$=token$+left$(aLine$,1)
                    aLine$=mid$(aLine$,2)
                    if aLine$="" then exit while
                wend
                if IsNumber(token$)=0 then
                    'one char extra
                    aLine$=right$(token$, 1)+aLine$
                    token$=left$(token$, len(token$)-1)
                end if
                if verbose then print "number: ";token$
                call storeToken token$, "num"
            else    'should be single "-" (or not compiles)
                token$=left$(aLine$,1)
                aLine$=mid$(aLine$,2)
                if verbose then print "   operator ";token$
                call storeToken token$, "op"
            end if


'should be moved after "numbers" so "-1" does not process as "-" "1"
        case instr("<>+-*/^", left$(aLine$,1))<>0
            token$=left$(aLine$,1)
            aLine$=mid$(aLine$,2)   'cut token
            if verbose then print "   operator ";token$
            call storeToken token$, "op"
```

```
        case else
                token$=left$(aLine$,1)
                aLine$=mid$(aLine$,2)
                if verbose then print "??char: ";token$
                call storeToken token$, "???"
        end select
    wend
wend
close #1
print "----------------"
print "nLines=",nLines
print "numTokens=";curTokenNum

print time$("ms")-t0;" ms"
t0=time$("ms")

goto [SkipPrint]
for i = 1 to curTokenNum
    print tokenPos(i, tp.line);" ";_
      tokenPos(i, tp.stmnt);" ";_
      tokenPos(i, tp.num),_
      token$(i, t.type),_
      token$(i, t.name)
next
[SkipPrint]

'second pass along token$ array, check for next "(" - to tell function
s/arrays
Print "second pass along token$ array..."
i=0
while i <= curTokenNum
    i=i+1
    if token$(i, t.type)="name" then
        if i+1<=curTokenNum then
            if token$(i+1, t.type)="(" and _
                tokenPos(i, tp.line) = tokenPos(i+1, tp.line) and _
                tokenPos(i, tp.stmnt) = tokenPos(i+1, tp.stmnt) then
                    if instr(funlist$, " ";lower$(token$(i, t.name));
" ")<>0 then
                        token$(i, t.type)="buildInFunc"
                    else
                        token$(i, t.type)="UDForArray"
                    end if
            end if
        end if
```

```
    end if
wend
print time$("ms")-t0;" ms"
t0=time$("ms")

'third pass, check for functions/subs/arrays
Print "third pass along token$ array..."
i=0
while i <= curTokenNum
    i=i+1
    if tokenPos(i, tp.num) = 1 then     'first token on a stmnt
        select case
        case lower$(token$(i, t.name)) ="dim" or  lower$(token$(i,
 t.name)) ="redim"
            curLine = tokenPos(i, tp.line)
            curStmnt = tokenPos(i, tp.stmnt)
            while i+1<=curTokenNum
                if curLine <> tokenPos(i+1, tp.line) _
                    or curStmnt <> tokenPos(i+1, tp.stmnt) then
exit while
                i=i+1
                if token$(i, t.type)="UDForArray" then token$(i,
 t.type)="dimmedArray"
                'mark all other instances
                for j = 1 to curTokenNum
                    if token$(j, t.name)=token$(i, t.name) and _
                        token$(j, t.type)="UDForArray" then
 token$(j, t.type)="dimmedArray"
                next
            wend

        case  lower$(token$(i, t.name)) ="sub" or  lower$(token$(i,
 t.name)) ="call"
            i=i+1
            token$(i, t.type)="sub"

        case  lower$(token$(i, t.name)) ="function"
            i=i+1
            token$(i, t.type)="UDF"
            'mark all other instances
            for j = 1 to curTokenNum
                if token$(j, t.name)=token$(i, t.name) and _
                    token$(j, t.type)="UDForArray" then token$(j,
 t.type)="UDF"
            next
        end select
```

```
    end if
wend
print time$("ms")-t0;" ms"
t0=time$("ms")

'all other "UDForArray" are undimmed arrays
Print "last pass along token$ array..."
for j = 1 to curTokenNum
    if token$(j, t.type)="UDForArray" then token$(j, t.type)=
"unDimmedArray"
next

goto [SkipPrint2]
print "----------------"
for i = 1 to curTokenNum
    print tokenPos(i, tp.line);" ";_
      tokenPos(i, tp.stmnt);" ";_
      tokenPos(i, tp.num),_
      token$(i, t.type),_
      token$(i, t.name)
next

print "----------------"

[SkipPrint2]

print time$("ms")-t0;" ms"
t0=time$("ms")

'count variables
Print "Counting ..."

un.typeName=0
un.count=1
dim uniqueName$(curTokenNum,1)

aIndex$ = ""
'aIndex would be in a form ######|word|, where ###### index in a()
    'You can guess ###### restricts max len to 999999
aLen=0
'POSSIBLE TYPES
'(    )    ,    :    ;    ???    =    buildInFunc    dimmedArray    hn
dl    kwrd    lbl    name    num    op    str    sub    UDF    unDimme
dArray
types2Skip$=
"(    )    ,    :    ;    =    |    num    op    str    buildInFunc    kw
```

```
rd "
for i = 1 to curTokenNum
    w$=token$(i, t.type);":";token$(i, t.name)
    if instr(types2Skip$, token$(i, t.type))<>0 then [skipToken]
    toFind$="|"+w$+"|"
    pos=instr(aIndex$, toFind$)
    if pos =0 then
        'add it in array
        aLen = aLen+1
        uniqueName$(aLen, un.typeName)=w$
        uniqueName$(aLen, un.count)="1" 'first time
        aIndex$=aIndex$+using("######",aLen)+toFind$
    else
        'get index for a word. FAST.
        j = val(mid$(aIndex$, pos-6,6))
        cnt = val(uniqueName$(j, un.count))
        uniqueName$(j, un.count)=str$(cnt+1)
    end if
[skipToken]
next

print time$("ms")-t0;" ms"
t0=time$("ms")

Print "sorting ..."
sort uniqueName$(),1,aLen,0

print time$("ms")-t0;" ms"

print "nDiffWords", aLen
print "----------------"
print "Counter", "Type", "Name"
print "-----------------------------------------"
for i = 1 to aLen
    print uniqueName$(i, un.count), word$(uniqueName$(i,
 un.typeName),1,":"), _
        word$(uniqueName$(i, un.typeName),2,":")
next

print "-over-----------"

end
'----------------------
[setLists]
    qq$ = chr$(34)    '(")
```

```
    digits$="1234567890"
    letters$=""
    for i=asc("A") to asc("Z")
        letters$=letters$+chr$(i)
    next
    letters$=letters$+lower$(letters$)
    firstVarChars$=letters$
    varChars$=letters$+digits$+".$"
    firstNumChars$=digits$+".-"

    'from Rod's solution
    'command list
    comlist$=
" xor while wend wait until unloadbmp trace to titlebar timer then tex
teditor "
    comlist$=comlist$+
"textbox sub stylebits struct stopmidi stop step statictext sort selec
t seek "
    comlist$=comlist$+
"scan run return resume restore rem redim readjoystick read randomize
"
    comlist$=comlist$+
"radiobutton put prompt printerdialog print popupmenu playwave playmid
i "
    comlist$=comlist$+
"password out or open oncomerror notice nomainwin next name mod menu "
    comlist$=comlist$+
"maphandle mainwin lprint loop loadbmp listbox line let kill input if
"
    comlist$=comlist$+
"groupbox graphicbox goto gosub global gettrim get function for fontdi
alog "
    comlist$=comlist$+
"files filedialog field exit error end else dump do dim data cursor co
nfirm "
    comlist$=comlist$+
"combobox colordialog cls close checkbox case callfn calldll callback
call "
    comlist$=comlist$+"button bmpsave bmpbutton beep as and "

    'type operators
    typlist$=
" word void ushort ulong short ptr none long dword double boolean "
    'command operators
    opelist$=
" window text random output graphics dll dialog byref binary append "
```

```
    opelist$=opelist$+
"horizscrollbar vertscrollbar on off min max window_nf window_popup gr
aphics_fs graphics_nsb graphics_fs_nsb graphics_nf_nsb "
    opelist$=opelist$+
"text_fs text_nsb text_nsb_ins dialog_modal dialog_nf dialog_nf_modal
dialog_fs dialog_nf_fs dialog_popup "
    opelist$=opelist$+
"yellow brown red darkred pink darkpink blue darkblue green darkgreen
"
    opelist$=opelist$+
"cyan darkcyan white black lightgray darkgray buttonface "

    'function list
    funlist$=
" word$ winstring val using  upper$ txcount trim$ time$ tan tab str$ s
qr "
    funlist$=funlist$+
"space$ sin rnd rmdir right$ not mkdir min midipos mid$ max lower$ log
 "
    funlist$=funlist$+
"lof loc len left$ int instr inputto$ input$ inp hwnd hexdec hbmp exp
"
    funlist$=funlist$+
"eval eval$ eof dechex$ date$ cos chr$ atn asn asc acs abs "
    funlist$=funlist$+"upto$ after$ afterlast$ endswith remchar$ "
return
'---------------------

sub storeToken tkn$, type$
    curTokenNum=curTokenNum+1
    curTokNum=curTokNum+1
    token$(curTokenNum, t.name)=tkn$
    token$(curTokenNum, t.type)=type$
    tokenPos(curTokenNum, tp.line)=iif(contLine<>0, contLine, nLines)
    tokenPos(curTokenNum, tp.stmnt)=curStmntNum
    tokenPos(curTokenNum, tp.num) = curTokNum
end sub

'---------------------
function remStartSpaces$(aLine$)
    whiteSpaces$=chr$(9)+" "
    for i = 1 to len(aLine$)
        c$=mid$(aLine$,i,1)
        if instr(whiteSpaces$, c$)=0 then
            remStartSpaces$=mid$(aLine$,i)
            exit function
```

```
        end if
    next
end function

'------------------------------------------------
function IsNumber(input$)
'checks input$ for being valid number. Returns 1 if yes, 0 otherwise.
    IsNumber = 0
    'check sign
    ns = eatUp(input$, "+-")
    if ns>1 then exit function 'with False
    'now, digits
    n1 = eatUp(input$, "0123456789")
    'could be decimal point
    nd = eatUp(input$, ".")
    if nd>1 then  exit function
    'then again, digits
    n2 = eatUp(input$, "0123456789")
    if n1+n2<1 then  exit function
    'now, exponent
    ne = eatUp(input$, "e")
    if ne<>0 then   'we have exponent
        if ne>1 then  exit function
        'check sign
        ns = eatUp(input$, "+-")
        if ns>1 then  exit function
        'now, digits
        n1 = eatUp(input$, "0123456789")
        if n1<1 then  exit function
    end if

    if input$="" then IsNumber = 1: exit function
    'else we have leftovers - over with False
end function

function eatUp(byRef input$, chars2eat$)
'trims all leading chars from  input$ that match chars in chars2eat$
'return count of trimmed characters
    count = 0
    while len(input$)>0
        if instr( chars2eat$, left$( input$,1))<>0 then
            input$ = mid$(input$,2)
            count = count +1
        else
            exit while
        end if
```

```
    wend
    eatUp = count
end function

function iif(test, valYes, valNo)
    iif =  valNo
    if test then iif =  valYes
end function
```