Subject: Re: quick testing of string variables
Posted by rivers on Tue, 23 Apr 1996 07:00:00 GMT
View Forum Message <> Reply to Message

In article <moninger-2304960900010001@zirkle.fsl.noaa.gov>, moninger@fsl.noaa.gov (Bill Moninger) writes:
> I have an array called station_name, dimensioned (6,n). Each item is a
> string 6 characters long. I would like to quickly test station_name
> against a particular string variable, find_this_station, another string of
> dimension 6.
>
> Is there any way to do this without using loops?
>
> If I have to use loops, does anyone have a tip on the fastest way to do so?
>
> Is there are better way to configure the array station_name to make such
> tests (against a particular station name) faster?

Here is how to do it:

```
IDL> a = ['s1', 's2', 's3', 's4', 's5', 's6']
IDL> t = where(a eq 's4')
IDL> print, t
         3
```

The where() function finds the index number of the string in the array which matches your test string. It returns -1 if there is no match.

Mark Rivers (312) 702-2279 (office)
CARS (312) 702-9951 (secretary)
Univ. of Chicago (312) 702-5454 (FAX)
5640 S. Ellis Ave. (708) 922-0499 (home)
Chicago, IL 60637 rivers@cars3.uchicago.edu (Internet)

Subject: Re: quick testing of string variables
Posted by meron on Wed, 24 Apr 1996 07:00:00 GMT
View Forum Message <> Reply to Message

In article <4lm1as$61f@news1.ucsd.edu>, David Foster <foster@bial1.ucsd.edu> writes:
> moninger@fsl.noaa.gov (Bill Moninger) wrote:
>>
>> I have an array called station_name, dimensioned (6,n). Each item is a
>> string 6 characters long. I would like to quickly test station_name
>> against a particular string variable, find_this_station, another string of
>> dimension 6.
>>

>
> I don't quite get from your message the dimensions of your
> second string variable. Is it of LENGTH 6?
>
> When you need to compare a STRARR(N) against a single string (or more
> generally any array against a scalar) use the WHERE function:
>
> indices = WHERE(station_name eq find_this_station)
>
> Read the online help on the WHERE() function...it's extremely useful.
>
> Also useful are UNIQ(), TOTAL(), SORT(), REFORM(), REVERSE()
> and SHIFT().
>

You can also use a function I wrote, STRMATCH, which allows for
comparisons using partial names (and other interesting things). Since
it is calling other functions in my library, I'm attaching a package
of six routines: CAST, DEFAULT, TYPE, STREQ, STRMATCH, STRPARSE. See
below

*** CAST ***

Function Cast, x, low, high

```
;+
; NAME:
; CAST
; PURPOSE:
; Generalized type casting.  Converts all variables whose type code is
; out of the range [LOW,HIGH] into this range.
; CATEGORY:
; Type conversion
; CALLING SEQUENCE:
; Result = CAST( X, [LOW [,HIGH]])
; INPUTS:
;    X
; Numerical, arbitrary, or a character representation of a number(s).
;    LOW
; Number representing a type code, range (1:9).  If greater than 9, it is
; set to 9.  If less then 1, or not given, it is set to 1.
; OPTIONAL INPUT PARAMETERS:
;    HIGH
; Type code, same as LOW.  Default value is 9.  If provided and less then
; LOW, it is set to LOW.
```

```
; KEYWORD PARAMETERS:
; None.
; OUTPUTS:
; If the type of X is < LOW, CAST returns X converted to type LOW.
; If the type of X is > HIGH, CAST returns X converted to type HIGH.
; Otherwise CAST returns X.
; OPTIONAL OUTPUT PARAMETERS:
; None.
; COMMON BLOCKS:
; None.
; SIDE EFFECTS:
; None.
; RESTRICTIONS:
; 1)  An attempt to convert a string which is NOT a character
;     representation of a number into a numeric type will yield error.
; 2)  X cannot be a structure (but can be a structure element).
; 3)  The value 8 for either LOW or HIGH is not allowed (since it
;     corresponds to structure type).
; PROCEDURE:
; Identifies the type of X, and if out of the range given by [LOW,HIGH]
; calls the proper conversion routine using the system routine
; CALL_FUNCTION.  Also uses TYPE from MIDL.
; MODIFICATION HISTORY:
; Created 25-DEC-1991 by Mati Meron.
; Modified 15-JUN-1995 by Mati Meron to accept the new DOUBLECOMPLEX type.
;-

    on_error, 1
    conv = ['nada', 'byte', 'fix', 'long', 'float', 'double', 'complex', $
     'string', 'nonap', 'dcomplex']
    if n_elements(low)  eq 0 then ilo = 1 else ilo = 1   > fix(low)  < 9
    if n_elements(high) eq 0 then ihi = 9 else ihi = ilo > fix(high) < 9

    ityp = Type(x)
    if ilo eq 8 or ihi eq 8 or ityp eq 8 or ityp eq 0 then $
    message, 'Can''t do that!' else $
    if ityp lt ilo then return, call_function(conv(ilo),x) else $
    if ityp gt ihi then return, call_function(conv(ihi),x) else return, x

end



                        *** DEFAULT ***

Function Default, x, y, strict = strit, dtype = deft, low = lot, high = hit

;+
```

```
; NAME:
; DEFAULT
; PURPOSE:
; Provides an automatic default value for nondefined parameters.
; CATEGORY:
; Programming.
; CALLING SEQUENCE:
; Result = DEFAULT( X, Y [, keywords])
; INPUTS:
;    X, Y
; Arbitrary, at least one needs to be defined.
; OPTIONAL INPUT PARAMETERS:
; None.
; KEYWORD PARAMETERS:
;    /STRICT
; Switch.  If set, X is considered defined only if it is of the same type
; as Y.
;    /DTYPE
; Switch.  If set, the result will be typecast into the type of Y.
; Explicit settings for LOW and/or HIGH (see below) override DTYPE.
;    LOW
; Numeric value between 1 to 9 (8 is excluded). If given, the result is
; of type >= LOW.
;    HIGH
; Numeric value between 1 to 9 (8 is excluded). If given, the result is
; of type <= HIGH.
; OUTPUTS:
; X if it is defined, otherwise Y.
; OPTIONAL OUTPUT PARAMETERS:
; None.
; COMMON BLOCKS:
; None.
; SIDE EFFECTS:
; None.
; RESTRICTIONS:
; All type casting is bypassed if the result is of type 8 (STRUCTURE).
; PROCEDURE:
; Uses the functions CAST and TYPE from MIDL.
; MODIFICATION HISTORY:
; Created 15-JUL-1991 by Mati Meron.
; Modified 15-NOV-1993 by Mati Meron.  The keyword TYPE has been replaced
; by STRICT.  Added keywords DTYPE, LOW and HIGH.
;-

   on_error, 1
   xtyp = Type(x)
   ytyp = Type(y)
```

```
    if not (xtyp eq 0 or keyword_set(strit)) then atyp = xtyp else $
    if ytyp ne 0 then atyp = ytyp else message,'Insufficient data!'

    if xtyp eq atyp then res = x else res = y

    if keyword_set(deft) then begin
 if n_elements(lot) eq 0 then lot = ytyp
 if n_elements(hit) eq 0 then hit = ytyp
    end

    if atyp eq 8 then return, res else return, Cast(res,lot,hit)
end



                        *** STREQ ***

Function Streq, str1, str2, len, caseon = cas, warn = wn

;+
; NAME:
; STREQ
; PURPOSE:
; Compares for equality the first LEN characters of STR1, STR2.
; If LEN is 0, or absent, the whole strings are compared.
; CATEGORY:
; String Processing
; CALLING SEQUENCE:
; Result = STREQ( STR1, STR2 [,LEN] [, keywords])
; INPUTS:
;    STR1, STR2
; character strings, mandatory.
; OPTIONAL INPUT PARAMETERS:
;    LEN
; Number of characters to compare.  Default is 0, translating to a full
; comparison.
; KEYWORD PARAMETERS:
;    /CASEON
; Switch. If set the comparison is case sensitive. Default is ignore case.
;    /WARN
; Switch.  If set, a warning is issued whenever STR1 or STR2 is not a
; character variable.  Default is no warning.
; OUTPUTS:
; 1b for equal, 0b for nonequal.
; OPTIONAL OUTPUT PARAMETERS:
; None.
; COMMON BLOCKS:
; None.
```

```
; SIDE EFFECTS:
; None.
; RESTRICTIONS:
; None.
; PROCEDURE:
; Straightforward.  Using DEFAULT and TYPE from MIDL.
; MODIFICATION HISTORY:
; Created 15-JUL-1991 by Mati Meron.
;-

   if Type(str1) ne 7 or Type(str2) ne 7 then begin
 if keyword_set(wn) then message, 'Not a string!', /continue
 return, 0b
   endif

   dlen = Default(len,0)
   if dlen eq 0 then dlen = max([strlen(str1),strlen(str2)])
   if not keyword_set(cas) then begin
 dum1 = strupcase(str1)
 dum2 = strupcase(str2)
   endif else begin
 dum1 = str1
 dum2 = str2
   endelse

   return, strmid(dum1,0,dlen) eq strmid(dum2,0,dlen)
end



                        *** STRMATCH ***

Function StrMatch, str, list, len, caseon = cas, all = all

;+
; NAME:
; STRMATCH
; PURPOSE:
; Compares the string STR with the strings in the array LIST.  Comparison
; is done for the first LEN characters, or all of them if LEN is 0.  If a
;  match is found, STR is replaced by the full string from the list (or
; if the keyword /ALL is set, by an array containing all the matching
; strings).
; CATEGORY:
; String Processing
; CALLING SEQUENCE:
; Result = STRMATCH( STR, LIST [, LEN] [, keywords])
; INPUTS:
```

```
;    STR
; Character string.
;    LIST
; Character array.
; OPTIONAL INPUT PARAMETERS:
;    LEN
; The number of characters to compare.  Default is full comparison.
; KEYWORD PARAMETERS:
;    /CASEON
; Switch. If set the comparison is case sensitive. Default is ignore case.
;    /ALL
; Switch.  If set, returns the indices of all the matching elements.
; OUTPUTS:
; Returns the index of the first match, or -1l if no match is found.
; Optionally (see keyword ALL above) returns all the matching indices.
; OPTIONAL OUTPUT PARAMETERS:
; None.
; COMMON BLOCKS:
; None.
; SIDE EFFECTS:
; None other then the substitution in STR.
; RESTRICTIONS:
; None.
; PROCEDURE:
; Uses the function STREQ from MIDL.
; MODIFICATION HISTORY:
; Created 15-JUL-1991 by Mati Meron.
; Modified 20-NOV-1993 by Mati Meron.  Added keyword ALL.
;-

   match = where(Streq(str,list,len,caseon = cas), nmatch)
   if not keyword_set(all) then match = match(0)
   if nmatch gt 0 then str = list(match)

   return, match
end



                        *** STRPARSE ***

Function StrParse, line, delim, list

;+
; NAME:
; STRPARSE
; PURPOSE:
; Parses the string LINE using the characters in DELIM as delimiters.
```

```
; Puts individual pieces into consecutive locations in LIST.
; CATEGORY:
; String Processing
; CALLING SEQUENCE:
; Result = STRPARSE( LINE, DELIM [, LIST])
; INPUTS:
;    LINE
; Character string.
;    DELIM
; Character string.  Each Character of DELIM is used as a delimiter.
; OPTIONAL INPUT PARAMETERS:
; None.
; KEYWORD PARAMETERS:
; None.
; OUTPUTS:
; Returns the number of pieces found minus one i.e. the index of the last
; element of LIST if LIST is provided.  If LINE is a null string or not a
; string, the function returns -1l.
; OPTIONAL OUTPUT PARAMETERS:
;    LIST
; Character array.  If name is provided, the pieces of LINE resulting
; from the parsing process are returned in consecutive locations in LIST.
; COMMON BLOCKS:
; None.
; SIDE EFFECTS:
; None.
; RESTRICTIONS:
; None.
; PROCEDURE:
; Straightforward.  Using the function TYPE from MIDL.
; MODIFICATION HISTORY:
; Created 15-JUL-1991 by Mati Meron.
;-

    if Type(line) ne 7 then return, -1l
    index = -1l
    list = ''
    len = strlen(line)
    for i = 0l, len - 1 do begin
 if strpos(delim,strmid(line,i,1)) ne -1 then index = [index,i]
    endfor
    index = [index,len]
    for i = 0l, n_elements(index) - 2 do begin
 list = [list,strmid(line,index(i) + 1,index(i+1) - index(i) -1)]
    endfor
    inlist = where(list ne '',items)
    if items ne 0 then list = list(inlist) else list = list([0])
```

```
    return, long(items - 1)
end
```

*** TYPE ***

Function Type, x

```
;+
; NAME:
; TYPE
; PURPOSE:
; Finds the type class of a variable.
; CATEGORY:
; Programming.
; CALLING SEQUENCE:
; Result = TYPE(X)
; INPUTS:
;    X
; Arbitrary, doesn't even need to be defined.
; OPTIONAL INPUT PARAMETERS:
; None.
; KEYWORD PARAMETERS:
; None.
; OUTPUTS:
; Returns the type of X as a long integer, in the (0,9) range.
; OPTIONAL OUTPUT PARAMETERS:
; None.
; COMMON BLOCKS:
; None.
; SIDE EFFECTS:
; None.
; RESTRICTIONS:
; None.
; PROCEDURE:
; Extracts information from the SIZE function.
; MODIFICATION HISTORY:
; Created 15-JUL-1991 by Mati Meron.
;-

    dum = size(x)
    return, dum(dum(0) + 1)
end
```

Mati Meron | "When you argue with a fool,
meron@cars.uchicago.edu | chances are he is doing just the same"

Subject: Re: quick testing of string variables
Posted by David Foster on Wed, 24 Apr 1996 07:00:00 GMT
View Forum Message <> Reply to Message

moninger@fsl.noaa.gov (Bill Moninger) wrote:
>
> I have an array called station_name, dimensioned (6,n). Each item is a
> string 6 characters long. I would like to quickly test station_name
> against a particular string variable, find_this_station, another string of
> dimension 6.
>

I don't quite get from your message the dimensions of your
second string variable. Is it of LENGTH 6?

When you need to compare a STRARR(N) against a single string (or more
generally any array against a scalar) use the WHERE function:

```
 indices = WHERE( station_name eq find_this_station )
```

Read the online help on the WHERE() function...it's extremely useful.

Also useful are UNIQ(), TOTAL(), SORT(), REFORM(), REVERSE()
and SHIFT().

Dave Foster
foster@bial1.ucsd.edu

Subject: Re: quick testing of string variables
Posted by steinhh on Wed, 24 Apr 1996 07:00:00 GMT
View Forum Message <> Reply to Message

In article <moninger-2304960900010001@zirkle.fsl.noaa.gov>, moninger@fsl.noaa.gov (Bill Moninger) writes:
|>
|> I have an array called station_name, dimensioned (6,n). Each item is a
|> string 6 characters long. I would like to quickly test station_name
|> against a particular string variable, find_this_station, another string of
|> dimension 6.
|>
|> Is there any way to do this without using loops?
|>

|> If I have to use loops, does anyone have a tip on the fastest way to do so?
|>
|> Is there are better way to configure the array station_name to make such
|> tests (against a particular station name) faster?
|>

It depends a little on what you mean my "testing A against B".
If it means, (as your variable names cleverly point to)
that you want to find out which entry - if any - in station_name
equals the value of find_this_station, then there's hope (I think).
For clarity assume that the strings have length "len", and station_name
has dimension (n1,n2). len, n1 and n2 should be LONGs for safety.

```
byt = byte(station_name)  ; Should be a byte array, dimension (len,n1,n2)

str = string(reform(byt,len*n1*n2,/overwrite)) ; Make a looong string.

pos = strpos(str,find_this_station)

; Of course, if pos eq -1 then nothing was found.
; But if station (0,0) is "AB" and station (1,0) is "AC", they
; will end up in str as "ABAC....".
; If you are looking for station "BA" then you could be in trouble,
; but the "pos mod len ne 0" test discovers this.

if pos eq -1 then print,"No such station found" $
else begin
   ;; We'll have to be careful here:
   if pos mod len ne 0 then print,"Bummer -- wrong number - indecisive"
   x = (pos/len) mod n1
   y = (pos/len)/n1
end

;; You'll find your station in station_name(x,y)
```

The problem with AB + AC = ABAC can be fixed by adding a separator
character that's never used in the station names, e.g.:

```
byt = byte(station_name + '@')    ;; (and len = len + 1 !!)
```

This is of course at the cost of one extra string array copying
operation. You might never need it given specific station names.

Regards,

Subject: Re: quick testing of string variables
Posted by plugge on Wed, 24 Apr 1996 07:00:00 GMT
View Forum Message <> Reply to Message

In article <moninger-2304960900010001@zirkle.fsl.noaa.gov>, moninger@fsl.noaa.gov (Bill Moninger) writes:
|>I have an array called station_name, dimensioned (6,n). Each item is a
|>string 6 characters long. I would like to quickly test station_name
|>against a particular string variable, find_this_station, another string of
|>dimension 6.
|>
|>Is there any way to do this without using loops?
|>
|>If I have to use loops, does anyone have a tip on the fastest way to do so?
|>
|>Is there are better way to configure the array station_name to make such
|>tests (against a particular station name) faster?
|>
|>I shall appreciate any help anyone can provide.
|>
|>-Bill Moninger
|>--
|>Bill Moninger, NOAA/Forecast Systems Laboratory.
|>home: 303-494-1709, work: 303-497-6435
|>

Bill,
you can simply do the logical test `eq' to the two arrays:

eql_array = station_name eq find_this_station

The result is an array eql_array containing 1 or 0 according to the comparison
of the corresponding elements of the two arrays.

---------- ----------
Michael Plugge _ Fachhochschule Mannheim
Institute for Statistics / \ Hochschule fuer Technik und Gestaltung
and Image Processing / \ Email: plugge@biv7.sr.fh-mannheim.de
Speyerer Str. 4 / \ Tel: 0621 2926208
68163 Mannheim / \ --o /\
Germany / -- - \<,- / \/\
/ \ (_)/ (_) / \/\
---------- ----------