Subject: reading files into arrays
Posted by gromm on Fri, 19 Jul 1996 07:00:00 GMT
View Forum Message <> Reply to Message

Hi, I'm new at this. Can anyone tell me how to read a text file with a
bunch of numbers in it into an array without specifying how many numbers
are in the file. (I don't want to count them!)

Thanks

Gadi

Subject: Re: reading files into arrays
Posted by meron on Fri, 19 Jul 1996 07:00:00 GMT
View Forum Message <> Reply to Message

In article <4sojao$39b@netnews.upenn.edu>, gromm@blue.seas.upenn.edu (Gideon Z Romm)
writes:
>
> Hi, I'm new at this. Can anyone tell me how to read a text file with a
> bunch of numbers in it into an array without specifying how many numbers
> are in the file. (I don't want to count them!)
>
I wrote a function named READ_ASCII, awhile ago, which will do just
this, as long as the number of numeric entries per line is constant
(i.e. a table format). It'll even skip over test lines, so you can
include comments in the text file. The function is provided below,
and since it calls some other stuff from my library I included all
that's relevant (the other routines are valuable in their own right)

```
Function Read_ascii, filnam, buffer = buf, double = doub, $
   npoints = ncr, header = head, show = sho, skip = skip

;+
; NAME:
; READ_ASCII
; PURPOSE:
; Reads data from an ASCII file into an array.  It is assumed that the
; file contains columns of numbers, with the same number of entries in
; each row.  The numbers may be separated by commas, spaces and/or tabs.
; The file may contain a header.  The first line in which the first
; non-blank character is one of ".+-0123456789" will be considered the
; beginning of the data.  Text lines imbedded in the data are skipped.
; CATEGORY:
; Input/Output.
```

```
; CALLING SEQUENCE:
; Result = READ_ASCII( FILNAM [, optional keywords])
; INPUTS:
;    FILNAM
; Char. value, the name of the data file.  Default extension is '.DAT'.
; OPTIONAL INPUT PARAMETERS:
; None.
; KEYWORD PARAMETERS:
;    BUFFER
; Initial number of data rows.  Default is 256.  In any case the result
; Is trimmed to the actual number.
;    /DOUBLE
; If set, the data is input as DOUBLE.  Default is FLOAT.
;    /SHOW
; If set, the header (if one exists) will be printed to the screen.
;    NPOINTS
; Optional output, see below.
;    HEADER
; Optional output, see below.
;    SKIP
; Number of lines to skip at the beginning of the file.  This keyword can
; be used if the header of the file contains lines beginning with
; ".+-0123456789" which would otherwise be read as data.  Default is 0.
; OUTPUTS:
; Returns the data in a (NC,NR) floating (or double precision if DOUBLE
; is set) array, where NC, NR, are the numbers of columns and rows,
; respectively.  In case of error returns 0.
; OPTIONAL OUTPUT PARAMETERS:
;    NPOINTS
; The name of a 2-dim vector to receive the values of NC, NR (see above).
; Doesn't need to be defined prior to the call.  In case of an error
; returns [0,0]
;    HEADER
; The name of a character array to receive the header lines.  Doesn't
; need to be defined prior to the call.  In case of an error, or if no
; header exists, returns a zero length string.
; COMMON BLOCKS:
; None.
; SIDE EFFECTS:
; None.
; RESTRICTIONS:
; None.
; PROCEDURE:
; Straightforward.  Uses DEFAULT, STREQ and STRPARSE from MIDL.
; MODIFICATION HISTORY:
; Created 25-JAN-1992 by Mati Meron.
; Modified 25-MAY-1994 by Mati Meron.  Added buffering and DOUBLE option.
; Modified 14-FEB-1995 by Mark Rivers.  Added keyword SKIP.
```

```
; Modified 24-JUL-1995 by Mati Meron.  Removed a UNIX/VMS conflict in the
; file OPENR statement.
;-

   ncr = [0l,0l]
   bufsiz = (round(Default(buf,256)/256.) > 1l)*256l
   nrtem = bufsiz
   dtyp = 4 + keyword_set(doub)
   on_ioerror, file_no_good
   if Streq(!version.os,'vms',3) then begin
openr, datun, filnam, default = '.dat', /get_lun
   endif else openr, datun, filnam, /get_lun

   line = ''
   head = ''
   nc = 0l
   on_ioerror, data_no_good
   if n_elements(skip) ne 0 then for i = 0l, skip-1 do $
readf, datun, line, prompt = ''
   while nc eq 0 and not eof(datun) do begin
finf = fstat(datun)
readf, datun, line, prompt = ''
if line ne '' then begin
    bline = byte(strtrim(line,1))
    if Strparse(' .+-0123456789 ',string(bline(0))) eq 0 then begin
 head = [temporary(head),line]
 if keyword_set(sho) then print, line
    endif else nc = 1l + Strparse(line, ' , ')
endif
   endwhile
   head = transpose(head(n_elements(head) gt 1:*))

   if nc gt 0 then begin
point_lun, datun, finf.cur_ptr
datline = make_array(nc, type = dtyp)
data = make_array(nc,bufsiz, type = dtyp)
on_ioerror, next
nr = 0l
next:
while not eof(datun) do begin
    readf, datun, datline, prompt = ''
    data(*,nr) = datline
    nr = nr + 1l
    if nr eq nrtem then begin
 data = [[data],[make_array(nc,bufsiz, type = dtyp)]]
 nrtem = nrtem + bufsiz
    endif
endwhile
```

```
 data = data(*,0:nr-1)
 ncr = [nc,nr]
    endif else data = 0

    free_lun, datun
    return, data

    data_no_good:
    free_lun, datun
    file_no_good:
    print, !err_string
    return, 0

end
```

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
```

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
```

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
```

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
```

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"
