Flexible Tabular Data Input in Swatplus

Motivation

SWAT+ currently reads tabular data in a free form format where the data columns must follow the order of the data in the derived type data object that is being read in. This causes the following limitations:

1. If the order of the columns in the tabular data do not exactly match the order in the derived type object, then the data will be read in incorrectly and assigned to the incorrect swatplus variables.
2. If a new data column is added, it will invalidate older data sets.
3. New data can only be added to the end as the last column. This complicates collaboration.
4. The tabular data set cannot be annotated with comments which prevents data sets from being commented with meta data as needed.
5. Column header names are currently ignored during input reading. They have no relevance.
6. Reading of the table data always starts at line 1 of the input file.
7. If the derived data variable is declared as in integer but the data in the data table for that variable appears to be a fortran real value, a gfortran compiled executable will stop reading that particular line from that point on without a runtime error or warning. Any remaining values on that line would be ignored and the variables would retain their default values as defined in the derived data type.

Proposed Solution

The proposed solution is to change the way tabular data is read in so that:

1. Column order in the input tabular data does not matter. Column order could be rearranged if desired. This would be implemented by matching the column header name to the individual data element in the derived data type that is being read in using a select case statement in Fortran.
2. Omitted/missing columns in the tabular data do not cause an error but would use their default values as defined in the derived data type object.
3. Additional columns in the tabular data but not yet defined in the derived data type object and mapped are skipped and a warning printed out to the terminal and to the swatplus diagnostic file.
4. Annotation within the tabular data file are handled by ignoring the "#" character anything after it on the line being read in.

5. All blank lines are ignored.
6. If table read starts on line 1, the first line in the tabular data input file is read but ignored. This line is generally the title line and sometimes contains the version of the input editor that generated the file. It is a meta data line without the leading "#" character.
7. If an input field has a Fortran real type value but the code expects an integer, throw a runtime exception with `gfortran` compiled executable.
8. Suggested, but not required, have the column header name be the actual name of the variable in the derived data type. This will help with automated code documentation.
9. Adds the ability to start reading the table at a specified start row number.

For example, take the following swatplus derived data type that represents tabular data to be read in:

```
module landuse_data_module

type conservation_practice_table
   character(len=40) :: name = ""              !name of conservation practice
   real :: pfac = 1.0                          !usle p factor
   real :: sl_len_mx = 1.0             !m      !maximum slope length
end type conservation_practice_table
type (conservation_practice_table), dimension (:), allocatable :: cons_prac

...
end module
```

And the following abbreviated tabular input data table cons_practice.lum:

```
# cons_practice.lum: written by SWAT+ editor v2.3.3 on 2023-10-25 08:58
# This data is from NRCS Engineering field manual

name                    pfac       sl_len_mx  # description
up_down_slope          1.00000     121.00000  # Up_and_down_slope
#cross_slope            0.75000     121.00000  # Cross_slope_tillage
contour_farming        0.50000     121.00000  # Contour_tillage
strip_cros_slope       0.37000     121.00000  # Strip_cropping_cross_slope
# Contour tillage info
strip_contour          0.25000     121.00000  # Strip_cropping_contour
contour_1-2            0.30000     121.00000  # Contour_tillage_1-2%_slopes
contour_3-5            0.50000      91.00000  # Contour_tillage_3-5%_slopes
contour_6-8            0.50000      61.00000  # Contour_tillage_6-8%_slopes
contour_9-12           0.60000      36.00000  # Contour_tillage_9-12%_slopes
contour_13-16          0.70000      24.00000  # Contour_tillage_13-16%_slopes
 . . .
# the end
```

The proposed solution would read the input data file and:

1. Read but ignore the first line.
2. Ignore the "#" character and anything after the "#" character on the second line so essentially the whole line is ignored.
3. Ignore the third line since it is blank
4. Read the fourth line and split it into column headers.
5. Read the remainder of the lines as actual data but ignore blank lines and anything after the "#" character
6. If the number of data fields in a line does not equal the number of columns, the line will be ignored but a warning will be output to the terminal and to the diagnostics.txt file.
7. If there is a column that is in the input file but not in the defined data type, a warning will be output to the terminal and to the diagnostics.txt file and that column will be ignored.
8. If a column is missing in the input data file then the variable associated with that column will retain its default value as initialized in the derived data type.
9. Column order in the input file does not have to be the same as the order defined in the derived data type.

The example swatplus subroutine that reads this tabular data is called cons_prac_read and is below. The functions and subroutines that are highlighted in yellow in code below are in the utils.f90 module and should not need to be modified to read another flat table read. These are:

- `init()` - initializes lu_tbl vars to zero or null
- `get_num_data_lines` - returns the number of actual valid data lines in the input file.
- `get_header_columns` - opens the input file and determines the number of columns and column names.
- `get_row_fields` - gets the data fields in a row of valid data.
- `output_column_warning` - output warning on unrecognized column

The subroutines in utils.f90 also call other helper subroutines in utils.f90. These are:

- subroutine `left_of_delim` - remove comments from a line of data -
- subroutine `split_line` - split a line of data into fields
- function `to_lower` - convert a string to lower case.

The actual code to read in the data table is as follows. Calls to subroutines in utils.f90 that are highlighted in yellow. Code highlighted in green would need to be modified on a per case basis.

```
subroutine cons_prac_read

use input_file_module
use maximum_data_module
use landuse_data_module
use utils

implicit none

integer :: eof = 0      ! end of file
integer :: imax = 0     ! number of elements to be allocated
integer :: i

type(table_reader) :: lu_tbl
call lu_tbl%init(unit=107, file_name=in_lum%cons_prac_lum)

if (lu_tbl%file_exists .eqv. .false.) then
  allocate (cons_prac(0:0))
else
  imax = lu_tbl%get_num_data_lines()  !get number of valid data lines
  allocate (cons_prac(0:imax))

  if (imax /= 0) then
    ! get the column headers
    call lu_tbl%get_header_columns(eof)

if (eof == 0) then   ! proceed if not at the end of the file.
```

```
do
  ! next row of data
  call lu_tbl%get_row_fields(eof)
  if (eof /= 0) exit  ! exit if at the end of the file.

  ! all columns in row
  do i = 1, lu_tbl%get_col_count()
    select case (lu_tbl%header_cols(i))
      ! Assign data to cons_prac fields based on header column names
      case ("name")
          cons_prac(lu_tbl%get_row_idx())%name = trim(lu_tbl%row_field(i))
      case ("pfac")
          read(lu_tbl%row_field(i), *) cons_prac(lu_tbl%get_row_idx())%pfac
      case ("sl_len_mx")
          read(lu_tbl%row_field(i), *) cons_prac(lu_tbl%get_row_idx())%sl_len_mx
      case default
          ! Output warning for unknown column header
          call lu_tbl%output_column_warning(i)
     end select
    end do  ! column iteration
   end do  ! row iteration
  endif
 endif
endif

db_mx%cons_prac = imax

close(107)

return
end subroutine cons_prac_read
```

Swatplus Executable CMAKE Build Types

System	Distribution*	IDE	Compiler	Static	Dynamic
Microsoft	Windows 10&11	Visual Studio	IFX	no	yes
Microsoft	Windows 10&11	Vscode	Gfortran	yes	No
Linux	Debian	Vscode	IFX	yes	No
Linux	Debian	Vscode	Gfortran	yes	No
Linux	Arch	Vscode	IFX	yes	No
Linux	Arch	Vscode	Gfortran	no	yes
Linux	Fedora	Vscode	IFX	no	yes
Linux	Fedora	Vscode	Gfortran	no	yes
Codespaces	Debian	Vscode	IFX	yes	no
Codespaces	Debian	Vscode	Gfortran	yes	no

The distributions listed in the table above are the base distribution from which other distributions are derived. For example, Ubuntu is based on Debian. Linux Mint is based on Ubuntu which is based on Debian. All distributions based on Debian "should" behave the same in regards to the IFX and Gfortran compilers. However this has not been fully tessted. There are many distributions based on the Arch distribution and since they generally share the same Arch package managers, the IFX and Gfortran compilers "should" behave the same as the base Arch distribution. Again, this has not been fully tested.