HOMEWORK: Fortran

In this homework, write a Fortran program that demonstrates the use of modules, interfaces, types, and operator overloading.

Consider the problem of storing a sparse matrix, and the creation of operators to add two sparse matrices.

A 2D matrix of size $n \times m$ has mn elements a_{ij} where $i = 1, \cdots, m$ and $j = 1, \cdots, n$. This matrix is sparse if most of its elements are zero. Note that the matrix is not necessarily a square metrix.

We need two integer variables: `nr`$=m$ and `nc`$=n$ (not necessarily equal). We will not store the full matrix, since with $m = n = 10^6$, would require memory of 10^{12} floats, which is more than you have available on your desktop computers. Instead, we will simply track each non-zero element by its row, column and array value at that location.

This matrix will be defined by the Fortran 90 type:

```
type sparse
   integer :: nr, nc   ! number of rows and columns
   integer :: nel      ! number of nonzero elements

   ! row (ia) and column (ja) of each nonzero element
   integer, dimension(:), pointer :: ia, ja

   ! double precision: nel elements
   real(8), dimension(:), pointer :: a
end type sparse
```

To better understand the storage format, consider the matrix of real(8)

$$A = \begin{pmatrix} 0. & 2. & 5. & 0. & 0. & 3. \\ 0. & 0. & 1. & 2. & 0. & 4. \\ 1. & 5. & 0. & 0. & 2. & 1. \end{pmatrix}$$

Instead of storing 18 real(8) = 144 bytes, we will represent the matrix with three arrays. First, `a` is a 1D array whose size is `nel`, equal to the number of nonzero elements of A, written out in the same order of appearance as found in A (scanning along rows).

$$a = (2., 5., 3., 1., 2., 4., 1., 5., 2., 1.)$$

Second, the columns of each nonzero element of A are stored in `ja`, also of size `nel`:

$$ja = (2, 3, 6, 3, 4, 6, 1, 2, 5, 6)$$

Finally for each of the three rows of A, we identify the first nonzero element: row 1 gives 2, row two gives 1, and row 3 gives 1. Each of these elements is found in the array `a`, and its position is stored in array `ia`:

$$ia = (1, 4, 7, 11)$$

The size of `ia` is `nr+1`, and the last value of `ia` is `nel+1=11`. In this example, `nel=10`. So we have stored 10 real(8) in `a`, 10 integers in `ja` and 4 integers in `ia`, for a total of (10*8+10*4+4*4=136 bytes). We have already saved 8 bytes. Not much. But in this case, there are 10 nonzero elements in a matrix of size 18. In the homework, the matrix will be of size 10^7, and there will be many less nonzero elements. Thus the savings are much more significant.

Consider row i (where $i = 1, 2, 3$). The nonzero elements of A can be found in `a(ia(i))` to `a(ia(i+1)-1)`. Thus for row 2, `ia(2)=4` and `ia(3)=7`. Therefore, the nonzero elements of A are `a(4)=3`, `a(5)=4`, and `a(6)=6`. The columns of these three non-zero elements are `ja(4)=3`, `ja(5)=4`, and `ja(6)=6`.

In the case that i-th row is full of zeros, we have $ia(i) = ia(i+1)$ =the number of nonzeros up to the i-th row+1 Example for

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 6 & 0 & 9 \\ 0 & 0 & 0 \end{pmatrix}$$

$$a = (6, 9)$$

$$ja = (1, 3)$$

$$ia = (1, 1, 3, 3)$$

. There are more examples later.

In summary, to scan this array, scan each row: $i = 1, nr$. For each row i, calculate `ia(i)` and `ia(i+1)-1`. The nonzero columns are `ja(ia(i))` through `ja(ia(i+1)-1)`, and the nonzero elements of A are `A(i,ja(ia(i)))` through `A(i,ja(ia(i+1)-1))`.

When adding two arrays, for each row, identify the nonzero columns and add the corresponding elements of the two arrays. Be careful: while the two matrices to be added are both sparse, the nonzero elements are not necessarily in the same locations. You should check that the addition works properly on small 3×3 or 4×4 matrices with nonzero elements in different locations. .

Under non circumstances are you to store the full matrix. The matrix should on be stored in compressed format. However, you are allowed to allocate temporary storage for a single row or column if you feel that is necessary.

- Create a module called `sparse_mod`, that defines the sparse_matrix type which contains all necessary information on the matrix: number of rows and columns (nr, nc), number of nonzero elements, and pointers to arrays `a`, `ia`, `ja`.

- Allocate `nel` elements for `a` and `ja` and `nr+1` elements for ia.

- Create a matrix of size $1,000$ by $10,000$, and fill it with $10,000$ elements. Use a constructor for this purpose. Thus on average, each row contains 10 elements. You should use a random number generator to assign the columns for each row, and you should use a random number generator to fill the nonzero elements of the matrix with values between 0 and 10.

- Create an operator(+) in the module to add two sparse matrices. For example, consider the two matrices A and B:

$$A = \begin{pmatrix} 0 & 3 & 0 \\ 5 & -7 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & 3 & 0 \\ 2 & -6 & 0 \\ 4 & 0 & -3 \end{pmatrix}$$

  Notice that A and B matrices do not have zeros in the same elements. The sum is:

$$C = A + B = \begin{pmatrix} 0 & 6 & 0 \\ 7 & -13 & 0 \\ 4 & 0 & -2 \end{pmatrix}$$

- When adding two sparse matrices of equal dimension, it is therefore important to consider all the non-zero elements of both matrices.

- Create a main program. This main program should have a structure similar to:

```
program main
use sparse_mod
implicit none
type(sparse) :: mat1, mat2, mat3, matsum
....
! Initialize mat1, mat2, and mat3, which you can manually set their a, ia and ja.

...

    matsum = mat1 + mat2 + mat3

print results     ! (e.g. the nonzoer elements of rows 5 for mat1, mat2
                  !                  and matsum
end program main
```

- Test your program with the following example:

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 5 & -7 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & 3 & 0 \\ 0 & -6 & 0 \\ 4 & 0 & -3 \end{pmatrix}$$

$$C = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 8 & 0 \end{pmatrix}$$

$$A + B + C = D = \begin{pmatrix} 0 & 3 & 0 \\ 5 & -13 & 0 \\ 4 & 8 & -3 \end{pmatrix}$$

  Note that for matrix A,

$$a = (5, -7)$$

$$ja = (1, 2)$$

$$ia = (1, 1, 3, 3)$$

for matrix B

$$a = (3, -6, 4, -3)$$

$$ja = (2, 2, 1, 3)$$

$$ia = (1, 2, 3, 5)$$

for matrix C

$$a = (8)$$

$$ja = (2)$$

$$ia = (1, 1, 1, 2)$$

for matrix D

$$a = (3, 5, -13, 4, 8, -3)$$

$$ja = (2, 1, 2, 1, 2, 3)$$

$$ia = (1, 2, 4, 7)$$

Your code should past this test. Please print out your test result.

- Make sure you have NOTES/README/INSTALL files to describe ideas you may have had while you were working (NOTES), what the program does (README), and what to do to get the program to run (INSTALL).
- Make sure the code is documented (you are graded on this)