SOD-ORG

Peculiar requirements:

1. In multicomponent, or Z'> 1 structures, SOD-ORG identifies the disordered molecule by directly comparing the coordinates in the res files, without any tolerance. This means that the non-disordered molecule(s) must have EXACTLY the same coordinates in the major and minor component res files.
2. The axis file generated by neighcrys will have a space in the beginning of each line, except the first. SOD-ORG must have axis files without such spaces.
3. SOD-ORG writes res files with (some) coordinates in exponential notation, which makes them unreadable to other computer programs.
4. The res files cannot contain UNIT, FVAR or REM lines, in fact SOD-ORG tolerates only a few specific SHELX commands in the res files.
5. SOD-ORG seems to require very simple atom labels in the input res files, the labels must consist of the chemical element and followed by an integer, only. Unacceptable labels will cause SOD-ORG to crash on line 508.
6. The atoms in the res files given to SOD-ORG must be in the same order in both res files.
7. SOD-ORG needs multipoles in GDMA's raw output format, without force field labels. SOD-ORG internally calls gdmaneighcrys to make the neighcrys-compatible multipole file. You must make sure that the length unit used by gdmaneighcrys corresponds to the unit used by GDMA. The GEOM files are always in Ångström, but GDMA can use either bohr or Ångström.
8. SOD-ORG requires a bondlength file. SOD-ORG assumes this file is named 'bondlength'. However, if there is a file named 'cutoff' in the working dir, neighcrys will use that and never asks for the name of the bondlength file, which messed up things.
9. SOD-ORG runs neighcrys several times. For Z'=2 or multicomponent structures, it can happen that neighcrys orders the molecules differently in different runs, say for the original res file and its corresponding symmetry-reduced P1 structure. This will cause the geom and dma files to be numbered inconsistently and gdmaneighcrys will fail. SOD-ORG makes no attempt to make sure the files are numbered correctly.
10. You cannot re-run SOD-ORG if the files generated by a previous run are still in the working directory.

There are a couple of mistakes in the SOD-ORG source code that were previously acceptable, but will cause modern compilers to complain, or the program to crash.

1. The first is on line 572, where it writes to a file:

   ```
   WRITE (UNIT=unt,FMT='(A,F9.5,F9.5,F9.5)') cnamefull, allcoord(i,1)+ctrs(h,1),
   allcoord(i,2)+ctrs(h,2), allcoord(i,3)+ctrs(h,3)
   ```

   The problem is that SOD-ORG tries to write to the end of a file that already has an end-of-file character in the end. The solution is to tell the program to first take one step back, and then write:

   ```
   IF (i==1) BACKSPACE unt
   WRITE (UNIT=unt,FMT='(A,F9.5,F9.5,F9.5)') cnamefull, allcoord(i,1)+ctrs(h,1),
   allcoord(i,2)+ctrs(h,2), allcoord(i,3)+ctrs(h,3)
   ```

2. On line 475 it tries to do ALLOCATE (sinchk(scount)), but the variable sinchk is already allocated. A couple of lines below, the sinchk variable is set to '0', so this looks like a coding

mistake.
The problem seems to go away if you instead do:
IF(.NOT. ALLOCATED(sichk)) ALLOCATE (sichk(scount))

3. SOD-ORG never checks the exit codes or output files of neighcrys or gdmaneighrys, so it will print that these programs ran successfully even when they did not.
4. Fortran code should not have lines that exceed 80 characters in length, but SOD-ORG contains lines longer than that. It is therefore necessary to instruct the compiler to allow this. With the gfortran compiler, this is done with -ffree-line-length-none