D:\tmp\gs_tutorial_code\stephan_ruehl_fortran_soln_gms_tutorial\sort-merge_sr.f

```
      program sortmerge

      implicit none

      integer::n_reflection,n_rejected,n_unique,n_centric,n_new_unique
      real::rint,fob,sf,temp
      integer::rots_symm(3,3,6)
      real::tran_symm(3,6)
      integer::ind_h,ind_k,ind_l,temp_h,temp_k,temp_l
      integer::index_h(400000),index_k(400000),index_l(400000)
      real::fobs(400000),sigmaf(400000)
      integer::flag,centric,new_h,new_k,new_l,n_symm,centric_flag

      integer::a,b

C-------------------------------------------------------------------------------C
C-- Read reflection file                                                      --C
C-------------------------------------------------------------------------------C

      centric = 0
      n_reflection = 1
      n_rejected = 0
      rint = 0
      n_unique = 0
      n_symm=6
      n_centric = 0

      open(1,file='in')
      read(1,*)
      read(1,*)
      read(1,*)

      do a = 1 , n_symm
        read(1,*)rots_symm(1,1,a),rots_symm(1,2,a),rots_symm(1,3,a),
     +           rots_symm(2,1,a),rots_symm(2,2,a),rots_symm(2,3,a),
     +           rots_symm(3,1,a),rots_symm(3,2,a),rots_symm(3,3,a),
     +           tran_symm(1,a),tran_symm(2,a),tran_symm(3,a)
      end do

      do
        read ( 1 , * , end = 999 ) ind_h , ind_k , ind_l , fob , sf

C-------------------------------------------------------------------------------C
C-- Reorder indices to a standard form:                                       --C
C--                                                                           --C
C-- l positive, l=0 => k positive, l=k=0 => h positive                        --C
C-------------------------------------------------------------------------------C

        new_h = ind_h
        new_k = ind_k
        new_l = ind_l

        do a = 1 , n_symm

          temp_h = ind_h * rots_symm(1,1,a) + ind_k * rots_symm(2,1,a)
     +           + ind_l * rots_symm(3,1,a)
          temp_k = ind_h * rots_symm(1,2,a) + ind_k * rots_symm(2,2,a)
     +           + ind_l * rots_symm(3,2,a)
          temp_l = ind_h * rots_symm(1,3,a) + ind_k * rots_symm(2,3,a)
     +           + ind_l * rots_symm(3,3,a)

          if ( (temp_l .LT. 0) .OR. (temp_l .EQ. 0 .AND. temp_k .LT. 0)
     +         .OR. (temp_l .EQ. 0 .AND. temp_k .EQ. 0 .AND. temp_h
     +         .LT. 0) ) then
            temp_h = -temp_h
            temp_k = -temp_k
            temp_l = -temp_l
```

```
          end if

          if ( (temp_l .GT. new_l) .OR. (temp_l .EQ. new_l .AND. temp_k
     +        .GT. new_k) .OR. (temp_l .EQ. new_l .AND. temp_k .EQ. new_k
     +        .AND. temp_h .GT. new_h)) then
            new_h = temp_h
            new_k = temp_k
            new_l = temp_l
          end if
        end do

        index_h(n_reflection) = new_h
        index_k(n_reflection) = new_k
        index_l(n_reflection) = new_l

        fobs(n_reflection) = fob
        sigmaf(n_reflection) = sf

        n_reflection = n_reflection + 1

      end do

999   continue

      n_reflection = n_reflection - 1

C-------------------------------------------------------------------------------C
C-- Processing the data.                                                      --C
C-------------------------------------------------------------------------------C

      call sort_hkl(n_reflection,index_l,index_k,index_h,fobs,sigmaf)

      call calculate_rint_value(n_reflection,index_h,index_k,index_l,
     +                          fobs,sigmaf,rint)

      call merge_equivalent_reflections(n_reflection,index_h,index_k,
     +                                  index_l,fobs,sigmaf,n_unique)

C-------------------------------------------------------------------------------C
C-- Identify a reflection, that should be systematically absent and count    --C
C-- the number of centric reflections.                                        --C
C-------------------------------------------------------------------------------C

      n_new_unique = 0
      centric_flag = 0
      outer: do b = 1 , n_unique
        do a = 2 , n_symm

          temp_h = index_h(b) * rots_symm(1,1,a)
     +           + index_k(b) * rots_symm(2,1,a)
     +           + index_l(b) * rots_symm(3,1,a)
          temp_k = index_h(b) * rots_symm(1,2,a)
     +           + index_k(b) * rots_symm(2,2,a)
     +           + index_l(b) * rots_symm(3,2,a)
          temp_l = index_h(b) * rots_symm(1,3,a)
     +           + index_k(b) * rots_symm(2,3,a)
     +           + index_l(b) * rots_symm(3,3,a)

          ! Look for systematically absent reflections

          if ( temp_h .EQ. index_h(b) .AND. temp_k .EQ. index_k(b) .AND.
     +         temp_l .EQ. index_l(b) ) then
            temp = index_h(b) * tran_symm(1,a)
     +           + index_k(b) * tran_symm(2,a)
     +           + index_l(b) * tran_symm(3,a) + 999.5
            if ( temp - int(temp) .LT. 0.4 ) then
              n_rejected = n_rejected + 1
```

```
              print *,'Reflection ',index_h(b), index_k(b), index_l(b)
     +               ,' is systematically absent with I/sigma = '
     +               ,fobs(b)/sigmaf(b)
              centric_flag = 0
              cycle outer
            end if

          ! flag centric reflections

          elseif ( temp_h .EQ. -index_h(b) .AND.
     +             temp_k .EQ. -index_k(b) .AND.
     +             temp_l .EQ. -index_l(b) ) then
            centric_flag = 1
          end if

        end do
        n_centric = n_centric + centric_flag
        centric_flag = 0
        n_new_unique = n_new_unique +1
        index_h(n_new_unique) = index_h(b)
        index_k(n_new_unique) = index_k(b)
        index_l(n_new_unique) = index_l(b)
        fobs(n_new_unique) = fobs(b)
        sigmaf(n_new_unique) = sigmaf(b)
      end do outer

      print *,n_reflection,' Reflections were reduced to ',n_new_unique
     +       , ' Reflections '
      print *,'of which ',n_centric,' were centric.'
      print *,'R(int)=',rint
      print *,n_rejected,' Reflections were systematically absent.'

      close(1)

      end



C=======================================================================C
C                                                                       C
C Soubroutine for sorting hkl-indices                                   C
C                                                                       C
C This routine sorts the hkl indices in the following way:              C
C fastest changing index: index_3                                       C
C slowest changing index: index_1                                       C
C                                                                       C
C=======================================================================C

      subroutine sort_hkl(n_reflection,index_1,index_2,index_3,
     +                    fobs,phi)

      implicit none

      integer::n_reflection
      integer::index_1(n_reflection),index_2(n_reflection),
     +         index_3(n_reflection)
      real::fobs(n_reflection),phi(n_reflection)

C-----------------------------------------------------------------------C
C-- Local variables                                                   --C
C-----------------------------------------------------------------------C

      integer::t1,t2,position,start
      integer,allocatable::n_index_1(:),n_index_2(:),n_index_3(:)
      integer,allocatable::pointer_1(:),pointer_2(:),
     +                     pointer_3(:,:),pointer_4(:,:)
      integer::max_1,min_1,max_2,min_2,max_3,min_3
```

```
      integer::index_1_temp(n_reflection),index_2_temp(n_reflection),
     +         index_3_temp(n_reflection)
      real::fobs_temp(n_reflection),phi_temp(n_reflection)

      integer::i,j,k

C-------------------------------------------------------------------------C
C-- Sort on index_1                                                     --C
C-------------------------------------------------------------------------C

C-------------------------------------------------------------------------C
C-- Determine minimum and maximum index in index_1                      --C
C-------------------------------------------------------------------------C

      max_1 = -500
      min_1 = 500
      max_2 = -500
      min_2 = 500

      do i = 1 , n_reflection
        if ( index_1(i) .GT. max_1 ) then
          max_1 = index_1(i)
        end if
        if ( index_1(i) .LT. min_1 ) then
          min_1 = index_1(i)
        end if
        if ( index_2(i) .GT. max_2 ) then
          max_2 = index_2(i)
        end if
        if ( index_2(i) .LT. min_2 ) then
          min_2 = index_2(i)
        end if
      end do

      allocate (n_index_1(min_1:max_1))

      do i = min_1 , max_1 + 1
        n_index_1(i) = 0
      end do

C-------------------------------------------------------------------------C
C-- Scan list and count how often each index occurs                     --C
C-------------------------------------------------------------------------C

      do i = 1 , n_reflection
        t1 = index_1(i)
        n_index_1(t1) = n_index_1(t1) + 1
        index_1_temp(i) = index_1(i)
        index_2_temp(i) = index_2(i)
        index_3_temp(i) = index_3(i)
        fobs_temp(i) = fobs(i)
        phi_temp(i) = phi(i)
      end do

C-------------------------------------------------------------------------C
C-- Define an array with pointers to the positions in the sorted list   --C
C-------------------------------------------------------------------------C

      allocate (pointer_1(min_1:max_1))
      allocate (pointer_2(min_1:max_1))
      allocate (pointer_3(min_1:max_1,min_2:max_2))
      allocate (pointer_4(min_1:max_1,min_2:max_2))

      t1 = 0

      do i = min_1 , max_1
        t2 = n_index_1(i)
```

```
        n_index_1(i) = t1
        pointer_1(i) = t1 + 1
        t1 = t1 + t2
        pointer_2(i) = t1
      end do

C-------------------------------------------------------------------------------C
C-- Scan list again and put each index at the final position                  --C
C-------------------------------------------------------------------------------C

      do i = 1 , n_reflection
        t1 = index_1_temp(i)
        position = n_index_1(t1) + 1
        index_1(position) = index_1_temp(i)
        index_2(position) = index_2_temp(i)
        index_3(position) = index_3_temp(i)
        fobs(position) = fobs_temp(i)
        phi(position) = phi_temp(i)
        n_index_1(t1) = position
      end do

      deallocate (n_index_1)

C-------------------------------------------------------------------------------C
C-- Sort on index_2                                                           --C
C-------------------------------------------------------------------------------C

      start = 0

      do i = min_1 , max_1

        max_2 = -500
        min_2 = 500

        do j = pointer_1(i) , pointer_2(i)
          if ( index_2(j) .GT. max_2 ) then
            max_2 = index_2(j)
          end if
          if ( index_2(j) .LT. min_2 ) then
            min_2 = index_2(j)
          end if
        end do

        allocate (n_index_2(min_2:max_2))

        do j = min_2 , max_2 + 1
          n_index_2(j) = 0
        end do

        do j = pointer_1(i) , pointer_2(i)
          t1 = index_2(j)
          n_index_2(t1) = n_index_2(t1)+1
          index_1_temp(j) = index_1(j)
          index_2_temp(j) = index_2(j)
          index_3_temp(j) = index_3(j)
          fobs_temp(j) = fobs(j)
          phi_temp(j) = phi(j)
        end do

        t1 = start

        do j = min_2 , max_2
          t2 = n_index_2(j)
          n_index_2(j) = t1
          pointer_3(i,j) = t1 + 1
          t1 = t1 + t2
          pointer_4(i,j) = t1
```

```
        end do

        start = pointer_4(i,max_2)

        do j = pointer_1(i) , pointer_2(i)
          t1 = index_2_temp(j)
          position = n_index_2(t1) + 1
          index_1(position) = index_1_temp(j)
          index_2(position) = index_2_temp(j)
          index_3(position) = index_3_temp(j)
          fobs(position) = fobs_temp(j)
          phi(position) = phi_temp(j)
          n_index_2(t1) = position
        end do

        deallocate (n_index_2)

      end do

C-------------------------------------------------------------------------------C
C-- Sort on index_3                                                           --C
C-------------------------------------------------------------------------------C

      start = 0

      do i = min_1 , max_1

        max_2 = -500
        min_2 = 500

        do j = pointer_1(i) , pointer_2(i)
          if ( index_2(j) .GT. max_2 ) then
            max_2 = index_2(j)
          end if
          if ( index_2(j) .LT. min_2 ) then
            min_2 = index_2(j)
          end if
        end do

        do j = min_2 , max_2

          max_3 = -500
          min_3 = 500

          do k = pointer_3(i,j) , pointer_4(i,j)
            if ( index_3(k) .GT. max_3 ) then
              max_3 = index_3(k)
            end if
            if ( index_3(k) .LT. min_3 ) then
              min_3 = index_3(k)
            end if
          end do

          allocate (n_index_3(min_3:max_3))

          do k = min_3 , max_3 + 1
            n_index_3(k) = 0
          end do

          do k = pointer_3(i,j) , pointer_4(i,j)
            t1 = index_3(k)
            n_index_3(t1) = n_index_3(t1) + 1
            index_1_temp(k) = index_1(k)
            index_2_temp(k) = index_2(k)
            index_3_temp(k) = index_3(k)
            fobs_temp(k) = fobs(k)
            phi_temp(k) = phi(k)
```

```
          end do

          t1 = start

          do k = min_3,max_3
            t2 = n_index_3(k)
            n_index_3(k) = t1
            t1 = t1 + t2
          end do

          start = t1

          do k = pointer_3(i,j) , pointer_4(i,j)
            t1 = index_3_temp(k)
            position = n_index_3(t1) + 1
            index_1(position) = index_1_temp(k)
            index_2(position) = index_2_temp(k)
            index_3(position) = index_3_temp(k)
            fobs(position) = fobs_temp(k)
            phi(position) = phi_temp(k)
            n_index_3(t1) = position
          end do

          deallocate (n_index_3)

        end do

      end do

      deallocate (pointer_1,pointer_2,pointer_3,pointer_4)

      return
      end


C=====================================================================C
C                                                                     C
C Subroutine for merging equivalent reflections.                      C
C                                                                     C
C=====================================================================C

      subroutine merge_equivalent_reflections(nref,inh,ink,inl,fob,
     +                                        sig,uni)

      implicit none

      integer::nref,inh(nref),ink(nref),inl(nref)
      integer::uni
      real::fob(nref),sig(nref)

      real::fmean,sum_s,sum_1,fobs_new,sig_new

      integer::i,j,k

      i = 1
      j = 1
      uni = 0
      fmean = fob(i)
      sum_s = 1/sig(i)**2

      do
        if (inh(i) .EQ. inh(i+j) .AND. ink(i) .EQ. ink(i+j) .AND.
     +      inl(i) .EQ. inl(i+j) ) then
          fmean = fmean + fob(i+j)
          sum_s = sum_s + 1/sig(i+j)**2
          j = j + 1
          if ( ( i + j ) .LE. nref ) then
```

```
            cycle
          end if
        end if
        fobs_new = fmean / j
        sig_new = 1/sqrt(sum_s)
        uni = uni + 1
        inh(uni) = inh(i)
        ink(uni) = ink(i)
        inl(uni) = inl(i)
        fob(uni) = fobs_new
        sig(uni) = sig_new

        i = i + j
        j = 1
        if ( i + j .LE. nref ) then
          fmean = fob(i)
          sum_s = 1/sig(i)**2
          fobs_new = 0
          sig_new = 0
        else
          uni = uni + 1
          inh(uni) = inh(i)
          ink(uni) = ink(i)
          inl(uni) = inl(i)
          fob(uni) = fmean
          sig(uni) = 1/sqrt(sum_s)
          exit
        end if
      end do

      return
      end

C=======================================================================C
C                                                                       C
C Subroutine for calculating Rint values.                               C
C                                                                       C
C=======================================================================C

      subroutine calculate_rint_value(uni,indh,indk,indl,fob,sig,ri)

      implicit none

      integer::uni,indh(uni),indk(uni),indl(uni)
      real::fob(uni),sig(uni),ri

      real::sum_i,temp1,temp2

      integer::a,b,c

      temp1 = 0
      temp2 = 0
      a = 1
      do while ( a < uni + 1 )
        b = 1
        sum_i = fob(a)
        do
          if (indh(a) .EQ. indh(a+b) .AND. indk(a) .EQ. indk(a+b) .AND.
     +        indl(a) .EQ. indl(a+b) ) then
            sum_i = sum_i + fob(a+b)
            b = b + 1
            if (a+b .LE. uni) then
              cycle
            end if
          end if
          if (b.NE.1) then
            sum_i = sum_i / b
```

```
            do c = a , a + b - 1
              temp1 = temp1 + abs(fob(c) - sum_i)
              temp2 = temp2 + fob(c)
            end do
          end if
          a = a + b
          b = 1
          exit
        end do
      end do

      ri = temp1 / temp2

      return
      end
```