nag_reorder_vector (m01esc)

1. Purpose

nag_reorder_vector (m01esc) rearranges a vector of arbitrary type data objects into the order specified by a vector of indices.

2. Specification

```
#include <nag.h>
#include <nag_stddef.h>
#include <nagm01.h>

void nag_reorder_vector(Pointer vec[], size_t n, size_t size, ptrdiff_t stride,
     size_t indices[], NagError *fail)
```

3. Description

nag_reorder_vector uses a variant of list merging as described by Knuth (1973). The function rearranges a set of n data objects of arbitrary type, which are stored in an array at intervals of length **stride**, into the order specified by an array of indices.

4. Parameters

vec[ ]

Input: the array of objects to be rearranged.
Output: the objects rearranged according to array **indices**.

n

Input: the number, n, of objects to be rearranged.
Constraint: $\mathbf{n} \geq 0$.

size

Input: the size of each object to be rearranged.
Constraint: $\mathbf{size} \geq 1$.

stride

Input: the increment between data items in **vec** to be rearranged.

Note: if **stride** is positive, **vec** should point at the first data object; otherwise **vec** should point at the last data object.
Constraint: $|\mathbf{stride}| \geq \mathbf{size}$.

indices[**n**]

Input: the indices specifying the order in which the elements of vector are to be rearranged.

fail

The NAG error parameter, see the Essential Introduction to the NAG C Library.

5. Error Indications and Warnings

NE_INT_ARG_LT

On entry, **n** must not be less than 0: $\mathbf{n} = \langle value \rangle$.
On entry, **size** must not be less than 1: $\mathbf{size} = \langle value \rangle$.

NE_INT_ARG_GT

On entry, **n** must not be greater than $\langle value \rangle$: $\mathbf{n} = \langle value \rangle$.
On entry, $|\mathbf{stride}|$ must not be greater than $\langle value \rangle$: $|\mathbf{stride}| = \langle value \rangle$.
On entry, **size** must not be greater than $\langle value \rangle$: $\mathbf{size} = \langle value \rangle$.

These parameters are limited to an implementation-dependent size which is printed in the error message.

NE_2_INT_ARG_LT

On entry, $|\mathbf{stride}| = \langle value \rangle$ while $\mathbf{size} = \langle value \rangle$. These parameters must satisfy $|\mathbf{stride}| \geq \mathbf{size}$.

NE_BAD_RANK
Invalid **indices** vector.

NE_ALLOC_FAIL
Memory allocation failed.

6. Further Comments

The average time taken by the function is approximately proportional to **n**.

6.1. References

Knuth D E (1973) *The Art of Computer Programming (Vol 3, Sorting and Searching)* Addison-Wesley.

7. See Also

None.

8. Example

The example program.

8.1. Program Text

```
/* nag_reorder_vector(m01esc) Example Program
 *
 * Copyright 1990 Numerical Algorithms Group.
 *
 * Mark 2 revised, 1992.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nag_stddef.h>
#include <nagm01.h>

#ifdef NAG_PROTO
static Integer compare(const Pointer a, const Pointer b)
#else
     static Integer compare(a, b)
     Pointer a, b;
#endif
{
  double x = *((double *)a) - *((double *)b);
  return (x<0.0 ? -1 : (x==0.0 ? 0 : 1));
}

#define MMAX 20
#define NMAX 20

main()
{
  double a[MMAX][NMAX];
  size_t i, j, n, k, m, indices[MMAX];
  static NagError fail;

  fail.print = TRUE;
  /* Skip heading in data file */
  Vscanf("%*[^\n]");
  Vprintf("m01esc Example Program Results\n");
  Vscanf("%d%d%d", &m, &n, &k);
  if (m>=0 && m<=MMAX && n>=0 && n<=NMAX && k>=1 && k<=n)
    {
      for (i=0; i<m; ++i)
        for (j=0; j<n; ++j)
          Vscanf("%lf",&a[i][j]);
      m01dsc((Pointer) &a[0][k-1], m, (ptrdiff_t)(NMAX*sizeof(double)),
              compare, Nag_Ascending, indices, &fail);
      if (fail.code != NE_NOERROR)
        exit (EXIT_FAILURE);
      m01zac(indices, m, &fail);
      if (fail.code != NE_NOERROR)
        exit (EXIT_FAILURE);
      for (j=0; j<n; ++j)
        {
          m01esc((Pointer) &a[0][j], m, sizeof(double),
                 (ptrdiff_t)(NMAX*sizeof(double)), indices, &fail);
          if (fail.code != NE_NOERROR)
            exit (EXIT_FAILURE);
        }
      Vprintf ("\nMatrix with column %d sorted\n", k);
      for (i=0; i<m; ++i)
        {
          for (j=0; j<n; ++j)
            Vprintf("  %7.1f  ", a[i][j]);
          Vprintf("\n");
        }
      exit(EXIT_SUCCESS);
    }
  else
    {
      Vfprintf(stderr, "Data error: program terminated\n");
      exit(EXIT_FAILURE);
    }
}
```

8.2. Program Data

```
m01esc Example Program Data
12 3 1
6.0 5.0 4.0
5.0 2.0 1.0
2.0 4.0 9.0
4.0 9.0 6.0
4.0 9.0 5.0
4.0 1.0 2.0
3.0 4.0 1.0
2.0 4.0 6.0
1.0 6.0 4.0
9.0 3.0 2.0
6.0 2.0 5.0
4.0 9.0 6.0
```

8.3. Program Results

```
m01esc Example Program Results

Matrix with column 1 sorted
      1.0        6.0        4.0
      2.0        4.0        9.0
      2.0        4.0        6.0
      3.0        4.0        1.0
      4.0        9.0        6.0
      4.0        9.0        5.0
      4.0        1.0        2.0
      4.0        9.0        6.0
      5.0        2.0        1.0
      6.0        5.0        4.0
      6.0        2.0        5.0
      9.0        3.0        2.0
```