nag_chain_sort (m01cuc)

1. Purpose

nag_chain_sort (m01cuc) rearranges the links of a linked list into ascending or descending order of a specified data field of arbitrary type.

2. Specification

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

void nag_chain_sort(Pointer *base, ptrdiff_t offset,
     Integer (*compare) (const Pointer, const Pointer), Nag_SortOrder order,
     NagError *fail)
```

3. Description

nag_chain_sort uses a variant of list merging, as described in Knuth (1973). It uses a local stack to avoid the need for a flag bit in each pointer.

4. Parameters

base

Input: pointer to the pointer to the first structure of the linked list.
Output: the pointer to which **base** points is updated to point to the first element of the ordered list.

offset

Input: the offset within the structure of the pointer field which points to the next element of the linked list.
Note: this field in the last element of the linked list must have the value **NULL**.

compare

User-supplied function: this function compares the data fields of two elements of the list. Its arguments are pointers to the structure, therefore this function must allow for the offset of the data field in the structure (if it is not the first).
The function must return:

−1 if the first data field is less than the second,
0 if the first data field is equal to the second,
1 if the first data field is greater than the second.

order

Input: specifies whether the array will be sorted into ascending or descending order.
Constraint: **order** = **Nag_Ascending** or **Nag_Descending**.

fail

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

5. Error Indications and Warnings

NE_CH_LOOP

Too many elements in chain or chain in a loop.
The linked list may have become corrupted.

NE_BAD_PARAM

On entry, **order** had an illegal value.

6. Further Comments

The time taken by the function is approximately proportional to $n \log n$.

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 declares a structure containing a data field, a pointer to the next record and an index field. It generates an array of such structures assigning random data to the data field. The index field is randomly assigned a unique value. The pointer to next record fields are assigned to create a linked list in the order of the index field. It sorts this linked list into ascending order according to the value of the data field.

8.1. Program Text

```
/* nag_chain_sort(m01cuc) Example Program
 *
 * Copyright 1996 Numerical Algorithms Group.
 *
 * Mark 4, 1996.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nag_stddef.h>
#include <nagg05.h>
#include <nagm01.h>

struct recd
{
  double data;
  struct recd *next;
  Integer index;
};

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

main()
{
  struct recd vec[20];
  struct recd *base, *origbase, *address;
  size_t i, j, l[20], n;
  ptrdiff_t ptroffset;
  static NagError fail;

  ptroffset = (ptrdiff_t) (((char *) &(vec->next)) - ((char *) vec));
  g05cbc((Integer)0);
  n = 10;
  /* Set data field to random number between 0 and 5 */
  for (i=0; i<n; ++i)
    vec[i].data = (double) g05dyc((Integer)0,(Integer)5);
  /* Randomly set index fields from 0 to 9*/
  for (i=0; i<n; ++i)
    {
```

```
      j = (int)((i+1)*g05cac());
      /* j is less than or equal to i */
      (vec[i]).index = (vec[j]).index;
      (vec[j]).index = i;
    }
  /* Set next pointers to make linked list in index field order */
  for (i=0; i<n; ++i)
    l[(vec[i]).index] = i;
  base = origbase = &vec[l[0]];
  for (i=0; i<n-1; ++i)
    vec[l[i]].next = &vec[l[i+1]];
  vec[l[n-1]].next = NULL;
  /* Print Input Data */
  Vprintf("m01cuc Example Program Results\n");
  Vprintf("\nDATA\n\n");
  Vprintf("Matrix Order:\n");
  Vprintf("   Matrix Index      Linked List Index      Data\n");
  for (i=0; i<n; ++i)
    Vprintf("%10d%20ld%20.6f\n",i,vec[i].index,vec[i].data);
  Vprintf("Linked List Order:\n");
  Vprintf("   Matrix Index      Linked List Index      Data\n");
  address = base;
  while (address != NULL)
    {
      Vprintf("%10d%20ld%20.6f\n", (int)(address-base), (*address).index,
              (*address).data);
      address = (*address).next;
    }
  /* Sort the linked list on the data field */
  fail.print= TRUE;
  m01cuc((Pointer *)&base, ptroffset, compare, Nag_Ascending, &fail);
  if (fail.code != NE_NOERROR)
    exit (EXIT_FAILURE);
  /* Output results */
  Vprintf("\nRESULTS\n\n");
  /* The order in the input matrix is unchanged */
  Vprintf("Matrix Order:\n");
  Vprintf("   Matrix Index      Linked List Index      Data\n");
  for (i=0; i<n; ++i)
    Vprintf("%10d%20ld%20.6f\n",i,vec[i].index,vec[i].data);
  /* But the linked list pointers have been changed to reflect
     the ascending sort on the data field */
  Vprintf("Linked List Order:\n");
  Vprintf("   Matrix Index      Linked List Index      Data\n");
  address = base;
  while (address != NULL)
    {
      Vprintf("%10d%20ld%20.6f\n", (int)(address-origbase), (*address).index,
              (*address).data);
      address = (*address).next;
    }
  exit(EXIT_SUCCESS);
}
```

8.2. Program Data

None.

8.3. Program Results

```
m01cuc Example Program Results

DATA

Matrix Order:
   Matrix Index      Linked List Index      Data
          0                  0           4.000000
          1                  9           1.000000
          2                  8           2.000000
          3                  7           1.000000
          4                  2           5.000000
          5                  5           0.000000
          6                  3           1.000000
          7                  1           2.000000
          8                  6           0.000000
          9                  4           3.000000
Linked List Order:
   Matrix Index      Linked List Index      Data
          0                  0           4.000000
          7                  1           2.000000
          4                  2           5.000000
          6                  3           1.000000
          9                  4           3.000000
          5                  5           0.000000
          8                  6           0.000000
          3                  7           1.000000
          2                  8           2.000000
          1                  9           1.000000

RESULTS

Matrix Order:
   Matrix Index      Linked List Index      Data
          0                  0           4.000000
          1                  9           1.000000
          2                  8           2.000000
          3                  7           1.000000
          4                  2           5.000000
          5                  5           0.000000
          6                  3           1.000000
          7                  1           2.000000
          8                  6           0.000000
          9                  4           3.000000
Linked List Order:
   Matrix Index      Linked List Index      Data
          5                  5           0.000000
          8                  6           0.000000
          6                  3           1.000000
          3                  7           1.000000
          1                  9           1.000000
          7                  1           2.000000
          2                  8           2.000000
          9                  4           3.000000
          0                  0           4.000000
          4                  2           5.000000
```