

NAG Library Function Document

nag_search_char (m01ncc)

1 Purpose

nag_search_char (m01ncc) examines an ordered vector of null terminated strings and returns the index of the first value equal to the sought-after item. Character items are compared according to the ASCII collating sequence.

2 Specification

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

```
Integer nag_search_char (Nag_Boolean validate, const char *ch[], Integer m1,
    Integer m2, const char *item, NagError *fail)
```

3 Description

nag_search_char (m01ncc) is based on Professor Niklaus Wirth's implementation of the Binary Search algorithm (see Wirth (2004)), but with two modifications. First, if the sought-after item is less than the value of the first element of the array to be searched, -1 is returned. Second, if a value equal to the sought-after item is not found, the index of the immediate lower value is returned.

4 References

Wirth N (2004) *Algorithms and Data Structures* 35–36 Prentice Hall

5 Arguments

- 1: **validate** – Nag_Boolean *Input*
On entry: if **validate** is set to Nag_TRUE argument checking will be performed. If **validate** is set to Nag_FALSE nag_search_char (m01ncc) will be called without argument checking, which includes checking that array **ch** is sorted in ascending order and the function will return with **fail.code** = NE_NOERROR. See Section 8 for further details.
- 2: **ch**[**m2** + 1] – const char * *Input*
On entry: elements **m1** to **m2** contain null terminated strings to be searched.
Constraint: elements **m1** to **m2** of **ch** must be sorted in ascending order. The length of each element of **ch** must not exceed 255. Trailing space characters are ignored.
- 3: **m1** – Integer *Input*
On entry: the index of the first element of **ch** to be searched.
Constraint: **m1** \geq 0.
- 4: **m2** – Integer *Input*
On entry: the index of the last element of **ch** to be searched.
Constraint: **m2** \geq **m1**.
- 5: **item** – const char * *Input*
On entry: the sought-after item. Trailing space characters are ignored.

6: **fail** – NagError *

Input/Output

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

NE_CHAR_LEN_INVALID

On entry, the length of each element of **ch** must be ≤ 255 : maximum string length = $\langle value \rangle$.

NE_INT

On entry, **m1** = $\langle value \rangle$.
Constraint: **m1** ≥ 0 .

NE_INT_2

On entry, **m1** = $\langle value \rangle$, **m2** = $\langle value \rangle$.
Constraint: **m2** \geq **m1**.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

NE_NOT_INCREASING

On entry, **ch** must be sorted in ascending order: **ch** element $\langle value \rangle >$ element $\langle value \rangle$.

7 Accuracy

Not applicable.

8 Further Comments

The argument **validate** should be used with caution. Set it to Nag_FALSE only if you are confident that the other arguments are correct, in particular that array **ch** is in fact arranged in ascending order. If you wish to search the same array **ch** many times, you are recommended to set **validate** to Nag_TRUE on first call of nag_search_char (m01ncc) and to Nag_FALSE on subsequent calls, in order to minimize the amount of time spent checking **ch**, which may be significant if **ch** is large.

The time taken by nag_search_char (m01ncc) is $O(\log n)$, where $n = \mathbf{m2} - \mathbf{m1} + 1$, when **validate** = Nag_FALSE.

9 Example

This example reads a list of character data and sought-after items and performs the search for these items.

9.1 Program Text

```

/* nag_search_char (m01ncc) Example Program.
 *
 * Copyright 2008, Numerical Algorithms Group.
 *
 * Mark 9, 2009.
 */
/* Pre-processor includes */
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <nag.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagm01.h>

int main(int argc, char *argv[])
{
    FILE          *fpin, *fpout;
    /*Logical scalar and array declarations */
    Nag_Boolean validate;
    /*Integer scalar and array declarations */
    Integer       exit_status = 0;
    Integer       chlen, i, index, lench, m1, m2;
    /*Character scalar and array declarations */
    char          item[255], chtmp[255];
    char          **ch;
    NagError      fail;

    INIT_FAIL(fail);

    /* Check for command-line IO options */
    fpin = nag_example_file_io(argc, argv, "-data", NULL);
    fpout = nag_example_file_io(argc, argv, "-results", NULL);

    fprintf(fpout, "%s\n", "nag_search_char (m01ncc) Example Program Results");
    fprintf(fpout, "\n");
    fscanf(fpin, "%*[\n] ");
    fscanf(fpin, "%ld%*[\n]", &lench);
    if (!(ch = NAG_ALLOC(lench, char *)))
    {
        fprintf(fpout, "Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    /* Read in Reference Vector ch*/
    for (i = 0; i < lench; i++)
    {
        fscanf(fpin, "%s", chtmp);
        chlen = strlen(chtmp);
        if (!(ch[i] = NAG_ALLOC(chlen+1, char)))
        {
            fprintf(fpout, "Allocation failure\n");
            exit_status = -1;
            goto END;
        }
        strncpy(ch[i], chtmp, chlen+1);
    }
    fscanf(fpin, "%*[\n] ");
    /* Read items sought in the reference vector*/
    validate = Nag_TRUE;
    m1 = 0;
    m2 = lench-1;
    while (fscanf(fpin, "%s%*[\n] ", item) != EOF)
    {
        /*
         * nag_search_char (m01ncc)
         * Binary search in set of character data
         */
        index = nag_search_char(validate, ch, m1, m2, item, &fail);
    }
}

```

```

if (fail.code != NE_NOERROR)
{
    fprintf(fpout, "Error from nag_search_char (m01ncc).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
}
if (validate)
{
    /* Print the reference vector*/
    fprintf(fpout, "%s\n", "Reference Vector is:");
    for (i = 0; i < lench; i++)
    {
        fprintf(fpout, "%s%s", ch[i], (i+1)%10?" ":"\n");
    }
    fprintf(fpout, "\n");
    validate = Nag_FALSE;
}
fprintf(fpout, "\n");
fprintf(fpout, " Search for item %s returned index: %4ld\n", item,
        index);
}

END:
if (fpin != stdin) fclose(fpin);
if (fpout != stdout) fclose(fpout);
for (i = 0; i < lench; i++)
{
    if (ch[i]) NAG_FREE(ch[i]);
}
if (ch) NAG_FREE(ch);

return exit_status;
}

```

9.2 Program Data

```

nag_search_char (m01ncc) Example Program Data
10                                     : lench
a02aac a02abc a02acc c02adc
c02aec c05adc c05agc c05ajg
c05axc c05azc                         : ch
c02adc                                 : item 1
a01aac                                 : item 2
c04ayc                                 : item 3
d01nbc                                 : item 4

```

9.3 Program Results

nag_search_char (m01ncc) Example Program Results

Reference Vector is:

a02aac a02abc a02acc c02adc c02aec c05adc c05agc c05ajg c05axc c05azc

```

Search for item c02adc returned index:    3
Search for item a01aac returned index:   -1
Search for item c04ayc returned index:    4
Search for item d01nbc returned index:    9

```