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

```c
#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


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 9 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 ≥ 0.

4: m2 – Integer

   *Input*

   On entry: the index of the last element of ch to be searched.

   Constraint: m2 ≥ m1.
5: \textbf{item} – const char * \hspace{1cm} \textit{Input}
   
   \textit{On entry:} the sought-after item. Trailing space characters are ignored.

6: \textbf{fail} – NagError * \hspace{1cm} \textit{Input/Output}
   
   The NAG error argument (see Section 3.6 in the Essential Introduction).

6 \ Error Indicators and Warnings

\textbf{NE_ALLOC_FAIL}

Dynamic memory allocation failed. 
See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE_BAD_PARAM}

On entry, argument \textit{\langle value\rangle} had an illegal value.

\textbf{NE_CHAR_LEN_INVALID}

On entry, the length of each element of \textbf{ch} must be at most 255: maximum string length
= \textit{\langle value\rangle}.

\textbf{NE_INT}

On entry, \textit{m1 = \langle value\rangle}.
   
   Constraint: \textit{m1} \geq 0.

\textbf{NE_INT_2}

On entry, \textit{m1 = \langle value\rangle} and \textit{m2 = \langle value\rangle}.
   
   Constraint: \textit{m2} \geq \textit{m1}.

\textbf{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.

An unexpected error has been triggered by this function. Please contact NAG. 
See Section 3.6.6 in the Essential Introduction for further information.

\textbf{NE_NO_LICENCE}

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

\textbf{NE_NOT_INCREASING}

On entry, \textbf{ch} must be sorted in ascending order: \textbf{ch} element \textit{\langle value\rangle} > element \textit{\langle value\rangle}.

7 \ Accuracy

Not applicable.

8 \ Parallelism and Performance

Not applicable.
9 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 \(n = m2 - m1 + 1\), when validate = Nag_FALSE.

10 Example

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

10.1 Program Text

```c
/* nag_search_char (m01ncc) Example Program.  
 * Copyright 2014 Numerical Algorithms Group.  
 * Mark 9, 2009.  
 */
/* Pre-processor includes */
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagm01.h>

int main(void)
{
    /* 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);

    printf("nag_search_char (m01ncc) Example Program Results\n");
    printf("\n");
    #ifdef _WIN32
    scanf_s("%*[\n] ");
    #else
    scanf("%*[\n] ");
    #endif
    #ifdef _WIN32
    scanf_s("%NAG_FMT%*[\n]", &lench);
    #else
    scanf("%NAG_FMT%*[\n]", &lench);
    #endif
    if (!(ch = NAG_ALLOC(lench, char *)) )
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    /* Read in Reference Vector ch*/
    for (i = 0; i < lench; i++)
    {
        #ifdef _WIN32

        #endif
    }

END:
exit_status
return exit_status;
} /* End of main */
```

```c
/*
 * nag_search_char (m01ncc)
 * Binary search in set of character data
 */
index = nag_search_char(validate, (const char**)ch, m1, m2, item, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_search_char (m01ncc)\n", fail.message);
    exit_status = 1;
    goto END;
}
if (validate)
{
    /* Print the reference vector*/
    printf("%s\n", "Reference Vector is:");
    for (i = 0; i < lench; i++)
    {
        printf("%s", ch[i], (i+1)%10?" ":"
        printf("\n");
    validate = Nag_FALSE;
    }
    printf("\n");
    printf(" Search for item %s returned index: %4"NAG_IFMT", item, index);
}
END:
for (i = 0; i < lench; i++)
{
    NAG_FREE(ch[i]);
}
NAG_FREE(ch);
return exit_status;
}
```
10.2 Program Data

nag_search_char (m01ncc) Example Program Data

<table>
<thead>
<tr>
<th>lench</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
</tr>
<tr>
<td>a02aac a02abc a02acc c02adc</td>
</tr>
<tr>
<td>c02aec c05auc c05awc c05axc</td>
</tr>
<tr>
<td>c05ayc c05azc</td>
</tr>
<tr>
<td>ch</td>
</tr>
<tr>
<td>c02adc : item 1</td>
</tr>
<tr>
<td>a01aac : item 2</td>
</tr>
<tr>
<td>c04ayc : item 3</td>
</tr>
<tr>
<td>d01nbc : item 4</td>
</tr>
</tbody>
</table>

10.3 Program Results

nag_search_char (m01ncc) Example Program Results

Reference Vector is:
a02aac a02abc a02acc c02adc c02aec c05auc c05awc c05axc c05ayc 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