```
/*
 * This is a free program sample that may be reproduced in any form.
 * The author's information should be retained to preserve its identity.
 *
 * Date written: October 14, 2003
 * Written by: Peraphon Sophatsathit
 * Department of Mathematics, Faculty of Science, Chulalongkorn University.
 * email: Peraphon.S@chula.ac.th
 * http://pioneer.netserv.chula.ac.th/~sperapho
 *
 * Data Structures (2301263) classnote.
 * Description: This sample program demonstrates topological sort based
 *              reference (1).  Some details, however, were adopted
 *              from (2).
 * Input:       The input file contains the adjacency matrix (n X n) of
 *              the form:
 *              row 1:  0  1  1  1  0  0  0 ...
 *              row i:  ...
 *              row n:  0  0  0  0  0  1  0 ...
 * References: 1) Data Structures and Algorithm Analysis in C, Mark Allen
 *                Weiss, Second Edition, Addison-Wesley Longman, Inc., 1997.
 *              2) Introduction to Algorithm, Cormen T.H., Leiserson, C.E.,
 *                and Rivest, R.L., MIT Press, 1990.
 */

#include       <stdio.h>
#include       <stdlib.h>
#include       <string.h>

#define        rmode          "r"
#define        Normal         0
#define        Err_code1      1
#define        Err_code2      2
#define        Err_code3      3
#define        Err_code4      4
#define        PG_code1       11
#define        Illegal        -1
#define        Invalid_value  -1

/*
 * function prototype
 */
void    print_mat(int **, int *, int, int);
int     proc_loop(int **, int *, int, int);
int     topsort(int **, int *, int, int *);
int     find_zero(int *, int);

/*
 * main processing
 */
int
main(int argc, char **argv)
{
       FILE    *fp;
       char    buf[BUFSIZ+1];
       int     i, j;
       int     rows = 0, cols = 0;
       int     **matr, *indeg;
       int     rt_code = Normal;
       size_t  num;

       if (argc != 2)
       {
              printf("missing argument: %s  infile_name\n", argv[0]);
              return Err_code1;
       }
       if ((fp = fopen(argv[1], rmode)) == NULL)
       {
              printf("Unable to open input data file %s\n", argv[1]);
              return Err_code2;
       }
       while (fgets(buf, BUFSIZ, fp) != NULL)
              rows++;
```

```
        /*
         * rows = columns in the adjacency matrix.
         */
        cols = rows;
        if ((matr  = (int **)malloc(sizeof(int) * rows)) == NULL ||
            (indeg = (int *)malloc(sizeof(int) * rows)) == NULL)
        {
                printf("Out of memory\n");
                fclose(fp);
                return Err_code3;
        }
        /*
         * allocate one row (of cols elements per row) at a time.
         */
        num = sizeof(int) * cols;
        for (i = 0; i < rows; i++)
                if ((matr[i] = (int *)malloc(num)) == NULL)
                {
                        printf("Out of memory\n");
                        rt_code = Err_code3;
                        break;
                }
        rewind(fp);
        if (rt_code == Normal)
        {
                /*
                 * fill the adjacency matrix from the input file.
                 */
                for (i = 0; i < rows; i++)
                        for (j = 0; j < cols; j++)
                                (void)fscanf(fp, "%d", &matr[i][j]);
        }
        fclose(fp);

        proc_loop(matr, indeg, rows, cols);

        /*
         * in reverse order of allocation.
         */
        for (i = 0; i < rows; i++)
                free((void *)matr[i]);
        free(matr);
        free((void *)indeg);
        return rt_code;
}

/*
 * description: compute in-degree for all nodes.
 * Note that despite being 'row major', the order of processing
 *      is reversed because it only makes sense to count the
 *      in-degree columnwise.
 * input:  adjacency matrix, in-degree array, rows, and columns.
 * output: success/failure.
 */
int
proc_loop(int **mat, int *indeg, int rows, int cols)
{
        int     i, j, rt;
        int     *sorta;

        /*
         * allocate sorted output array.
         */
        if ((sorta = (int *)malloc(sizeof(int) * rows)) == NULL)
        {
                printf("Out of memory\n");
                return Err_code3;
        }
        for (i = 0; i < rows; i++)
                indeg[i] = 0;
```

```
        for (j = 0; j < cols; j++)
        {
                for (i = 0; i < rows; i++)
                {
                        if (mat[i][j] > 0)
                                indeg[j]++;
                }
        }
        print_mat(mat, indeg, rows, cols);
        rt = topsort(mat, indeg, rows, sorta);
        return rt;
}

/*
 * description: sort the network of nodes.
 * input:  adjacency matrix, in-degree array, no. of rows, and the sorted array.
 * output: succeeded or failed code.
 */
int
topsort(int **mat, int *indeg, int numv, int *sorted_array)
{
        int     nt, j, v;
        int     tally = 0;

        for (nt = 0; nt < numv; nt++)
        {
                v = find_zero(indeg, numv);
                if (v == Illegal)
                {
                        printf("Error: graph has a cycle\n");
                        return PG_code1;
                }
                /*
                 * add one to reflect the 'real' node number
                 */
                sorted_array[tally++] = v + 1;
                for (j = 0; j < numv; j++)
                {
                        /*
                         * skip itself
                         */
                        if (j == v)
                        {
                                indeg[v] = Invalid_value;
                                continue;
                        }
                        /*
                         * scan the adjacency matrix for all connected neighbors
                         */
                        if (mat[v][j] > 0)
                        {
                                (indeg[j])--;
                        }
                }
        }
        printf("\nsorted list:\n");
        for (j = 0; j < numv; j++)
                printf("%d   ", sorted_array[j]);
        printf("\n\n");
        return Normal;
}
```

```
/*
 * description: find the first node having in-degree equals to zero.
 * input:  indeg array and no. of rows.
 * output: the desired node or failed.
 */
int
find_zero(int *indeg, int rows)
{
	int	i, rt = Illegal;

	for (i = 0; i < rows; i++)
	{
		if (indeg[i] == Invalid_value)
			continue;
		if (indeg[i] == 0)
		{
			rt = i;
			break;
		}
	}
	return rt;
}

/*
 * description: print the adjacency matrix and in-degree array.
 * input:  adjacency matrix, in-degree array, rows, and columns.
 * output: printed adjacency matrix and in-degree array.
 */
void
print_mat(int **mat, int *indeg, int rows, int cols)
{
	int	i, j;

	printf("\nadjacency matrix:\n");
	for (i = 0; i < rows; i++)
	{
		for (j = 0; j < cols; j++)
			printf("%d   ", mat[i][j]);
		printf("\n");
	}
	printf("\nin-degree:\n");
	for (i = 0; i < rows; i++)
		printf("%d   ", indeg[i]);
	printf("\n\n");
}
```