```c
/*This is a program code to check what happens in the array declaration*/
#include <stdio.h>
#include <stdlib.h>              //need to be included

void message_error(char error_text[]);
double *d_vector(long size);          //return value is the pointer to double

double **d_matrix(long size_row, long size_column);     //return value is pointer
variable pointing to [a pointer variable pointing to double]
void free_d_matrix(double **x);
void free_d_vector(double *x);

int main(void)
{

     double *vector_test;   //pointer variable pointing to double
     double **matrix_test; //pointer variable pointing to [a pointer variable pointing to
double]

     int i, j, k;
     int size_vector = 5;    //size of vector
     int size_r = 3;  //size of row in matrix
     int size_c = 2;  //ise of column in matrix

     vector_test = d_vector(size_vector); //allocation of memory
     matrix_test = d_matrix(size_r, size_c); //allocation of memory

     for(i = 1; i <= size_vector; i++) {
          vector_test[i] = 1.0*i;
          printf("%lf\n", vector_test[i]);
     }
     printf("\n");
     for(j = 1; j <= size_r; j++) {
          for(k = 1; k <= size_c;k++) {
               matrix_test[j][k] = 1.0*j + 2.0*k;
               printf("%lf\t", matrix_test[j][k]);
          }
          printf("\n");
     }


     free_d_vector(vector_test);          //must release memory space!!
     free_d_matrix(matrix_test);          //must release memory space!!

  return 0;
}
```

```c
void message_error(char error_text[])
/*Standard error handler*/
{
     printf("There are some errors...\n");
     printf("%s\n", error_text);
     printf("...now existing to system...\n");
     exit(1);
}

double *d_vector(long size) //for generating double vector[1]...vector[size]
{
     double *x;         //pointer
     x = (double *) malloc((size_t) ((size + 1)*sizeof(double)));
     //allocation of memory space to stock (size + 1) double variables & malloc returns
the address of the top of the allocated memory space to x
     //the reason to prepare (size + 1) is just to use from x[1] to x[size]
     if(x == NULL) message_error("allocation failure in d_vector()");//if memory
allocation was failed, malloc returns NULL and x becomes NULL
     return x;     //return the address of the top of the allocated memory space to x
}

double **d_matrix(long size_row, long size_column) //for generating double matrix[1]
[1]...matrix[size_row][size_column]
{
     double **x;             //pointer to 'pointer valuable', valuable to stock the
address of the pointer valuable
     long i;
     long size_row_P = size_row + 1;  //technical (not necessary) statement just to start
from [1][1]
    long size_column_P = size_column + 1;  //technical (not necessary) statement just to
start from [1][1]

    x = (double **) malloc((size_t)  (size_row_P*sizeof(double *)));            //
allocation of memory spate to stock (size_row) pointer valuables to double, & malloc
returns the address of the top of the allocated memory space to x
    if(x == NULL) message_error("allocation failure in d_vector()");//if memory
allocation was failed, malloc returns NULL and x becomes NULL

    x[0] = (double *) malloc((size_t) (size_row_P*size_column_P*sizeof(double)));
    //allocation of memory scape to stock (size_row*zsize_column) doubles variables &
malloc returns the address of the top of the allocated memory space to x[0]
    //Note that x[0] (== *x) is the value (with type pointer to double) of the pointer
valuable , pointed by x

    if(x[0] == NULL) message_error("allocation failure in d_vector()"); //if memory
allocation was failed, malloc returns NULL and x becomes NULL

    for(i = 1; i < size_row_P; i++) x[i] = x[0] + i*size_column_P;  //operating on
pointer

    return x;

}

void free_d_matrix(double **x)
{
    free(x[0]);
    free(x);
}

void free_d_vector(double *x)
{
    free(x);
}
```