16 Allocating 1-dimensional and 2-dimensional arrays

It is easy to create a 1-dimensional array of doubles, say:

```
double * a = (double*) calloc ( n, sizeof ( double ) );
```

The `sizeof()` function, or pseudo-function since its argument is a C type, gives the number of bytes occupied. For example, `sizeof(int)` is 4, and `sizeof(int*)` is 8 on 64-bit machines. The above code allocates `n * sizeof(double)` or $8n$ bytes.

To allocate something which resembles a 2-dimensional array of doubles, the type should be

```
double ** a;
```

Construction is not obvious. Now

```
If a is to resemble an m x n array,

a   should be an array of m 'rows,' so
the correct allocation is
a = (double **)
  calloc ( m, sizeof ( double * ) );
```

Then each 'row' `a[i]` should be an array of n doubles.

```
a[i] = (double*) calloc ( n, sizeof ( double ) );
```

Putting these together,

```
double * make_vector ( int n )
{
  double * b = (double*) calloc (n, sizeof(double));
}

double ** quasi_2d ( int m, int n )
{
  double ** mat = (double **) calloc ( m, sizeof ( double* ) );
  int i;
  for (i=0; i<m; ++i)
  {
    mat[i] = (double *) calloc ( n, sizeof ( double ) );
  }
  return mat;
}
```

16.1 Revised matrix-by-vector program

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

        /*
         * This program is an improvement on
         * earlier versions which multiply a matrix
         * by a vector, first a 3x4 matrix by a 4-vector,
         * and the 6th programming assignment which work
         * with 2-dimensional arrays of fixed size, 10x10.
         *
         * This program uses calloc() first to construct
         * a vector of n doubles, and then to construct
         * a 'quasi-2-dimensional' array.  It inputs
         * and multiplies them and prints the product.
         *
         * Construction of the 'quasi-2-dimensional' array
         * is not obvious, but once it is done one can
         * take a 'quasi-matrix' stored in a variable a,
         * declared double ** a, and write a[i][j] as if
         * a was an ordinary 2-dimensional array.
         *
         * One other point is that the product routine in earlier
         * versions has been replaced by a product function which
         * returns the matrix by vector product.  The final
         * print statement calls and prints the product in one
         * statement.
         *
         * It is necessary for these routines to be passed
         * information about the dimensions (height and width
         * of matrices, size of vectors).
         */

double * make_vector ( int n )
{
  double * b = (double*) calloc (n, sizeof(double));
}

double ** quasi_2d ( int m, int n )
{
  double ** mat = (double **) calloc ( m, sizeof ( double* ) );
  int i;
  for (i=0; i<m; ++i)
  {
```

```
    mat[i] = (double *) calloc ( n, sizeof ( double ) );
  }
  return mat;
}


double * product( int m, int n, double ** a, double *b)
{
  int i,j;
  double sum;
  double * c = make_vector ( n );

  for (i=0; i<m; ++i)
  {
    sum = 0;
    for (j=0; j<n; ++j)
    { sum += a[i][j] * b[j] ; }
    c[i] = sum;
  }
  return c;
}

void print_matrix ( int m, int n, double ** a )
{
  int i, j;
  for ( i=0; i<m; ++i )
  {
    for ( j=0; j<n; ++j )
    { printf(" %8.3f", a[i][j]); }
    printf("\n");
  }
}

void print_vector ( int n, double *b )
{
  int j;
    for ( j=0; j<n; ++j )
    { printf(" %8.3f", b[j]); }
    printf("\n");
}

int main()
{
  double **a, *b, *c;
  int ell,m,n;
```

```
  int i,j;

  scanf ("%d %d", &ell, &m); //height and width of a

  a = quasi_2d ( ell, m );

  for (i=0; i<ell; i = i+1)
  for (j=0; j<m; j = j+1)
  { scanf("%lf", &( a[i][j] )) ; }


  scanf ("%d", &n); //height of b
                        // no check that height of b is width of a.
  b = make_vector ( n );
  for (j=0; j<n; j = j+1)
  {  scanf("%lf", &( b[j] )) ; }

        // a is equivalent to an ell by m matrix,
        // and b is a vector of height n,
        // and the product function assumes that b has
        // height m without checking.

  printf("matrix\n"); print_matrix (ell,m, a );

  printf("vector\n"); print_vector (n, b );

  printf("product \n");
  print_vector (ell, product ( ell,m, a, b ));
}
```