elements. The choice of one method over another is often a matter of personal preference. In applications involving numerical arrays, it is often easier to define the arrays in the conventional manner, thus avoiding any possible subtleties associated with initial memory assignments. The following example, however, illustrates the use of pointer notation to process multidimensional numerical arrays.

Example 10.22 Adding Two Tables of Numbers In Example 9.19 we developed a C program to calculate the sum of the corresponding elements of two tables of integers. That program required three separate, two-dimensional arrays, which were defined and processed in the conventional manner. Here is a variation of this program, in which each two-dimensional array is defined as a pointer to a set of one-dimensional integer arrays.

The complete program is shown below.

```
#include <stdio.h>

#define MAXCOLS  30

/* calculate the sum of the elements in two tables of integers */

/* each 2-dimensional array is processed as a pointer
   to a set of 1-dimensional integer arrays */

main()

{
    int nrows, ncols;

    /* pointer definitions */
    int (*a)[MAXCOLS], (*b)[MAXCOLS], (*c)[MAXCOLS];

    /* function prototypes */
    void readinput(int (*a)[MAXCOLS], int nrows, int ncols);
    void computesums(int (*a)[MAXCOLS], int (*b)[MAXCOLS],
                     int (*c)[MAXCOLS], int nrows, int ncols);
    void writeoutput(int (*c)[MAXCOLS], int nrows, int ncols);

    printf("How many rows? ");
    scanf("%d", &nrows);
    printf("How many columns? ");
    scanf("%d", &ncols);

    /* allocate initial memory */
    *a = (int *) malloc(nrows * ncols * sizeof(int));
    *b = (int *) malloc(nrows * ncols * sizeof(int));
    *c = (int *) malloc(nrows * ncols * sizeof(int));

    printf("\n\nFirst table:\n");
    readinput(a, nrows, ncols);

    printf("\n\nSecond table:\n");
    readinput(b, nrows, ncols);

    computesums(a, b, c, nrows, ncols);

    printf("\n\nSums of the elements:\n\n");
    writeoutput(c, nrows, ncols);
}
```

```
void readinput(int (*a)[MAXCOLS], int m, int n)
/* read in a table of integers */

{
    int row, col;

    for (row = 0; row < m; ++row)    {
        printf("\nEnter data for row no. %2d\n", row + 1);
        for (col = 0; col < n; ++col)
            scanf("%d", (*(a + row) + col));
    }
    return;
}

void computesums(int (*a)[MAXCOLS], int (*b)[MAXCOLS],
                                    int (*c)[MAXCOLS], int m, int n)
/* add the elements of two integer tables */

{
    int row, col;

    for (row = 0; row < m; ++row)
        for (col = 0; col < n; ++col)
            *(*(c + row) + col) = *(*(a + row) + col) + *(*(b + row) + col);
    return;
}

void writeoutput(int (*a)[MAXCOLS], int m, int n)
/* write out a table of integers */

{
    int row, col;

    for (row = 0; row < m; ++row)    {
        for (col = 0; col < n; ++col)
            printf("%4d", *(*(a + row) + col));
        printf("\n");
    }
    return;
}
```

In this program a, b and c are defined as pointers to groups of contiguous, one-dimensional integer arrays of size MAXCOLS. The function declarations within main and the formal argument declarations within the subordinate functions also represent the arrays in this manner.

Since a, b and c are defined as pointers rather than as arrays, we must allocate initial memory for each array using the malloc function, as described in Sec. 10.4. These memory allocations appear in main, after the values for nrows and ncols have been entered. Consider the first memory allocation, that is,

```
*a = (int *) malloc(nrows * ncols * sizeof(int));
```

In this statement, *a points to the first element in the first array. (Similarly, *(a + 1) points to the first element in the second array, *(a + 2) points to the first element in the third array, and so on, as explained in Example 10.21.) Thus, a block of memory large enough to store nrows × ncols integer quantities begins at the first element of the first array. Similar memory allocations are written for the other two arrays.

The individual array elements are processed by using the indirection operator repeatedly. In readinput, for example, each array element is referenced as

```
scanf("%d", (*(a + row) + col));
```

Similarly, the addition of the array elements within computesums is written as