Chapter 6

Arrays

Chapter 6 - Arrays

Outline

6.1 Introduction
6.2 Arrays
6.3 Declaring Arrays
6.4 Examples Using Arrays
6.5 Passing Arrays to Functions
6.6 Sorting Arrays
6.7 Case Study: Computing Mean, Median and Mode Using Arrays
6.8 Searching Arrays
6.9 Multiple-Subscripted Arrays

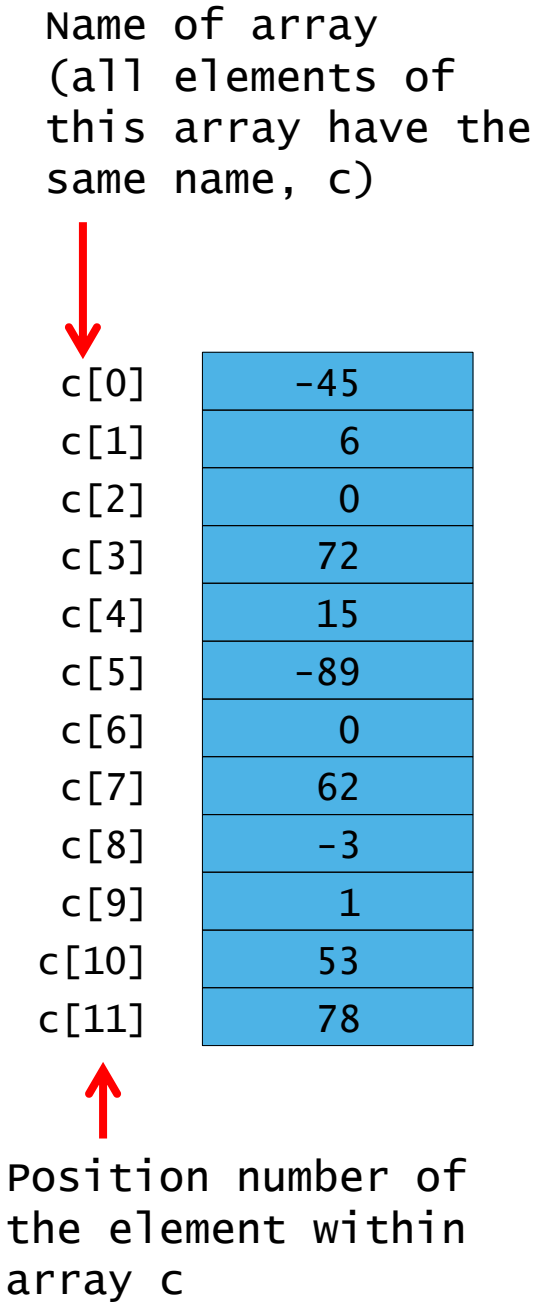
6.1 Introduction

- Arrays
  - Structures of related data items
  - Static entity – same size throughout program
  - Dynamic data structures will be discussed in Chapter 12

6.2 Arrays

- Array
  - Group of consecutive memory locations
  - Same name and data type
- To refer to an element, specify
  - Array name
  - Position number
- Format:

  arrayname`[` *position number* `]`
  - First element at position 0
  - n element array named `c:`
    - `c[ 0 ], c[ 1 ]...c[ n - 1 ]`



Example: Array Declaration and Initialization

METHOD - 1:

```
int  c[12] = { -45, 6, 0, 72, 15, -89, 0, 62, -3, 1, 53, 78 };
```

METHOD - 2:

```
int  c[12];
c[0] = -45;
c[1] = 6;
c[2] = 0;
c[3] = 72;
. . . . . .
c[11] = 78;
```

METHOD - 3:

```
int  c[12], i;
for (i=0; i <= 11; i++)
{
    printf("Enter %d. element ", i);
    scanf("%d", &c[i]);
}
```

6.2 Arrays

- Array elements are like normal variables

```
c[ 0 ] =  3;
printf( "%d", c[ 0 ] );
```

  - Perform operations in subscript. If x equals 3

```
c[ 5 - 2 ] , c[ 3 ] , c[ x ]
```

  - We can also use another array element as a subscript.

```
c[ a[4] ]

c[ a[4] + 2 ]
```

6.3 Defining Arrays

- When defining arrays, specify
  - Name
  - Type of array
  - Number of elements

```
arrayType  arrayName [ numberOfElements ];
```

  - Examples:

```
int c[ 10 ];
float myArray[ 300 ];
```

- Defining multiple arrays of same data type
  - Format similar to regular variables
  - Example:

```
int b[ 100 ], x[ 27 ];
```

6.4 Examples Using Arrays

- Initializers

```
int a[ 5 ] = { 10, 20, 30, 40, 50 };
```

  - If not enough initializers, rightmost elements become 0

```
int a[ 5 ] = { 0 };  // All elements are set to zero
int a[ 5 ] = { 10 };  // First element is 10, other elements are zero
```

  - If too many elements, then a syntax error is produced
  - C arrays have no bounds checking

- If size omitted, count of initializers will determine the size

```
int a[ ] = { 10, 20, 30, 40, 50 };
```

  - 5 initializers, therefore compiler knows that array has 5 elements

Example: Initializing an array with a loop

```c
/* Fig. 6.3: fig06_03.c
   initializing an array */
#include <stdio.h>

int main()
{
   int n[ 10 ]; // n is an array of 10 integers
   int i; // counter

   // initialize elements of array n to 0
   for ( i = 0; i < 10; i++ ) {
      n[ i ] = 0; // set element at location i to 0
   }

   printf( "%s%13s\n", "Element", "Value" );

   // output contents of array n in tabular format
   for ( i = 0; i < 10; i++ ) {
      printf( "%7d%13d\n", i, n[ i ] );
   }
} // end main
```

Program Output

```
Element         Value
      0             0
      1             0
      2             0
      3             0
      4             0
      5             0
      6             0
      7             0
      8             0
      9             0
```

Example: Initializing an array with a declaration

```
/* Fig. 6.4: fig06_04.c
   Initializing an array with a initializer list */
#include <stdio.h>

int main()
{
   // use initializer list to initialize array n
   int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
   int i; // counter

   printf( "%s%13s\n", "Element", "Value" );

   // output contents of array in tabular format
   for ( i = 0; i < 10; i++ ) {
      printf( "%7d%13d\n", i, n[ i ] );
   }

} // end main
```

Program Output

```
Element         Value
      0            32
      1            27
      2            64
      3            18
      4            95
      5            14
      6            90
      7            70
      8            60
      9            37
```

Example: Setting array elements

```c
/* Fig. 6.5: fig06_05.c
   Initialize the elements of array s to
   the even integers from 2 to 20 */
#include <stdio.h>
#define SIZE 10

int main() {
   // symbolic constant SIZE can be used to specify array size
   int s[ SIZE ]; // array s has 10 elements
   int j; // counter

   for ( j = 0; j < SIZE; j++ ) { // set the values
      s[ j ] = 2 + 2 * j;
   }

   printf( "%s%13s\n", "Element", "Value" );

   // output contents of array s in tabular format
   for ( j = 0; j < SIZE; j++ ) {
      printf( "%7d%13d\n", j, s[ j ] );
   }
} // end main
```

Program Output

```
Element         Value
      0             2
      1             4
      2             6
      3             8
      4            10
      5            12
      6            14
      7            16
      8            18
      9            20
```

Example: Sum of elements in an array

```c
/* Fig. 6.6: fig06_06.c
   Compute the sum of the elements of the array */
#include <stdio.h>
#define SIZE 12

int main()
{
   // use initializer list to initialize array
   int a[ SIZE ] = { 1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45 };
   int i; // counter
   int total = 0; // sum of array

   // sum contents of array a
   for ( i = 0; i < SIZE; i++ ) {
      total += a[ i ];
   }

   printf( "Total of array element values is %d\n", total );
} // end main
```

Program Output

```
Total of array element values is 383
```

Example: Student Polling Results

- Assume that 40 students were participated in a polling (i.e, survey).
- For example, subject of the poll was: “How do you utilize your free times?”
- The answers were encoded (rated) between 1 and 10.
  1) Read books
  2) Watch TV
  3) Play football
  4) Study for courses
  5) etc…
  6) etc…
  7) etc…
  8) etc…
  9) etc…
  10) etc….

Example: Student polling

```
/* Fig. 6.7: fig06_07.c
   Student poll program */
#include <stdio.h>
#define RESPONSE_SIZE 40 // define array sizes
#define FREQUENCY_SIZE 10

int main()
{
   int answer; // counter to loop through 40 responses
   int rating; // counter to loop through frequencies 1-10

   // initialize frequency counters to 0
   int frequency[ FREQUENCY_SIZE ] = { 0 };

   // place the survey responses in the responses array
   int responses[ RESPONSE_SIZE ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
        1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
        5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
```

Part 1 of 2

Example: Student polling

```
 /* for each answer, select value of an element of array responses
  and use that value as subscript in array frequency to
  determine element to increment */
   for ( answer = 0; answer < RESPONSE_SIZE; answer++ ) {
      ++frequency[ responses [answer] - 1 ];
   }

   // display results
   printf( "%s%17s\n", "Rating", "Frequency" );

   // output the frequencies in a tabular format
   for ( rating = 1; rating <= FREQUENCY_SIZE; rating++ ) {
      printf( "%6d%17d\n", rating, frequency[ rating-1 ] );
   }

} // end main
```

Part 2 of 2

Program Output

```
Rating          Frequency
     1                  2
     2                  2
     3                  2
     4                  2
     5                  5
     6                 11
     7                  5
     8                  7
     9                  1
    10                  3
```

Example: Histogram

```c
/* Fig. 6.8: fig06_08.c
   Histogram printing program */
#include <stdio.h>
#define SIZE 10

int main() {
   // use initializer list to initialize array n
   int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
   int i; // outer for counter for array elements
   int j; // inner for counter counts *s in each histogram bar

   printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );

   // for each element of array n, output a bar of the histogram
   for ( i = 0; i < SIZE; i++ ) {
      printf( "%7d%13d          ", i, n[ i ]) ;

      for ( j = 1; j <= n[ i ]; j++ ) {   // print one bar
         printf( "%c", '*' );
   } // end inner for

      printf( "\n" ); // end a histogram bar
   } // end outer for
} // end main
```

Program Output

```
Element         Value       Histogram
      0            19       ********************
      1             3       ***
      2            15       ***************
      3             7       *******
      4            11       ***********
      5             9       *********
      6            13       *************
      7             5       *****
      8            17       *****************
      9             1       *
```

Example: Dice Simulation (with frequency counting)

(Same as in Chapter5, but the switch command is replaced with the array usage.)

```c
/* Fig. 6.9: fig06_09.c
   Roll a six-sided die 6000 times */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define SIZE 7

int main() {
   int face; // random die value 1 - 6
   int roll; // roll counter
   int frequency[ SIZE ] = { 0 }; // clear counts

   srand( time( NULL ) ); // seed random-number generator
   // roll die 6000 times
   for ( roll = 1; roll <= 6000; roll++ ) {
      face = 1 + rand() % 6;
      ++frequency[ face ]; // replaces 26-line switch of Fig. 5.8
   }

   printf( "%s%17s\n", "Face", "Frequency" );
   // output frequency elements 1-6 in tabular format
   for ( face = 1; face < SIZE; face++ ) {
      printf( "%4d%17d\n", face, frequency[ face ] );
   }
} // end main
```

Program Output

```
Face        Frequency
   1             1029
   2              951
   3              987
   4             1033
   5             1010
   6              990
```

Character Arrays (Strings)

- Character arrays can be initialized using string literals

```
char string1[] = “apple";
```

- String “`apple`” is really a static array of characters
- Null character `'\0'` terminates strings (added automatically)
- `string1` actually has 6 elements
- It is equivalent to

```
char string1[]={‘a’,‘p',‘p',‘l',‘e','\0'};
```

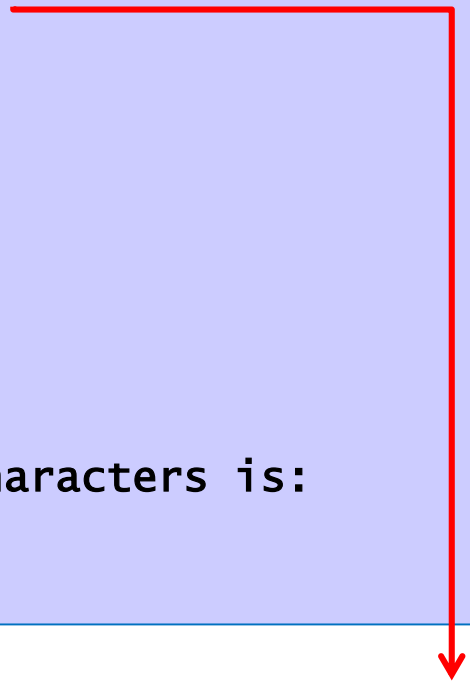
- We can access individual characters
  - `string1[4] is character ‘e’`
- Array name is address of array, so **&** not needed for scanf
  - `scanf("%s", string2);`
    - Reads characters until whitespace encountered
    - Can write beyond end of array in memory, be careful

Example: Character arrays (strings)

```c
/* Fig. 6.10: fig06_10.c
   Treating character arrays as strings */
#include <stdio.h>
int main() {
   char string1[ 20 ]; // reserves 20 characters
   char string2[] = "string literal"; // reserves 15 characters
   int i; // counter

   // read string from user into array string1
   printf("Enter a string: ");
   scanf( "%s", string1 ); // input ended by whitespace character

   // output strings
   printf( "string1 is: %s\nstring2 is: %s\n"
           "string1 with spaces between characters is:\n",
           string1, string2 );

   // output characters until null character is reached
   for ( i = 0; string1[ i ] != '\0'; i++ ) {
      printf( "%c ", string1[ i ] );
   }
} // end main
```

Program Output

```
Enter string1 : Hello there

string1 is: Hello

string2 is: Apple  Orange

string1 with dashes between characters is:
H  - e  - l  - l  - o -
```



- User enters “Hello there”
- But only the first word (Hello) was read by computer because the space character is interpreted as a string stopper.

String Reading Example : The gets() function

```
#include <stdio.h>

int main()
{
      char mesaj[30];

      printf("Bir mesaj girin :");
      gets(mesaj);

      printf("Girilen mesaj = %s \n", mesaj);

} // end main
```

When user enters “Hello there”, both words are read by computer.

Example: Statistical Calculations

- Get student scores from user and store into an array X
- Calculate the followings

$$Avg\ (\bar{x}) = \frac{\sum_{i=1}^{N} X_i}{N} \qquad Variance = \frac{\sum_{i=1}^{N} (x_i - \bar{x})^2}{N}$$

$$\text{Standard deviation} = \sqrt{Variance}$$

$$\text{Absolute deviation} = \frac{\sum_{i=1}^{N} |x_i - \bar{x}|}{N}$$

Example: Statistical Calculations

Part 1 of 2

```
#include <stdio.h>
#include <math.h>  // fabs,sqrt functions
#define MAXSTUDENTS 100

int main()
{
    int score[MAXSTUDENTS];
    int N = 0; // Number of students
    float avg, variance, std_dev, abs_dev;
    float total = 0.0, sqr_total = 0.0, abs_total = 0.0;
    int i = 0;

    printf("How many students are there ? ");
    scanf("%d", &N);
    for (i = 0; i < N; i++)
    {
        printf("Enter grade of student # %d : ", i + 1);
        scanf("%d", &score[i]);
        total += score[i];
    }
```

Example: Statistical Calculations

Part 2 of 2

```
    avg = total / N;

    for (i = 0; i < N; i++)
    {
        sqr_total += pow( score[i] – avg , 2);
        abs_total += fabs(score[i] - avg);
    }

    variance = sqr_total / N;
    std_dev = sqrt(variance);
    abs_dev = abs_total / N;

    printf("Average            =  %f\n", avg);
    printf("Variance           =  %f\n", variance);
    printf("Standard deviation =  %f\n", std_dev);
    printf("Absolute deviation =  %f\n", abs_dev);

} // end main
```

Program Output

```
How many students are there ? 4

Enter grade of student # 1 :  92
Enter grade of student # 2 :  62
Enter grade of student # 3 :  70
Enter grade of student # 4 :  51


Average            =  68.750000
Variance           =  300.916656
Standard deviation =  17.346949
Absolute deviation =  12.250000
```

6.5 Passing Arrays to Functions

- Passing entire array
  - To pass an array argument to a function, specify the name of the array without any brackets

```
int  myArray[24];
myFunction1( myArray , 24 );
```

    - Array size usually passed to function
  - Entire array is passed with **CALL-BY-REFERENCE** method
  - Name of array is address of first element
  - Function knows where the array is stored
    - Modifies original memory locations

- Passing a specific element of array
  - An element is passed with **CALL-BY-VALUE** method
  - Pass subscripted name to function

```
myFunction2( myArray[3] );
```

6.5 Passing Arrays to Functions

- Function prototype

```
void modifyArray( int b[], int arraySize );
```

  - Parameter names optional in prototype
    - `int b[]` could be written `int []`
    - `int arraySize` could be simply `int`

Example: Array address

```c
/* Fig. 6.12: fig06_12.c
   The name of an array is the same as &array[ 0 ] */
#include <stdio.h>

int main()
{
   char array[ 5 ]; // define an array of size 5

   printf( "    array = %p \n &array[0] = %p \n"
      "   &array = %p \n",
      array, &array[ 0 ], &array );

} // end main
```

Program Output

```
    array = 0012FF78
&array[0] = 0012FF78
   &array = 0012FF78
```

Example: Passing entire array and an element to functions

Part 1 of 3

```c
/* Fig. 6.13: fig06_13.c
   Passing arrays and individual array elements to functions */
#include <stdio.h>
#define SIZE 5

// function prototypes
void modifyArray( int b[], int size );
void modifyElement( int e );

// function main begins program execution
int main()
{
   int a[ SIZE ] = { 0, 1, 2, 3, 4 }; // initialize a
   int i; // counter

   printf( "Effects of passing entire array by reference:\n\nThe "
           "values of the original array are:\n" );

   // output original array
   for ( i = 0; i < SIZE; i++ ) {
      printf( "%3d", a[ i ] );
   }

   printf( "\n" );
```

Example: Passing entire array and an element to functions

Part 2 of 3

```
   // pass array a to modifyArray by reference
   modifyArray( a, SIZE );

   printf( "The values of the modified array are:\n" );

   // output modified array
   for ( i = 0; i < SIZE; i++ ) {
      printf( "%3d", a[ i ] );
   }

   // output value of a[ 3 ]
   printf( "\n\n\nEffects of passing array element "
           "by value:\n\nThe value of a[3] is %d\n", a[ 3 ] );

   modifyElement( a[ 3 ] ); // pass array element a[ 3 ] by value

   // output value of a[ 3 ]
   printf( "The value of a[ 3 ] is %d\n", a[ 3 ] );

} // end main
```

Example: Passing entire array and an element to functions

Part 3 of 3

```
/* in function modifyArray, "b" points to the original array "a"
   in memory */
void modifyArray( int b[], int size )
{
   int j; // counter

   // multiply each array element by 2
   for ( j = 0; j < size; j++ ) {
      b[ j ] *= 2;
   }

} // end function modifyArray
```

```
/* in function modifyElement, "e" is a local copy of array element
   a[ 3 ] passed from main */
void modifyElement( int e )
{
   // multiply parameter by 2
   printf( "Value in modifyElement is %d\n", e *= 2 );
} // end function modifyElement
```

Program Output

```
Effects of passing entire array By Reference:

The values of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8


Effects of passing array element By Value:

The value of a[3] is 6
Value in modifyElement is 12
The value of a[3] is 6
```

The original array changed

The original element remains unchanged !

6.6 Sorting Arrays

- Sorting data
  - Important computing application
  - Virtually every organization must sort some data
- Bubble sort (sinking sort)
  - Several passes through the array
  - Successive pairs of elements are compared
    - If increasing order (or identical), no change
    - If decreasing order, elements exchanged
  - Repeat above
- Example:
  - original: 3 4 2 6 7
  - pass 1: 3 2 4 6 7
  - pass 2: 2 3 4 6 7
  - Small elements "bubble" to the top

Example: Bubble Sort Method

```c
/* Fig. 6.15: fig06_15.c
   This program sorts an array's values into ascending order */
#include <stdio.h>
#include <stdlib.h>

#define SIZE 10

// function main begins program execution
int main()
{
   // initialize a
   int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
   int pass; // passes counter
   int i;    // comparisons counter
   int hold; // temporary location used to swap array elements

   printf( "Data items in original order\n" );

   // output original array
   for ( i = 0; i < SIZE; i++ ) {
      printf( "%4d", a[ i ] );
   }
```

```
   // Bubble Sort
   // loop to control number of passes
   for ( pass = 1; pass < SIZE; pass++ ) {

      // loop to control number of comparisons per pass
      for ( i = 0; i < SIZE - 1; i++ ) {

         /* compare adjacent elements and swap them if first
         element is greater than second element */
         if ( a[ i ] > a[ i + 1 ] ) {
            hold = a[ i ];
            a[ i ] = a[ i + 1 ];
            a[ i + 1 ] = hold;
         } // end if

      } // end inner for

   } // end outer for

   printf( "\nData items in ascending order\n" );
   // output sorted array
   for ( i = 0; i < SIZE; i++ ) {
      printf( "%4d", a[ i ] );
   }
} // end main
```

Part 2 of 2

Program Output

```
Data items in original order

   2   6   4   8  10  12  89  68  45  37


Data items in ascending order

   2   4   6   8  10  12  37  45  68  89
```

6.7 Case Study: Computing Mean, Median and Mode Using Arrays

- **Mean** : average
- **Median** : number in middle of sorted list
  - 10, 20, 30, 40, 50
  - 30 is the median
- **Mode** : number that occurs most often
  - 10, 10, 10, 20, 30, 30, 40, 50
  - 10 is the mode

Example: Student polling (Finding the Mean, Median, Mode)

Part 1 of 7

```c
/* Fig. 6.16: fig06_16.c
   This program introduces the topic of survey data analysis.
   It computes the mean, median and mode of the data */
#include <stdio.h>
#define SIZE 99

// function prototypes
void mean( const int answer[] );
void median( int answer[] );
void mode( int freq[], const int answer[] ) ;
void bubbleSort( int a[] );
void printArray( const int a[] );

int main() {
   int frequency[ 10 ] = { 0 }; // initialize array frequency

   // initialize array response
   int response[ SIZE ] =
      { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
        7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
        6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
        7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
        6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
        7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
        5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
        7, 8, 9, 6, 8, 7, 8, 9, 7, 8,
        7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
        4, 5, 6, 1, 6, 5, 7, 8, 7 };
```

Part 2 of 7

```
   // process responses
   mean( response );
   median( response );
   mode( frequency, response );
} // end main
```

```
// calculate average of all response values
void mean( const int answer[] )
{
   int j; // counter for totaling array elements
   int total = 0; // variable to hold sum of array elements

   printf( "%s\n%s\n%s\n", "********", "  Mean", "********" );

   // total response values
   for ( j = 0; j < SIZE; j++ ) {
      total += answer[ j ];
   }

   printf( "The mean is the average value of the data\n"
           "items. The mean is equal to the total of\n"
           "all the data items divided by the number\n"
           "of data items ( %d ). The mean value for\n"
           "this run is: %d / %d = %.4f\n\n",
           SIZE, total, SIZE, (  double  ) total / SIZE );
} // end function mean
```

Part 3 of 7

```
// sort array and determine median element's value
void median( int answer[] )
{
   printf( "\n%s\n%s\n%s\n%s",
           "********", " Median", "********",
           "The unsorted array of responses is" );

   printArray( answer ); // output unsorted array

   bubbleSort( answer ); // sort array

   printf( "\n\nThe sorted array is" );
   printArray( answer ); // output sorted array

   // display median element
   printf( "\n\nThe median is element %d of\n"
           "the sorted %d element array.\n"
           "For this run the median is %d\n\n",
           SIZE / 2, SIZE, answer[ SIZE / 2 ] );
} // end function median
```

Part 4 of 7

```
// determine most frequent response
void mode( int freq[], const int answer[] )
{
   int rating; // counter for accessing elements 1-9 of array freq
   int j; // counter for summarizing elements 0-98 of array answer
   int h; // counter for diplaying histograms of elements in array freq
   int largest = 0; // represents largest frequency
   int modeValue = 0; // respesents most frequent response

   printf( "\n%s\n%s\n%s\n",
           "********", "  Mode", "********" );

   // initialize frequencies to 0
   for ( rating = 1; rating <= 9; rating++ ) {
      freq[ rating ] = 0;
   }

   // summarize frequencies
   for ( j = 0; j < SIZE; j++ ) {
      ++freq[ answer[ j ] ];
   }

   // output headers for result columns
   printf( "%s%11s%19s\n\n%54s\n%54s\n\n",
           "Response", "Frequency", "Histogram",
           "1    1    2    2", "5    0    5    0    5" );
```

Part 5 of 7

```
   // output results
   for ( rating = 1; rating <= 9; rating++ ) {
      printf( "%8d%11d           ", rating, freq[ rating ] );

      // keep track of mode value and largest frequency value
      if ( freq[ rating ] > largest ) {
         largest = freq[ rating ];
         modeValue = rating;
      }

      // output histogram bar representing frequency value
      for ( h = 1; h <= freq[ rating ]; h++ ) {
         printf( "*" );
      } // end inner for

      printf( "\n" ); /* being new line of output */
   } // end outer for

   // display the mode value
   printf( "The mode is the most frequent value.\n"
           "For this run the mode is %d which occurred"
           " %d times.\n", modeValue, largest );
} // end function mode
```

Part 6 of 7

```
// function that sorts an array with bubble sort algorithm
void bubbleSort( int a[] )
{
   int pass; // pass counter
   int j;    // comparison counter
   int hold; // temporary location used to swap elements

   // loop to control number of passes
   for ( pass = 1; pass < SIZE; pass++ ) {

      // loop to control number of comparisons per pass
      for ( j = 0; j < SIZE - 1; j++ ) {

         // swap elements if out of order
         if ( a[ j ] > a[ j + 1 ] ) {
            hold = a[ j ];
            a[ j ] = a[ j + 1 ];
            a[ j + 1 ] = hold;
         } // end if

      } // end inner for

   } // end outer for

} // end function bubbleSort
```

Part 7 of 7

```
// output array contents (20 values per row)
void printArray( const int a[] )
{
   int j; // counter

   // output array contents
   for ( j = 0; j < SIZE; j++ ) {

      if ( j % 20 == 0 ) { // begin new line every 20 values
         printf( "\n" );
      }

      printf( "%2d", a[ j ] );
   } // end for

} // end function printArray
```

Program Output 1

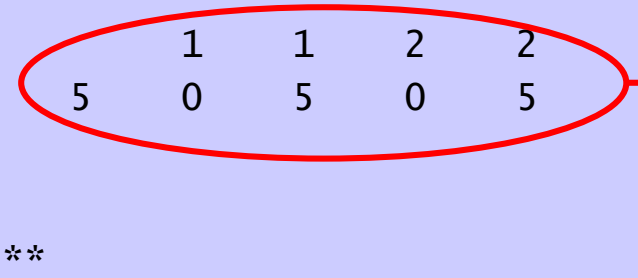
```
********
  Mean
********
The mean is the average value of the data
items. The mean is equal to the total of
all the data items divided by the number
of data items ( 99 ). The mean value for
this run is:  681 / 99 = 6.8788
```

Program Output 2

```
********
 Median
********
The unsorted array of responses is
 6 7 8 9 8 7 8 9 8 9 7 8 9 5 9 8 7 8 7 8
 6 7 8 9 3 9 8 7 8 7 7 8 9 8 9 8 9 7 8 9
 6 7 8 7 8 7 9 8 9 2 7 8 9 8 9 8 9 7 5 3
 5 6 7 2 5 3 9 4 6 4 7 8 9 6 8 7 8 9 7 8
 7 4 4 2 5 3 8 7 5 6 4 5 6 1 6 5 7 8 7

The sorted array is
 1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5 5 5 5 5
 5 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7
 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9


The median is element 49 of
the sorted 99 element array.
For this run the median is 7
```

Program Output 3

```
********
  Mode
********
Response   Frequency          Histogram

                          1    1    2    2
                     5    0    5    0    5

       1           1      *
       2           3      ***
       3           4      ****
       4           5      *****
       5           8      ********
       6           9      *********
       7          23      ***********************
       8          27      ***************************
       9          19      *******************
      10           0

The mode is the (index of) most frequent value.
For this run the mode is 8 which occurred 27 times.
```



Horizontal ruler marks:

5, 10, 15, 20, 25

6.8 Searching Arrays: Linear Search

- Search an array for a *Key Value*
- Linear search
  - Simple
  - Compare each element of array with the given key value
  - Useful for small and unsorted arrays

Example: Linear search

Part 1 of 2

```c
/* Fig. 6.18: fig06_18.c
   Linear search of an array */
#include <stdio.h>
#include <stdlib.h>

#define SIZE 100

// function prototype
int linearSearch( const int array[], int key, int size );

int main()
{
   int a[ SIZE ]; // create array a
   int x; // counter for initializing elements 0-99 of array a
   int searchKey; // value to locate in array a
   int element; // variable to hold location of searchKey or -1

   // create data
   for ( x = 0; x < SIZE; x++ ) {
      a[ x ] = 2 * x;
   }

   printf( "Enter integer search key:\n" );
   scanf( "%d", &searchKey );
```

Part 2 of 2

```
   // attempt to locate searchKey in array a
   element = linearSearch( a, searchKey, SIZE );

   // display results
   if ( element != -1 ) {
      printf( "Found value in element %d\n", element );
   }
   else {
      printf( "Value not found\n" );
   }

} // end main
```

```
/* compare key to every element of array until the location is found
   or until the end of array is reached; return subscript of element
   if key or -1 if key is not found */
int linearSearch( const int array[], int key, int size ) {
   int n; // counter

   // loop through array
   for ( n = 0; n < size; ++n ) {
      if ( array[ n ] == key ) {
         return n; // return location of key
      } // end if
   } // end for

   return -1; // key not found
} // end function linearSearch
```

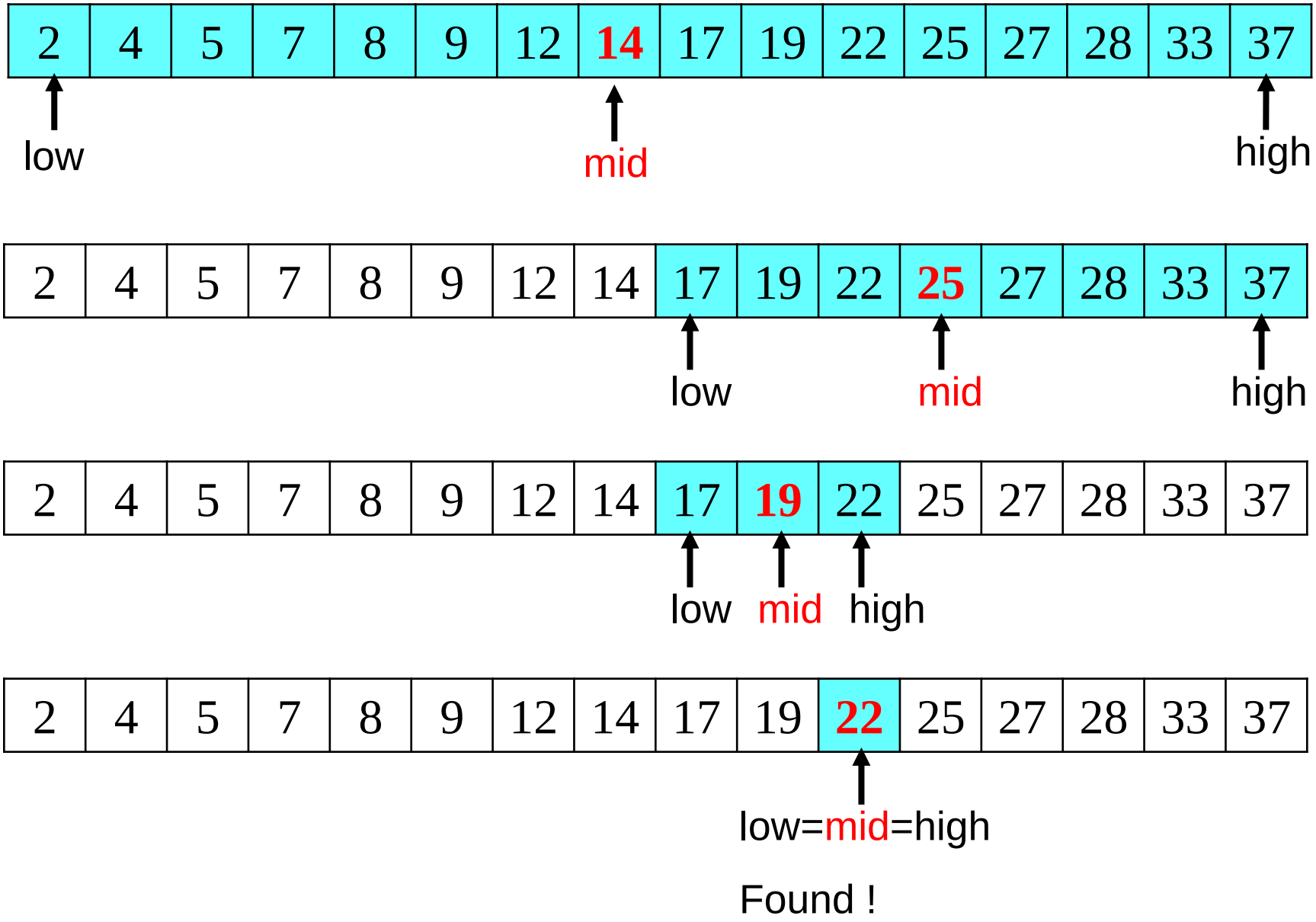
Program Output

```
Enter integer search key:
36
Found value in element 18


Enter integer search key:
37
Value not found
```

6.8 Searching Arrays: Binary Search

- Binary search
  - Can be use for **only** sorted arrays
  - Compares `middle` element with `key`
    - If equal, match found
    - If `key < middle`, looks in first half of array
    - If `key > middle`, looks in last half
    - Set a new `middle` and repeat the steps above
  - Very fast; at most x steps, where 2^x > Number of elements
    - 30 element array takes at most 5 steps
      - 2^5 > 30 so at most 5 steps

Binary Search Example: Find 22



Example: Binary search

Part 1 of 5

```
/* Fig. 6.19: fig06_19.c
   Binary search of an array */
#include <stdio.h>

#define SIZE 15

// function prototypes
int binarySearch( const int b[], int searchKey, int low, int high );
void printHeader( void );
void printRow( const int b[], int low, int mid, int high );

int main()
{
   int a[ SIZE ]; // create array a
   int i; // counter for initializing elements 0-14 of array a
   int key; // value to locate in array a
   int result; // variable to hold location of key or -1

   // create data
   for ( i = 0; i < SIZE; i++ ) {
      a[ i ] = 2 * i;
   }
```

Part 2 of 5

```
   printf( "Enter a number between 0 and 28: " );
      scanf( "%d", &key );

      printHeader();

      // search for key in array a
      result = binarySearch( a, key, 0, SIZE - 1 );

      // display results
      if ( result != -1 ) {
         printf( "\n%d found in array element %d\n", key, result );
      }
      else {
         printf( "\n%d not found\n", key );
      }

} // end main
```

Part 3 of 5

```
// function to perform binary search of an array
int binarySearch( const int b[], int searchKey, int low, int high )
{
   int middle; // variable to hold middle element of array

   // loop until low subscript is greater than high subscript
   while ( low <= high ) {
      // determine middle element of subarray being searched
      middle = ( low + high ) / 2;
      // display subarray used in this loop iteration
      printRow( b, low, middle, high );

      // if searchKey matched middle element, return middle
      if ( searchKey == b[ middle ] ) {
         return middle;
      } // end if
      // if searchKey less than middle element, set new high
      else if ( searchKey < b[ middle ] ) {
         high = middle - 1; // search low end of array
      } // end else if
      // if searchKey greater than middle element, set new low
      else {
         low = middle + 1; // search high end of array
      } // end else

   } // end while
   return -1;   // searchKey not found
} // end function binarySearch
```

Part 4 of 5

```
// Print a header for the output
void printHeader( void )
{
   int i; // counter

   printf( "\nSubscripts:\n" );

   // output column head
   for ( i = 0; i < SIZE; i++ ) {
      printf( "%3d ", i );
   }

   printf( "\n" ); // start new line of output

   // output line of - characters
   for ( i = 1; i <= 4 * SIZE; i++ ) {
      printf( "-" );
   }

   printf( "\n" ); // start new line of output
} // end function printHeader
```

Part 5 of 5

```c
/* Print one row of output showing the current
   part of the array being processed. */
void printRow( const int b[], int low, int mid, int high )
{
   int i; // counter for iterating through array b

   // loop through entire array
   for ( i = 0; i < SIZE; i++ ) {

      // display spaces if outside current subarray range
      if ( i < low || i > high ) {
         printf( "    " );
      } // end if
      else if ( i == mid ) { // display middle element
         printf( "%3d*", b[ i ] ); // mark middle value
      } // end else if
      else { // display other elements in subarray
         printf( "%3d ", b[ i ] );
      } // end else

   } // end for

   printf( "\n" ); // start new line of output
} // end function printRow
```

Program Output 1

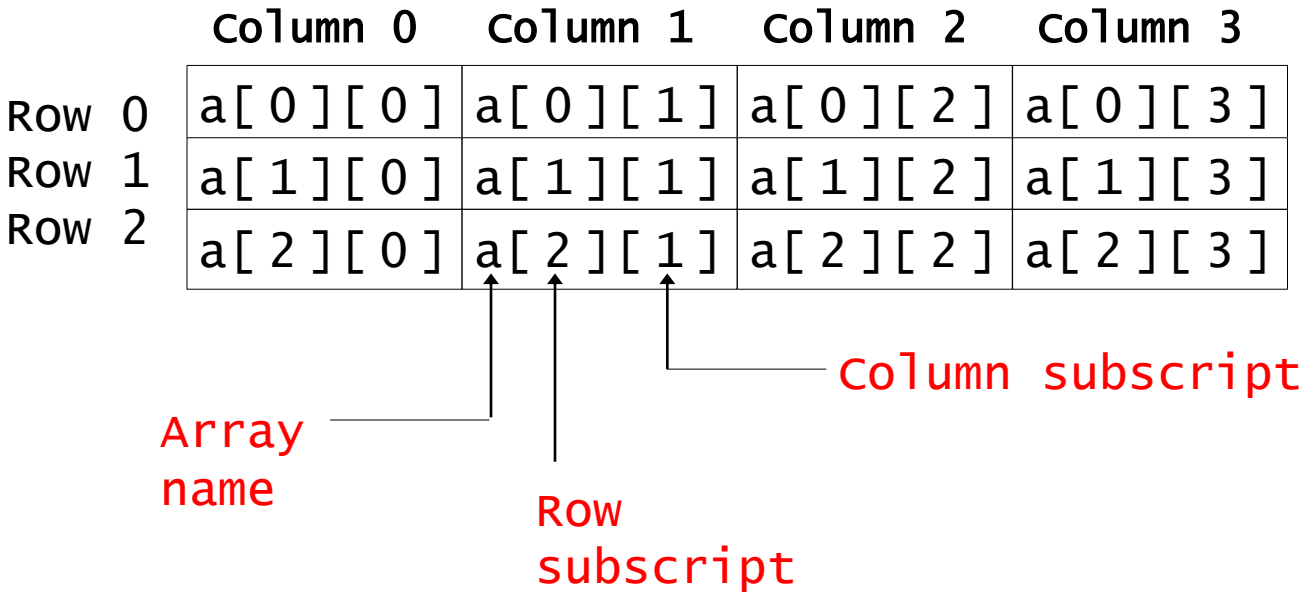
```
Enter a number between 0 and 28: 25

Subscripts:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
---------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
                              16  18  20  22* 24  26  28
                                          24  26* 28
                                          24*

25 not found
```

Program Output 2

```
Enter a number between 0 and 28: 8

Subscripts:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
---------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
  0   2   4   6*  8  10  12
                  8  10* 12
                  8*

8 found in array element 4
```

6.9 Multiple-Subscripted Arrays

- Multiple subscripted arrays
  - Tables with rows and columns (`m` by `n` array)
  - Like matrices: specify row, then column



6.9 Multiple-Subscripted Arrays

- Initialization
  - `int b[ 2 ][ 2 ] = { { 10, 20 }, { 30, 40 } };`
  - Initializers grouped by row in braces

10	20
30	40

  - If not enough, unspecified elements set to zero

```
int b[ 2 ][ 2 ] = { { 10}, { 30, 40 } };
```

10	0
30	40

- Referencing elements
  - Specify row, then column

```
printf( "%d", b[ 1 ][ 0 ] );
```
//Displays 30

Example: Matrix initialization

Part 1 of 2

```c
/* Fig. 6.21: fig06_21.c
   Initializing multidimensional arrays */
#include <stdio.h>

void printArray( const int a[][ 3 ] ); // function prototype

int main()
{
   // initialize array1, array2, array3
   int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
   int array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };
   int array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };

   printf( "Values in array1 by row are:\n" );
   printArray( array1 );

   printf( "Values in array2 by row are:\n" );
   printArray( array2 );

   printf( "Values in array3 by row are:\n" );
   printArray( array3 );

} // end main
```

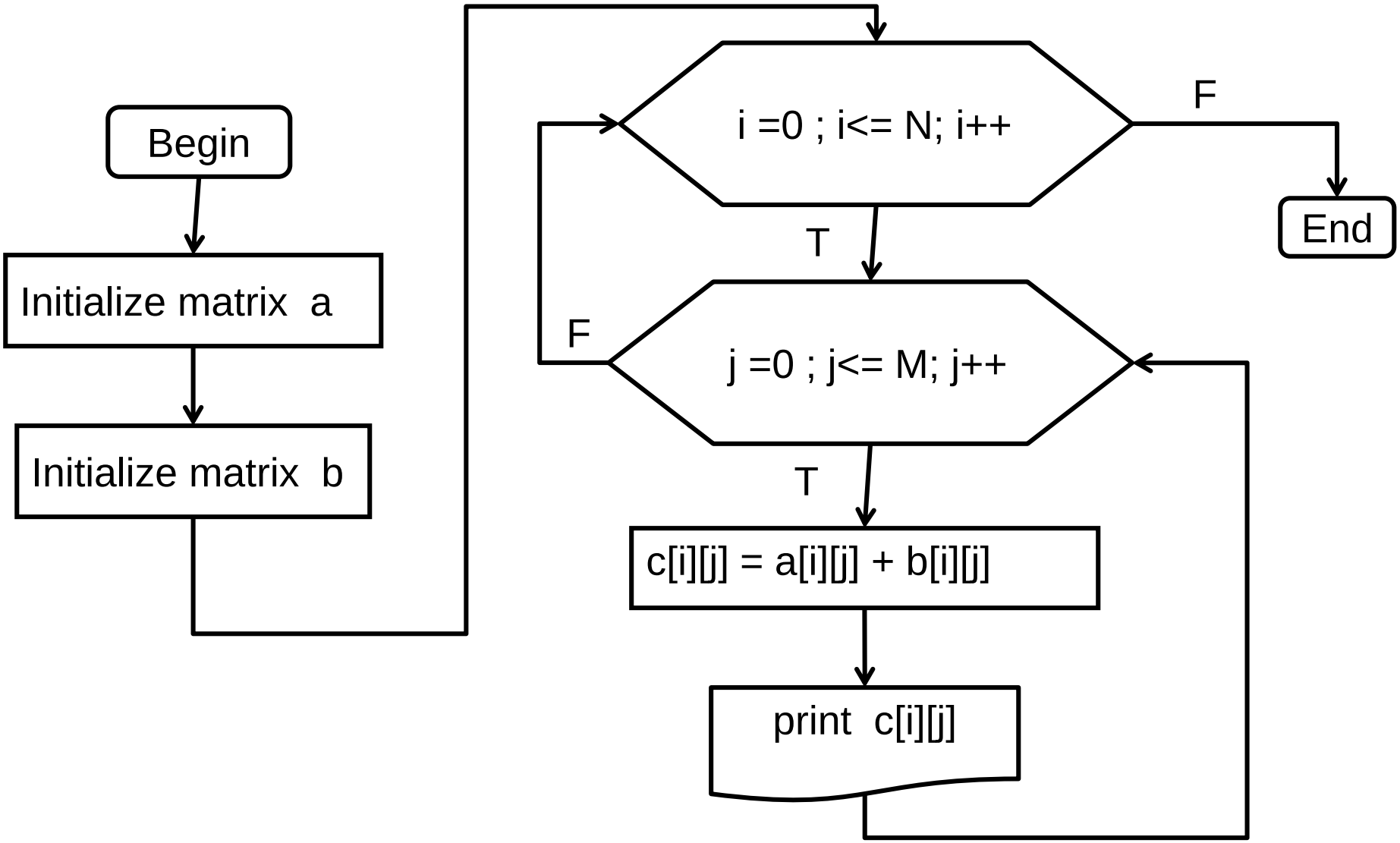
Part 2 of 2

```
// function to output array with two rows and three columns
void printArray( const int a[][ 3 ] )
{
   int i; // row counter
   int j; // column counter

   // loop through rows
   for ( i = 0; i <= 1; i++ ) {

      // output column values
      for ( j = 0; j <= 2; j++ ) {
         printf( "%d ", a[ i ][ j ] );
      } // end inner for

      printf( "\n" ); // start new line of output
   } // end outer for

} // end function printArray
```

Program Output

```
Values in array1 by row are:
1 2 3
4 5 6
Values in array2 by row are:
1 2 3
4 5 0
Values in array3 by row are:
1 2 0
4 0 0
```

Example: Adding Two Matrices



Example: Adding Two Matrices

```
#include <stdio.h>
int main() {
    int a[2][2] = {{10, 15}, {20, 5}};  // Matrix a
    int b[2][2] = {{25,  5}, { 6, 0}};  // Matrix b
    int c[2][2]; // Matrix c
    int i, j;
    printf ("RESULTING ADDITION MATRIX \n\n");

    for(i=0; i<2; i++)  {
        for(j=0; j<2; j++)   {
            c[i][j] = a[i][j] + b[i][j];
          printf ("%d\t", c[i][j]);
        } // end inner for
        printf ("\n"); // new line
     } // end outer for
} // end main
```

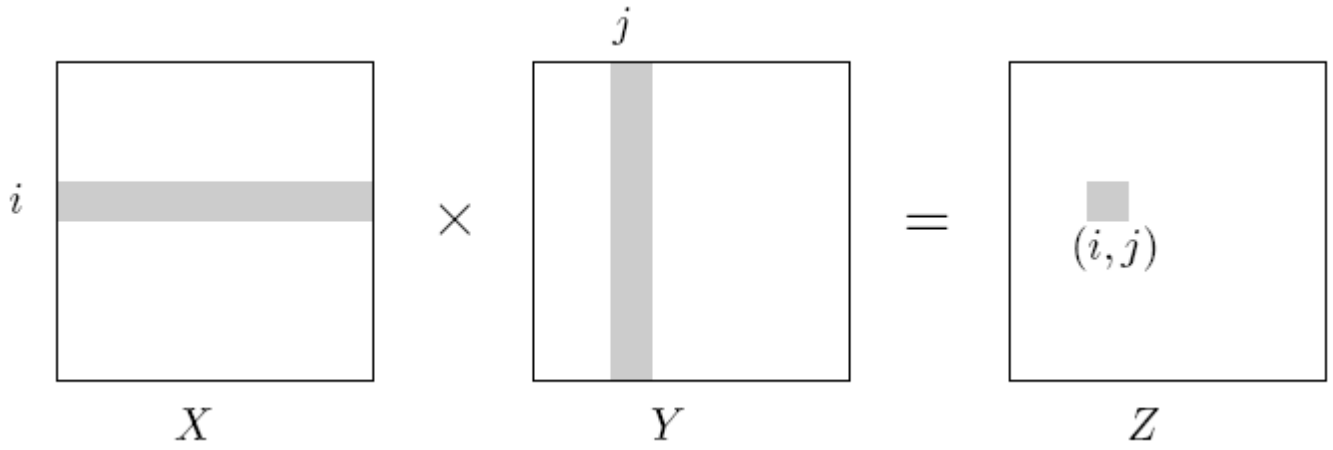
Program Output

```
RESULTING ADDITION MATRIX

35      20
26      5
```

Example: Multiplying Two Matrices (1)

- The product of two nxn matrices X and Y is a third nxn matrix Z = X.Y , with (i,j)th entry

$$Z_{ij} = \sum_{k=1}^{N} X_{ik} Y_{kj}$$



Example: Multiplying Two Matrices (2)

- Example: X and Y are two 2x2 matrices.
- Z is also a 2x2 matrix (Z = X . Y)

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \quad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

Example: Multiplying Two Matrices

Part 1 of 3

```
#include <stdio.h>
#define N 2
#define M 4
#define L 3

int main()
{
    int a[N][M] = {{ 8,  5, -6,  7},
                   { 0,  2,  1,  4}
                  };

    int b[M][L] = {{ 3, -9,  1},
                   { 2,  5,  8},
                   {-2,  4,  0},
                   { 1,  7,  6}
                  };

    int c[N][L];

    int i,j,k;
```

Example: Multiplying Two Matrices

Part 2 of 3

```
// Compute the multiplication
   for (i = 0; i < N; i++)
   {
       for (j = 0; j < L; j++)
       {
           c[i][j] = 0;
           for (k = 0; k < M; k++)
               c[i][j] += a[i][k] * b[k][j];
       }
   }
```

Example: Multiplying Two Matrices

Part 3 of 3

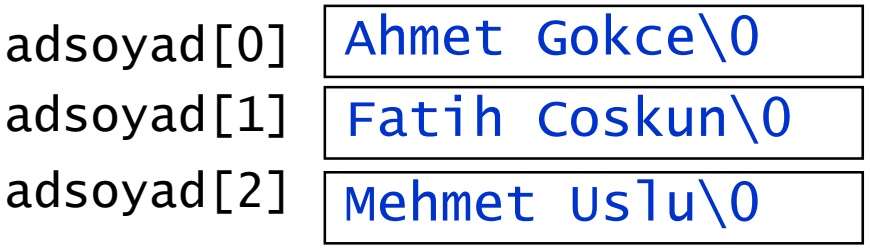
```
    // Display the result matrix

    printf ("RESULTING MULTIPLICATION MATRIX \n\n");

    for(i=0; i < N; i++)
    {
        for (j=0; j < L; j++)
            printf ("%d\t", c[i][j]);

        printf ("\n");
     }

} // end main
```

Program Output

```
RESULTING MULTIPLICATION MATRIX

53      -22     90
6       42      40
```

Example: Array of Strings

```
#include <stdio.h>
#define  N  3    // Number of persons

int main()
{
 int i;
 char  adsoyad[N][20] = {"Ahmet Gokce",
                         "Fatih Coskun",
                         "Mehmet Uslu"};
 for (i=0; i < N; i++)
      printf("%s \n", adsoyad[i]);

} // end main
```



Example: Inputting an Array of Strings

```
#include <stdio.h>
#define  N  3      // Number of persons

int main()
{
 int i;
 char  adsoyad[N][20];
 for (i=0; i < N; i++)
 {
   printf("Enter name of %d. person : ", i+1);
   scanf("%s", adsoyad[i]);
 }

 for (i=0; i < N; i++)
      printf("%s \n", adsoyad[i]);

} // end main
```

Example:

Matrix for Students and Exams

Example: Matrix for students and exams

Part 1 of 6

```
/* Fig. 6.22: fig06_22.c
   Double-subscripted (two-dimensional) array example */
#include <stdio.h>

#define STUDENTS 3
#define EXAMS 4

// function prototypes
int minimum( const int grades[][ EXAMS ], int pupils, int tests );
int maximum( const int grades[][ EXAMS ], int pupils, int tests );
double average( const int setOfGrades[], int tests );
void printArray( const int grades[][ EXAMS ], int pupils, int tests );

int main()
{
   int student; // student counter

   // initialize student grades for three students (rows)
   const int studentGrades[ STUDENTS ][ EXAMS ] =
      { { 77, 68, 86, 73 },
        { 96, 87, 89, 78 },
        { 70, 90, 86, 81 } };
```

Part 2 of 6

```
   // output array studentGrades
   printf( "The array is:\n" );
   printArray( studentGrades, STUDENTS, EXAMS );

   // determine smallest and largest grade values
   printf( "\n\nLowest grade: %d\nHighest grade: %d\n",
      minimum( studentGrades, STUDENTS, EXAMS ),
      maximum( studentGrades, STUDENTS, EXAMS ) );

   // calculate average grade for each student
   for ( student = 0; student < STUDENTS; student++ ) {
      printf( "The average grade for student %d is %.2f\n",
         student, average( studentGrades[ student ], EXAMS ) );
   } // end for

} // end main
```

Part 3 of 6

```
// Find the minimum grade
int minimum( const int grades[][ EXAMS ], int pupils, int tests )
{
   int i; // student counter
   int j; // exam counter
   int lowGrade = 100; // initialize to highest possible grade

   // loop through rows of grades
   for ( i = 0; i < pupils; i++ ) {

      // loop through columns of grades
      for ( j = 0; j < tests; j++ ) {

         if ( grades[ i ][ j ] < lowGrade ) {
            lowGrade = grades[ i ][ j ];
         } // end if

      } // end inner for

   } // end outer for

   return lowGrade; // return minimum grade

} // end function minimum
```

Part 4 of 6

```
// Find the maximum grade
int maximum( const int grades[][ EXAMS ], int pupils, int tests )
{
   int i; // student counter
   int j; // exam counter
   int highGrade = 0; // initialize to lowest possible grade

   /*/ loop through rows of grades
   for ( i = 0; i < pupils; i++ ) {

      // loop through columns of grades
      for ( j = 0; j < tests; j++ ) {

         if ( grades[ i ][ j ] > highGrade ) {
            highGrade = grades[ i ][ j ];
         } // end if

      } // end inner for

   } // end outer for

   return highGrade; // return maximum grade

} // end function maximum
```

Part 5 of 6

```
// Determine the average grade for a particular student
double average( const int setOfGrades[], int tests )
{
   int i; // exam counter
   int total = 0; // sum of test grades

   // total all grades for one student
   for ( i = 0; i < tests; i++ ) {
      total += setOfGrades[ i ];
   } // end for

   return ( double ) total / tests;   // average

} // end function average
```

Part 6 of 6

```
// Print the array
void printArray( const int grades[][ EXAMS ], int pupils, int tests )
{
   int i; // student counter
   int j; // exam counter

   // output column heads
   printf( "                [0]  [1]  [2]  [3]" );

   // output grades in tabular format
   for ( i = 0; i < pupils; i++ ) {

      // output label for row
      printf( "\nstudentGrades[%d] ", i );

      // output grades for one student
      for ( j = 0; j < tests; j++ ) {
         printf( "%-5d", grades[ i ][ j ] );
      } // end inner for

   } // end outer for

} // end function printArray
```

Program Output

```
The array is:
                    [0]  [1]  [2]  [3]
studentGrades[0] 77   68   86   73
studentGrades[1] 96   87   89   78
studentGrades[2] 70   90   86   81


Lowest grade: 68
Highest grade: 96


The average grade for student 0 is 76.00
The average grade for student 1 is 87.50
The average grade for student 2 is 81.75
```