Chapter 6 - Arrays

Outline

Objectives

- In this chapter, you will learn:
  - To introduce the array data structure.
  - To understand the use of arrays to store, sort and search lists and tables of values.
  - To understand how to define an array, initialize an array and refer to individual elements of an array.
  - To be able to pass arrays to functions.
  - To understand basic sorting techniques.
  - To be able to define and manipulate multiple subscript arrays.

6.1 Introduction

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

6.2 Arrays

- Array
  - Group of consecutive memory locations
  - Same name and 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]

Name of array (Note that all elements of this array have the same name, c)

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Position number of the element within array c

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 ]
```

6.2 Arrays

Operators						Associativity	Type
[]	()					left to right	highest
++	--	!	(*type*)			right to left	unary
*	/	%				left to right	multiplicative
+	-					left to right	additive
<	<=	>	>=			left to right	relational
==	!=					left to right	equality
&&						left to right	logical and
||						left to right	logical or
?:						right to left	conditional
=	+=	-=	*=	/=	%=	right to left	assignment
,						left to right	comma

Fig. 6.2 Operator precedence.

6.3 Defining Arrays

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

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

  - Examples:

```
int c[ 10 ];
float myArray[ 3284 ];
```

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

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

6.4 Examples Using Arrays

- Initializers

```
int n[ 5 ] = { 1, 2, 3, 4, 5 };
```

  - If not enough initializers, rightmost elements become 0

```
int n[ 5 ] = { 0 }
```

    - All elements 0
  - If too many a syntax error is produced syntax error
  - C arrays have no bounds checking
- If size omitted, initializers determine it

```
int n[ ] = { 1, 2, 3, 4, 5 };
```

  - 5 initializers, therefore 5 element array

Outline

fig06_03.c

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

/* function main begins program execution */
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 */
   } /* end for */

   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 for */

   return 0; /* indicates successful termination */

} /* end main */
```

Outline

Program Output

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

6.4 Examples Using Arrays

- Character arrays
  - String "`first`" is really a static array of characters
  - Character arrays can be initialized using string literals

```
char string1[] = "first";
```

    - Null character `'\0'` terminates strings
    - `string1` actually has 6 elements
      - It is equivalent to

```
char string1[] = { 'f', 'i', 'r', 's', 't', '\0' };
```

  - Can access individual characters

```
string1[ 3 ] is character 's'
```

  - 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, be careful

Outline

fig06_04.c

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

/* function main begins program execution */
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 for */

   return 0; /* indicates successful termination */

} /* 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
```

Outline

fig06_05.c

```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

/* function main begins program execution */
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;
   } /* end for */

   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 for */

   return 0; /* indicates successful termination */

} /* 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
```

Outline

fig06_06.c

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

/* function main begins program execution */
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 ];
   } /* end for */

   printf( "Total of array element values is %d\n", total );

   return 0; /* indicates successful termination */

} /* end main */
```

Program Output

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

Outline

fig06_07.c (Part 1 of 2)

```c
/* Fig. 6.7: fig06_07.c
   Student poll program */
#include <stdio.h>
#define RESPONSE_SIZE 40 /* define array sizes */
#define FREQUENCY_SIZE 11

/* function main begins program execution */
int main()
{
   int answer; /* counter */
   int rating; /* counter */

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

   /* place survey responses in array responses */
   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 };

```

Outline

fig06_07.c (Part 2 of 2)

```
   /* 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 ] ];
   } /* end for */

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

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

   return 0; /* indicates successful termination */

} /* end main */
```

Program Output

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

Outline

fig06_08.c (Part 1 of 2)

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

/* function main begins program execution */
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 counter */
    int j; /* inner counter */

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

    /* for each element of array n, output a bar in 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" ); /* start next line of output */
   } /* end outer for */

   return 0; /* indicates successful termination */

} /* end main */
```

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


fig06_08.c (Part 2 of 2)

Program Output

Outline

fig06_09.c (Part 1 of 2)

```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

/* function main begins program execution */
int main()
{
   int face;                      /* random number with value 1 - 6 */
   int roll;                      /* roll counter */
   int frequency[ SIZE ] = { 0 }; /* initialize array to 0 */

   srand( time( NULL ) ); /* seed random-number generator */

   /* roll die 6000 times */
   for ( roll = 1; roll <= 6000; roll++ ) {
      face = rand() % 6 + 1;
      ++frequency[ face ]; /* replaces 26-line switch of Fig. 5.8 */
   } /* end for */

   printf( "%s%17s\n", "Face", "Frequency" );

```

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

   return 0; /* indicates successful termination */

} /* end main */
```

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

Outline

fig06_09.c (Part 2 of 2)

Program Output

Outline

fig06_10.c (Part 1 of 2)

```c
/* Fig. 6.10: fig06_10.c
   Treating character arrays as strings */
#include <stdio.h>

/* function main begins program execution */
int main()
{
   char string1[ 20 ];                /* reserves 20 characters */
   char string2[] = "string literal"; /* reserves 15 characters */
   int i;                             /* counter */

   /* read string from user into array string2 */
   printf("Enter a string: ");
   scanf( "%s", string1 );

   /* 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 for */

```

```
   printf( "\n" );

   return 0; /* indicates successful termination */

} /* end main */
```

```
Enter a string: Hello there
string1 is: Hello
string2 is: string literal
string1 with spaces between characters is:
H e l l o
```


fig06_10.c (Part 2 of 2)

Outline

fig06_11.c (Part 1 of 3)

```c
/* Fig. 6.11: fig06_11.c
   Static arrays are initialized to zero */
#include <stdio.h>

void staticArrayInit( void );    /* function prototype */
void automaticArrayInit( void ); /* function prototype */

/* function main begins program execution */
int main()
{
   printf( "First call to each function:\n" );
   staticArrayInit();
   automaticArrayInit();

   printf( "\n\nSecond call to each function:\n" );
   staticArrayInit();
   automaticArrayInit();

   return 0; /* indicates successful termination */

} /* end main */

```

Outline

fig06_11.c (Part 2 of 3)

```c
/* function to demonstrate a static local array */
void staticArrayInit( void )
{
   /* initializes elements to 0 first time function is called */
   static int array1[ 3 ];
   int i; /* counter */

   printf( "\nValues on entering staticArrayInit:\n" );

   /* output contents of array1 */
   for ( i = 0; i <= 2; i++ ) {
      printf( "array1[ %d ] = %d  ", i, array1[ i ] );
   } /* end for */

   printf( "\nValues on exiting staticArrayInit:\n" );

   /* modify and output contents of array1 */
   for ( i = 0; i <= 2; i++ ) {
      printf( "array1[ %d ] = %d  ", i, array1[ i ] += 5 );
   } /* end for */

} /* end function staticArrayInit */

```

Outline

fig06_11.c (Part 3 of 3)

```c
/* function to demonstrate an automatic local array */
void automaticArrayInit( void )
{
   /* initializes elements each time function is called */
   int array2[ 3 ] = { 1, 2, 3 };
   int i; /* counter */

   printf( "\n\nValues on entering automaticArrayInit:\n" );

   /* output contents of array2 */
   for ( i = 0; i <= 2; i++ ) {
      printf("array2[ %d ] = %d  ", i, array2[ i ] );
   } /* end for */

   printf( "\nValues on exiting automaticArrayInit:\n" );

   /* modify and output contents of array2 */
   for ( i = 0; i <= 2; i++ ) {
      printf( "array2[ %d ] = %d  ", i, array2[ i ] += 5 );
   } /* end for */

} /* end function automaticArrayInit */
```

Program Output

```
First call to each function:

Values on entering staticArrayInit:
array1[ 0 ] = 0  array1[ 1 ] = 0  array1[ 2 ] = 0
Values on exiting staticArrayInit:
array1[ 0 ] = 5  array1[ 1 ] = 5  array1[ 2 ] = 5

Values on entering automaticArrayInit:
array2[ 0 ] = 1  array2[ 1 ] = 2  array2[ 2 ] = 3
Values on exiting automaticArrayInit:
array2[ 0 ] = 6  array2[ 1 ] = 7  array2[ 2 ] = 8

Second call to each function:

Values on entering staticArrayInit:
array1[ 0 ] = 5  array1[ 1 ] = 5  array1[ 2 ] = 5
Values on exiting staticArrayInit:
array1[ 0 ] = 10  array1[ 1 ] = 10  array1[ 2 ] = 10

Values on entering automaticArrayInit:
array2[ 0 ] = 1  array2[ 1 ] = 2  array2[ 2 ] = 3
Values on exiting automaticArrayInit:
array2[ 0 ] = 6  array2[ 1 ] = 7  array2[ 2 ] = 8
```

6.5 Passing Arrays to Functions

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

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

    - Array size usually passed to function
  - Arrays passed call-by-reference
  - Name of array is address of first element
  - Function knows where the array is stored
    - Modifies original memory locations
- Passing array elements
  - Passed by call-by-value
  - Pass subscripted name (i.e., `myArray[ 3 ]`) to function

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`

Outline

fig06_12.c

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

/* function main begins program execution */
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 );

   return 0; /* indicates successful termination */

} /* end main */
```

Program Output

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

Outline

fig06_13.c (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 ] );
   } /* end for */

   printf( "\n" );

```

Outline

fig06_13.c (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 ] );
   } /* end for */

   /* 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 ] );

   return 0; /* indicates successful termination */

} /* end main */
```

Outline

```
/* 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 for */

} /* 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 */
```

fig06_13.c (Part 3 of 3)

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
```

Outline

fig06_14.c (Part 1 of 2)

```c
/* Fig. 6.14: fig06_14.c
   Demonstrating the const type qualifier with arrays */
#include <stdio.h>

void tryToModifyArray( const int b[] ); /* function prototype */

/* function main begins program execution */
int main()
{
   int a[] = { 10, 20, 30 }; /* initialize a */

   tryToModifyArray( a );

   printf("%d %d %d\n", a[ 0 ], a[ 1 ], a[ 2 ] );

   return 0; /* indicates successful termination */

} /* end main */

```

Outline

fig06_14.c (Part 2 of 2)

```
/* in function tryToModifyArray, array b is const, so it cannot be
   used to modify the original array a in main. */
void tryToModifyArray( const int b[] )
{
   b[ 0 ] /= 2;    /* error */
   b[ 1 ] /= 2;    /* error */
   b[ 2 ] /= 2;    /* error */
} /* end function tryToModifyArray */
```

Program Output

```
Compiling...
FIG06_14.C
fig06_14.c(24) : error C2166: l-value specifies const object
fig06_14.c(25) : error C2166: l-value specifies const object
fig06_14.c(26) : error C2166: l-value specifies const object
```

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
- 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

Outline

fig06_15.c (Part 1 of 3)

```c
/* Fig. 6.15: fig06_15.c
   This program sorts an array's values into ascending order */
#include <stdio.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 i;    /* inner counter */
   int pass; /* outer 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 ] );
   } /* end for */

```

Outline

fig06_15.c (Part 2 of 3)

```
   /* 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" );

```

Outline

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

   printf( "\n" );

   return 0; /* indicates successful termination */

```

fig06_15.c (Part 3 of 3)

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
```

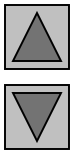
6.7 Case Study: Computing Mean, Median and Mode Using Arrays

- Mean – average
- Median – number in middle of sorted list
  - 1, 2, 3, 4, 5
  - 3 is the median
- Mode – number that occurs most often
  - 1, 1, 1, 2, 3, 3, 4, 5
  - 1 is the mode

Outline

fig06_16.c (Part 1 of 8)

```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[] );

/* function main begins program execution */
int main()
{
   int frequency[ 10 ] = { 0 }; /* initialize array frequency */

```



Outline

fig06_16.c (Part 2 of 8)

```
   /* 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 };

   /* process responses */
   mean( response );
   median( response );
   mode( frequency, response );

   return 0; /* indicates successful termination */

} /* end main */
```

Outline

fig06_16.c (Part 3 of 8)

```
/* calculate average of all response values */
void mean( const int answer[] )
{
   int j;         /* counter */
   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  ];
   } /* end for */

   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 */

```

Outline

fig06_16.c (Part 4 of 8)

```
/* 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 */

```

Outline

fig06_16.c (Part 5 of 8)

```
/* determine most frequent response */
void mode( int freq[], const int answer[] )
{
   int rating;        /* counter */
   int j;             /* counter */
   int h;             /* counter */
   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;
   } /* end for */

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

```

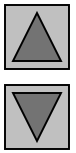
Outline

fig06_16.c (Part 6 of 8)

```
/* 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" );

/* 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;
   } /* end if */

   /* 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 */

```

fig06_16.c (Part 7 of 8)

```
   /* 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 */

/* function that sorts an array with bubble sort algorithm */
void bubbleSort( int a[] )
{
   int pass; /* counter */
   int j;    /* 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 */

```



Outline

fig06_16.c (Part 8 of 8)

```
      } /* end inner for */

   } /* end outer for */

} /* end function bubbleSort */

/* 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" );
      } /* end if */

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

} /* end function printArray */
```

Program Output

```
********
  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


********
 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
```

Outline

Program Output (continued)

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


********
  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          *******************
The mode is the most frequent value.
For this run the mode is 8 which occurred 27 times.
```

6.8 Searching Arrays: Linear Search and Binary Search

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

6.8 Searching Arrays: Linear Search and Binary Search

- Binary search
  - For 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
    - Repeat
  - Very fast; at most n steps, where 2^n > number of elements
    - 30 element array takes at most 5 steps
      - $2^5 > 30$ so at most 5 steps

Outline

fig06_18.c (Part 1 of 3)

```c
/* Fig. 6.18: fig06_18.c
   Linear search of an array */
#include <stdio.h>
#define SIZE 100

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

/* function main begins program execution */
int main()
{
   int a[ SIZE ]; /* create array a */
   int x;         /* counter */
   int searchKey; /* value to locate in a */
   int element;   /* variable to hold location of searchKey or -1 */

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

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

```

Outline

fig06_18.c (Part 2 of 3)

```
   /* 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 );
   } /* end if */
   else {
      printf( "Value not found\n" );
   } /* end else */

   return 0; /* indicates successful termination */

} /* 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 ) {

```

Outline

```
      if ( array[ n ] == key ) {
         return n; /* return location of key */
      } /* end if */

   } /* end for */

   return -1; /* key not found */

} /* end function linearSearch */
```

fig06_18.c (Part 3 of 3)

Program Output

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


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

Outline

fig06_19.c (Part 1 of 5)

```c
/* 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 );

/* function main begins program execution */
int main()
{
   int a[ SIZE ]; /* create array a */
   int i;         /* counter */
   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;
   } /* end for */

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

```

Outline

fig06_19.c (Part 2 of 5)

```
   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 );
   } /* end if */
   else {
      printf( "\n%d not found\n", key );
   } /* end else */

   return 0; /* indicates successful termination */

} /* end main */

/* 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 */

```

Outline

fig06_19.c (Part 3 of 5)

```
   /* 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 */

```

Outline

fig06_19.c (Part 4 of 5)

```
   return -1;   /* searchKey not found */

} /* end function binarySearch */

/* 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 );
   } /* end for */

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

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

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

```

Outline

fig06_19.c (Part 5 of 5)

```
/* 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 */

   /* 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

```
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

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
```

Outline

Program Output (continued)

```
Enter a number between 0 and 28: 6

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

6 found in array element 3
```

6.9 Multiple-Subscripted Arrays

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

	Column	Column 1	Column	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Column subscript

Array name

Row subscript

6.9 Multiple-Subscripted Arrays

- Initialization
  - `int b[2][2]={{1,2},{3,4}};`
  - Initializers grouped by row in braces
  - If not enough, unspecified elements set to zero

    `int b[2][2]={{1},{3,4}};`

1	2
3	4

1	0
3	4

- Referencing elements
  - Specify row, then column

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

Outline

fig06_21.c (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 */

/* function main begins program execution */
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 );

   return 0; /* indicates successful termination */

} /* end main */

```

Outline

fig06_21.c (Part 2 of 2)

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

Outline

fig06_22.c (Part 1 of 6)

```c
/* Fig. 6.22: fig06_22.c
   Double-subscripted 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 );

/* function main begins program execution */
int main()
{
   int 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 } };

```

Outline

fig06_22.c (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 - 1; student++ ) {
      printf( "The average grade for student %d is %.2f\n",
         student, average( studentGrades[ student ], EXAMS ) );
   } /* end for */

   return 0; /* indicates successful termination */

} /* end main */

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

```

Outline

fig06_22.c (Part 3 of 6)

```
   /* 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 */

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

```

Outline

fig06_22.c (Part 4 of 6)

```
   /* 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 */

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

```

Outline

fig06_22.c (Part 5 of 6)

```
   /* 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 */

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

```

Outline

fig06_22.c (Part 6 of 6)

```
      /* 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 */
```

```
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
```