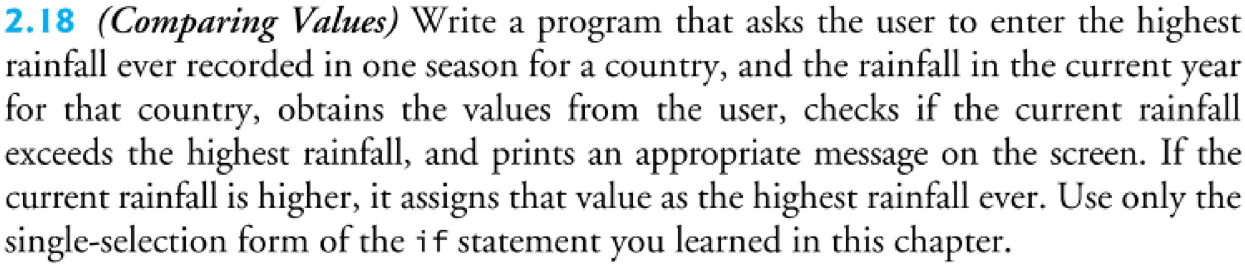
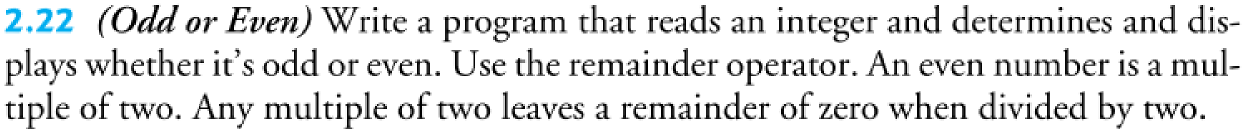
King Saud University **Procedural Programming** **CSC215**
Collage of Computer and Information Sciences **2023 - Semester 451**
Computer Science Department **Lab Tutorial** **Before Exam**



2.18 *(Comparing Values)* Write a program that asks the user to enter the highest rainfall ever recorded in one season for a country, and the rainfall in the current year for that country, obtains the values from the user, checks if the current rainfall exceeds the highest rainfall, and prints an appropriate message on the screen. If the current rainfall is higher, it assigns that value as the highest rainfall ever. Use only the single-selection form of the `if` statement you learned in this chapter.

```
// Exercise 2.18 Solution
#include <stdio.h>

int main(void) {
   int x = 0; // define first number
   int y = 0; // define second number
   printf("%s", "Enter two numbers: "); // prompt
   scanf("%d%d", &x, &y); // read two integers

   // compare the two numbers
   if (x > y) {
      printf("%d is larger\n", x);
   }
   if (x < y) {
      printf("%d is larger\n", y);
   }
   if (x == y) {
      puts("These numbers are equal");
   }
}
```



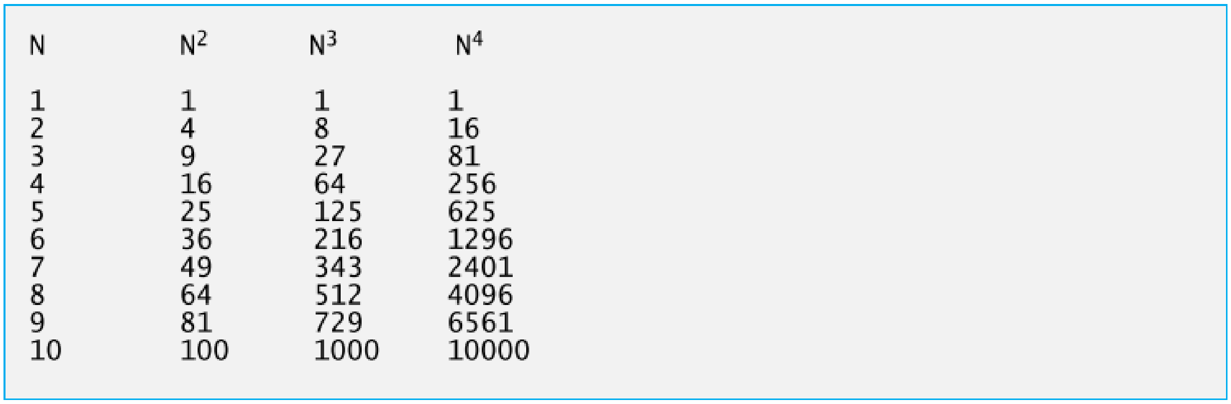
2.22 *(Odd or Even)* Write a program that reads an integer and determines and displays whether it's odd or even. Use the remainder operator. An even number is a multiple of two. Any multiple of two leaves a remainder of zero when divided by two.

```
// Exercise 2.22 Solution
#include <stdio.h>

int main(void) {
   int integer = 0; // integer input by user

   printf("%s", "Input an integer: "); // prompt
   scanf("%d", &integer); // read integer

   // test if integer is even
   if (integer % 2 == 0) {
      printf("%d is an even integer\n", integer);
   }
   // test if integer is odd
   if (integer % 2 != 0) {
      printf("%d is an odd integer\n", integer);
   }
}
```

2.27 *(Summing the Digits of an Integer)* Write a program that inputs one 4-digit number, sums each of the individual digits, and displays the result. [*Hint*: Use division and remainder operation]. For example, if the input is 3581, the output should be 17. (Explanation: 3 + 5 + 8 + 1 = 17).

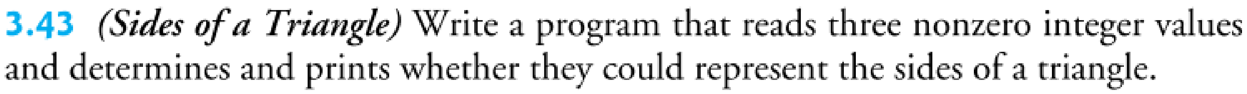
```c
#include <stdio.h>

int main() {
  int n, sum = 0, remainder;

  printf("Enter a 4-digit integer: ");
  scanf("%d", &n);
  int temp = n;

  remainder = n % 10;
  sum += remainder;
  n /= 10;

  remainder = n % 10;
  sum += remainder;
  n /= 10;

  remainder = n % 10;
  sum += remainder;
  n /= 10;

  remainder = n % 10;
  sum += remainder;
  n /= 10;


  n = temp;
  printf("The sum of the digits
of %d is %d\n", n, sum);
  return 0;
}

/** using loop **/

#include <stdio.h>

int main() {
  int n, sum = 0, remainder;

  printf("Enter a 4-digit integer: ");
  scanf("%d", &n);
  int temp = n;

  while (n > 0) {
    remainder = n % 10;
    sum += remainder;
    n /= 10;
  }

  n = temp;

  printf("The sum of the digits of %d
is %d\n", n, sum);

  return 0;
}
```

3.24 *(Tabular Output)* Write a program that uses looping to print the following table of values. Use the tab escape sequence, \t, in the printf statement to separate the columns with tabs.



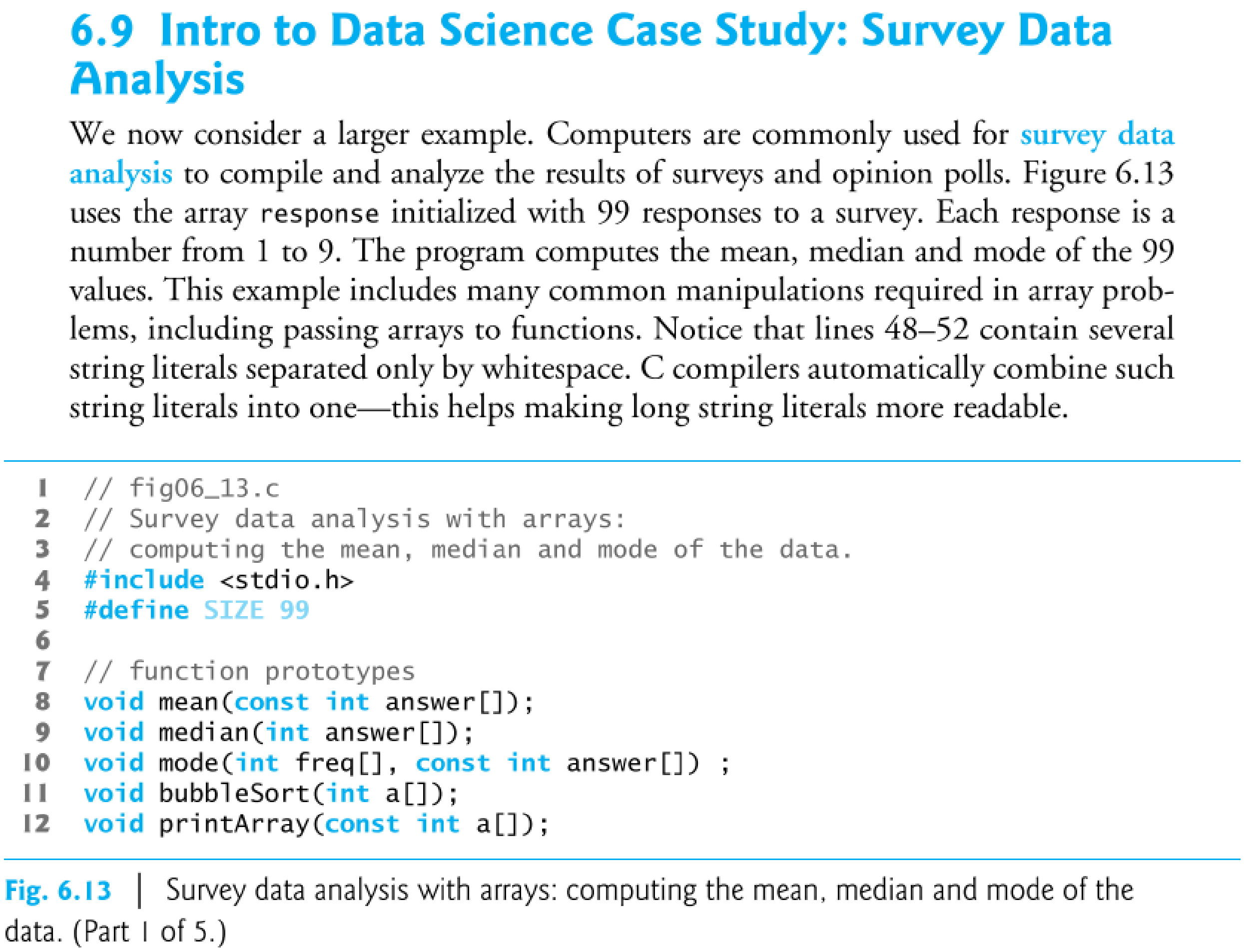
```
N         N^2       N^3       N^4

1         1         1         1
2         4         8         16
3         9         27        81
4         16        64        256
5         25        125       625
6         36        216       1296
7         49        343       2401
8         64        512       4096
9         81        729       6561
10        100       1000      10000
```

```c
// Exercise 2.27 Solution
#include <stdio.h>

int main(void) {
   int count = 0; // initialize count to zero

   // calculate the squares and cubes for the numbers 0 to 10
   puts("number\tsquare\tcube\tN^4");
   printf("%d\t%d\t%d\t%d\n", count, count * count,
      count * count * count, count * count * count * count);

   count = count + 1; // increment count by 1
   printf("%d\t%d\t%d\t%d\n", count, count * count,
      count * count * count, count * count * count * count);
   count = count + 1;
   printf("%d\t%d\t%d\t%d\n", count, count * count,
      count * count * count, count * count * count * count);
   count = count + 1;
   printf("%d\t%d\t%d\t%d\n", count, count * count,
      count * count * count, count * count * count * count);
   count = count + 1;
   printf("%d\t%d\t%d\t%d\n", count, count * count,
      count * count * count, count * count * count * count);
   count = count + 1;
   printf("%d\t%d\t%d\t%d\n", count, count * count,
      count * count * count, count * count * count * count);
   count = count + 1;
   printf("%d\t%d\t%d\t%d\n", count, count * count,
      count * count * count, count * count * count * count);
   count = count + 1;
   printf("%d\t%d\t%d\t%d\n", count, count * count,
      count * count * count, count * count * count * count);
   count = count + 1;
   printf("%d\t%d\t%d\t%d\n", count, count * count,
      count * count * count, count * count * count * count);
   count = count + 1;
   printf("%d\t%d\t%d\t%d\n", count, count * count,
      count * count * count, count * count * count * count);
   count = count + 1;
   printf("%d\t%d\t%d\t%d\n", count, count * count,
      count * count * count, count * count * count * count);
}

/** using loop **/

#include <stdio.h>

int main(void) {
   int count;

   // calculate the squares and cubes for the numbers 0 to 10
   puts("number\tsquare\tcube\tN^4");
   for(count = 0; count<=10; count++){
      printf("%d\t%d\t%d\t%d\n", count, count * count,
         count * count * count, count * count * count * count);
   }
}
```



3.43 *(Sides of a Triangle)* Write a program that reads three nonzero integer values and determines and prints whether they could represent the sides of a triangle.

```
// Exercise 3.43 Solution
#include <stdio.h>

int main(void) {
   int a = 0; // first number
   int b = 0; // second number
   int c = 0; // third number

   // input 3 numbers
   printf("%s", "Enter three non-zero integers: ");
   scanf("%d%d%d", &a, &b, &c);

   // check whether the sum of any two sides is shorter than the third
   if (a + b < c) {
      puts("The three integers cannot be the sides of a triangle");
   }
   else if (b + c < a) {
      puts("The three integers cannot be the sides of a triangle");
   }
   else if (c + a < b) {
      puts("The three integers cannot be the sides of a triangle");
   }
   else {
      puts("The three integers could be the sides of a  triangle ");
   }
}
```

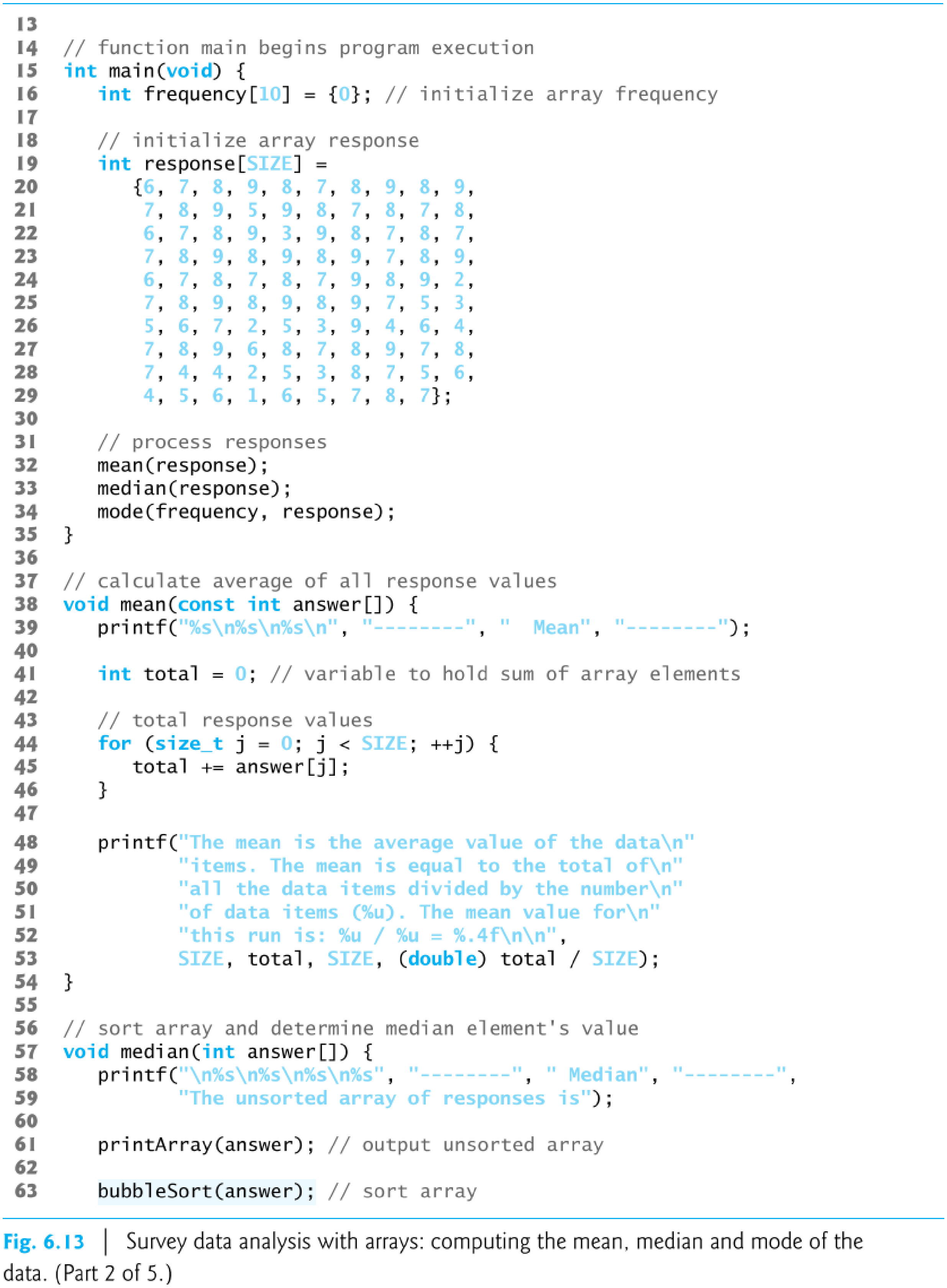


6.9 Intro to Data Science Case Study: Survey Data Analysis

We now consider a larger example. Computers are commonly used for survey data analysis to compile and analyze the results of surveys and opinion polls. Figure 6.13 uses the array `response` initialized with 99 responses to a survey. Each response is a number from 1 to 9. The program computes the mean, median and mode of the 99 values. This example includes many common manipulations required in array problems, including passing arrays to functions. Notice that lines 48–52 contain several string literals separated only by whitespace. C compilers automatically combine such string literals into one—this helps making long string literals more readable.

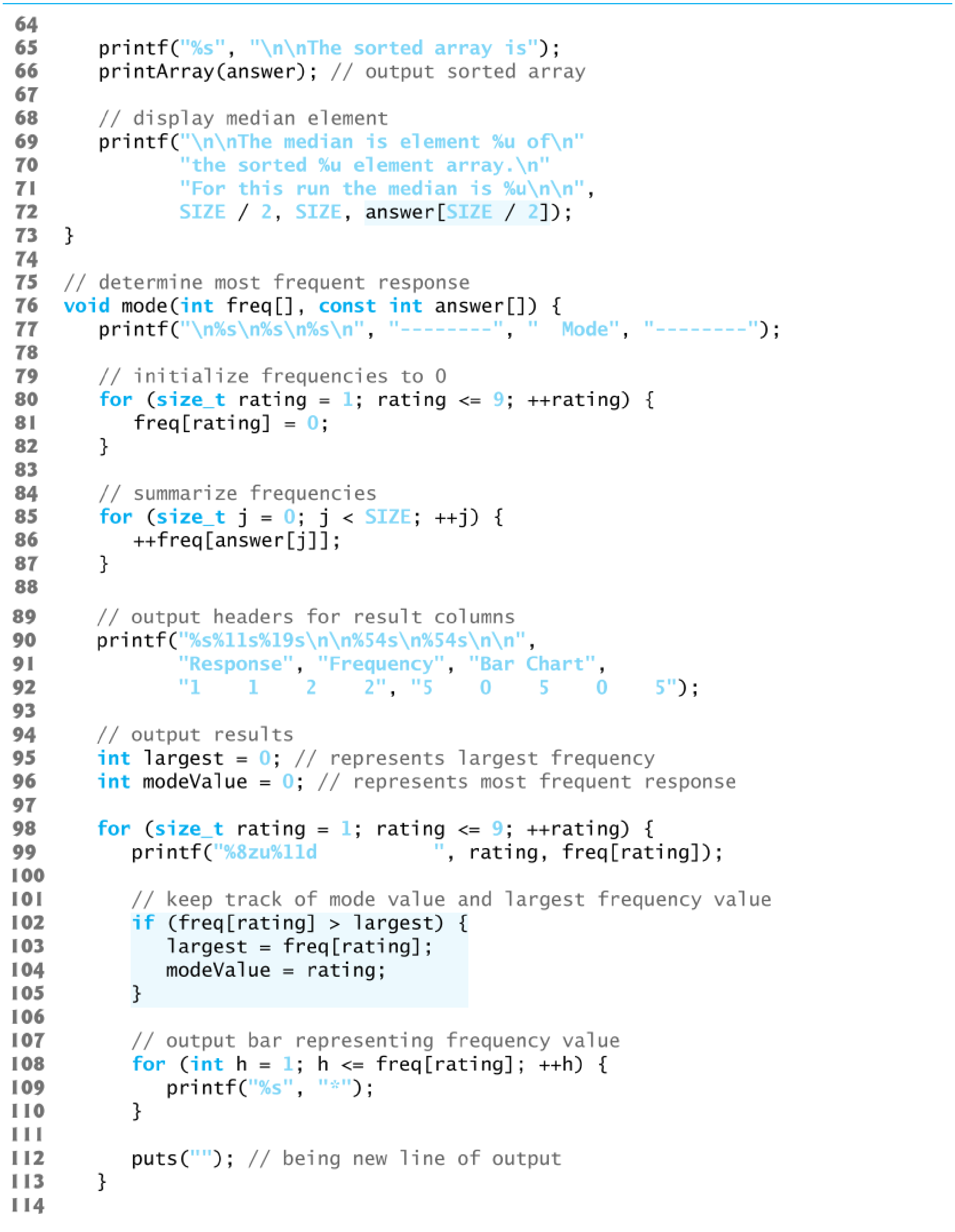
```
 1  // fig06_13.c
 2  // Survey data analysis with arrays:
 3  // computing the mean, median and mode of the data.
 4  #include <stdio.h>
 5  #define SIZE 99
 6
 7  // function prototypes
 8  void mean(const int answer[]);
 9  void median(int answer[]);
10  void mode(int freq[], const int answer[]) ;
11  void bubbleSort(int a[]);
12  void printArray(const int a[]);
```

Fig. 6.13 | Survey data analysis with arrays: computing the mean, median and mode of the data. (Part 1 of 5.)



```c

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

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

// calculate average of all response values
void mean(const int answer[]) {
   printf("%s\n%s\n%s\n", "--------", "  Mean", "--------");

   int total = 0; // variable to hold sum of array elements

   // total response values
   for (size_t 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 (%u). The mean value for\n"
          "this run is: %u / %u = %.4f\n\n",
          SIZE, total, SIZE, (double) total / SIZE);
}

// 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
```

Fig. 6.13 | Survey data analysis with arrays: computing the mean, median and mode of the data. (Part 2 of 5.)



```c

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

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

// determine most frequent response
void mode(int freq[], const int answer[]) {
   printf("\n%s\n%s\n%s\n", "--------", "  Mode", "--------");

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

   // summarize frequencies
   for (size_t 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", "Bar Chart",
          "1    1    2    2", "5    0    5    0    5");

   // output results
   int largest = 0; // represents largest frequency
   int modeValue = 0; // represents most frequent response

   for (size_t rating = 1; rating <= 9; ++rating) {
      printf("%8zu%11d          ", rating, freq[rating]);

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

      // output bar representing frequency value
      for (int h = 1; h <= freq[rating]; ++h) {
         printf("%s", "*");
      }

      puts(""); // being new line of output
   }

```

Fig. 6.13 | Survey data analysis with arrays: computing the mean, median and mode of the data. (Part 3 of 5.)

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

// function that sorts an array with bubble sort algorithm
void bubbleSort(int a[]) {
   // loop to control number of passes
   for (int pass = 1; pass < SIZE; ++pass) {
      // loop to control number of comparisons per pass
      for (size_t j = 0; j < SIZE - 1; ++j) {
         // swap elements if out of order
         if (a[j] > a[j + 1]) {
            int hold = a[j];
            a[j] = a[j + 1];
            a[j + 1] = hold;
         }
      }
   }
}

// output array contents (20 values per row)
void printArray(const int a[]) {
   // output array contents
   for (size_t j = 0; j < SIZE; ++j) {

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

      printf("%2d", a[j]);
   }
}
```

```
--------
  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

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


--------
  Mode
--------
Response  Frequency          Bar Chart

                                      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.
```

Mean

The mean is the arithmetic average of the 99 values. Function `mean` (lines 38–54) computes the mean by totaling the 99 elements and dividing the result by 99.

Median

The median is the middle value. Function `median` (lines 57–73) first sorts the responses by calling function `bubbleSort` (defined in lines 122–135). Then it determines the median by picking the sorted array's middle element, `answer[SIZE / 2]`. When the number of elements is even, the median should be calculated as the mean of the two middle elements—function `median` does not currently provide this capability. Lines 61 and 66 call function `printArray` (lines 138–148) to output the `response` array before and after the sort.

Mode

The mode is the value that occurs most frequently among the 99 responses. Function `mode` (lines 76–119) determines the mode by counting the number of responses of each type, then selecting the value with the greatest count. This version of function `mode` does not handle a tie (see Exercise 6.14). Function `mode` also produces a bar chart to aid in determining the mode graphically.

```c
// Exercise 6.14 Solution
#include <stdio.h>
#define SIZE 100


void mean(int answer[]);              // function prototype
void median(int answer[]);            // function prototype
void mode(int freq[], int answer[]); // function prototype


int main(void) {

   // array of responses
   int response[SIZE] = {6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
                         7, 8, 9, 5, 9, 8, 7, 8, 7, 1,
                         6, 7, 8, 9, 3, 9, 8, 7, 1, 7,
                         7, 8, 9, 8, 9, 8, 9, 7, 1, 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, 1,
                         7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
                         4, 5, 6, 1, 6, 5, 7, 8, 7, 9};
   int frequency[10] = {0}; // array of response frequencies


   mean(response); // process mean
   median(response); // process median
   mode(frequency, response); // process mode
}


// calculate average of all response values
void mean(int answer[]) {
   printf("%s\n%s\n%s\n", "******", " Mean", "******");
   int total = 0; // total of all response values

   // total response values
   for (int j = 0; j <= SIZE - 1; j++) {
      total += answer[j];
   }

   // output results
   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). ", SIZE);
   printf("The mean value for this run is: "
          "%d / %d = %.4f\n\n", total, SIZE, (double) total / SIZE);
}
```

```
// sort an array and determine median element's value
void median(int answer[]) {
   printf("\n%s\n%s\n%s\n", "******", "Median", "******");
   puts("The unsorted array of responses is");
   // display unsorted array
   for (int loop = 0, firstRow = 1; loop <= SIZE - 1; loop++) {
      // start a new line
      if (loop % 20 == 0 && !firstRow) {
         printf("\n");
      }
      printf("%2d", answer[loop]);
      firstRow = 0;
   }

   printf("\n\n");
   // sort array
   for (int pass = 0; pass <= SIZE - 2; pass++) {
      // compare elements and swap if necessary
      for (int loop = 0; loop <= SIZE - 2; loop++) {

         // swap elements
         if (answer[loop] > answer[loop + 1]) {
            int hold = answer[loop];
            answer[loop] = answer[loop + 1];
            answer[loop + 1] = hold;
         }
      }
   }

   puts("The sorted array is");
   // display sorted array
   for (int loop = 0, firstRow = 1; loop <= SIZE - 1; loop++) {
      // start a new line
      if (loop % 20 == 0 && !firstRow) {
         printf("\n");
      }
      printf("%2d", answer[loop]);
      firstRow = 0;
   }

   puts("\n");
   // even number of elements
   if (SIZE % 2 == 0) {
      printf("The median is the average of elements %d", (SIZE + 1) / 2);
      printf(" and %d of", 1 + (SIZE + 1) / 2);
      printf(" the sorted %d element array.\n", SIZE);
      printf("For this run the median is %.1f\n\n",
         (double)(answer[(SIZE + 1) / 2] + answer[(SIZE + 1) / 2 + 1]) / 2);
   }
   else { // odd number of elements
      printf("The median is element %d of ", (SIZE + 1) / 2);
      printf("the sorted %d element array.\n", SIZE);
      printf("For this run the median is %d\n\n", answer[(SIZE + 1) / 2 - 1]);
   }
}
```

```c
// determine most frequent response
void mode(int freq[], int answer[]) {
   printf("\n%s\n%s\n%s\n", "******", " Mode", "******");
   // set all frequencies to 0
   for (int rating = 1; rating <= 9; rating++) {
      freq[rating] = 0;
   }
   // traverse array and increment corresponding frequency
   for (int loop = 0; loop <= SIZE - 1; loop++) {
      ++freq[answer[loop]];
   }
   printf("%s%11s%19s\n\n", "Response", "Frequency", "Histogram");
   printf("%54s\n", "1    1    2    2");
   printf("%54s\n\n", "5    0    5    0    5");
   // display values and frequency
   int largest = 0; // represents largest frequency
   int count = 0; // flag to count number of modes
   int array[10] = {0}; // array used to hold largest frequencies
   for (int rating = 1; rating <= 9; rating++) {
      printf("%8d%11d          ", rating, freq[rating]);
      // test if current frequency is greater than largest frequency
      if (freq[rating] > largest) {
         largest = freq[rating];
         // set values of array to 0
       for (int loop = 0; loop < 10; loop++) {
            array[loop] = 0;
         }
         // add new largest frequency to array
         array[rating] = largest;
         ++count;
      }
      // if current frequency equals largest, add current to array
      else if (freq[rating] == largest) {
         array[rating] = largest;
         ++count;
      }
      // display histogram
      for (int loop = 1; loop <= freq[rating]; loop++)
         printf("%s", "*");
      puts("");
   }
   puts("");
   // if more than one mode
   if (count > 1) {
      printf("%s", "The modes are:  ");
   }
   else { // only one mode
      printf("%s", "The mode is: ");
   }
   // display mode(s)
   for (int loop = 1; loop <= 9; loop++) {
      if (array[loop] != 0) {
         printf("%d with a frequency of %d\n\t\t", loop, array[loop]);
      }
   }
   puts("");
}
```