Solutions to Chapter 8

Review Questions

1. a. True

3. b. False

5. b. False

7. c. index

9. d. `ary[0] = x;`

11. e. sorting

13. e. sequential

15. a. A two-dimensional array can be thought of as an array of one-dimensional arrays.

Exercises

17.

```
2
2
1
2
```

19.

```
first pass:     7   8   13   44   26   23   98   57
second pass:    7   8   13   23   26   44   98   57
third pass:     7   8   13   23   26   44   98   57
```

21.

```
first pass:     3   7   13   26   44   23   98   57
second pass:    3   7   13   26   44   23   98   57
third pass:     3   7   13   26   44   23   98   57
```

23. Bubble sort, because the two smallest elements have been bubbled to the front.

25.

```
FIRST          LAST           MID
  0             7              3          // 88 > 26
  4             7              5          // 88 > 56
  6             7              6          // Found
  8             7              0          // Exit
```

27. In each pass, the first element of the unsorted sublist is picked up and transferred into the sorted sublist by inserting it at the appropriate place. When it locates the correct position, therefore, the data have already been moved right one position, and the current location is empty. So, the sort simply places the saved element in its proper location.

Problems

29. See Program 8-1.

Program 8-1 Solution to Problem 29

```
/* ================== reverseArray ==================
   This function reverses the elements of an array.
      Pre   an array and its size
      Post  the elements are reversed
*/
void reverseArray (int x[], int size)
{
// Statements
   for (int i = 0, j = size - 1, temp = 0;
        i < j;
        i++, j--)
       {
        temp = x[j];
        x[j] = x[i];
        x[i] = temp;
       } // for
   return;
}  //  reverseArray
```

31. See Program 8-2.

Program 8-2 Solution to Problem 31

```
/* ==================== ISBNtest ====================
   This function tests an ISBN to see if it is valid.
      Pre   array of character for ISBN code
      Post  returns true if valid, false if invalid
*/
int ISBNtest (char code[])
{
// Local Declarations
   int value = 0;
   int sum   = 0;

// Statements
   for (int i = 0, j = 10; i < 10; i++, j--)
     {
      // when the 10th digit (code[9]) is 'x')
      if (i == 9 && code[i] == 'x')
          value = 10 * j;
      else
         // (ASCII - 48) is value of numeric characters
         value = ((int) code[i] - 48) * j;
      sum += value;
     } // for

   // Verification code: Remove for production
   printf ("\t\t\t  Weighted Sum: %3d\n", sum);

   return ((sum % 11) == 0);
}  //  ISBNtest
```

33. See Program 8-3.

Program 8-3 Solution to Problem 33

```
/* ================= convertArray ===================
   This function copies a 1-dimensional array of n
   elements into  a 2-dimensional array of k rows and j
```

Program 8-3 Solution to Problem 33 (continued)

```
   columns. (Note: assume that MAX_ROW and MAX_COL are
   defined at global declarations.)
      Pre  number of elements in one-dimensional array
           number of rows in two-dimensional array
           number of columns in two-dimensional array
           one-dimensional array
           two-dimensional array
      Post Returns 0 if array cannot be copied
           1 if copied.
*/
int convertArray (int elements,
                  int toRow,
                  int toCol,
                  int array1[],
                  int array2[][MAX_COL])
{
// Statements
   if (elements != toRow * toCol)
       return 0;

   for (int from = 0, row = 0; row < toRow; row++)
       for (int col = 0; col < toCol; col++, from++)
           array2[row][col] = array1[from];
   return 1;
}  //  convertArray
```

35. See Program 8-4.

Program 8-4 Solution to Problem 35

```
/* This program repeats Problem 34 as a binary search.
      Written by:
      Date:
*/

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>

const int cElements = 100;
const int cSearches = 100;

// Function Declarations
   void bubbleSort    (int  list[],  int  last);
   bool binarySearch  (int  list[],
                       int  end,     int target,
                       int* locn,    int* compares);

int main (void)
{
// Local Declarations
   int ary [cElements];
   int locn;
   int target;
   int success     = 0;
   int compares    = 0;
   int numSearches;

// Statements
   printf ("*** Start of Binary Search ***\n\n");

   srand (time (NULL));

   for (int index = 0; index < cElements; index++)
        ary [index] = rand() % 200 + 1;
```

Program 8-4 Solution to Problem 35 (continued)

```
   bubbleSort (ary, cElements - 1);

   for (numSearches = 0;
        numSearches < cSearches;
        numSearches++)
       {
        target = rand()%200 + 1;
        success += binarySearch (ary,    cElements - 1,
                                 target,
                                 &locn, &compares);
       } // for

   printf ("Searches completed       : %4d\n",
           numSearches);
   printf ("Successful cSearches     : %4d\n",
           success);
   printf ("Percent successful       : %6.1f%%\n",\
           (float) success / cSearches * 100);
   printf ("Total comparisons        : %4d\n",
           compares);
   printf ("Average compares/search  : %6.1f\n",\
           (float) compares / cSearches);

   printf ("\n*** End of Binary Search ***\n\n");
   return 0;
}  //  main

/* ==================== bubbleSort ====================
   Sort list using bubble sort. Adjacent elements are
   compared and exchanged until list is ordered.
      Pre  the list must contain at least one item
           last contains index to last element in list
      Post list rearranged in sequence low to high
*/
void bubbleSort (int list [], int last)
{
// Local Declarations
   int temp;

// Statements
   // Outer loop
   for(int current = 0; current < last; current++)
      {
       // Inner loop: Bubble up one element each pass
       for (int walker = last;
                walker > current;
                walker--)
           if (list[walker]  < list[walker - 1])
              {
               temp             = list[walker];
               list[walker]     = list[walker - 1];
               list[walker - 1] = temp;
              } // if
      } // for current
   return;
}  // bubbleSort

/* ================== binarySearch ==================
   This algorithm searches an ordered array for target.
      Pre  list must contain at least one element
           end is index to largest element in list
           target is  value of element being sought
      Post FOUND    : locn is index to target element
                      returns true (found)
           NOT FOUND: locn is element below or above
                      target--returns false
*/
```

Program 8-4 Solution to Problem 35 (continued)

```
bool binarySearch (int  list[], int  end,
                   int  target,
                   int* locn,   int* compares)
{
// Local Declarations
   int first;
   int mid;
   int last;

// Statements
   first = 0;
   last  = end;
   while (first <= last)
      {
       mid = (first + last) / 2;
       if (++(*compares), target > list[mid])
          // look in upper half
          first = mid + 1;
       else if (++(*compares), target < list[mid])
          // look in lower half
          last = mid - 1;
       else
          // found equal => force exit
          first = last + 1;
      } // while
   *locn = mid;
   return target == list[mid];
}  // binarySearch
```

37. See Program 8-5.

Program 8-5 Solution to Problem 37

```
/* This program is modified from Problem 34 to
   incorporate the new search from Problem 36.
      Written by:
      Date:
*/
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>

const int cElements = 100;
const int cSearches = 100;

// Function Declarations
   bool seqSearch  (int  list[], int  last,  int target,
                    int* locn,   int* compares);
   void bubbleSort (int list[],  int  last);

int main (void)
{
// Local Declarations
   int ary [cElements];
   int locn;
   int target;
   int success   = 0;
   int compares  = 0;
   int numSearches;

// Statements
   printf ("*** Start of Sequential Search ***\n\n");

   srand (time (NULL));
```

Program 8-5 Solution to Problem 37 (continued)

```
   for (int index = 0; index < cElements; index++)
        ary [index] = rand() % 200 + 1;

   bubbleSort (ary, cElements - 1);

   for (numSearches = 0;
        numSearches < cSearches;
        numSearches++)
       {
        target = rand() % 200 + 1;
        success += seqSearch (ary,   cElements-1, target,
                              &locn, &compares);
       } // for

   printf ("Searches completed         : %4d\n",
             numSearches);
   printf ("Successful searches        : %4d\n",
             success);
   printf ("Percent successful         : %6.1f%%\n",
            (float) success / cSearches * 100);
   printf ("Total comparisons          : %4d\n",
             compares);
   printf ("Average comparisons/search : %6.1f\n",
            (float) compares / cSearches);

   printf ("\n*** End of Sequential Search ***\n\n");
   return 0;
}  //  main

/* ==================== bubbleSort ====================
   Sort list using bubble sort. Adjacent elements are
   compared and exchanged until list is ordered.
      Pre  the list must contain at least one item
           last contains index to last element in list
      Post list rearranged in sequence low to high
*/
void bubbleSort (int list [], int last)
{
// Local Declarations
   int temp;

// Statements
   // Outer loop
   for(int current = 0; current < last; current++)
      {
       // Inner loop: Bubble up one element each pass
       for (int walker = last;
                walker > current;
                walker--)
           if (list[walker]  < list[walker - 1])
              {
               temp             = list[walker];
               list[walker]     = list[walker - 1];
               list[walker - 1] = temp;
              } // if
      } // for current
   return;
}  // bubbleSort

/* ================== seqSearch ================
   Locate a target in a sorted list of integers.
     Pre   list must contain at least one item
           last contains index to last element in list
           target contains the data to be located
     Post  FOUND    : matching index stored in locn
                      returns 1 found
           NOT FOUND: last stored in locn address
                      returns zero not found
```

Program 8-5 Solution to Problem 37 (continued)

```
*/
bool seqSearch (int  list[], int  last, int target,
                int* locn,   int* compares)
{
// Local Declarations
   int looker;

// Statements
   if ((*compares)++, target >= list[last])
       looker = last;
   else
       for (looker = 0;
           (*compares)++, target > list[looker];
            looker++)
               ;
   *locn = looker;
   return (target == list[looker]);
}  //  seqSearch
```

39. See Program 8-6.

Program 8-6 Solution to Problem 39

```
/* ==================== bubbleSort ====================
   Sort list using bubble sort. Adjacent elements are
   compared and exchanged until list is ordered.
      Pre  the list must contain at least one item
           last contains index to last element in list
      Post list rearranged in sequence low to high
*/
void bubbleSort (int list [], int last)
{
// Local Declarations
   int  temp;
   bool sorted = false;

// Statements
   // Outer loop
   for(int current = 0;
      (current < last) && !sorted;
       current++)
      {
       sorted = true;
       // Inner loop: Bubble up one element each pass
       for (int walker = last;
                walker > current;
                walker--)
           if (list[walker]  < list[walker - 1])
              {
               sorted = false;
               temp             = list[walker];
               list[walker]     = list[walker - 1];
               list[walker - 1] = temp;
              } // if
      } // for current
   return;
}  // bubbleSort
```

41. See Program 8-7.

Program 8-7 Solution to Problem 41

```
/* This program repeats Problem 40 for the Bubble Sort.
      Written by:
```

Program 8-7 Solution to Problem 41 (continued)

```
      Date:
*/
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>

#define ELEMENTS 50

// Function Declarations
   int  bubbleSort (int list[], int last);
   int  bubbleUp   (int list[], int first, int last);
   void printArray (int ary[],  int size);

int main (void)
{
// Local Declarations
   int ary [ELEMENTS];
   int result;

// Statements
   printf ("*** Start of Bubble Sort ***\n\n");

   srand (time (NULL));
   for (int index = 0; index < ELEMENTS; index++)
        ary [index] = rand() % 200 + 1;

   printf ("Before Sorting\n");
   printArray (ary, ELEMENTS);
   printf ("\n\n");

   result = bubbleSort (ary, ELEMENTS - 1);

   printf ("After Sorting\n");
   printArray (ary, ELEMENTS);
   printf ("\n\n");

   printf ("Number of exchanges: %d\n", result);
   printf ("\n*** End of Bubble Sort ***\n\n");
   return 0;
}  //  main

/* ==================== bubbleSort ====================
   Sort list using bubble sort. Adjacent elements are
   compared and exchanged until list is ordered.
   Returns count of number of exchanges in sort.
      Pre  the list must contain at least one item
           last contains index to last element in list
      Post list rearranged in sequence low to high
*/
int bubbleSort (int list [], int last)
{
// Local Declarations
   int  temp;
   int  exchTotal = 0;
   bool sorted    = false;

// Statements
   // Outer loop
   for(int current = 0;
      (current < last) && !sorted;
       current++)
      {
       sorted = true;
       // Inner loop: Bubble up one element each pass
       for (int walker = last;
                walker > current;
                walker--)
```

Program 8-7 Solution to Problem 41 (continued)

```
            if (list[walker]  < list[walker - 1])
               {
                sorted = false;
                exchTotal++;
                temp              = list[walker];
                list[walker]      = list[walker - 1];
                list[walker - 1] = temp;
               } // if
      } // for current
   return exchTotal;
}  // bubbleSort

/* ================= printArray ===============
   Prints an array of integers, 10 to a line
      Pre   ary is array to print
            size is number of elements in array
      Post  array has been displayed on the monitor
*/
void printArray (int ary[], int size)
{
// Statements
   for (int index = 0; index < size; index++)
      {
       if (!(index % 10))
           printf ("\n");
       printf ("%5d", ary[index]);
      } // for
   printf ("\n");
   return;
}  // printArray
```

43. See Program 8-8.

Program 8-8 Solution to Problem 43

```
/* This program fills right-to-left diagonal of a
   square matrix with zeros, lower right triangle
   with -1s, and upper left triangle with +1s.
      Written by:
      Date:
*/
#include <stdio.h>

#define MAX_SIZE 21

int main (void)
{
// Local Declarations
   int square [MAX_SIZE][MAX_SIZE];
   int size;
   int real_size;

// Statements
   printf ("*** Start of Matrix ***\n\n");

   printf ("Enter the size of square (<= %d): ",
            MAX_SIZE);
   scanf  ("%d", &size);

   if (size > MAX_SIZE)
       size = MAX_SIZE;

   // Fill matrix
   real_size = size - 1;
   for (int row = 0; row < size; row++)
       {
```

Program 8-8 Solution to Problem 43 (continued)

```
        for (int col = 0; col < size; col++)
            {
             if (col <  real_size - row)
                 square [row][col] = 1;
             if (col == real_size - row)
                 square [row][col] = 0;
             if (col >  real_size - row)
                 square [row][col] = -1;
            } // for col
       } // for row

// print the matrix
   printf ("\n\n");
   for (int row = 0; row < size; row++)
       {
        for (int col = 0; col < size; col++)
            printf ("%2d ", square [row][col]);
        printf ("\n");
       } // for row

   printf ("\n\n *** End of Matrix ***\n");
   return 0;
}  //  main
```

45. See Program 8-9.

Program 8-9 Solution to Problem 45

```
/* Test driver for function that sorts parallel arrays.
      Written by:
      Date:
*/
#include <stdio.h>
#include <stdbool.h>

#define ARY_SIZE 10

// Function Declarations
   void insertionSort     (int ary1[],  int ary2[],
                           int last);
   void selectionSort     (int ary1[],  int ary2[],
                           int last);
   void bubbleSort        (int ary1[],  int ary2[],
                           int last);
   void printArrays       (int ary1[],  int ary2[],
                           int size);
int main (void)
{
// Local Declarations
   int IDAry_a []   =
           {18, 237, 35, 5, 76, 103, 189, 22, 156, 49};
   int ScoreAry_a[] =
           {90, 47, 105, 25, 739, 26, 38, 110, 31, 245};
   int IDAry_b []   =
           {18, 237, 35, 5, 76, 103, 189, 22, 156, 49 };
   int ScoreAry_b[] =
           {90, 47, 105, 25, 739, 26, 38, 110, 31, 245};
   int IDAry_c []   =
           {18, 237, 35, 5, 76, 103, 189, 22, 156, 49 };
   int ScoreAry_c[] =
           {90, 47, 105, 25, 739, 26, 38, 110, 31, 245};


// Statements
   printf ("*** Start Parallel Sort ***\n");

   printf ("First, insertion sort:\n\tBefore:\n");
```

Program 8-9 Solution to Problem 45 (continued)

```
   printArrays   (IDAry_a, ScoreAry_a, ARY_SIZE);
   insertionSort (IDAry_a, ScoreAry_a, ARY_SIZE - 1);

   printf ("\tAfter:\n");
   printArrays   (IDAry_a, ScoreAry_a, ARY_SIZE);

   printf ("\nNext, selection sort:\n\tBefore:\n");
   printArrays   (IDAry_b, ScoreAry_b, ARY_SIZE);
   selectionSort (IDAry_b, ScoreAry_b, ARY_SIZE - 1);

   printf ("\tAfter:\n");
   printArrays (IDAry_b, ScoreAry_b, ARY_SIZE);

   printf ("\nFinally, bubble sort:\n\tBefore:\n");
   printArrays (IDAry_c, ScoreAry_c, ARY_SIZE);
   bubbleSort  (IDAry_c, ScoreAry_c, ARY_SIZE - 1);

   printf ("\n\tAfter:\n");
   printArrays (IDAry_c, ScoreAry_c, ARY_SIZE);

   printf ("\n\n*** End Parallel Sort ***\n");
   return 0;
}  // main

/* ================= insertionSort ==================
   Sort parallel arrays using insertion sort.
      Pre    ary1 & ary2 must contain at least one item
      Post   arrays have been sorted
*/
void insertionSort (int ary1[], int ary2[], int last)
{
// Local Declarations
   int  walk;
   int  temp1;
   int  temp2;
   bool located;

// Statements
   for (int current = 1; current <= last; current++)
      {
       located = 0;
       temp1   = ary1[current];
       temp2   = ary2[current];

          for (walk = current - 1;
               walk >= 0 && !located;
              )
             if (temp1 < ary1[walk])
                {
                 ary1[walk + 1] = ary1[walk];
                 ary2[walk + 1] = ary2[walk];
                 walk--;
                } // if
             else
                 located = 1;
       } // for current

   ary1 [walk + 1] = temp1;
   ary2 [walk + 1] = temp2;

   return;
}  // insertionSort

/* ================== selectionSort ================
   Sort parallel arrays by selecting smallest element
   in unsorted  portion of array and exchanging it with
   element at the beginning of the unsorted list.
      Pre   ary1 & ary2 must contain at least one item
```

Program 8-9 Solution to Problem 45 (continued)

```
            last is index to last element in list
      Post  arrays rearranged smallest to largest.
*/
void selectionSort (int ary1[], int ary2[], int last)
{
// Local Declarations
   int smallest;
   int tempData;

// Statements
   for (int current = 0; current < last; current++)
       {
        smallest = current;
        // Inner Loop: One sort pass each loop
        for (int walk = current + 1;
                 walk <= last;
                 walk++)
            if (ary1[walk] < ary1[smallest])
                smallest = walk;

        // Smallest selected: exchange with current
        if (current != smallest)
           {
            // smallest selected: exchange with current
            tempData       = ary1[current];
            ary1[current]  = ary1[smallest];
            ary1[smallest] = tempData;
            tempData       = ary2[current];
            ary2[current]  = ary2[smallest];
            ary2[smallest] = tempData;
           } // if
        } // for current
   return;
}  // selectionSort

/* ================== bubbleSort ================
   Sort parallel arrays using bubble sort.
      Pre   arrays must contain at least one item
            last contains index to last element in lists
      Post  lists rearranged in sequence low to high
*/
void bubbleSort (int ary1[], int ary2[], int last)
{
//Local Declarations
   int temp;

// Statements
   for (int current = 0; current < last; current++)
      for(int current = 0; current < last; current++)
         {
          // Inner loop: Bubble up one element each pass
          for (int walker = last;
                   walker > current;
                   walker--)
              if (ary1[walker]  < ary1[walker - 1])
                 {
                  temp             = ary1[walker];
                  ary1[walker]     = ary1[walker - 1];
                  ary1[walker - 1] = temp;
                  temp             = ary2[walker];
                  ary2[walker]     = ary2[walker - 1];
                  ary2[walker - 1] = temp;
                 } // if
      } // for current
   return;
}  //  bubbleSort
```

Program 8-9 Solution to Problem 45 (continued)

```
/* ================ printArrays ===============
   Prints two parallel arrays of integers.
      Pre   ary1 and ary2 are arrays to print
            size is number of elements in arrays
      Post  arrays displayed on the monitor
*/
void printArrays (int ary1[], int ary2[], int size)
{

// Statements
   printf ("\n\tID\t\tScore\n");
   printf ("\t==\t\t=====\n");

   for (int index = 0; index < size; index++)
     {
      if (!(index % 10))
          printf ("\n");
      printf ("\t%4d\t\t%5d\n",
               ary1[index], ary2[index]);
     } // for

   printf ("\n");
   return;
}  // printArrays
```