8 Arrays

REVIEW QUESTIONS

1. The type of all elements in an array must be the same.

a. True

3. When an array is defined, C++ automatically sets the value of its elements to zero.

b. False

5. Because of its efficiency, the binary search is the best search for any array, regardless of its size and order.

b. False

7. A(n) _____ is an integral value used to access an element in an array.

e. index

9. Which of the following statements assigns the value stored in *x* to the first element on an array, *ary*?

d. ary[0] = x;

11. The process through which data are arranged according to their values is known as

d. sorting

13. The _____ search locates the target item by starting at the beginning and moving toward the end of the list.

d. sequential

15. Which of the following statements about two-dimensional arrays is true?

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

b. Only the size of the second dimension needs to be declared when the array is used as a parameter.

EXERCISES

17. The program prints the following on separate lines

2 2 1 2

19. Using a selection sort, after three more passes the array contains:

after first pass: 7 8 13 44 26 23 98 57

after second pass: 7 8 13 23 26 44 98 57

after third pass: 7 8 13 23 26 44 98 57

21. Using an insertion sort, after three more passes the array contains:

after first pass: 3 7 13 26 44 23 98 57

after second pass: 3 7 13 26 44 23 98 57

after 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. The tracing of a binary search for 88 is:

FIRST	LAST	MID	Comments
0	7	3	88 > 26
4	7	5	88 > 56
6	7	6	88 == 88 (found)
8	7	6	Terminates

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 has 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.

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

31.

```
/*  ==================== ISBN_test ===============
    This function tests an ISBN to see if it is valid.
       Pre  TheISBN code to be checked (an array)
       Post Returns true if valid, false if invalid
        Note Requires <cctype> library
```

```
*/
bool ISBN_test (char code[])
{
   int i;
   int j;
   int value = 0;
   int sum   = 0;

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

   // Verification code : Remove for production
   cout << "\t\t\t  Weighted Sum :   " << sum << endl;

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

33.

```
/*  ================== convert_array ============
   Copies a 1-dimentional array of n elements
   into a 2-dimentional array of k rows and j columns.
   Note: assumes that MAX_COL is a global constants
      Pre  The one-dimentional array
           Number of elements in one-dimentional array
           The two-dimentional array
           Number of rows in two-dimentional array
           Number of columns in two-dimentional array
      Post Returns false if array cannot be created,
           Returns true if created
*/
bool convert_array (int array1[], int one_size,
                    int array2[][MAX_COL],
                    int to_row,  int to_col)
{
   if (one_size  !=  to_row * to_col)
      return false;

   int from;
   int row;
   int col;
   for (from = 0, row = 0;  row < to_row;  row++)
        for (col = 0;  col < to_col;  col++, from++)
             array2[row][col] = array1[from];
   return true;
}  // convert_array
```

35.

```
/*  This program creates array of 150 random integers in
   the range 1 to 200. Then, using the binary search,
   searches the array 200 times using randomly
   generated targets in the same range.
      Written by:
```

```
      Date:
*/
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

const int ELEMENTS = 150;
const int SEARCHES = 200;

void bubble_sort   (int  ary[],  int last);
void bubble_up     (int  list[], int first, int last);
bool binary_search (int  ary[],  int end, int target,
                    int& locn,   int& tests);

int main ()
{
   cout << "\n*** start of program ***\n\n";
   int   ary [ELEMENTS];
   srand (time (NULL));
   for (int i = 0;  i < ELEMENTS;  i++)
       ary[i]  =  rand() % 200  +  1;

   bubble_sort (ary, ELEMENTS - 1);

   int   index;
   int   success     =  0;
   int   test_count  =  0;
   bool  result;
   int   target;
   for (int i = 0;  i < SEARCHES;  i++)
       {
        target = rand() % 200  +  1;
        result = binary_search (ary,
                                ELEMENTS - 1,
                                target,
                                index,
                                test_count);
       if (result == true)
           success++;
      } // for

   cout << "\nThe number of searches completed  :  "
        << SEARCHES << endl;
   cout << "\nThe number of successful searches :  "
        << success << endl;
   cout << "\nThe percent of successful searches:  "
        << ((double) success / SEARCHES * 100)
        << '%' << endl << endl;
   cout << "The average number to tests per search:  "
        << ((double) test_count / SEARCHES) << endl;

   cout << "\n***  end  of program ***\n\n";
   return 0;
}  // main

/* ================ bubble_sort ==============
   Sort list using bubble sort, Adjacent elements are
   compared & exchanged until list is ordered.
      Pre  list must contain at least one item
           last is index to last element in list
```

```
      Post list is in ascending sequence
*/
void bubble_sort (int list[], int last)
{
   for (int current = 0; current <= last; current++)
        bubble_up (list, current, last);
    return;
}  // bubble_sort

/* ============== bubble_up ===============
   Move lowest element in unsorted portion of an array
   to the current element in the unsorted portion.
      Pre  list must contain at least one item
           current is beginning of unsorted data
           last is the unsorted data
      Post Array segment rearranged so that lowest
             element is at beginning of unsorted data
*/
void bubble_up (int list[], int current, int last)
{
   int  temp;
   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
   return;
}  // bubble_up
/*  ================ binary_search =================
   This algorithm searches an orderd array for target.
      Pre  list must contain at least one element
           end is index to last (largest) element
           target is value of element being sought
      Post FOUND     : locn is index to target
                        returns true(found)
           NOT FOUND : locn undeterminable
                       returns false (not found)
*/
bool binary_search (int  list[], int  end, int target,
                    int& locn,   int& tests)
{
   int first = 0;
   int last  = end;
   int mid;
   while (first <= last)
      {
       mid = (first + last) / 2;
       if (tests++, target > list[mid])
           // look in upper half
           first = mid + 1;
       else if (tests++, 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]);
}   // binary_search
```

37.

```
/*  This program creates an array of 100 random integers
    in the range 1-200. Then, using the ordered list
    search, searches the array 200 times using randomly
    generated targets in the same range.
       Written by:
       Date:
*/
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

const int ELEMENTS = 100;
const int SEARCHES = 200;

void bubble_sort (int  list[], int  last);
void bubble_up   (int  list[], int  current,
                  int  last);
bool seq_search  (int  list[], int   last, int target,
                  int& locn,   long& tests);

int main ()
{
   cout << "\n*** start of program ***\n\n";
   srand (time (NULL));
   int  ary [ELEMENTS];
   for (int i = 0; i < ELEMENTS; i++)
        ary [i]  =  rand() % 200 + 1;

   bubble_sort (ary, ELEMENTS - 1);

   long test_count  = 0;
   int  success     = 0;
   int  locn;
   int  target;
   bool result;
   for (int i = 0; i < SEARCHES; i++)
       {
        target = rand() % 200 + 1;
        result = seq_search (ary,    ELEMENTS - 1,
                             target, locn,
                             test_count);
        if (result == true)
             success++;
       } // for

   cout << "\nNumber of searches completed       : "
        << SEARCHES << endl;
   cout << "\nNumber of successful searches      : "
        << success << endl;
   cout << "\nPercent of successful searches     : "
        << ((double) success / SEARCHES * 100)
        << '%' << endl;
   cout << "\nAverage number of tests per search : "
       << static_cast<double>(test_count) / SEARCHES;
```

```
    cout << "\n\n***  end  of prb08022.cpp ***\n";
    return 0;
}   // main

/*  ============= bubble_sort ================
    Sort list using bubble sort.
       Pre  The list must contain at least one item
            last is index to last element in list
       Post List has been ordered low to high
*/
void bubble_sort (int list[], int last)
{
    for (int current = 0;  current <= last;  current++)
         bubble_up (list, current, last);
    return;
}   // bubble_sort

/*  =============== bubble_up ================
    Move lowest element in unsorted portion of  array to
    the current element in the unsorted portion.
       Pre  list must contain at least one item
            current is beginning of unsorted data
            last is the unsorted data (index)
       Post Array segment rearranged so that lowest
            element is at beginning of unsorted data
*/
void bubble_up (int list[], int current, int last)
{
    int temp;
    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
    return;
}   // bubble_up

/*  ================== seq_search ===============
    Mdified sequential search to locate the target
    in a sorted list of size elements.  The search
    terminates when the target is less than the current.
       Pre  sorted list containilng at least 1 item
            last is index to last element in the list
            target contains the data to be located
       Post FOUND     : matching index stored in locn
                        returns true (found)
            NOT FOUND : last stored in locn address
                        returns false (not found)
*/
bool seq_search  (int   list[], int  last,
                  int   target, int& locn,
                  long& tests)
{
    int looker;
    if (target > list[last])
         looker = last;
    else
```

```
        for (looker = 0;
             tests++, target > list[looker];
             looker++)
          ;
    locn = looker;
    return (target == list[looker]);
}   // seq_search
```

39. Author's Note: While an interesting insight into efficiency, counting the number of exchanges is not a good measure. You may want to modify this problem to count the number of loops in the sort or to count both the exchanges and the compares needed to order the list.

```
/* Modification of Selection Sort to count the number
   of exchanges needed to order an array of 50
   random numbers.
      Written by:
      Date:
*/
#include <iostream>
#include <iomanip>
#include <ctime>
using namespace std;

const int ELEMENTS = 50;

int selection_sort    (int list[], int last);
int exchange_smallest (int list[], int current,
                       int last);

int main ()
{
   cout << "\n*** start of program ***\n\n";

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

   cout << "\nBefore Sorting :\n";
   for (int i = 0; i < ELEMENTS; i++)
       {
        if (!(i % 10))
           cout << endl;
        cout << setw (5) << ary [i];
       } // for

   int result = selection_sort (ary, ELEMENTS - 1);

   cout << "\n\nAfter Sorting :\n";
   for (int i = 0; i < ELEMENTS; i++)
       {
        if (!(i % 10))
           cout << endl;
        cout << setw (5) << ary[i];
       } // for

   cout << "\n\nThe number of exchanges is :  "
        << result << endl;
   cout << "\n*** end  of program ***\n";
   return 0;
```

```
}   // main

/*  ============== selection_sort =================
    Sorts by selecting smallest element in unsorted
    data and exchanging it with element at beginning
    of the unsorted data. Counts number of exchanges
    to order the array.
       Pre  list must contain at least one item
            last is index to last element in list
       Post list rearranged smallest to largest
*/
int  selection_sort (int list[], int last)
{
    int exchange_total = 0;
    for (int current = 0; current < last; current++)
        {
         exchange_total
           +=  exchange_smallest (list, current, last);
        } // for
    return exchange_total;
}   // selection_sort

/*  ================ exchange_smallest =============
    Given array of integers, place smallest element in
    position in array.
       Pre  list must contain at least one item
            crnt is beginning of array/array segment
            last is last element in array
       Post returns number of exchanges made
*/
int  exchange_smallest (int list[], int crnt,
                        int last)
{
    int exchanges  =  0;

    int smallest = crnt;
    for (int walker = crnt + 1; walker <= last;
walker++)
         if (list[walker] < list[smallest])
             smallest = walker;

    if (crnt != smallest)
       {
        // smallest selected: exchange with current
        int temp_data  = list[crnt];
        list[crnt]     = list[smallest];
        list[smallest] = temp_data;
        exchanges++;
       } // if

    return exchanges;
}   // exchange_smallest
```

41.

```
/*  This program reads data from keyboard, puts it in an
    array, builds a frequency array, and prints the data
    with its histogram.
       Written by:
       Date:
*/
```

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <ctime>
using namespace std;

const int MAX_ELMNTS = 100;
const int ANLYS_RNG  = 20;

void fill_array (int numbers[], int size, int range);
void print_data (int numbers[], int size, int
line_size);

void make_frequency  (int numbers[],   int size,
                      int frequency[], int range);
void make_histogram  (int frequency[], int range);

int main ()
{
   cout << "\n *** start of program ***\n\n";

   // fill array with values slightly out of range
   int nums      [MAX_ELMNTS];
   fill_array (nums, MAX_ELMNTS, ANLYS_RNG + 2);

   print_data (nums, MAX_ELMNTS, 10);

   // for 0 & out of range
   int frequency [ANLYS_RNG + 2];
   make_frequency (nums,      MAX_ELMNTS,
                   frequency, ANLYS_RNG);
   make_histogram (frequency, ANLYS_RNG);

   cout << "\n *** end  of program ***\n\n";
   return 0;
}  // main

/* =============== fill_array ===============
   Fill an array with random numbers within
   a given range.  (0 -> range)
      Pre  data is empty array
           size is the maximum elements in array
           range is  highest value that can be use
      Post array is filled
*/
void fill_array (int data[], int size, int range)
{
   srand (time (NULL));
   for (int i = 0;  i < size;  i++)
        data [i] = rand() % (range + 1);
   return;
}  // fill_array

/* ================== print_data ==================
   Prints the data in an array.
      Pre  data is a filled array
           size is number of elements in the array
           line_size is number of elements per line
      Post data have been printed
*/
void print_data (int data[], int size, int line_size)
```

```
{
    cout << "\n\nData in Array :\n";
    for (int i = 0;  i < size;  i++)
        {
         if (!(i % line_size))
              cout << endl;
         cout << setw (5) << data [i];
        } // for
    cout << "\n\n";
    return;
}   // print_data

/*  =============== make_frequency =========
    Analyzes data in array & build their frequency
    distribution array.
       Pre  nums is array of data to be analyzed
            size is number of elements in nums array
            frequency is accumulation array
       Post frequency array has been built
*/
void make_frequency (int nums[],      int size,
                     int frequency[], int range)
{
//  First initialize the frequency array
    for (int i = 0;  i <= range + 1;  i++)
        frequency[i] = 0;

//  Scan numbers and build frequency array
    for (int i = 0;  i < size;  i++)
        if (nums[i]  <=  range)
           frequency [nums[i]]++;
        else
           frequency [range + 1]++;
    return;
}   // make_frequency

/*  ================ make_histogram ===========
    Print the histogram.
       Pre  freq is times each value occurred in data
            range is value range for frequency array
       Post histogram array has been printed
*/
void make_histogram (int freq[], int range)
{
    for (int i = 0;  i <= range;  i++)
        {
         cout << setw (4) << i << "   "
              << setw (4) << freq[i];
         cout << "   ";
         for (int j = 1;  j <= freq[i]; j++)
             cout << '*';
         cout << endl;
        } // for

    cout << "   !" << "  " << setw (4) << freq[range + 1];
    cout << "   ";
    for (int j = 1;  j <= freq[range + 1];  j++)
        cout << '*';
    cout << endl;
    return;
}   // make_histogram
```