Pointers 9

REVIEW QUESTIONS

1. Pointer constants are drawn from the set of addresses for a computer.

   a. true

3. The underlying type of a pointer is address.

   b. false

5. Which of the following statements about pointers is false?

   a. Pointers are built on the standard type, address.

7. Which of the following statements will not add 1 to a variable?

   d. *p++;

9. Which of the following defines and initializes a pointer to the address of `x`?

   e. int* ptr = &x;

11. Given the following definitions, which answer points to the value stored in `x`?

```
int     x;
int*    p = &x;
int** pp = &p;
```

   d. **pp

13. Which of the following statements about pointer compatibility is true?

   d. When a pointer is cast, C++ automatically reformats the data to reflect the correct type.

15. Given a pointer to an array element, `ptr`, `ptr - 5` is a pointer to the value 5 elements toward the beginning of the array.

   a. true

17. The parameter declaration `int* a` can be used to declare an array of integers passed to a function.

   a. true

19. Which of the following statements about pointers and arrays is true?
    - c. The name of an array can be used with the indirection operator to reference data.

21. Which of the following defines a two-dimensional array of integers?
    - b. int ary[ ][SIZE2]

23. Which of the following statements about ragged arrays is false?
    - c. Ragged arrays can only be used with arrays of integers.

EXERCISES

25.
    - a. true
    - b. false (because a is not a pointer)

27. The values for the following expressions are:
    - a. 6
    - b. 6
    - c. 6
    - d. 6

29. The following program fragments are invalid:
    - a. Invalid results because data read into pointer variable.
    - c. Invalid code. Depending on compiler, will generate a warning or a compile error. Correct code is in b.
    - d. Invalid code. Pointer needs to be dereferenced.

31. The type of each expression is:
    - a. a pointer to a pointer to a pointer to an integer
    - b. a pointer to a pointer to an integer
    - c. a pointer to an integer
    - d. an integer

33. The following statements break the rules for lvalues and rvalues:
    - a. No error as long as p is a pointer variable initialized to point to a variable.
    - b. &a[0] must be stored in p instead of &p.
    - c. q must store (p + 2) instead of &(p + 2).

35. The prototype statement is:

```
int* spin (int& x, long double** py);
```

37. See Figure 9-1.

39. The rewritten expressions are:
    - a. *(tax + 6)
    - b. *(score + 7)
    - c. *(num + 4)
    - d. *(prices + 9)

41. If we interpret the sixth element as ary[5], and if p is pointing to a[3], we can access a[5] by coding:

    *(p + 2)   or   p[2]

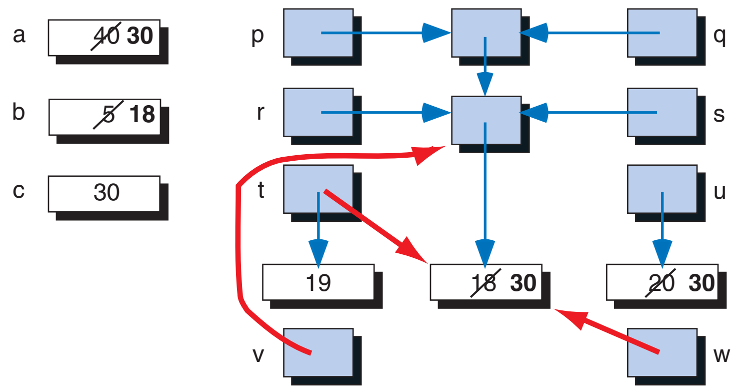

FIGURE 9-1 **Solution to Exercise 37**

43. The output is:

```
6  6
3  4
6  2
4  6
```

45. The prototype statement and call are:

Prototype:

```
void fun (int ary[][5]);
```

Call:

```
fun (table);
```

47. See Figure 9-2.

a. x is an array of five pointers to integer.

b. x is a pointer to an array of five integers.

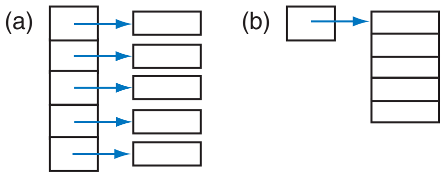

FIGURE 9-2 **Solution to Exercise 47.**

49. The output is:

```
452
769
```

Explanation: For the first call: p is pointing to the whole first row, so (*p) is the first row itself. Then using the first row, (*p)[0] refers to the first element (4), (*p)[1] refers to the second element (5), and (*p)[2] refers to the third element (2).

For the second call: p is pointing to the whole second row (x+1), so (*p) is the second row itself. Then using the second row, (*p)[0] refers to the first element (7), (*p)[1] refers to the second element (6), and (*p)[2] refers to the third element (9).

51. The expression values are:

a. e

b. m

53. The expression values are:

a. 4

b. 4

c. address of i

d. 4

e. address of i

f. address of i) + 8 (if &i is 10,000 and sizeof (int) is 4, then value is 10,032)

55. The equivalent index expressions are:

a. num [2]

b. num [j]

c. num [i + j]

d. num [i] + num [j]

e. num [num [1]]

57. The equivalent index expressions are:

a. &n [0]

b. n[0]

c. n[0] + 1

d. n[1]

e. n[j]

59. The following calls are invalid for the reason shown:

a. not valid: mushem needs two addresses.

b. not valid: mushem needs two addresses

.

61. The output is:

1. 9 4 17

2. 31 17 10 18 7 19 10

PROBLEMS

63.

```
/* This program adds, subtracts, multiplies, and
   divides two numbers using pointers.
        Written by:
        Date:
*/
#include <iostream>
```

```
using namespace std;

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

   cout << "\nEnter the  first number :  ";
   int  a;
   int* pa = &a;
   cin  >> *pa;

   int  b;
   int* pb = &b;
   cout << "\nEnter the second number :  ";
   cin  >> *pb;

   int  r;
   int* pr = &r;
   *pr = *pa + *pb;
   cout << *pa << " + " << *pb << " is " << *pr << endl;

   *pr = *pa - *pb;
   cout << *pa << " - " << *pb << " is " << *pr << endl;

   *pr = *pa * *pb;
   cout << *pa << " * " << *pb << " is " << *pr << endl;

   *pr   = *pa / *pb;

   int  rem;
   int* prem = &rem;
   *prem = *pa % *pb;
   cout << *pa << " / " << *pb << " is " << *pr
        << " with a remainder of " << *prem << endl;

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

65.

```
/* ================= time_convert ==================
   Given time in seconds pass back the time in hours,
   minutes, seconds, and an am/pm character.
      Pre  given the time in seconds
      Post time in hours, minutes, seconds and am/pm
           returns true for successs  false for error
*/
bool time_convert (long  time, int* hour,
                   int*  min,  int* sec,
                   char* am_pm)
{
   *hour = *min = *sec = 0;

   if (time >= 86400)
       return false;

   *sec  = time % 60;
   time /= 60;

   *min  = time % 60;
   time /= 60;

   *hour = time;

   if (*hour < 12)
      *am_pm = 'a';
   else
      {
       *am_pm = 'p';
       *hour -= 12;
      } // else
   return true;
}  // time_convert
```

67.

```
/* ================== gcd_lcm =====================
   This function receives 2 integers and returns
   the G.C.D. and L.C.M.
      Pre  given 2 integers
      Post returns the GCD and LCM
*/
void gcd_lcm (int  num1, int  num2,
              int* gcd,  int* lcm)
{
   int fact1;
   int fact2;
   if (num1 >= num2)
      {
       fact1  =  num1;
       fact2  =  num2;
      } // if
   else
      {
       fact1  =  num2;
       fact2  =  num1;
      } // else

   int rem = fact1 % fact2;
   while (rem != 0)
      {
       fact1 = fact2;
       fact2 = rem;
       rem   = fact1 % fact2;
      } // while

   *gcd = fact2;
   *lcm = (num1 * num2) / *gcd;
   return;
}  // gcd_lcm
```

69.

```
/* ============== arrays_equal ====================
   Compares every element of array 1 equal to its
   corresponding element in array 2.
      Pre  a pointer to array 1 and
           a pointer to array 2
           size is number of elements
      Post returns true if arrays are equal
             returns false if they are different
*/
```

```
bool arrays_equal (int* pAry1, int* pAry2, int size)
{
   bool equal = true;
   int* p1    = pAry1;
   int* pLast = p1 + size - 1;
   int* p2    = pAry2;

   while (equal && p1 <= pLast)
      {
       if (*p1 != *p2)
            equal = false;
       p1++;
       p2++;
      } // for
   return equal;
}  // arrays_equal
```

71.

```
/* ====================== Pascal ==================
   This function creates a ragged array representing a
   Pascal Triangle.
      Pre  size of triangle as an integer
      Post returns pointer to the ragged array
*/
int** Pascal (int size)
{
   int   total_rows;
   if (size < 1)
      total_rows = 1;
   else
      total_rows = size;

   int** p = new int* [total_rows];   // ragged array

   for (int row = 0; row < total_rows; row++)
      *(p + row) =  new int[row + 1];

// Filling the data for size of triangle
   for (int row = 0; row < total_rows; row++)
       {
        for (int col = 0; col <= row; col++)
            {
             if (col == 0 || col == row)
               *(*(p + row) + col)  =  1;
             else
               *(*(p + row) + col)
                = *(*(p + row - 1) + col - 1)
                + *(*(p + row - 1) + col);
            } // for col
       } // for row
   return p;
}  // Pascal
```

73.

```
/* ================ convert_array ==================
   This function copies one-dimentional array of n
   elements to two-dimentional array of k rows and j
   columns.
      Pre  number of elements in one-dimentionl array
           number of rows in two-dimentional array
           mumber of columns in two-dimentional array
           one-dimentional array
      Post if n != j * k, returns null pointer
           else returns pointer to 2-dimentional array
*/
int** convert_array (int total_elements, int to_row,
                     int to_col, int* from_ary)
{
   int** new_ary;
   if (total_elements != to_row * to_col)
        new_ary  =  NULL;
   else
      {
       new_ary = new int* [to_row];
       for (int row = 0; row < to_row; row++)
          *(new_ary + row) = new int [to_col];

       int from_index = 0;
       for (int row = 0; row < to_row; row++)
           {
            for (int col = 0;
                  col < to_col;
                  from_index++, col++)
           *(*(new_ary + row) + col)
                 = *(from_ary + from_index);
           } // for
      } // else
   return new_ary;
}  // convertArray
```

75. Read and fill array with characters

```
for (char* ptr = a;  ptr < a + 40;  ptr++)
   cin >> *ptr;
```

77. Rotate array one element left

```
char* ptr;
char  temp = *a;
for (ptr = a; ptr < a + 5; ptr++)
    *ptr = *(ptr + 1);
*ptr = temp;
```