Solutions to Chapter 9

Review Questions

1. a. True

3. a. True

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

7. e. `*p++;`

9. c. `int* ptr = &x;`

11. e. `**pp`

13. d. When a void pointer is dereferenced, it must be cast.

15. b. Runs efficiently on any hardware.

Exercises

17.

- **a.** True
- **b.** False (because `a` is not a pointer)

19.

- **a.** `6`
- **b.** `6`
- **c.** 6
- **d.** 6

21.

- **a.** Not valid. `p` is an uninitialized pointer to integer; we cannot read a pointer.
- **b.** Not valid. `p` is an uninitialized pointer to integer; while the address operator cancels the dereference, we cannot read an address.
- **c.** Not valid. `p` is an uninitialized pointer to integer. Even if it were initialized, dereferencing it provides an *int*, not an address of an *int*.
- **d.** No error.

23.

- **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

25.

- **a.** No error
- **b.** Error: `&a[0]` must be stored in `p` instead of `&p`.
- **c.** Error: `q` must store `p + 2` (pointer arithmetic) instead of `&(p+2)`.
- **d.** No error

27.

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

29. See Figure 9-1.

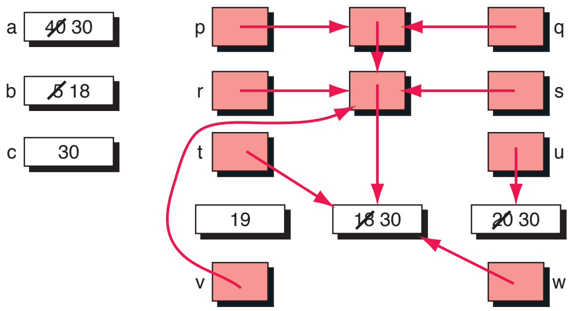

Figure 9-1 Solution for Exercise 29

Problems

31. See Program 9-1.

Program 9-1 Solution to Problem 31

```
/* ==================== julianDate ====================
   This function converts a Julian date to month & day.
      Pre     a julian date
      Post    month and day corresponding julian date
              -or- month & day are zero if date error
      Return  1 if conversion successful; zero if error
*/
int julianDate  (int  julYear,  int  julDay,
                 int* month,    int* day)
{
// Local Declarations
   int days  =  julDay;
   int retval;
   int monthArray [] =
      {0,31,28,31,30,31,30,31,31,30,31,30,31};

// Statements
   retval = 1;

   // check for too few days
   if (days <= 0)
      {
       printf ("\n\aError in JulianDate: ");
       printf ("days must be positive\n");
       retval = 0;
      } // if negative or 0 number of days

   else
      {
       // check for leap year
       if (! (julYear % 4)
          && (julYear % 100)
          || !(julYear % 400))
              monthArray[2]++;
        // Convert the day
       for (int i = 1; i <= 12  &&  days > 0; i++)
          {
           if (days <= monthArray[i])
              {
               *month = i;
               *day   = days;
              } // if
```

Program 9-1 Solution to Problem 31 (continued)

```
           days -= monthArray[i];
          } // for

       if (days > 0)
          {
           printf ("\n\aError in JulianDate: ");
           printf ("too may days\n");
           retval = 0;
          } // if days left over
      } // else positive number of days

       if (!retval)
          {
           *month = 0;
           *day   = 0;
          } // if bad date
   return retval;
}  // julianDate
```

33. See Program 9-2.

Program 9-2 Solution to Problem 33

```
/* ====================== change ======================
   This function receives a floating-point number
   representing the change from a purchase and
   determines coin distribution.
      Pre  floating-point number
      Post change in dollars, half-dollars, quarters,
             dimes, nickels, and pennies.
*/
void change  (float total,
              int* dlrs,     int* halfDlrs,
              int* quarters, int* dimes,
              int* nickels,  int* cents)
{
// Local Declarations
   int calc;

// Statements
   calc  = (int)(total * 100.0);

   *dlrs = calc / 100;
   calc -= (*dlrs * 100);

   *halfDlrs = calc / 50;
   calc -= (*halfDlrs * 50);

   *quarters = calc / 25;
   calc -= (*quarters * 25);

   *dimes = calc / 10;
   calc -= (*dimes * 10);

   *nickels = calc / 5;
   calc -= (*nickels * 5);

   *cents = calc;
   return;
}  // change
```

35. See Program 9-3.

Program 9-3 Solution to Problem 35

```
/* ==================== gcd_lcm ================
   This function receives 2 integers and passes back
   the G.C.D. and L.C.M.
      Pre   2 integers
      Post  the GCD and LCM
*/
void gcd_lcm (int  num1,  int  num2,
              int* gcd,   int* lcm)
{
// Local Declarations
   int rem;
   int fact1;
   int fact2;

// Statements
   if (num1 >= num2)
      {
       fact1 = num1;
       fact2 = num2;
      }
   else
      {
       fact1 = num2;
       fact2 = num1;
      } // if

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

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