Declare `p1` as a pointer that points to `f1` and `p2` as a pointer to `f2`. Declare `ap` as an array of five pointers of the same type as `p1`, and declare `pa` as a pointer to an array of ten pointers of the same type as `p2`. Use `typedef` as an aid.

Programming Exercises

1. Write a program that repeatedly asks the user to enter pairs of numbers until at least one of the pair is `0`. For each pair, the program should use a function to calculate the harmonic mean of the numbers. The function should return the answer to `main()`, which should report the result. The harmonic mean of the numbers is the inverse of the average of the inverses and can be calculated as follows:

   harmonic mean = 2.0 × x × y / (x + y)

2. Write a program that asks the user to enter up to 10 golf scores, which are to be stored in an array. You should provide a means for the user to terminate input prior to entering 10 scores. The program should display all the scores on one line and report the average score. Handle input, display, and the average calculation with three separate array-processing functions.

3. Here is a structure declaration:

```
struct box
{
      char maker[40];
      float height;
      float width;
      float length;
      float volume;
};
```

   a. Write a function that passes a `box` structure by value and that displays the value of each member.

   b. Write a function that passes the address of a `box` structure and that sets the `volume` member to the product of the other three dimensions.

   c. Write a simple program that uses these two functions.

4. Many state lotteries use a variation of the simple lottery portrayed by Listing 7.4. In these variations you choose several numbers from one set and call them the field numbers. For example, you might select five numbers from the field of 1–47). You also pick a single number (called a mega number or a power ball, etc.) from a second range, such as 1–27. To win the grand prize, you have to guess all the picks correctly. The chance of winning is the product of the probability of picking all the field numbers times the probability of picking the mega number. For instance, the

probability of winning the example described here is the product of the probability of picking 5 out of 47 correctly times the probability of picking 1 out of 27 correctly. Modify Listing 7.4 to calculate the probability of winning this kind of lottery.

5. Define a recursive function that takes an integer argument and returns the factorial of that argument. Recall that 3 factorial, written 3!, equals 3 × 2!, and so on, with 0! defined as 1. In general, if n is greater than zero, n! = n * (n - 1)!. Test your function in a program that uses a loop to allow the user to enter various values for which the program reports the factorial.

6. Write a program that uses the following functions:

   `Fill_array()` takes as arguments the name of an array of `double` values and an array size. It prompts the user to enter `double` values to be entered in the array. It ceases taking input when the array is full or when the user enters non-numeric input, and it returns the actual number of entries.

   `Show_array()` takes as arguments the name of an array of `double` values and an array size and displays the contents of the array.

   `Reverse_array()` takes as arguments the name of an array of `double` values and an array size and reverses the order of the values stored in the array.

   The program should use these functions to fill an array, show the array, reverse the array, show the array, reverse all but the first and last elements of the array, and then show the array.

7. Redo Listing 7.7, modifying the three array-handling functions to each use two pointer parameters to represent a range. The `fill_array()` function, instead of returning the actual number of items read, should return a pointer to the location after the last location filled; the other functions can use this pointer as the second argument to identify the end of the data.

8. Redo Listing 7.15 without using the `array` class. Do two versions:

   a. Use an ordinary array of `const char *` for the strings representing the season names, and use an ordinary array of `double` for the expenses.

   b. Use an ordinary array of `const char *` for the strings representing the season names, and use a structure whose sole member is an ordinary array of `double` for the expenses. (This design is similar to the basic design of the `array` class.)

9. This exercise provides practice in writing functions dealing with arrays and structures. The following is a program skeleton. Complete it by providing the described functions:

```
#include <iostream>
using namespace std;
```

```
const int SLEN = 30;
struct student {
    char fullname[SLEN];
    char hobby[SLEN];
    int ooplevel;
};
// getinfo() has two arguments: a pointer to the first element of
// an array of student structures and an int representing the
// number of elements of the array. The function solicits and
// stores data about students. It terminates input upon filling
// the array or upon encountering a blank line for the student
// name. The function returns the actual number of array elements
// filled.
int getinfo(student pa[], int n);

// display1() takes a student structure as an argument
// and displays its contents
void display1(student st);

// display2() takes the address of student structure as an
// argument and displays the structure's contents
void display2(const student * ps);

// display3() takes the address of the first element of an array
// of student structures and the number of array elements as
// arguments and displays the contents of the structures
void display3(const student pa[], int n);

int main()
{
     cout << "Enter class size: ";
     int class_size;
     cin >> class_size;
     while (cin.get() != '\n')
         continue;

    student * ptr_stu = new student[class_size];
    int entered = getinfo(ptr_stu, class_size);
    for (int i = 0; i < entered; i++)
    {
        display1(ptr_stu[i]);
        display2(&ptr_stu[i]);
    }
    display3(ptr_stu, entered);
    delete [] ptr_stu;
    cout << "Done\n";
    return 0;
}
```

10. Design a function `calculate()` that takes two type `double` values and a pointer to a function that takes two `double` arguments and returns a `double`. The `calculate()` function should also be type `double`, and it should return the value that the pointed-to function calculates, using the double arguments to `calculate()`. For example, suppose you have this definition for the `add()` function:

```
double add(double x, double y)
{
       return x + y;
}
```

Then, the function call in the following would cause `calculate()` to pass the values `2.5` and `10.4` to the `add()` function and then return the `add()` return value (`12.9`):

```
double q = calculate(2.5, 10.4, add);
```

Use these functions and at least one additional function in the `add()` mold in a program. The program should use a loop that allows the user to enter pairs of numbers. For each pair, use `calculate()` to invoke `add()` and at least one other function. If you are feeling adventurous, try creating an array of pointers to `add()`-style functions and use a loop to successively apply `calculate()` to a series of functions by using these pointers. Hint: Here's how to declare such an array of three pointers:

```
double (*pf[3])(double, double);
```

You can initialize such an array by using the usual array initialization syntax and function names as addresses.