a has the wrong type. When used as an argument, it's a pointer to an array, but `find_largest` is expecting a pointer to an integer. However, `a[0]` has type `int *`, so it's an acceptable argument for `find_largest`. This concern about types is actually good; if C weren't so picky, we could make all kinds of horrible pointer mistakes without the compiler noticing.

Exercises

Section 12.1

1. Suppose that the following declarations are in effect:

```
int a[] = {5, 15, 34, 54, 14, 2, 52, 72};
int *p = &a[1], *q = &a[5];
```

(a) What is the value of `*(p+3)`?
(b) What is the value of `*(q-3)`?
(c) What is the value of `q - p`?
(d) Is the condition `p < q` true or false?
(e) Is the condition `*p < *q` true or false?

W *2. Suppose that `high`, `low`, and `middle` are all pointer variables of the same type, and that `low` and `high` point to elements of an array. Why is the following statement illegal, and how could it be fixed?

```
middle = (low + high) / 2;
```

Section 12.2

3. What will be the contents of the a array after the following statements are executed?

```
#define N 10

int a[N] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int *p = &a[0], *q = &a[N-1], temp;

while (p < q) {
  temp = *p;
  *p++ = *q;
  *q-- = temp;
}
```

W 4. Rewrite the `make_empty`, `is_empty`, and `is_full` functions of Section 10.2 to use the pointer variable `top_ptr` instead of the integer variable `top`.

Section 12.3

5. Suppose that a is a one-dimensional array and p is a pointer variable. Assuming that the assignment `p = a` has just been performed, which of the following expressions are illegal because of mismatched types? Of the remaining expressions, which are true (have a nonzero value)?

(a) `p == a[0]`
(b) `p == &a[0]`
(c) `*p == a[0]`
(d) `p[0] == a[0]`

W 6. Rewrite the following function to use pointer arithmetic instead of array subscripting. (In other words, eliminate the variable `i` and all uses of the `[]` operator.) Make as few changes as possible.

```
int sum_array(const int a[], int n)
{
  int i, sum;

  sum = 0;
  for (i = 0; i < n; i++)
    sum += a[i];
  return sum;
}
```

7. Write the following function:

```
bool search(const int a[], int n, int key);
```

`a` is an array to be searched, `n` is the number of elements in the array, and `key` is the search key. `search` should return `true` if `key` matches some element of `a`, and `false` if it doesn't. Use pointer arithmetic—not subscripting—to visit array elements.

8. Rewrite the following function to use pointer arithmetic instead of array subscripting. (In other words, eliminate the variable `i` and all uses of the `[]` operator.) Make as few changes as possible.

```
void store_zeros(int a[], int n)
{
  int i;

  for (i = 0; i < n; i++)
    a[i] = 0;
}
```

9. Write the following function:

```
double inner_product(const double *a, const double *b,
                     int n);
```

`a` and `b` both point to arrays of length `n`. The function should return `a[0] * b[0] + a[1] * b[1] + ... + a[n-1] * b[n-1]`. Use pointer arithmetic—not subscripting—to visit array elements.

10. Modify the `find_middle` function of Section 11.5 so that it uses pointer arithmetic to calculate the return value.

11. Modify the `find_largest` function so that it uses pointer arithmetic—not subscripting—to visit array elements.

12. Write the following function:

```
void find_two_largest(const int *a, int n, int *largest,
                      int *second_largest);
```

`a` points to an array of length `n`. The function searches the array for its largest and second-largest elements, storing them in the variables pointed to by `largest` and `second_largest`, respectively. Use pointer arithmetic—not subscripting—to visit array elements.

Section 12.4

W 13. Section 8.2 had a program fragment in which two nested `for` loops initialized the array `ident` for use as an identity matrix. Rewrite this code, using a single pointer to step through the array one element at a time. *Hint:* Since we won't be using `row` and `col` index variables, it won't be easy to tell where to store 1. Instead, we can use the fact that the first element of the array should be 1, the next `N` elements should be 0, the next element should

be 1, and so forth. Use a variable to keep track of how many consecutive 0s have been stored; when the count reaches `N`, it's time to store 1.

14. Assume that the following array contains a week's worth of hourly temperature readings, with each row containing the readings for one day:

```
int temperatures[7][24];
```

Write a statement that uses the `search` function (see Exercise 7) to search the entire `temperatures` array for the value 32.

W 15. Write a loop that prints all temperature readings stored in row `i` of the `temperatures` array (see Exercise 14). Use a pointer to visit each element of the row.

16. Write a loop that prints the highest temperature in the `temperatures` array (see Exercise 14) for each day of the week. The loop body should call the `find_largest` function, passing it one row of the array at a time.

17. Rewrite the following function to use pointer arithmetic instead of array subscripting. (In other words, eliminate the variables `i` and `j` and all uses of the `[]` operator.) Use a single loop instead of nested loops.

```
int sum_two_dimensional_array(const int a[][LEN], int n)
{
  int i, j, sum = 0;

  for (i = 0; i < n; i++)
    for (j = 0; j < LEN; j++)
      sum += a[i][j];
  return sum;
}
```

18. Write the `evaluate_position` function described in Exercise 13 of Chapter 9. Use pointer arithmetic—not subscripting—to visit array elements. Use a single loop instead of nested loops.

Programming Projects

W 1. (a) Write a program that reads a message, then prints the reversal of the message:

```
Enter a message: Don't get mad, get even.
Reversal is: .neve teg ,dam teg t'noD
```

Hint: Read the message one character at a time (using `getchar`) and store the characters in an array. Stop reading when the array is full or the character read is `'\n'`.

(b) Revise the program to use a pointer instead of an integer to keep track of the current position in the array.

2. (a) Write a program that reads a message, then checks whether it's a palindrome (the letters in the message are the same from left to right as from right to left):

```
Enter a message: He lived as a devil, eh?
Palindrome

Enter a message: Madam, I am Adam.
Not a palindrome
```

Ignore all characters that aren't letters. Use integer variables to keep track of positions in the array.

(b) Revise the program to use pointers instead of integers to keep track of positions in the array.

W 3. Simplify Programming Project 1(b) by taking advantage of the fact that an array name can be used as a pointer.

4. Simplify Programming Project 2(b) by taking advantage of the fact that an array name can be used as a pointer.

5. Modify Programming Project 14 from Chapter 8 so that it uses a pointer instead of an integer to keep track of the current position in the array that contains the sentence.

6. Modify the `qsort.c` program of Section 9.6 so that `low`, `high`, and `middle` are pointers to array elements rather than integers. The `split` function will need to return a pointer, not an integer.

7. Modify the `maxmin.c` program of Section 11.4 so that the `max_min` function uses a pointer instead of an integer to keep track of the current position in the array.