```
int main()
{ int a[3][5];
  read(a);
  print(a);
}
void read(int a[][5])
{ cout << "Enter 15 integers, 5 per row:\n”;
  for (int i=0; i<3; i++)
  { cout << "Row " << i << ": ";
    for (int j=0; j<5; j++)
    cin >> a[i][j];
  }
}
void print(const int a[][5])
{ for (int i=0; i<3; i++)
  { for (int j=0; j<5; j++)
      cout << " " << a[i][j];
    cout << endl;
  }
}
```

```
Enter 15 integers, 5 per row:
Row 0: 44 77 33 11 44
Row 1: 60 50 30 90 70
Row 2: 85 25 45 45 55
 44 77 33 11 44
 60 50 30 90 70
 85 25 45 45 55
```

Notice that in the functions’ parameter lists, the first dimension is left unspecified while the second dimension (5) is specified. This is because the two-dimensional array `a[][]` is stored as a one-dimensional array of three 5-element arrays. The compiler does not need to know how many of these 5-element arrays are to be stored, but it does need to know that they are 5-element arrays.

When a multi-dimensional array is passed to a function, the first dimension is not specified, while all the remaining dimensions are specified.

EXAMPLE 6.20 Processing a Two-Dimensional Array of Quiz Scores

```
const NUM_STUDENTS = 3;
const NUM_QUIZZES = 5;
typedef int Score[NUM_STUDENTS][NUM_QUIZZES];
void read(Score);
void printQuizAverages(Score);
void printClassAverages(Score);
int main()
{ Score score;
  cout << "Enter " << NUM_QUIZZES << " scores for each student:\n";
  read(score);
  cout << "The quiz averages are:\n";
  printQuizAverages(score);
  cout << "The class averages are:\n";
  printClassAverages(score);
}
```