Chapter 8

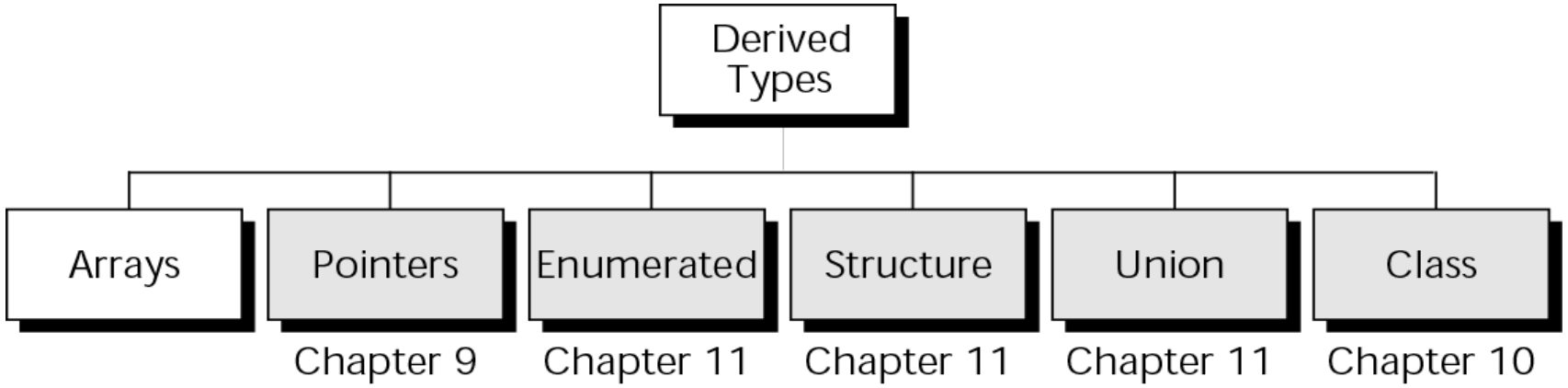
Arrays

OBJECTIVES

After studying this chapter you will be able to:

- Understand and use one, two, and three dimensional arrays in C++.
- Understand how to initialize arrays.
- Read data into an array or write data from an array.
- Understand that array range checking is the responsibility of the programmer.
- Write programs that pass arrays or array elements to functions.
- Sort data in an array using selection, bubble, or insertion sorting.
- Search an array using sequential or binary searches.
- Analyze the efficiency of sorting and searching algorithms.

Figure 8-1 Derived types



CONCEPTS

Figure 8-2 Ten variables

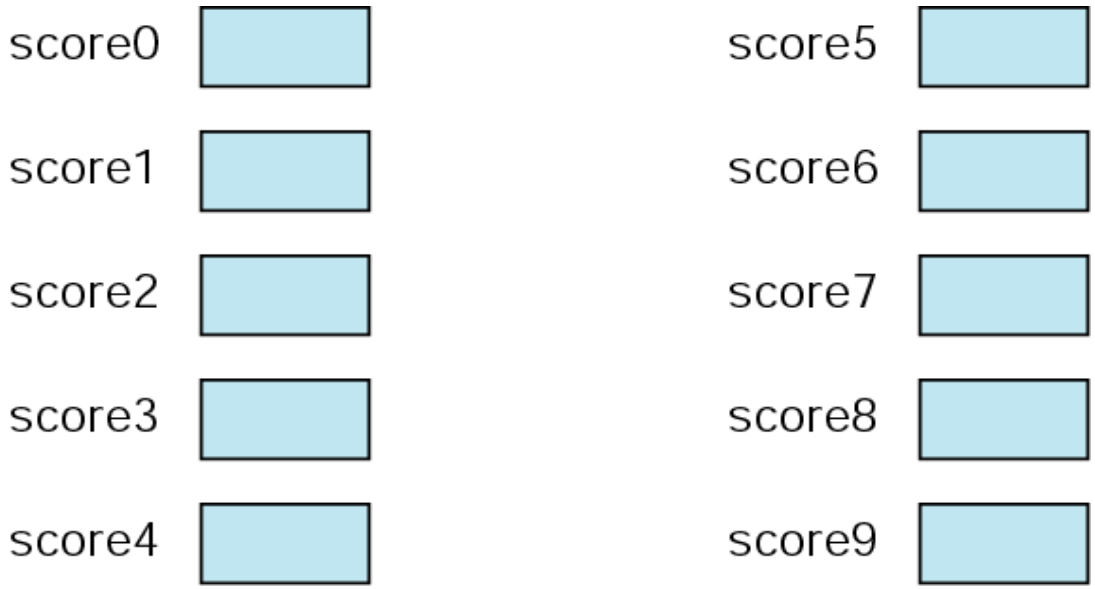


Figure 8-3 Process 10 variables

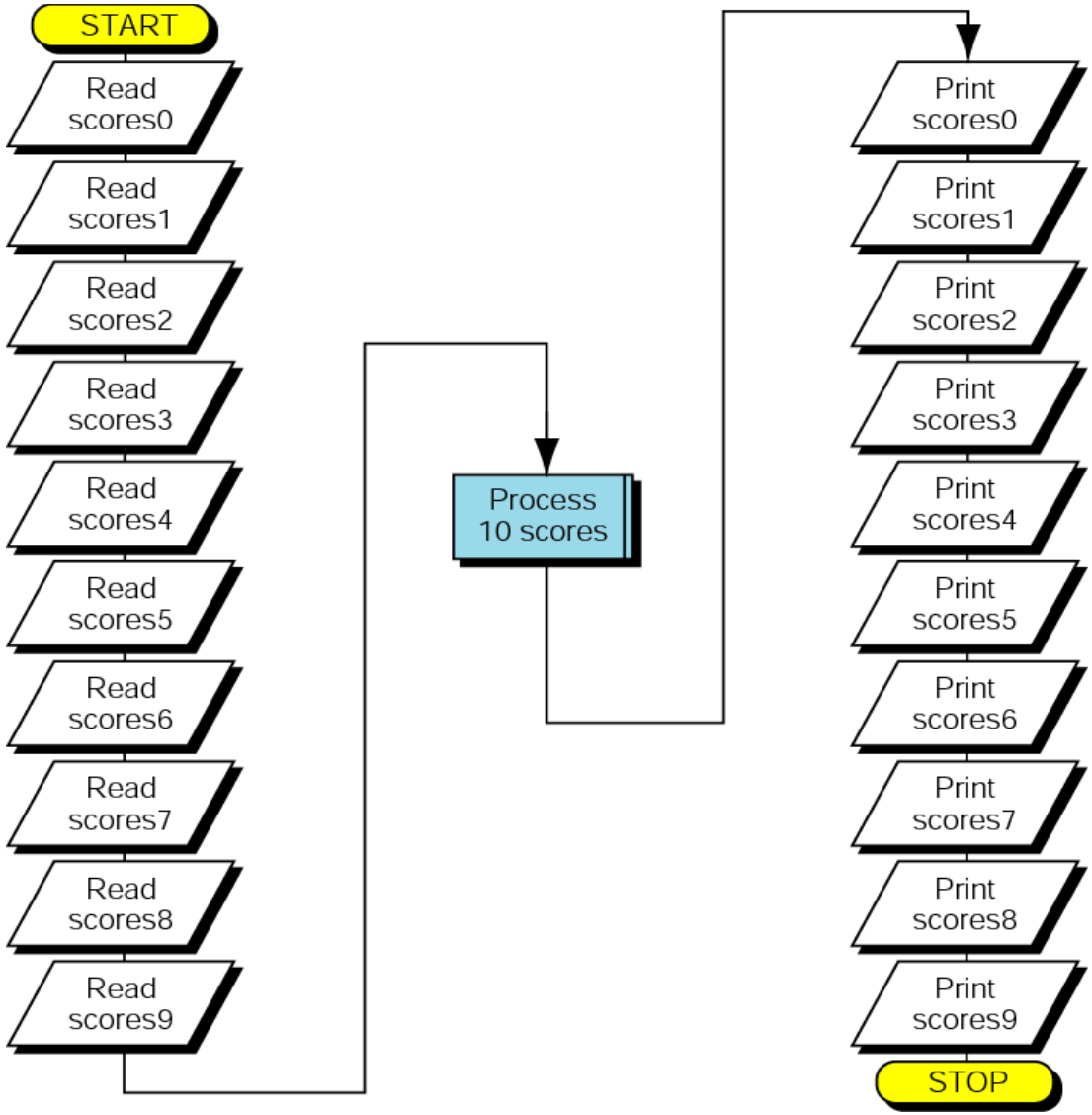


Figure 8-4 An array of scores

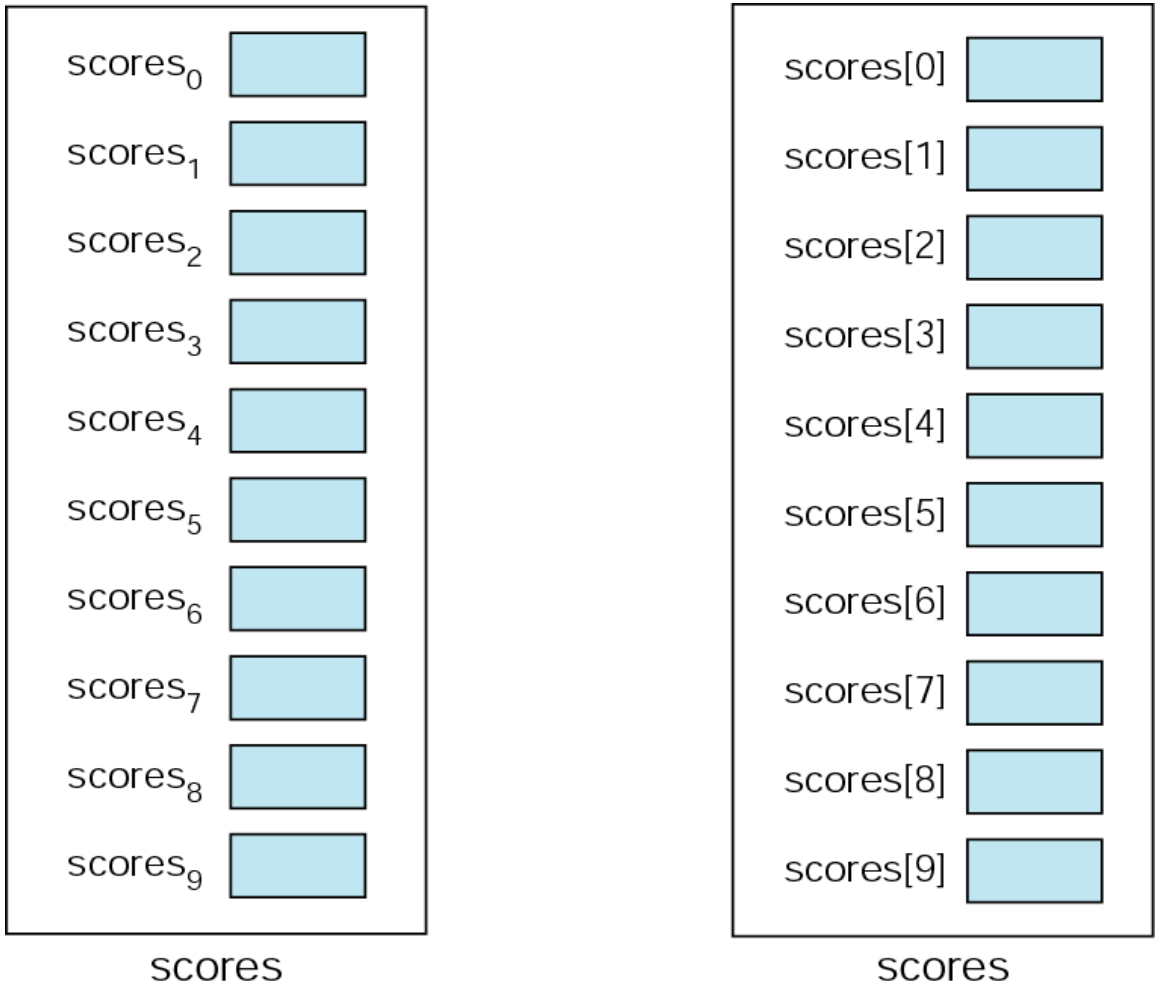
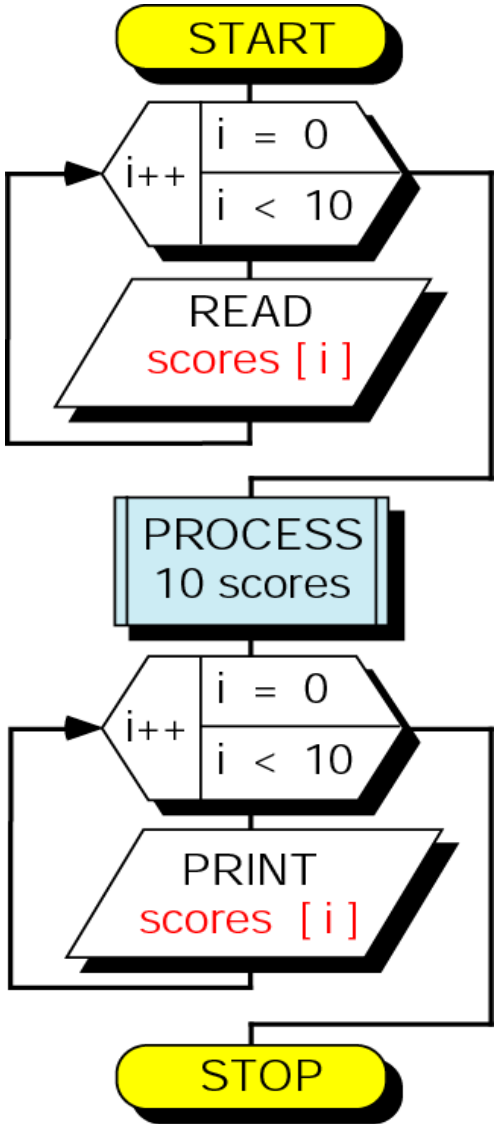


Figure 8-5 Loop for 10 scores



USING ARRAYS IN C++

Figure 8-6 The scores array

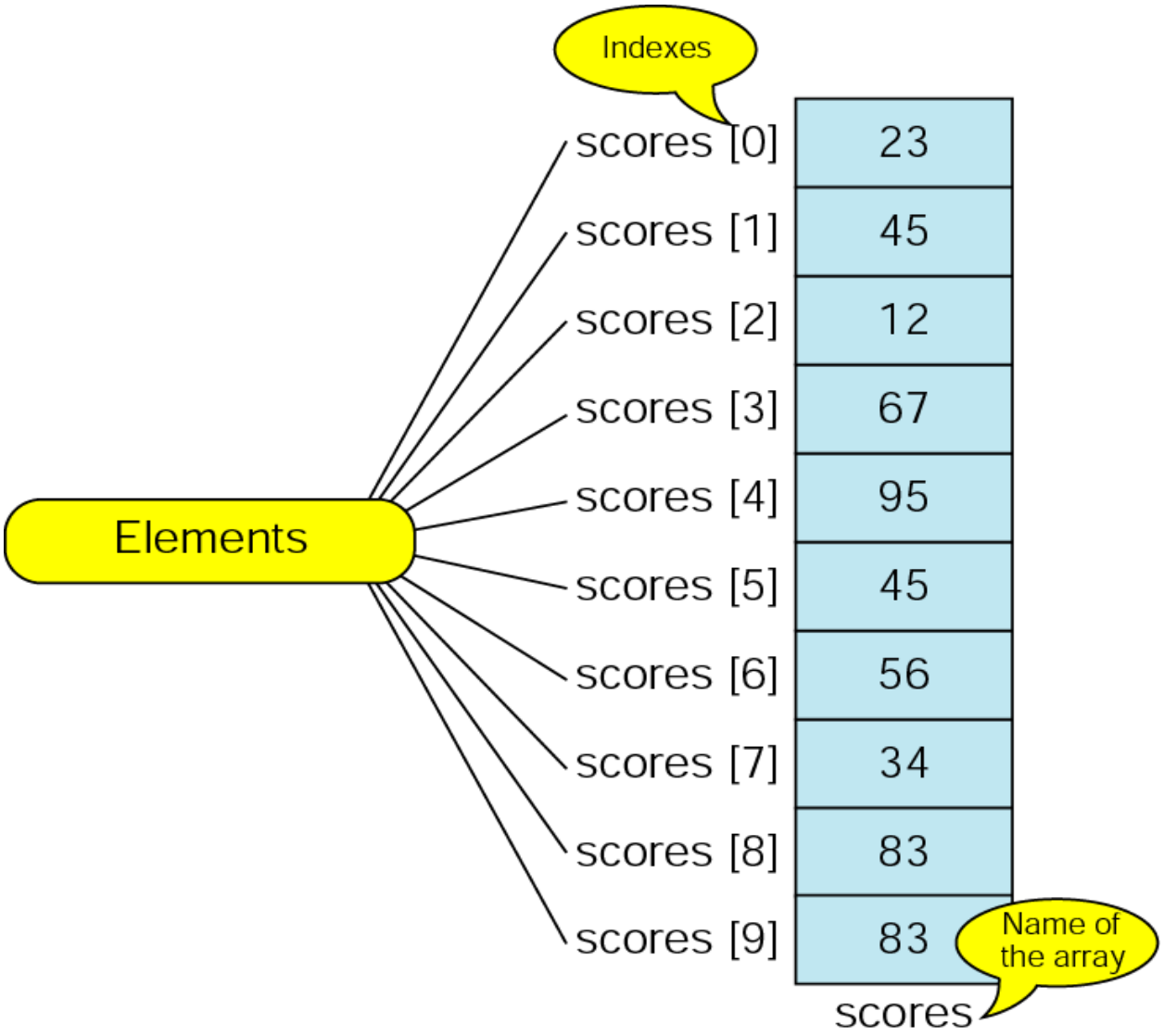


Figure 8-7 Declaring and defining arrays

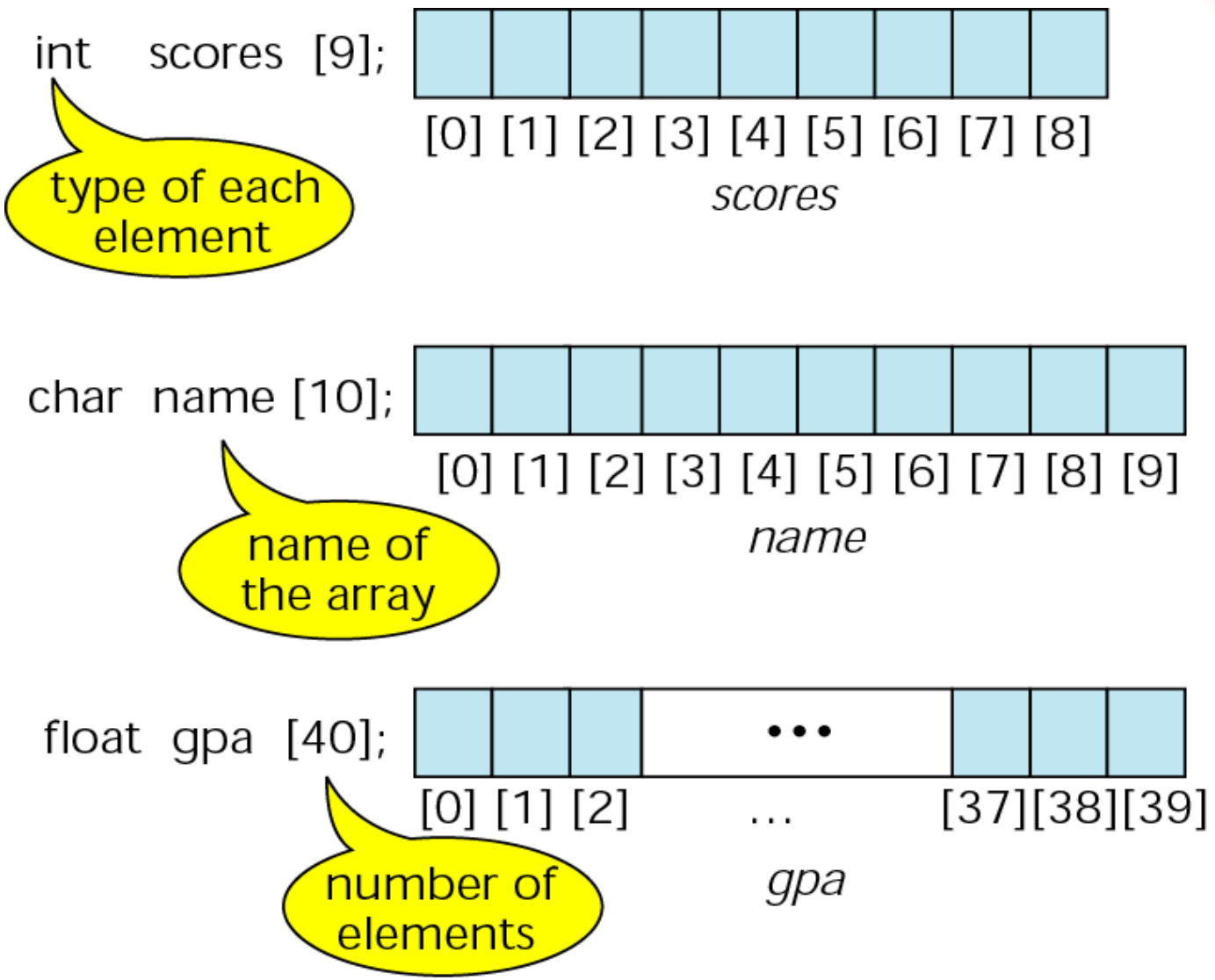


Figure 8-8 Initializing arrays

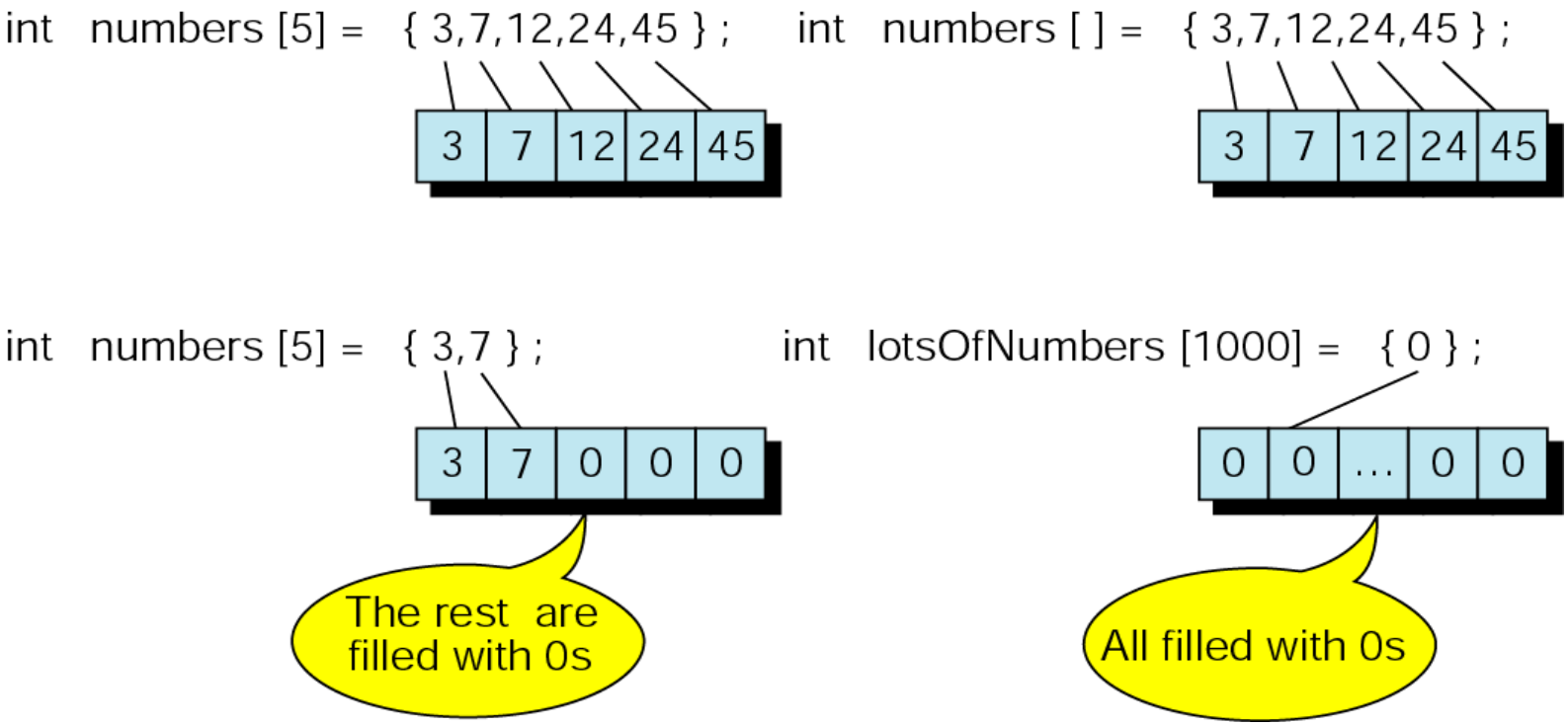


Figure 8-9 Exchanging scores—the wrong way

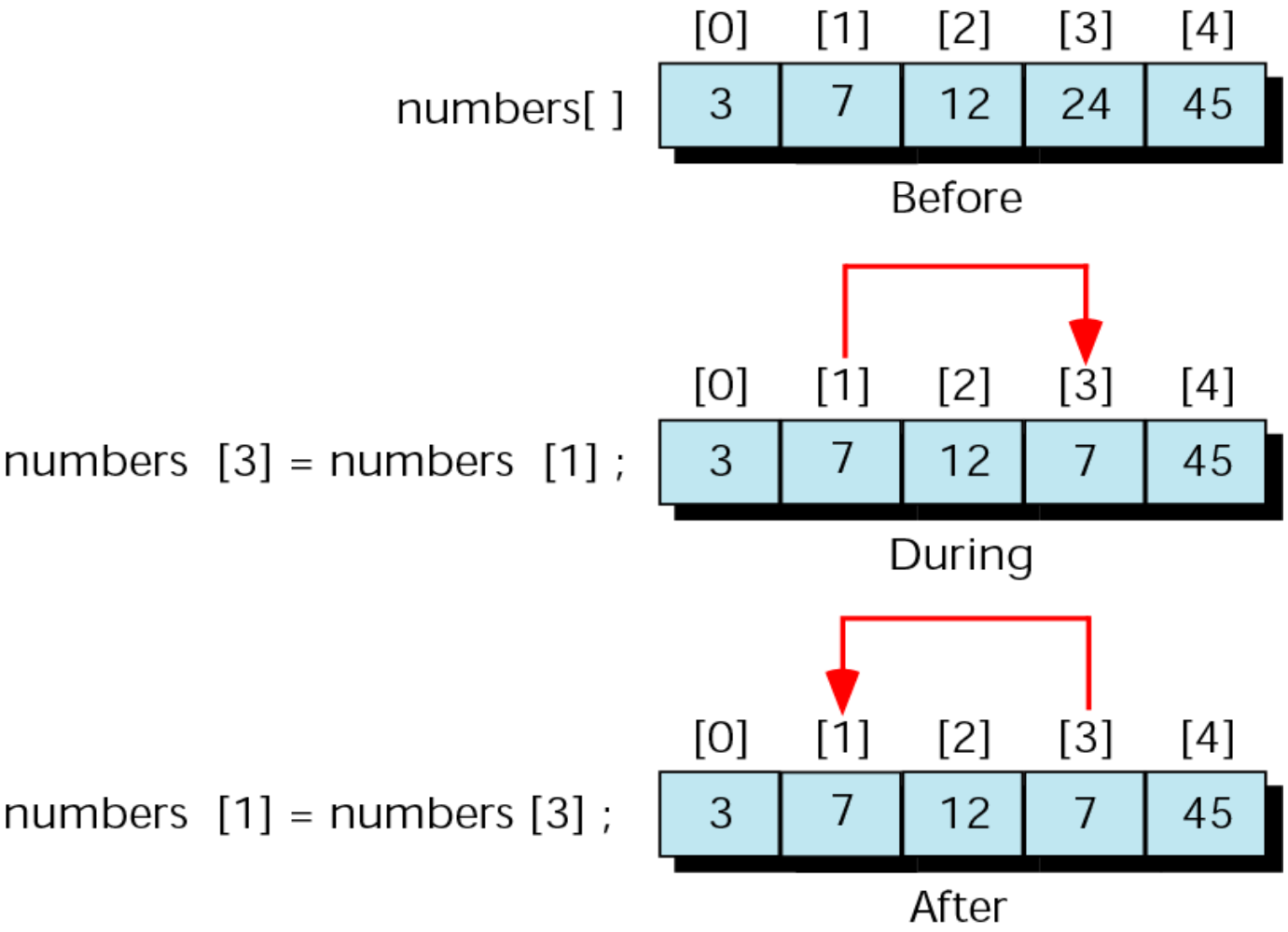
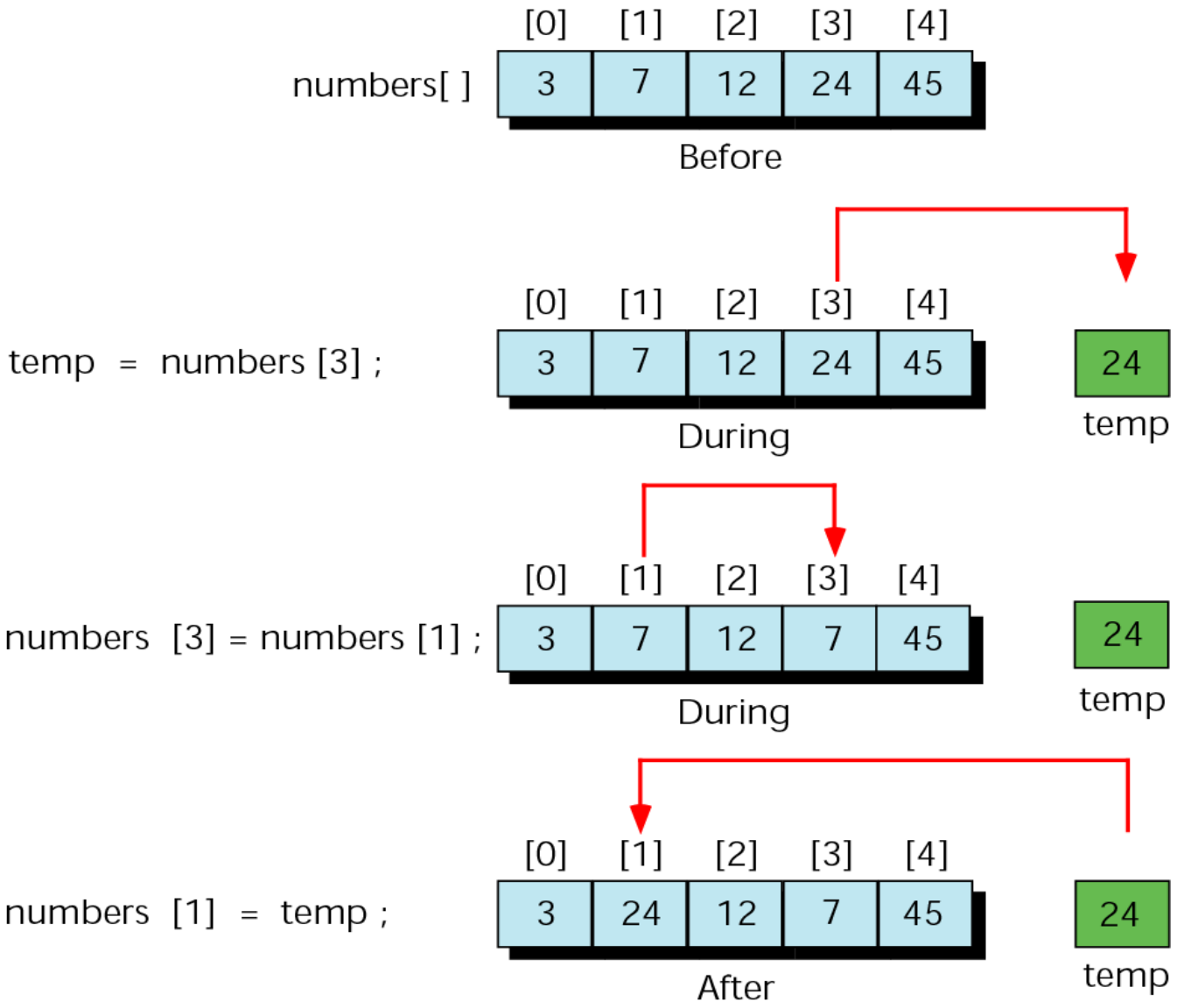


Figure 8-10 Exchanging scores with temporary variable



Arrays and Functions

Figure 8-11 Passing individual elements

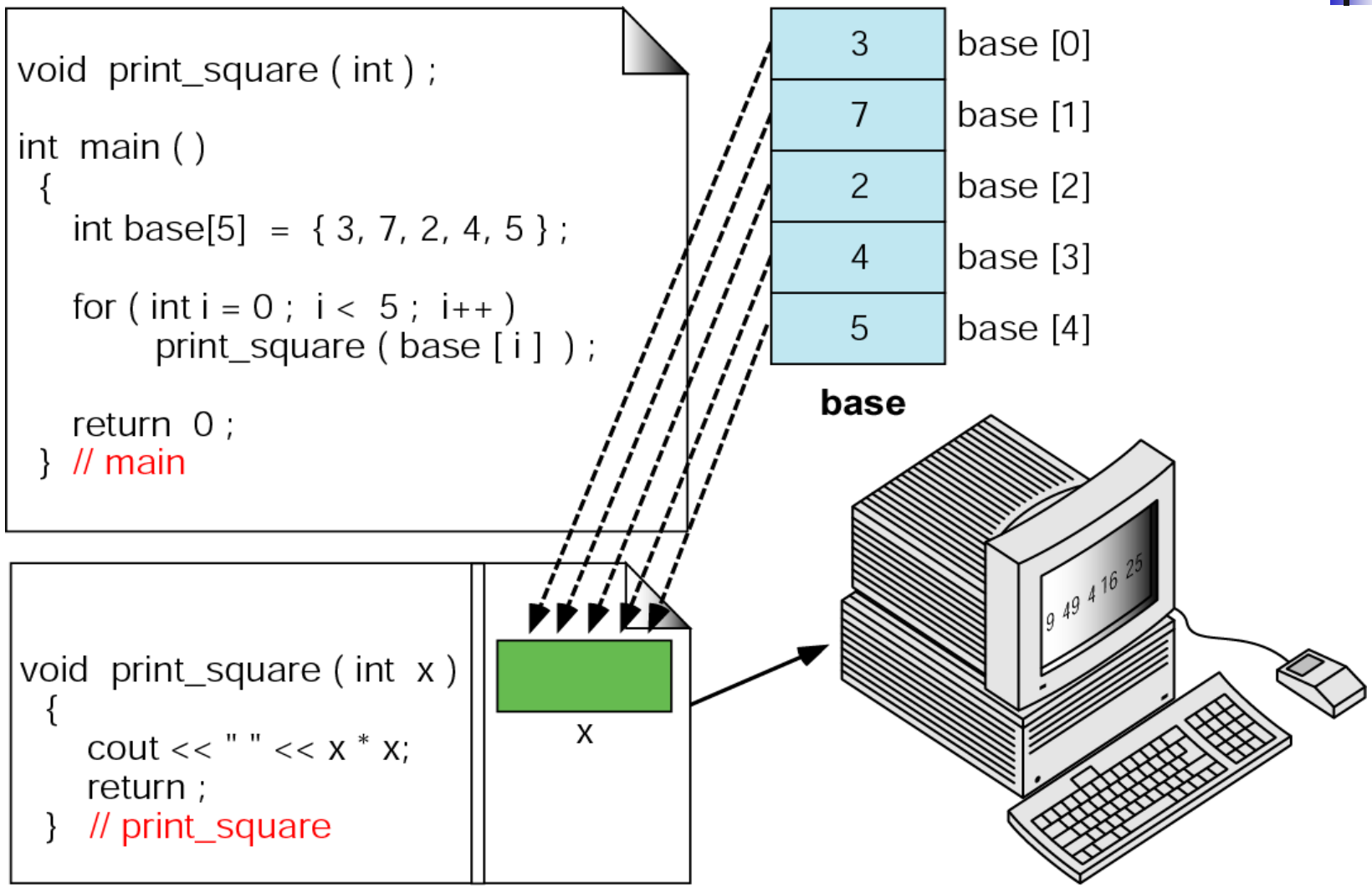


Figure 8-12 Passing arrays—average

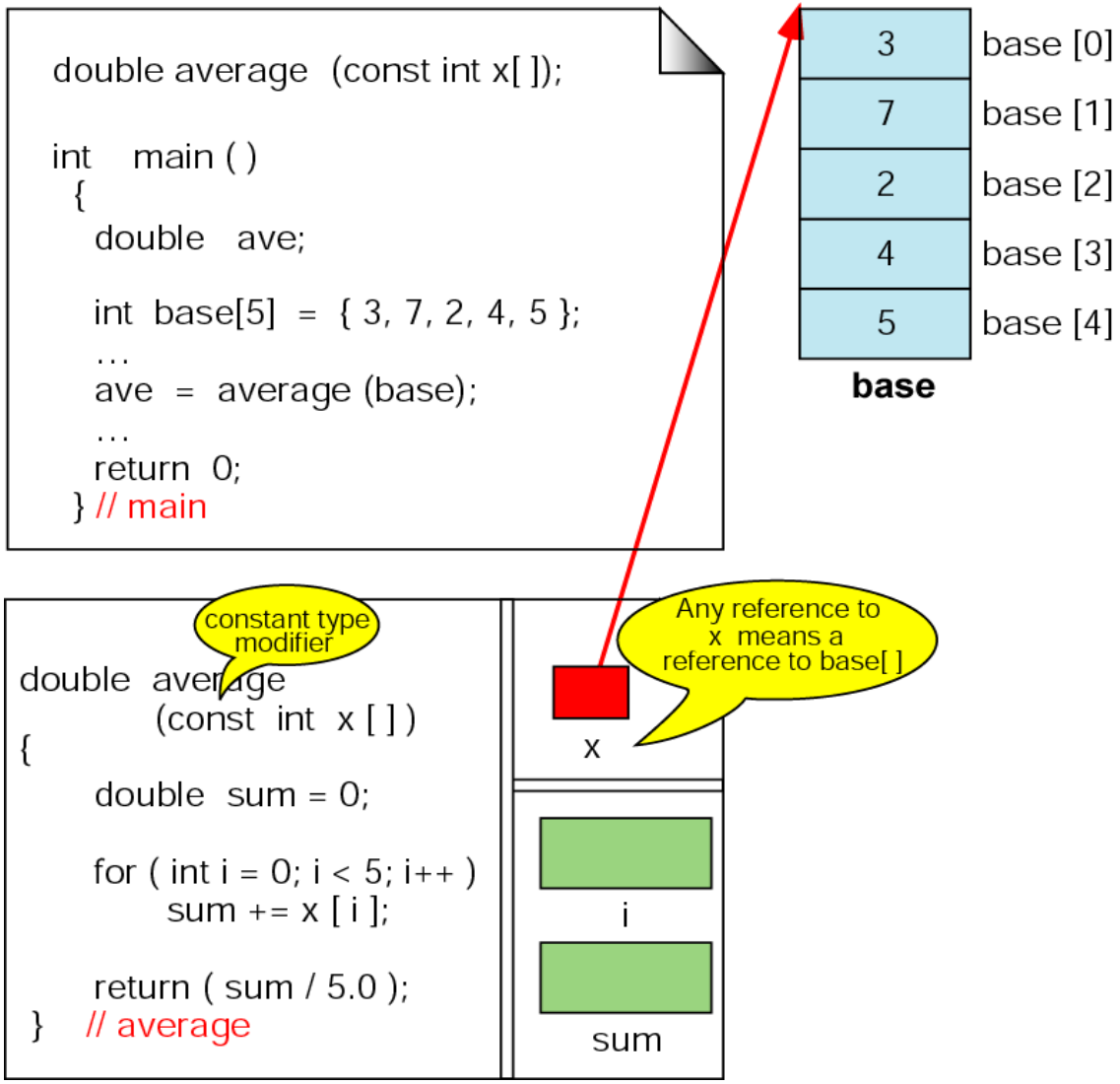
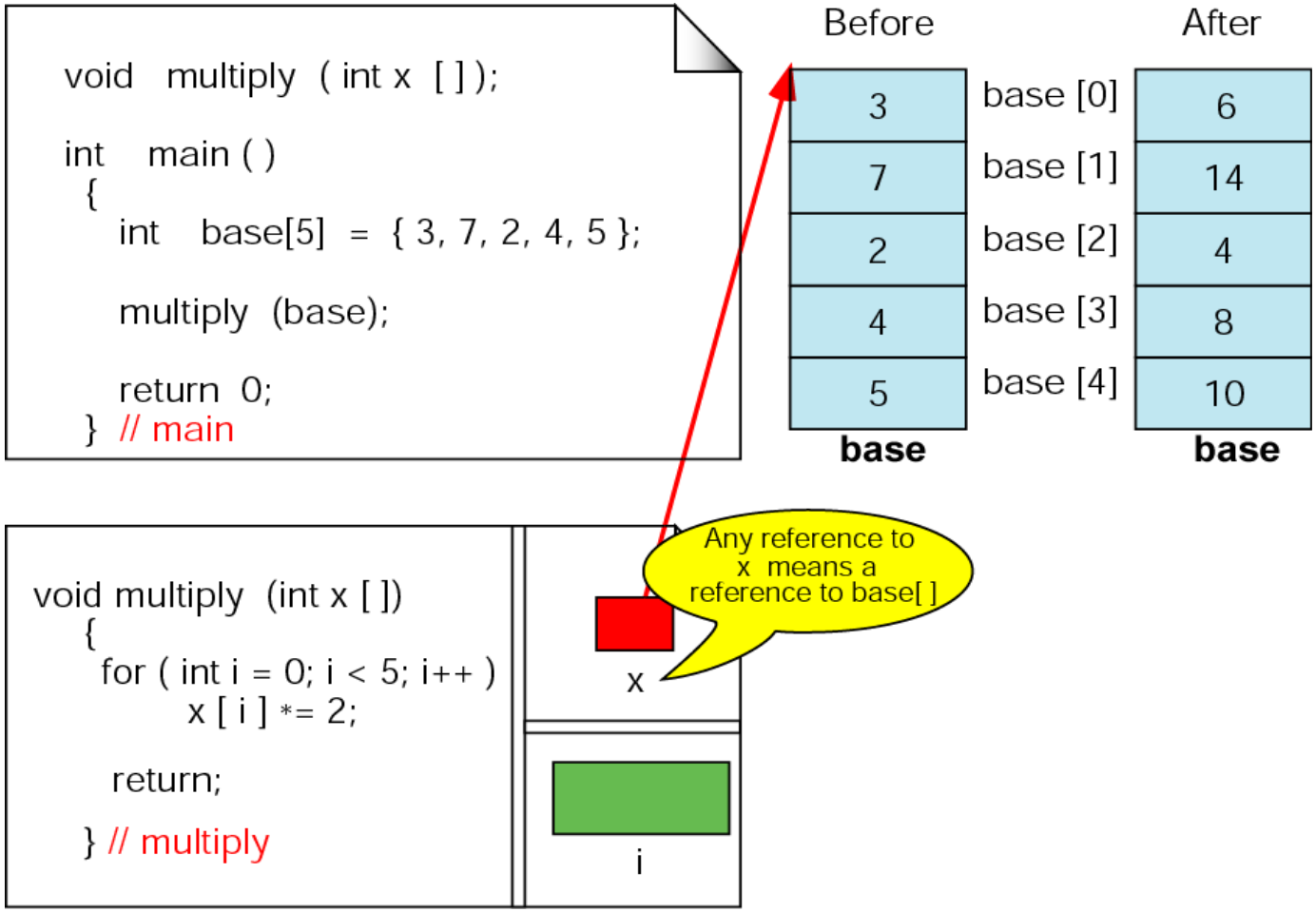


Figure 8-13 Changing values in arrays



ARRAY APPLICATIONS

Figure 8-14 Frequency array

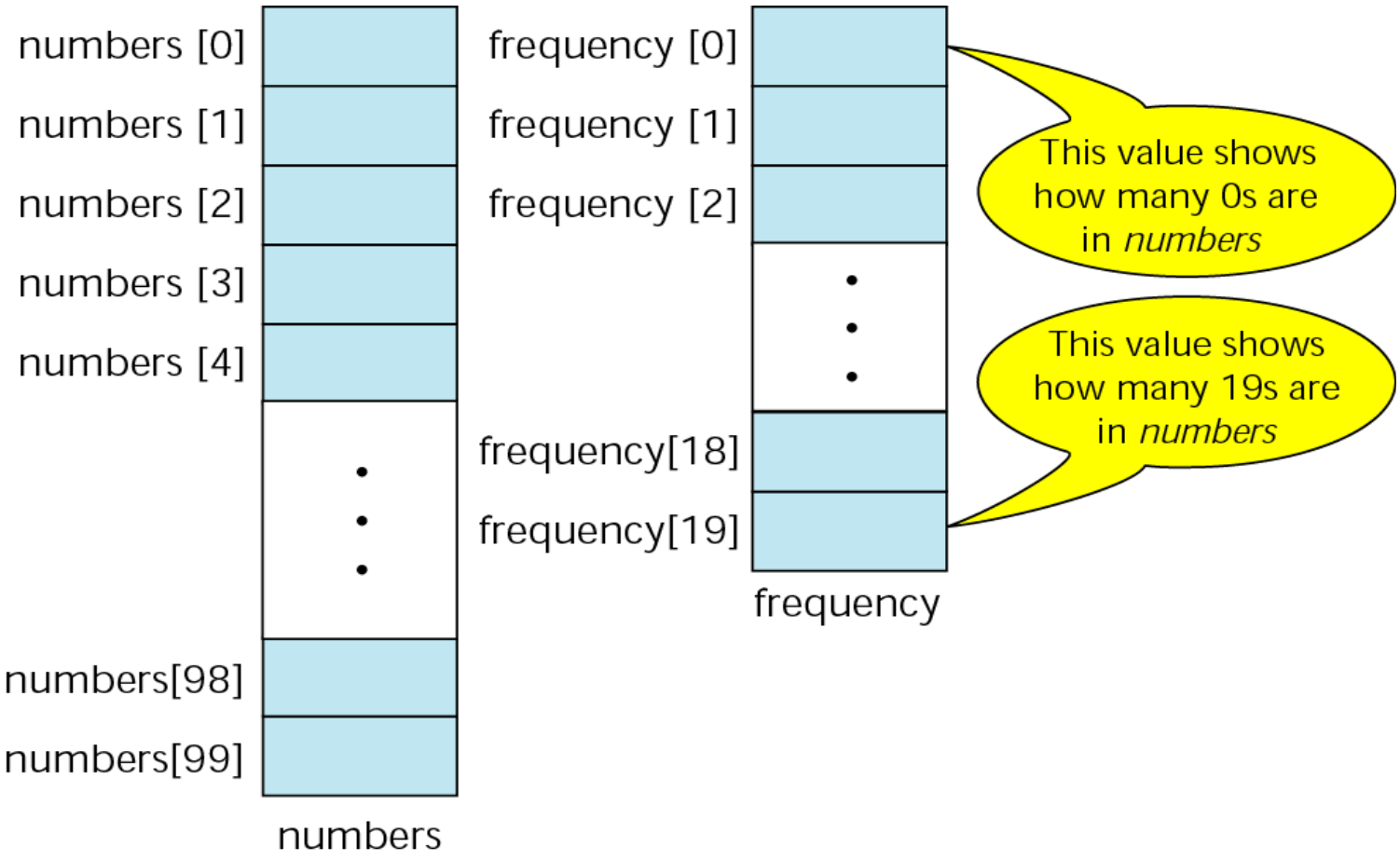


Figure 8-15 Frequency histogram

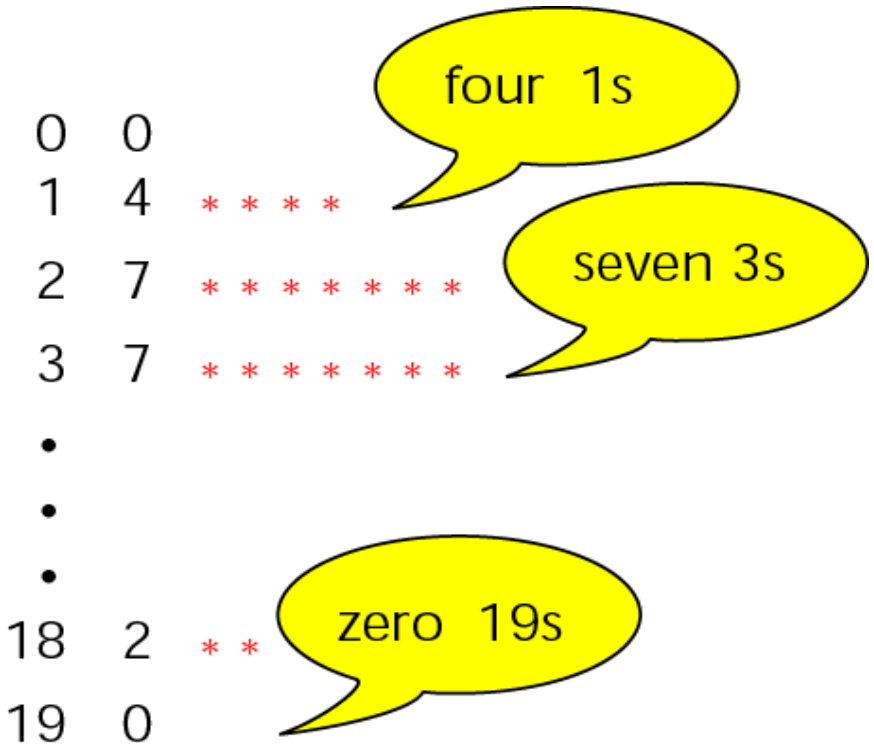


Figure 8-16 Histogram program design

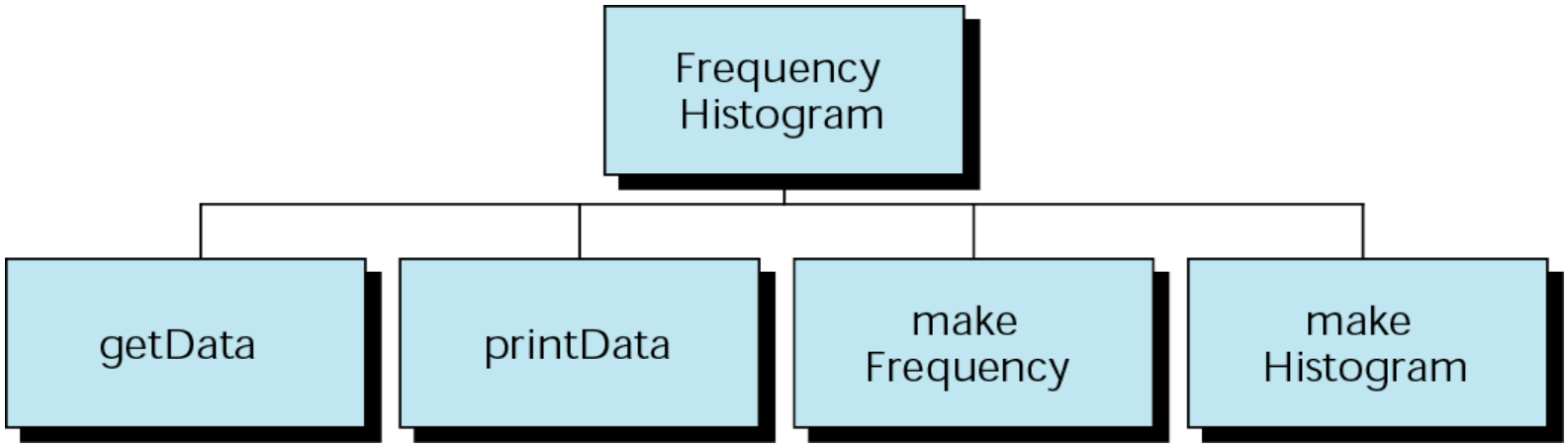
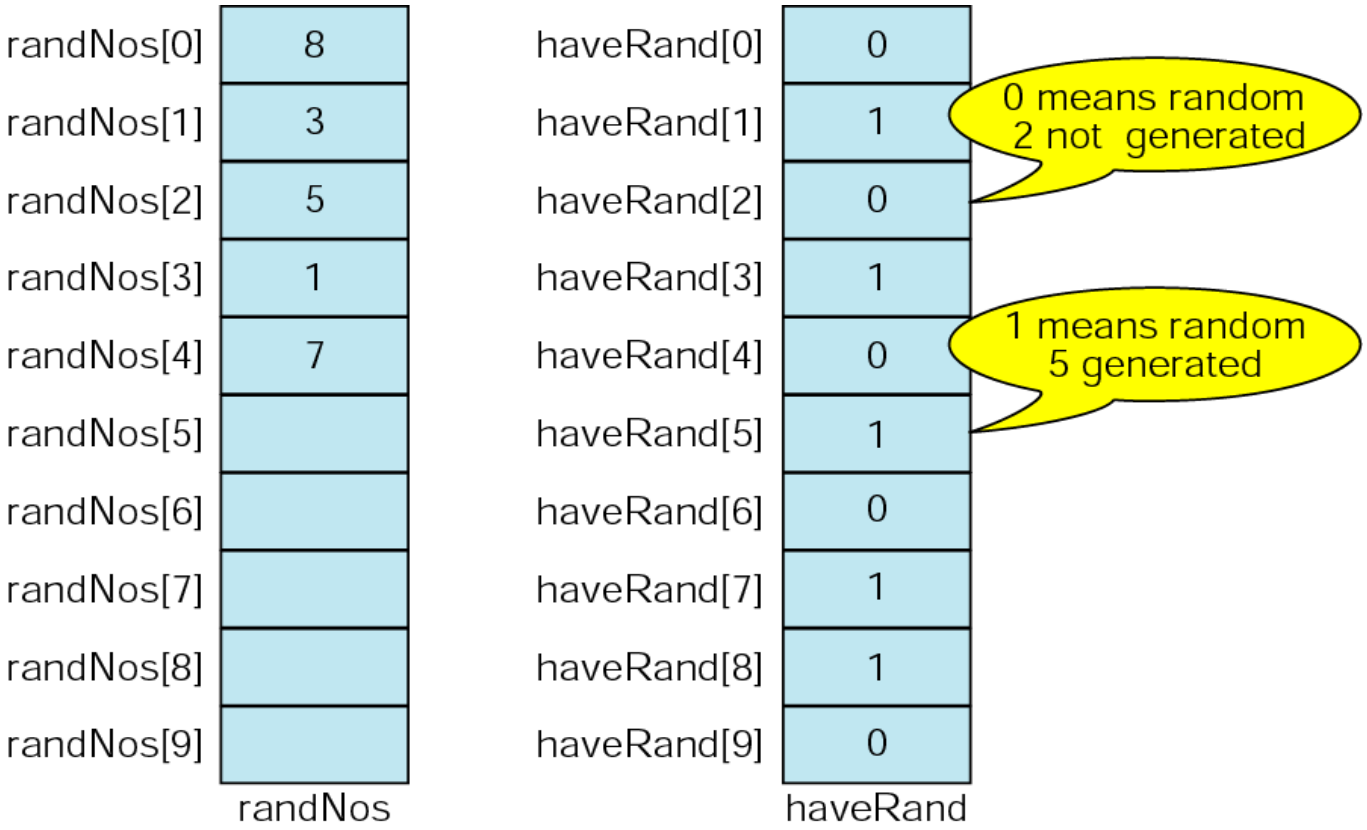


Figure 8-17 Design for random number permutations



After first five random numbers generated.

SORTING

Figure 8-18 Selection sort concept

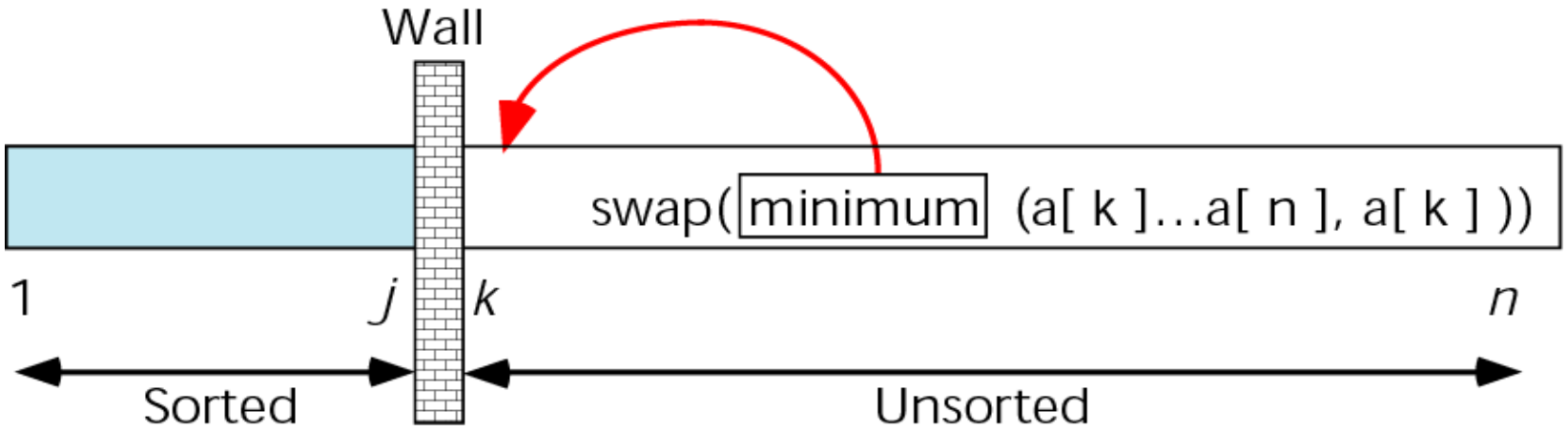


Figure 8-19 Selection sort example

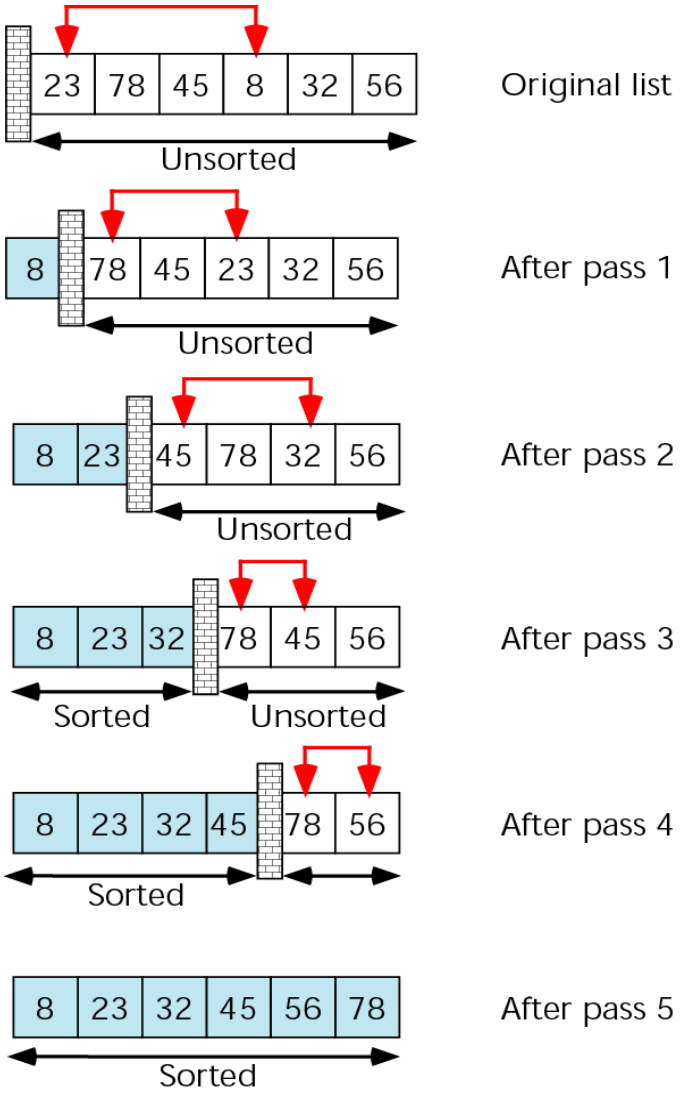


Figure 8-20 Bubble sort concept

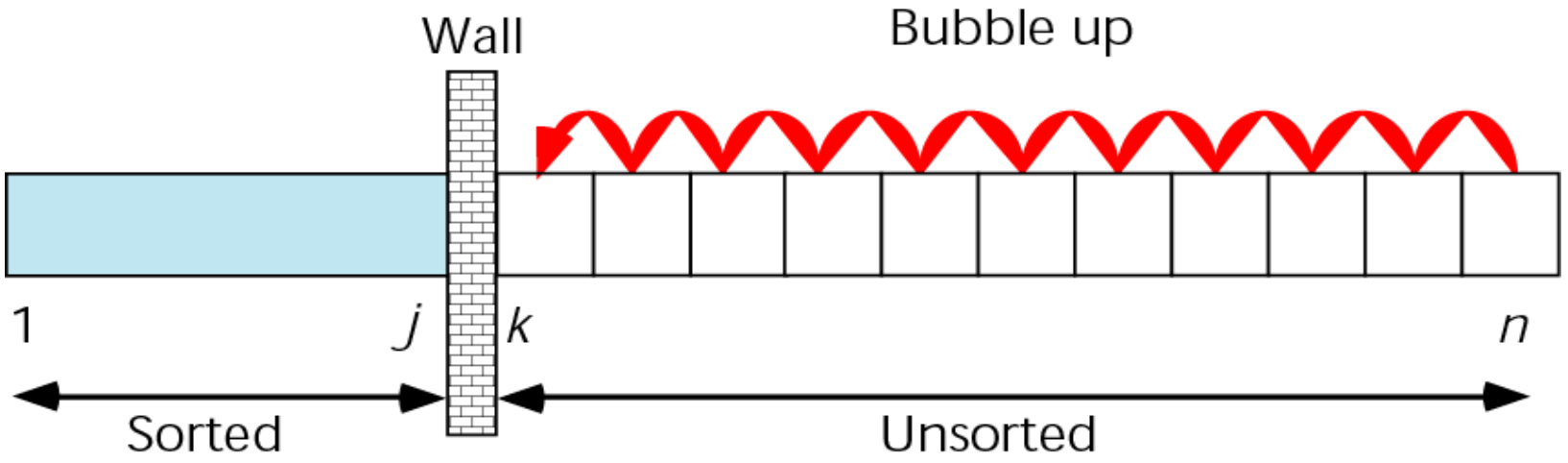


Figure 8-21 Bubble sort example

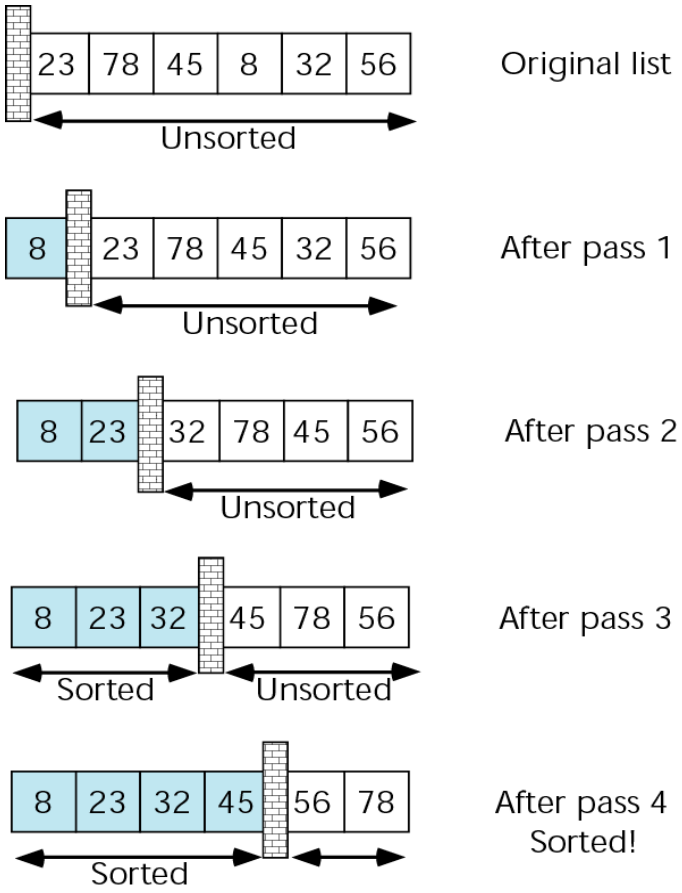


Figure 8-22 Insertion sort concept

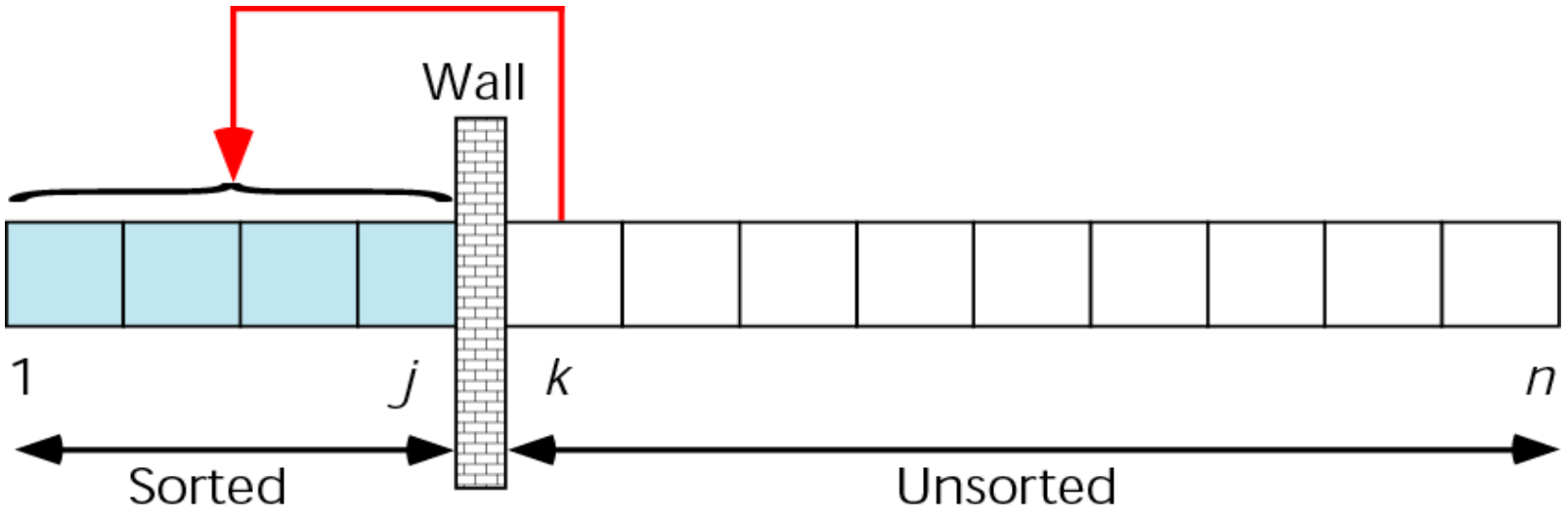
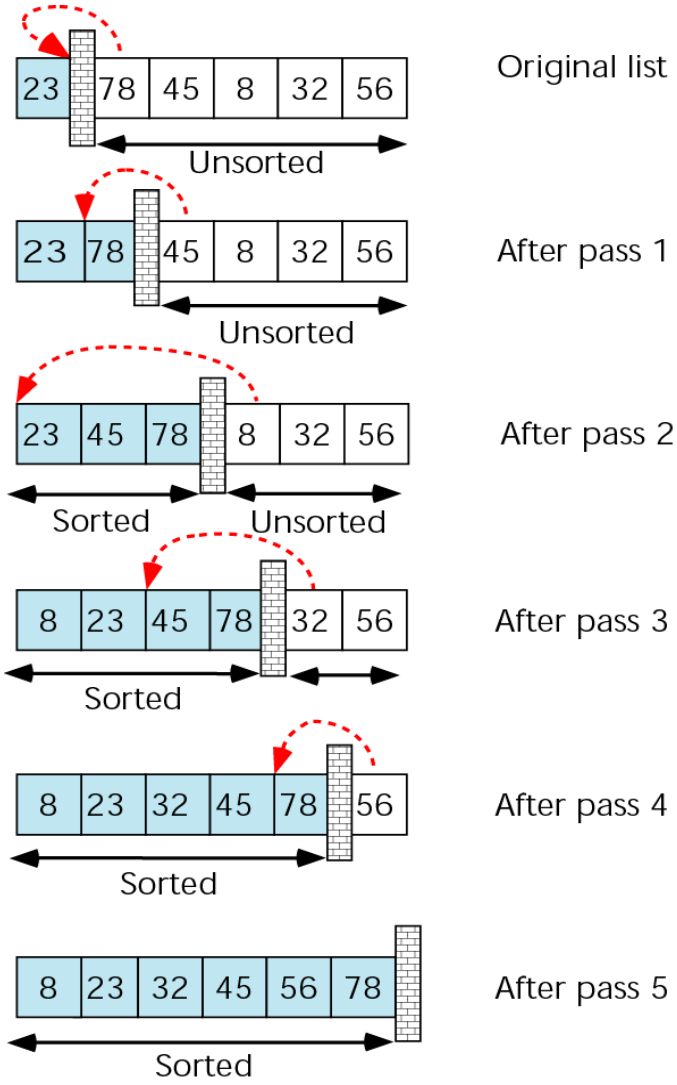


Figure 8-23 Insertion sort example



SEARCHING

Figure 8-24 Search concept

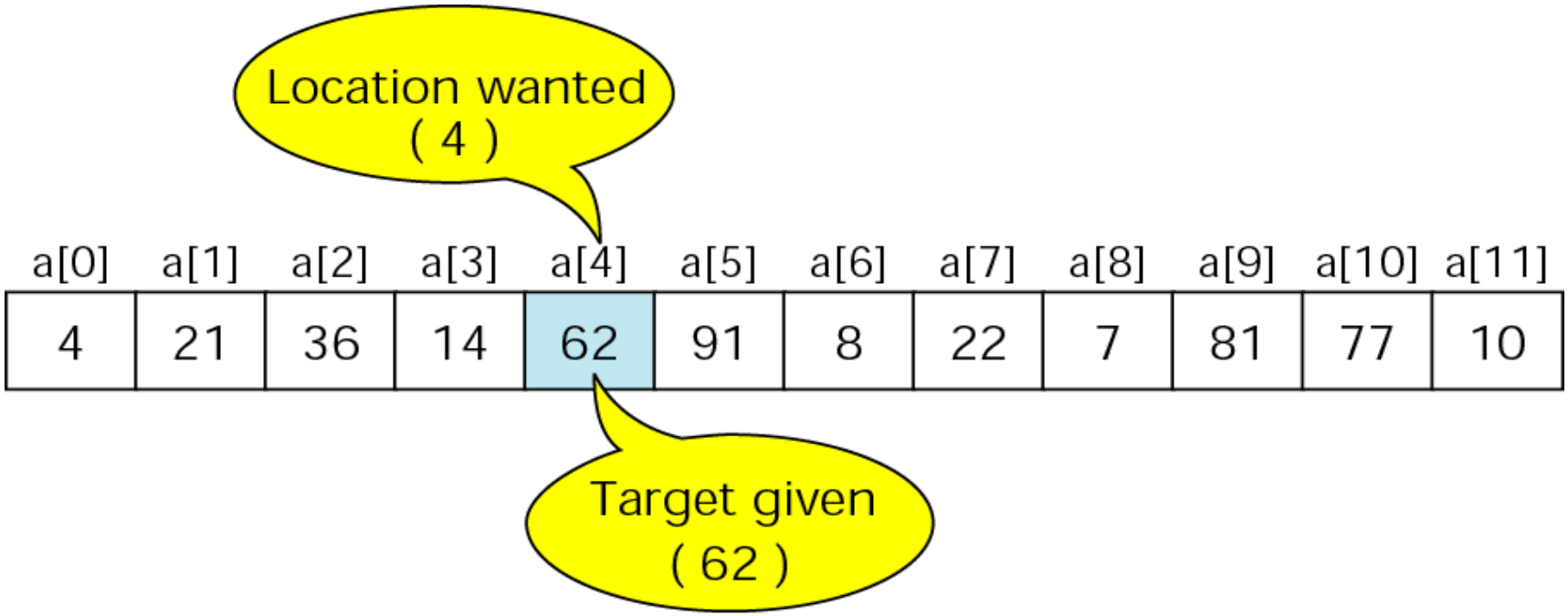


Figure 8-25 Locating data in an unordered list

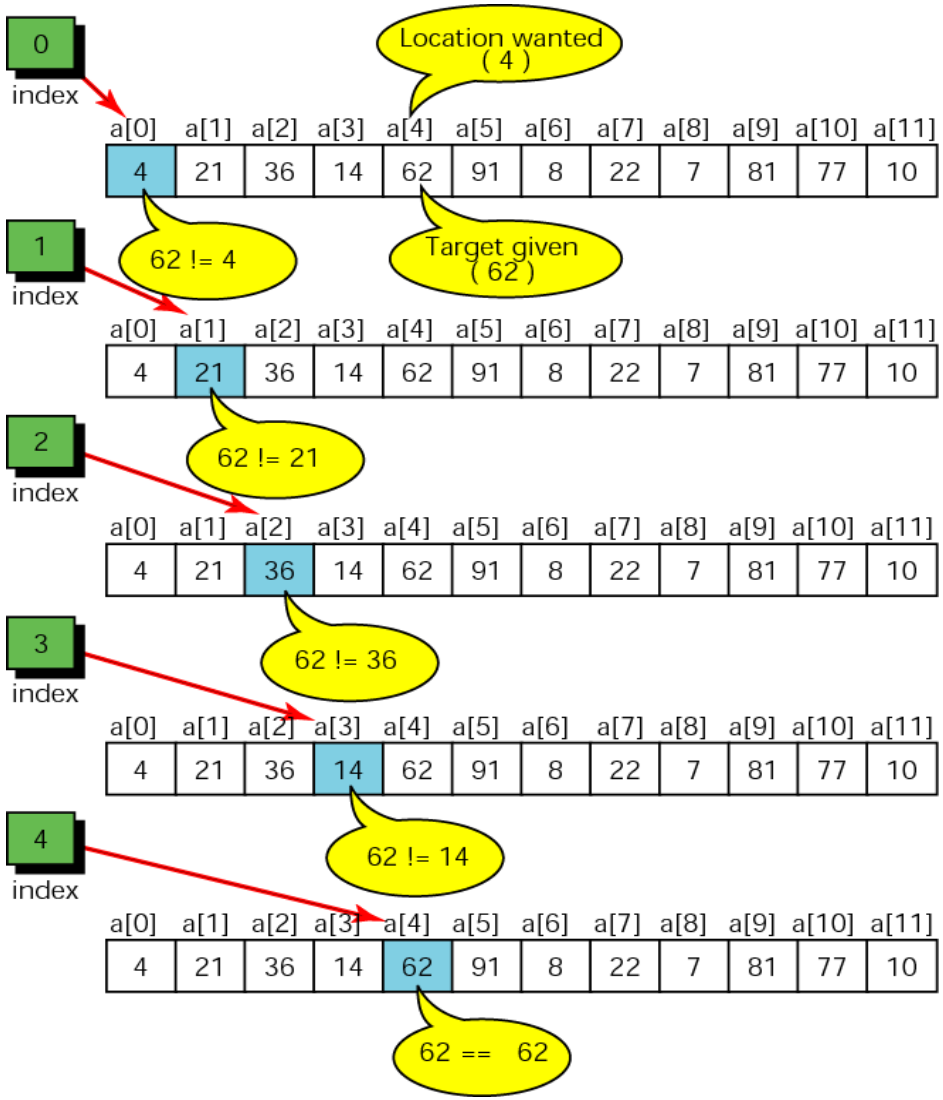
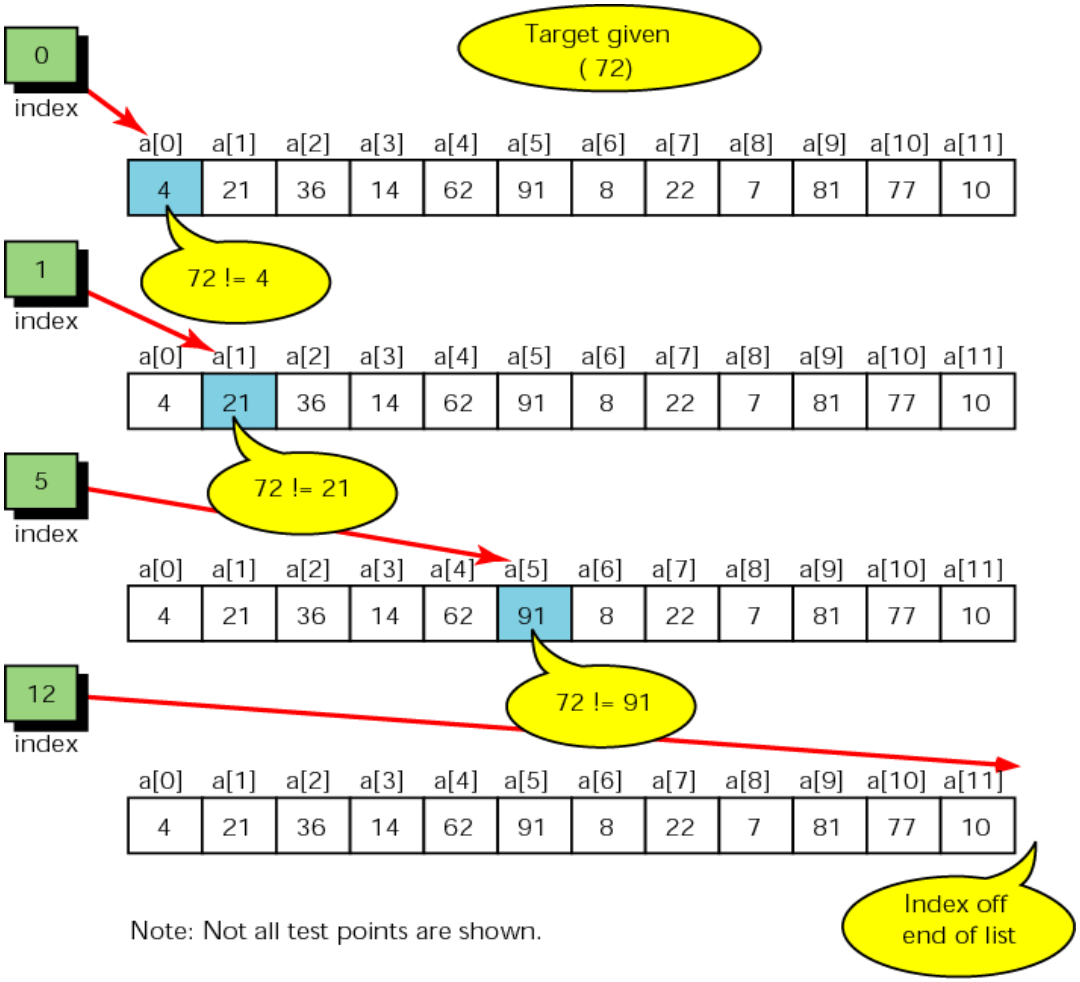


Figure 8-26 Unsuccessful search in an unordered list



Note: Not all test points are shown.

Figure 8-27 Binary search example

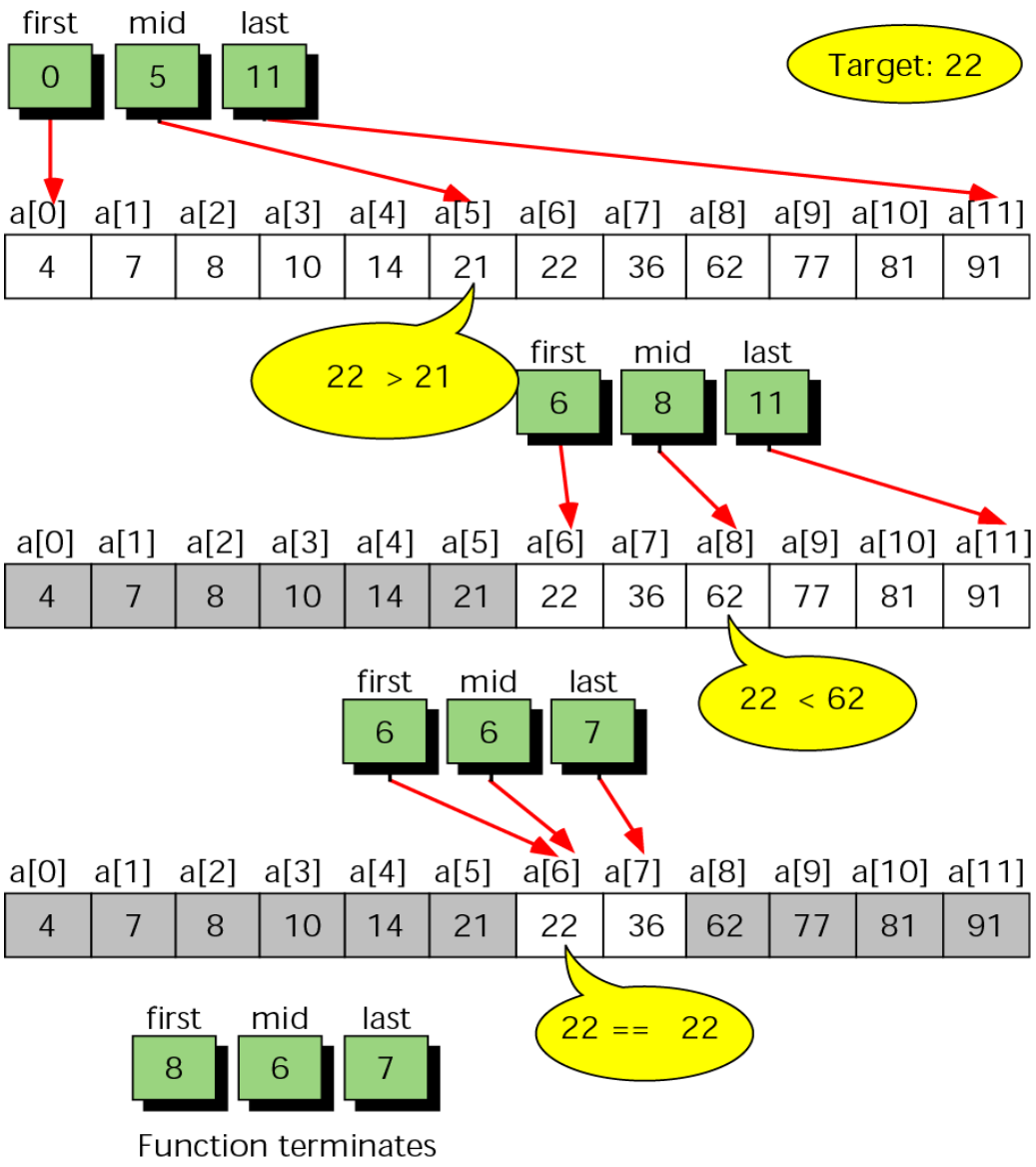
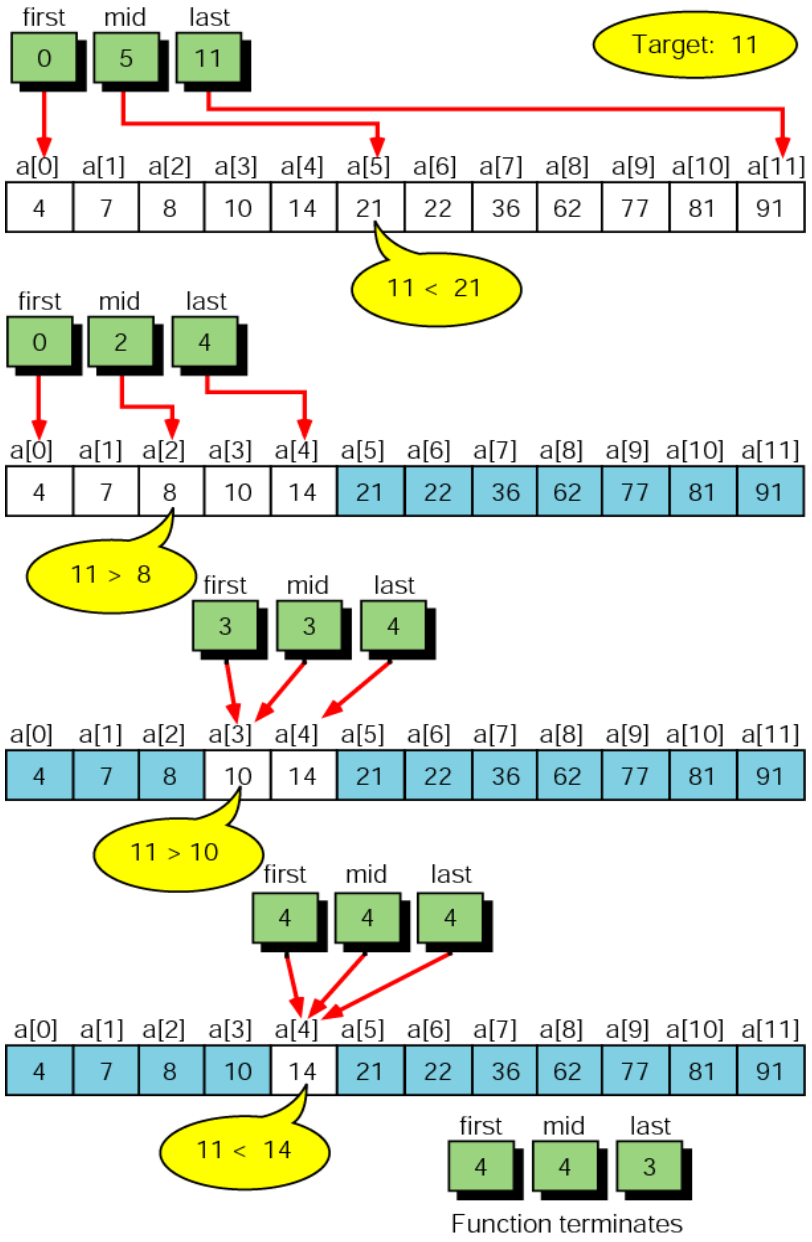


Figure 8-28 Unsuccessful binary search example



Two-Dimensional Arrays

Figure 8-29 Two-dimensional array

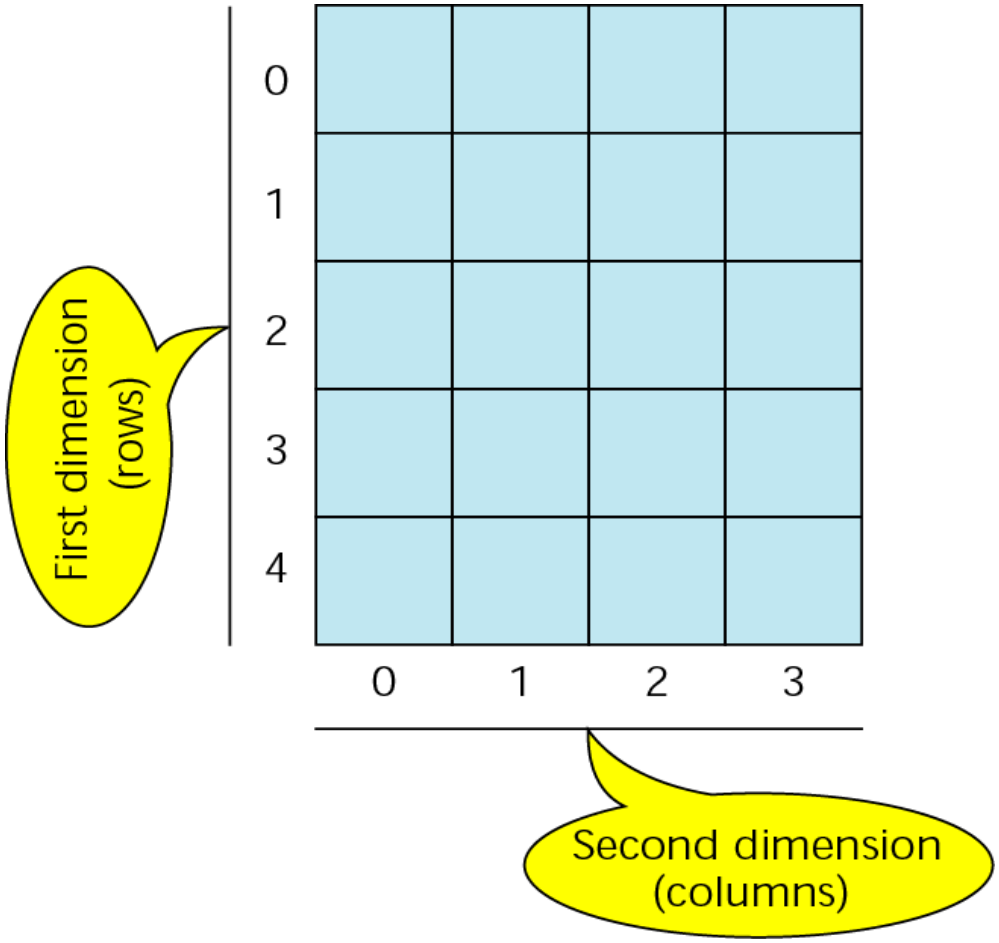


Figure 8-30 Array of arrays

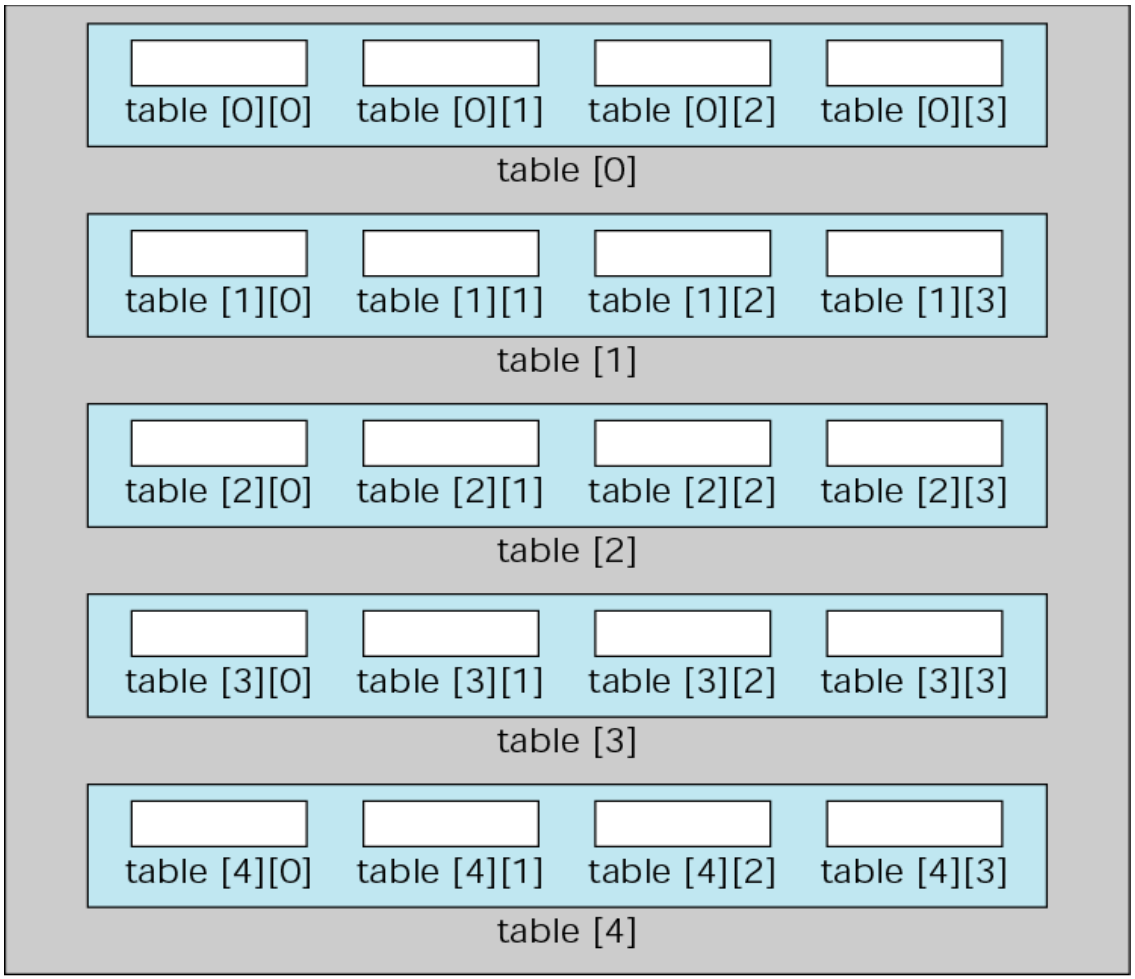
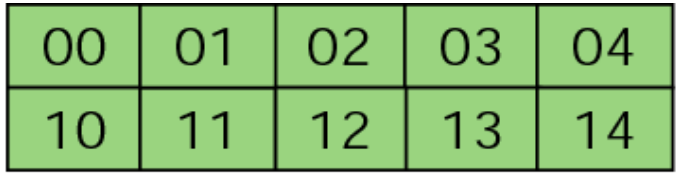
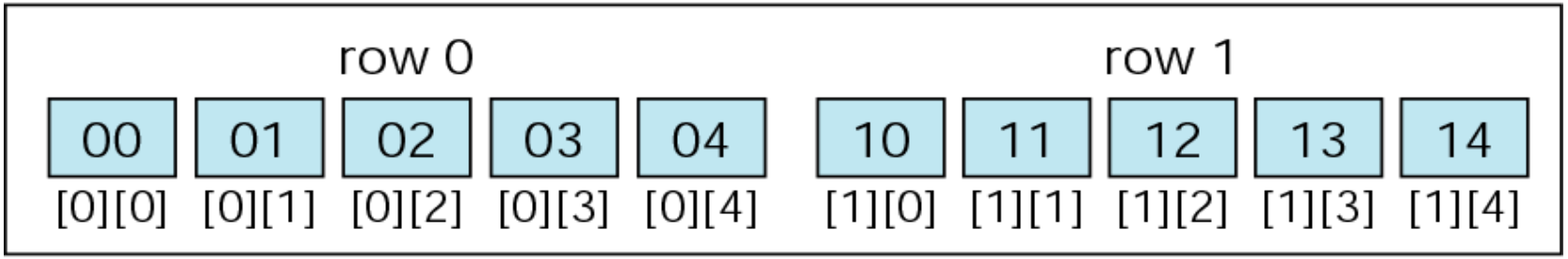


Figure 8-31 Memory layout



User's view



Memory view

Figure 8-32 Passing a row

```
const int cMAX_ROWS  = 5;
const int cMAX_COLS  = 4;

void   print_sqr ( const int [ ] ) ;

int    main ( )
{
  int table [cMAX_ROWS ] [cMAX_COLS]  =
              {
               { 0, 1, 2, 3 } ,
               { 10, 11, 12, 13 } ,
               { 20, 21, 22, 23 } ,
               { 30, 31, 32, 33 } ,
               { 40, 41, 42, 43 }
              } ;  // table
  …
  for ( int row = 0 ;  row <  cMAX_ROWS ; row++ )
                print_sqr ( table [ row ]  ) ;
  …
  return  0 ;
} // main
```

```
void print_sqr (const  int  x [ ] )
{
   for ( int col = 0 ; col < MAX_COLS ; col++ )
       cout << setw(6) <<  x [col] * x [col ] ;
   cout << endl ;
   return ;
}  // print_sqr
```

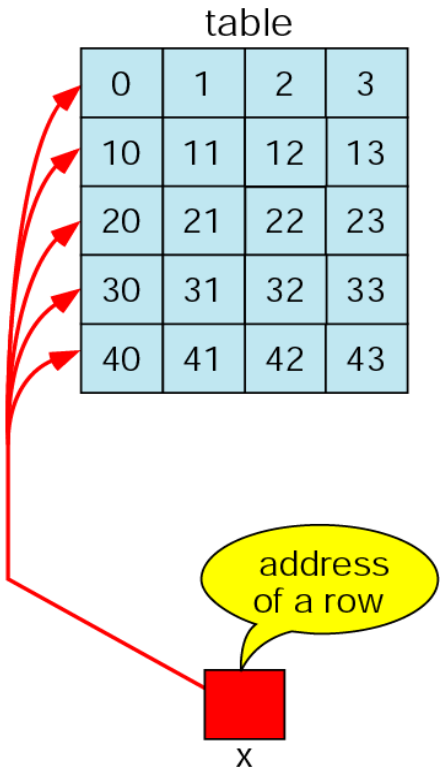


Figure 8-33 Calculate average of integers in array

```
const int cMAX_ROWS    = 5;
const int cMAX_COLS    = 4;
double   average ( int [ ] [cMAX_COLS] ) ;
int    main ( )
{
  int
        ave;
  int
        table [ MAX_ROWS ] [cMAX_COLS]  =
            {
             { 0, 1, 2, 3 } ,
             { 10, 11, 12, 13 } ,
             { 20, 21, 22, 23 } ,
             { 30, 31, 32, 33 } ,
             { 40, 41, 42, 43 }
            } ;
  ...
  ave  =  average ( table ) ;
  ...
  return  0 ;
}  // main
```

table

0	1	2	3
10	11	12	13
20	21	22	23
30	31	32	33
40	41	42	43

```
double  average  ( int  x [ ][ cMAX_COLS]  )
{
    double sum = 0 ;

    for (int  i = 0 ; i < cMAX_ROWS ; i++ )
        for (int  j = 0 ; j < cMAX_COLS ; j++ )
            sum += x [ i ] [ j ];

     return( sum / ( cMAX_ROWS *  cMAX_COLS );
}  // average
```

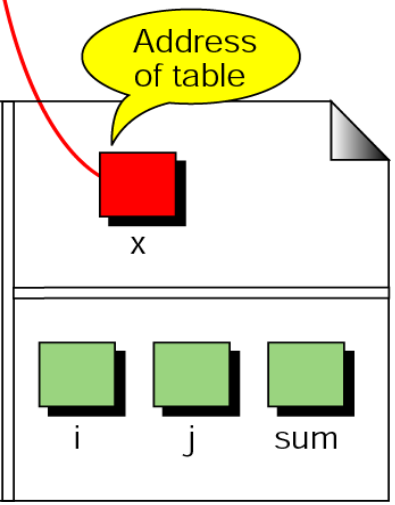
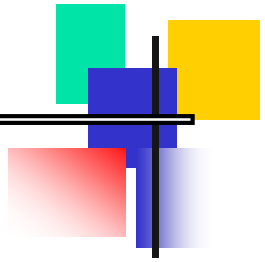


Figure 8-34 Example of filled matrix



0	1	1	1	1	1
-1	0	1	1	1	1
-1	-1	0	1	1	1
-1	-1	-1	0	1	1
-1	-1	-1	-1	0	1
-1	-1	-1	-1	-1	0

MULTIDIMENSIONAL ARRAYS

Figure 8-35 A three-dimensional array (3 x 5 x 4)

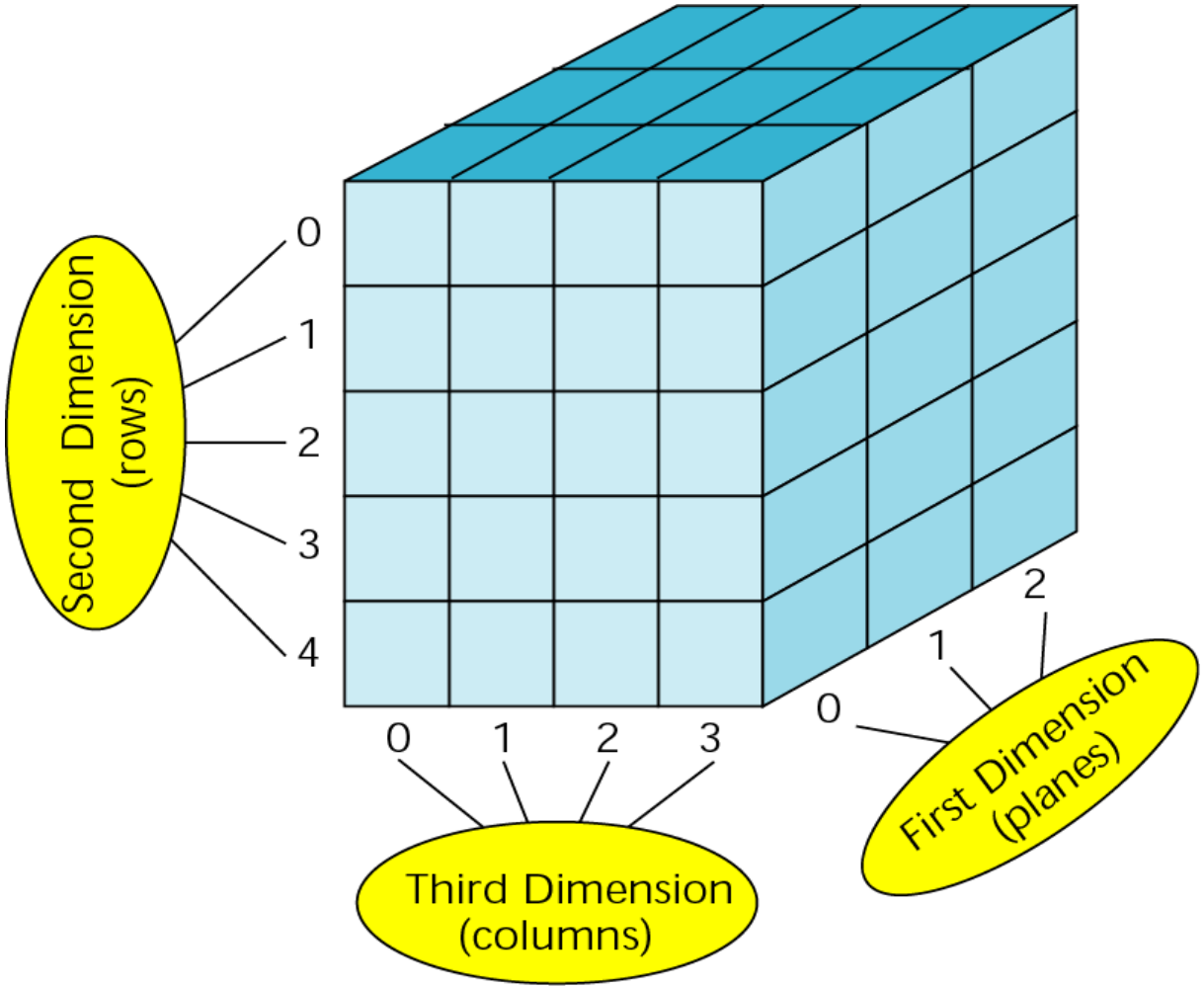


Figure 8-36 C++ view of three-dimensional array

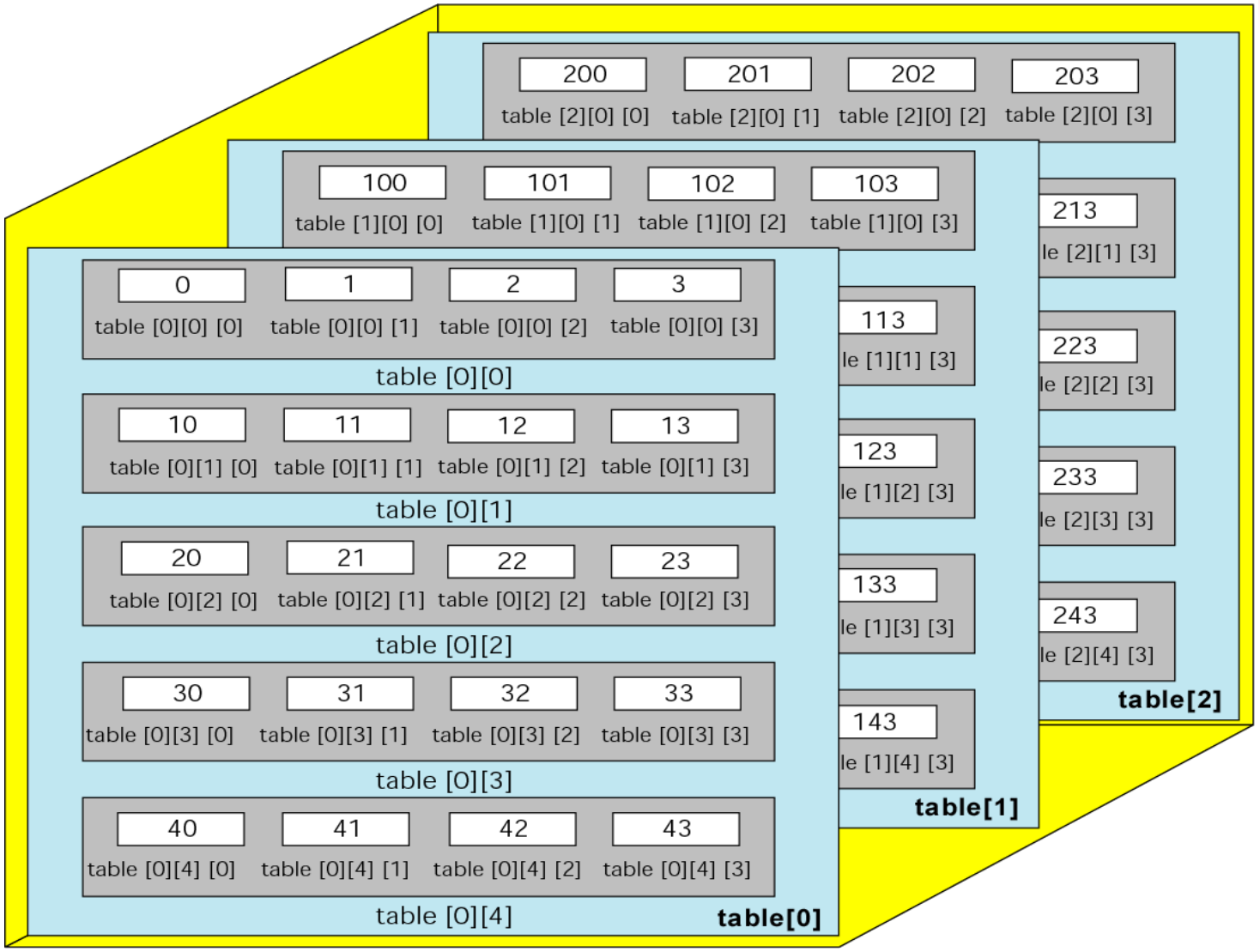


Figure 8-37 Initializing a three-dimensional array

```
int         table[3][5][4] =
{
    {           // Plane 0
                {0, 1, 2, 3},                   // Row 0
                            ...
                {40, 41, 42, 43}                // Row 4
    },
    {           // Plane 1
                 {100, 101, 102, 103},          // Row 0
                            ...
                 {140, 141, 142, 143}           // Row 4
    },
    {           // Plane 2
                  {200, 201, 202, 203},         // Row 0
                            ...
                  {240, 241, 242, 243}          // Row 4
    }

}; // table
```

Programming Example—Calculate Row and Column Averages

Figure 8-38 Data structures for row-column averages

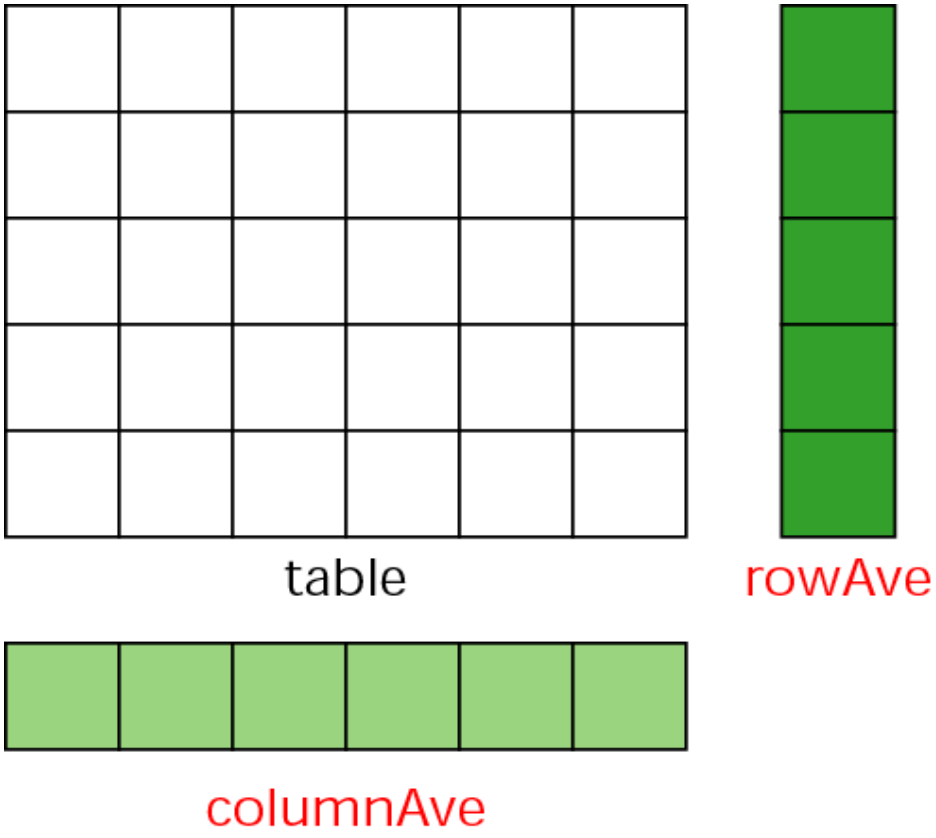
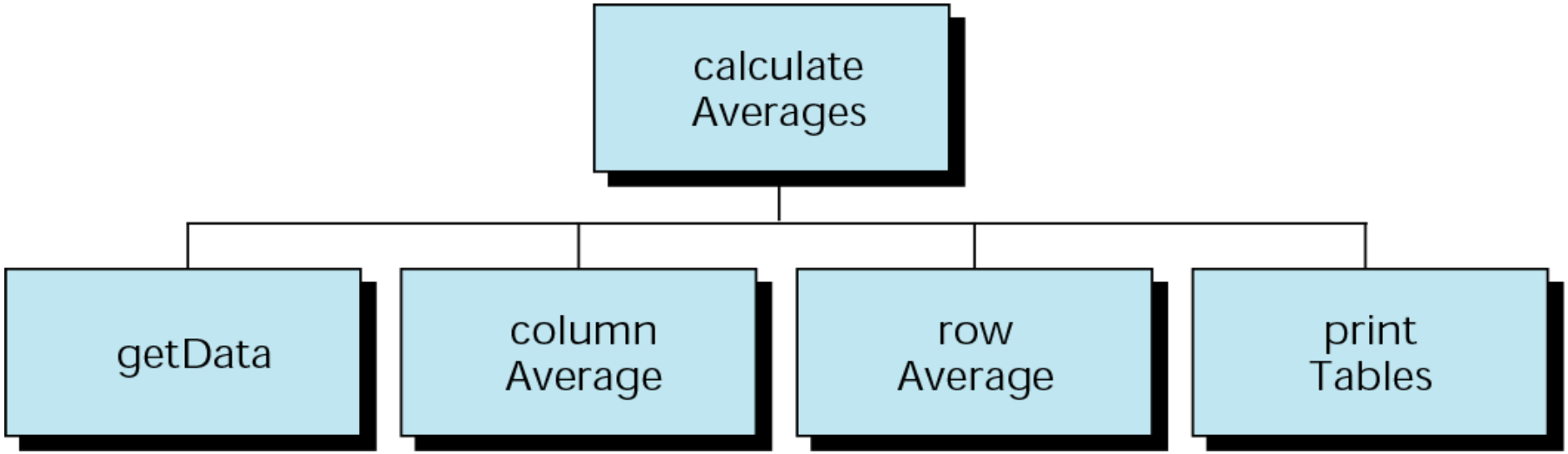


Figure 8-39 Calculate row-column average design



Software Engineering and Programming Style

Note:

The efficiency of the bubble sort is O(n2).

Note:

The efficiency of the selection sort is O(n2).

Note:

The efficiency of the insertion sort is O(n2).

Note:

The efficiency of the sequential search is O(n).

Note:

The efficiency of the binary search is O(log2n).