Programming Logic and Design
Ninth Edition

Chapter 6
Arrays

Objectives

In this chapter, you will learn about:

- Arrays
- How an array can replace nested decisions
- Using constants with arrays
- Searching an array for an exact match
- Using parallel arrays
- Searching an array for a range match
- Remaining within array bounds
- Using a `for` loop to process arrays

Understanding Arrays

- **Array**
  - A series or list of variables in computer memory
  - All variables share the same name, and must be the same data type
  - Each variable has a different subscript

How Arrays Occupy Computer Memory

- **Element**: an item in the array
  - Array elements are contiguous in memory
- **Size of the array**: the number of elements it will hold
- **Subscripts** or **indexes**
  - Position number of an item in an array starting from 0 to one less than the number of elements in array
  - Subscripts are always a sequence of integers
- Adding data values is called **populating the array**

How Arrays Occupy Computer Memory (continued)

When programmers refer to array element `prices[0]`, they say "`prices` sub 0" or simply "`prices` zero."

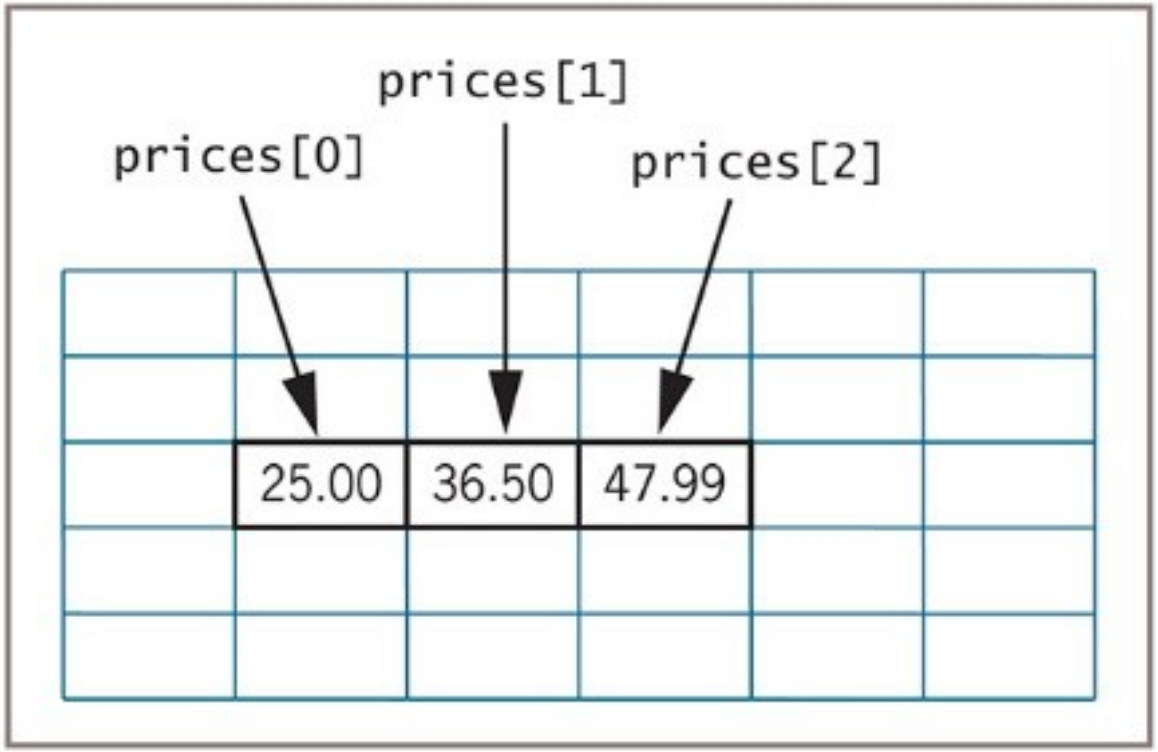


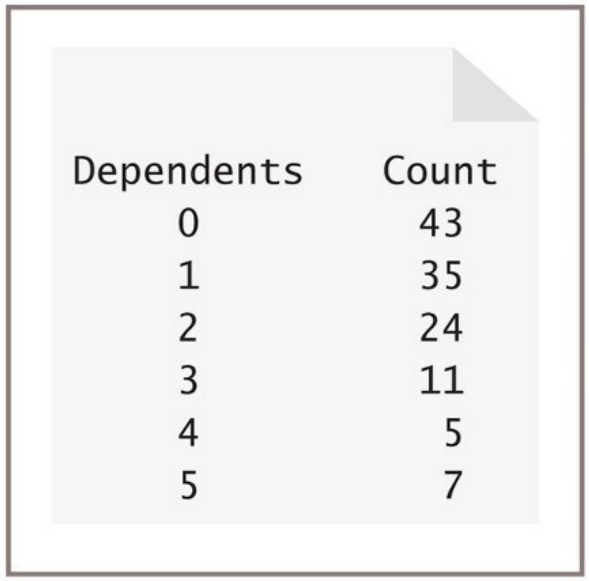
Figure 6-1 Appearance of a three-element array in computer memory

Characteristics of Arrays

- An array is a list of data items in contiguous memory locations
- Each data item in an array is an *element*
- Each array element is the same data type and the same size
- Each element is differentiated from the others by a subscript, which is a whole number
- Usable subscripts for an array range from 0 to one less than the number of elements in an array
- Each array element can be used in the same way as a single item of the same data type

How an Array Can Replace Nested Decisions

- Example: Human Resources Department Dependents report
  - List employees who have claimed zero through five dependents
    - Assume no employee has more than five dependents
- Application produces counts for dependent categories
  - Uses a series of decisions
- Application does not scale up to more dependents



Dependents	Count
0	43
1	35
2	24
3	11
4	5
5	7

Figure 6-2 Typical Dependents report

How an Array Can Replace Nested Decisions

(continued -1)

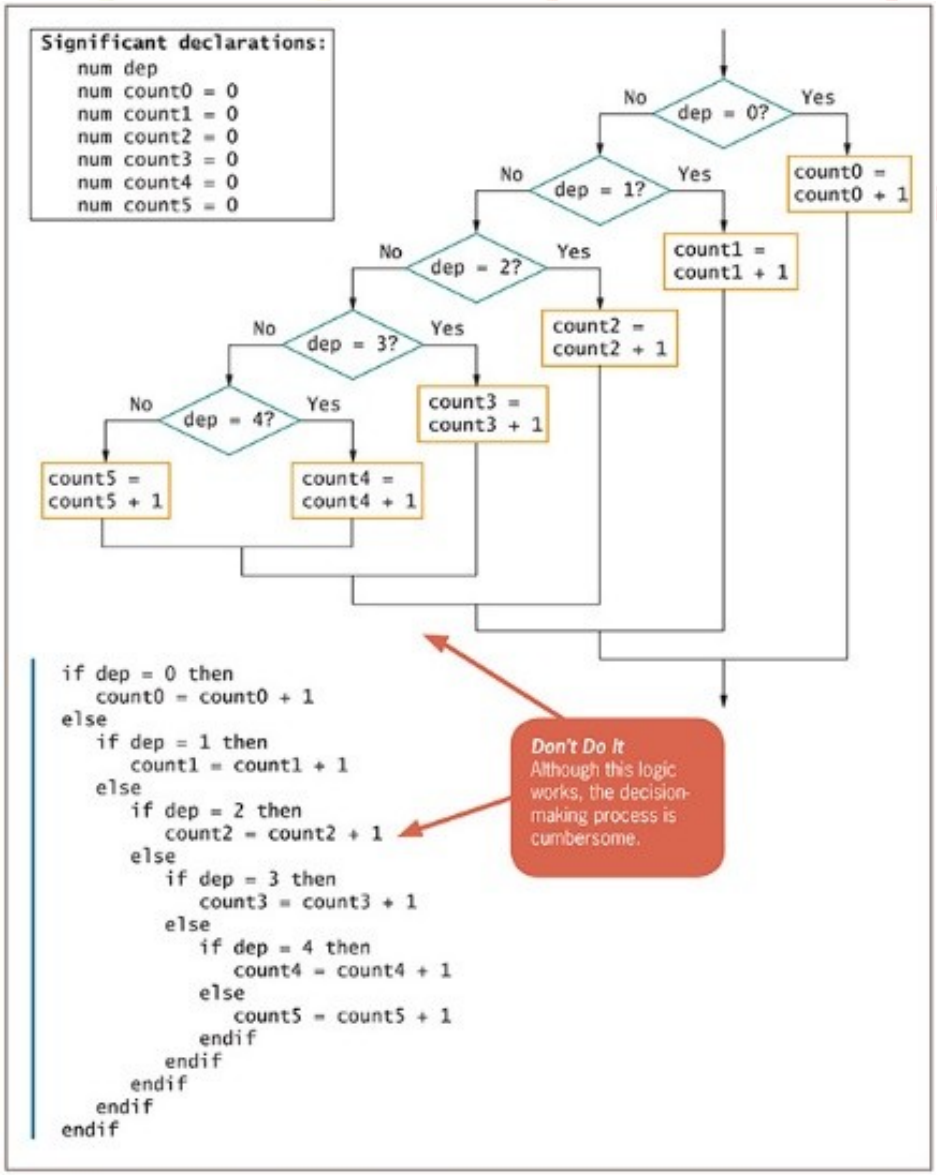


Figure 6-3 Flowchart and pseudocode of decision-making process using a series of decisions—the hard way

How an Array Can Replace Nested Decisions (continued -2)

- The array reduces the number of statements needed
- Six dependent count accumulators are redefined as a single array
- Variable as a subscript to the array
- Array subscript variable must be:
  - Numeric with no decimal places
  - Initialized to 0
  - Incremented by 1 each time the logic passes through the loop

How an Array Can Replace Nested Decisions

(continued -4)

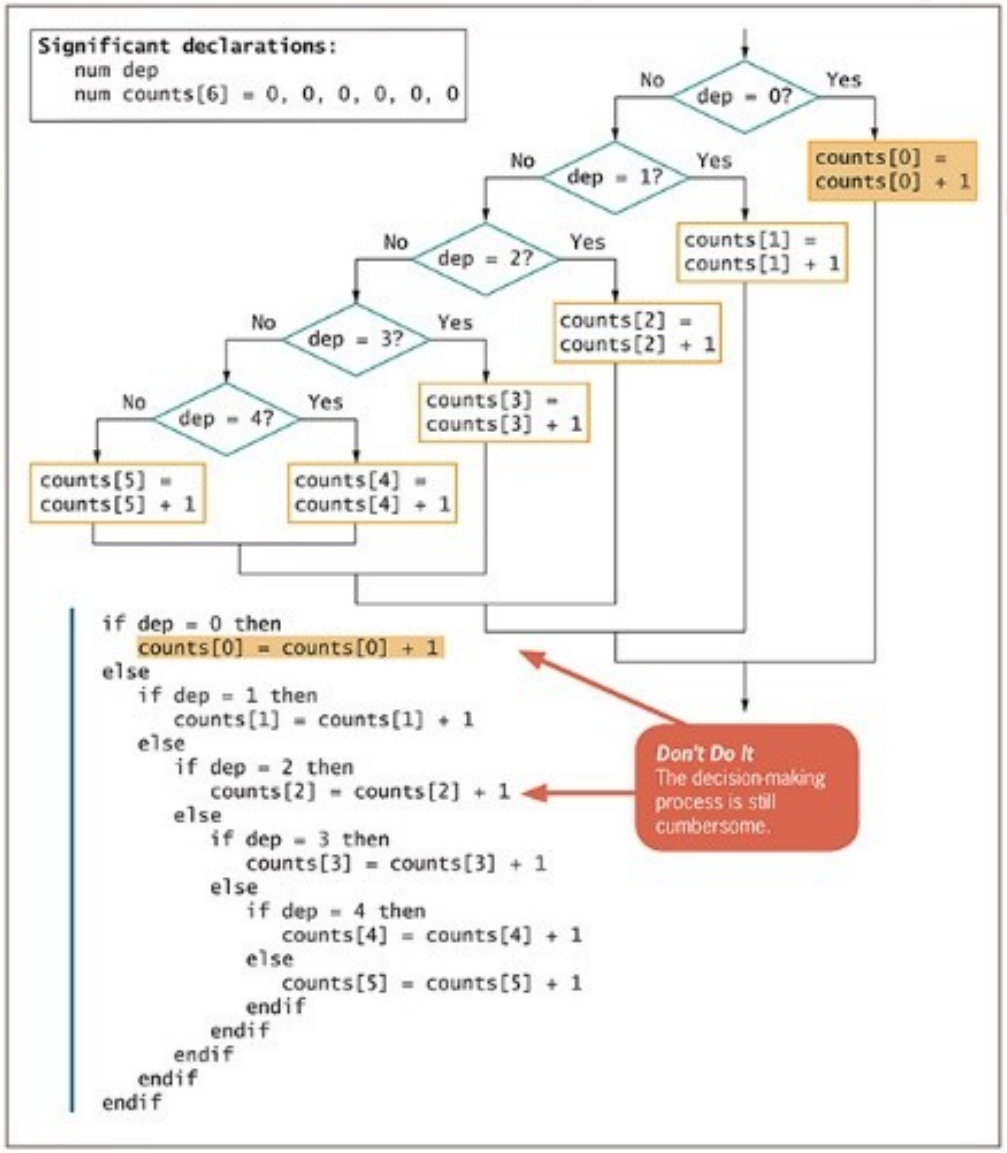


Figure 6-4 Flowchart and pseudocode of decision-making process—but still the hard way

How an Array Can Replace Nested Decisions

(continued -5)

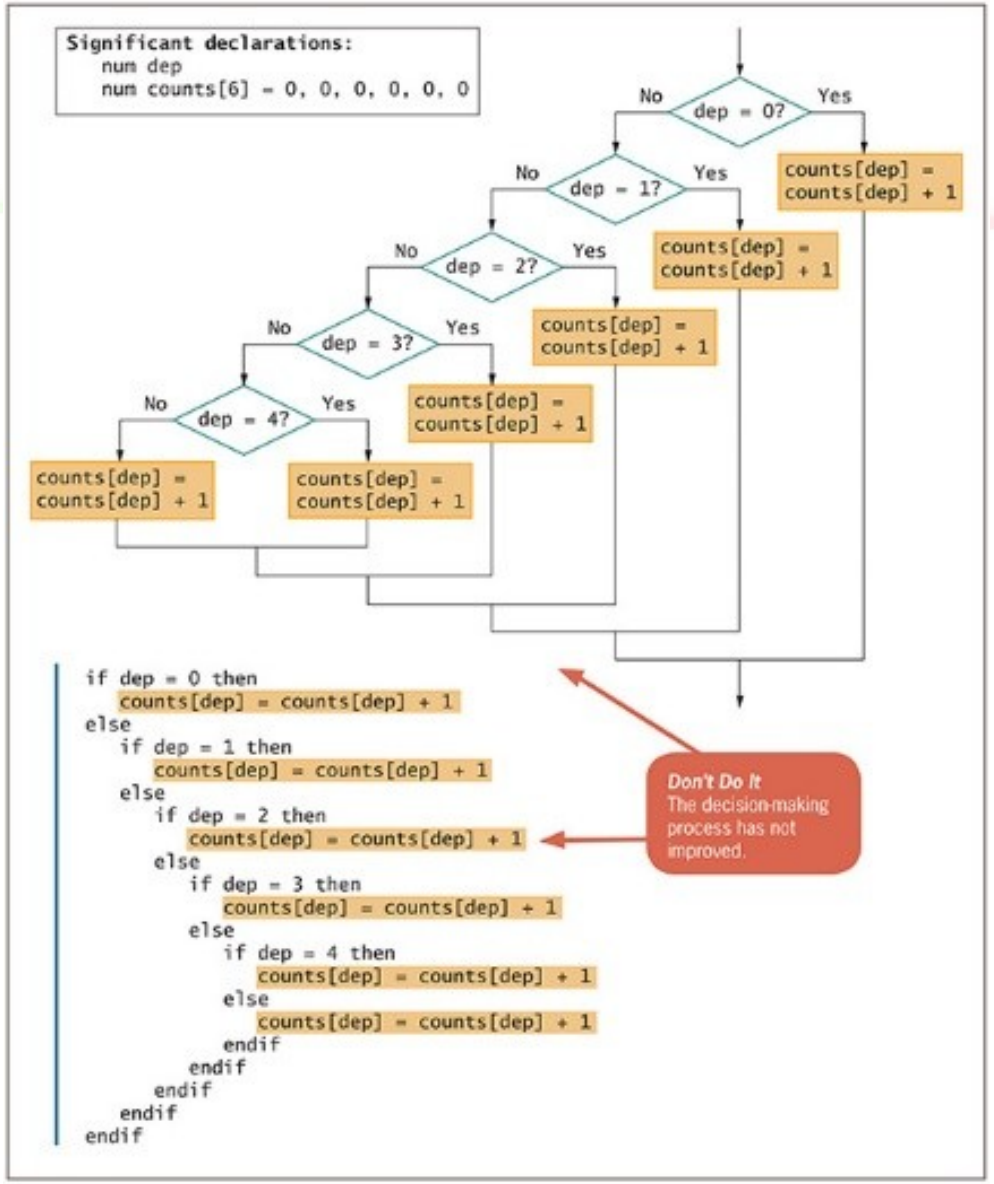


Figure 6-5 Flowchart and pseudocode of decision-making process using an array—but still a hard way

How an Array Can Replace Nested Decisions (continued -6)

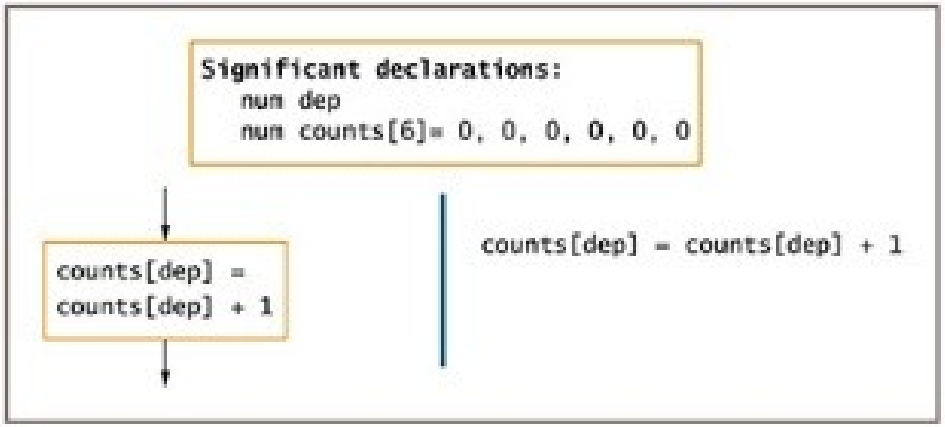


Figure 6-6 Flowchart and pseudocode of efficient decision-making process using an array

How an Array Can Replace Nested Decisions

(continued -7)

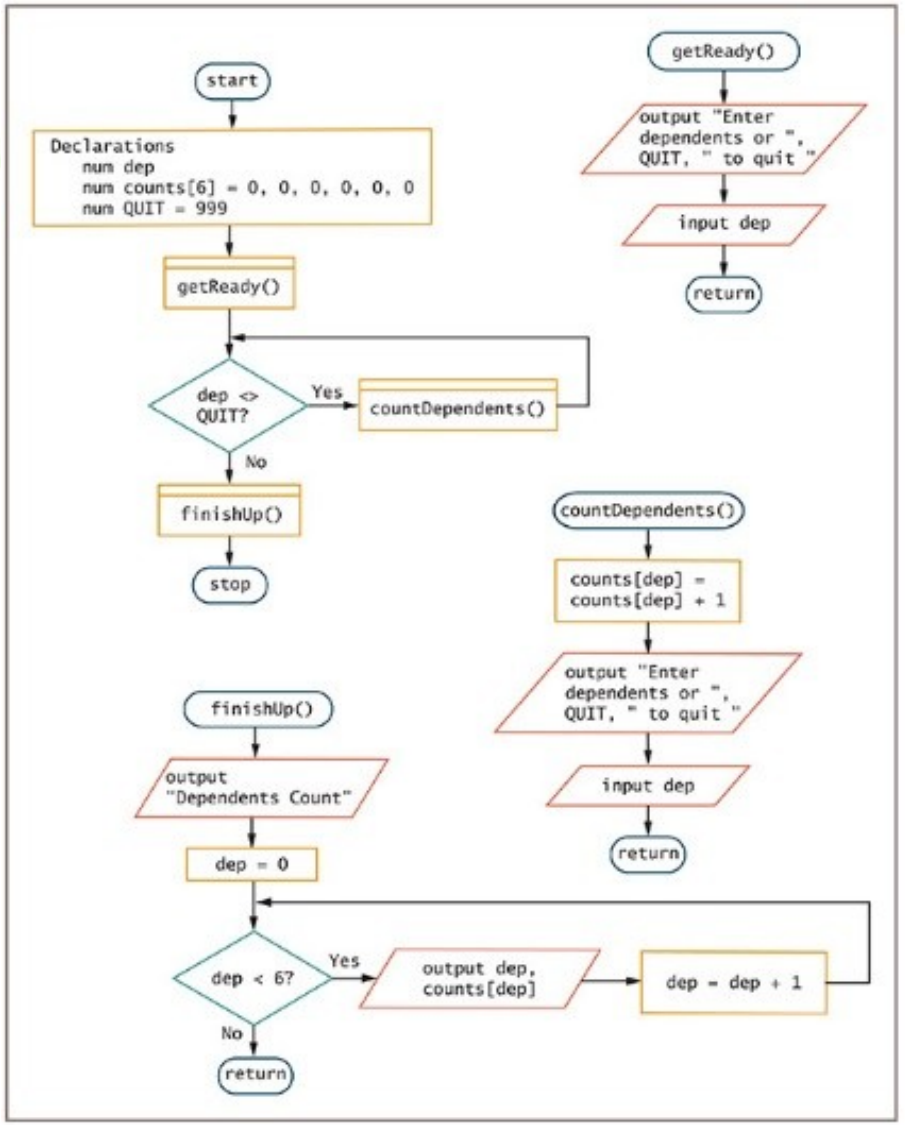


Figure 6-7 Flowchart and pseudocode for Dependents report program *(continues)*

How an Array Can Replace Nested Decisions

(continued -8)

(continued)

```
start
   Declarations
      num dep
      num counts[6] = 0, 0, 0, 0, 0, 0
      num QUIT = 999
   getReady()
   while dep <> QUIT
      countDependents()
   endwhile
   finishUp()
stop

getReady()
   output "Enter dependents or ", QUIT, " to quit "
   input dep
return

countDependents()
   counts[dep] = counts[dep] + 1
   output "Enter dependents or ", QUIT, " to quit "
   input dep
return

finishUp()
   output "Dependents Count"
   dep = 0
   while dep < 6
      output dep, counts[dep]
      dep = dep + 1
   endwhile
return
```

Figure 6-7 Flowchart and pseudocode for Dependents report program

Using Constants with Arrays

- Use constants in several ways:
  - To hold the size of an array
  - As the array values
  - As subscripts

Using a Constant as the Size of an Array

- Avoid “magic numbers” (unnamed constants)
- Declare a named numeric constant to be used every time the array is accessed
- Make sure any subscript remains less than the constant value
- Constants are created automatically in many languages

Using Constants as Array Element Values

- Sometimes the values stored in arrays should be constants
- Example

```
string MONTH[12] = "January",
"February", "March", "April", "May",
"June", "July", "August", "September",
"October", "November", "December"
```

Using a Constant as an Array Subscript

- Use a numeric constant as a subscript to an array
- Example
  - Declare a named constant as: `num INDIANA = 5`
  - Display value with:

```
output salesArray[INDIANA]
```

Searching an Array for an Exact Match

- Sometimes you must search through an entire array to find a value – called a **linear search**
- Example: mail-order business
  - Item numbers are three-digit, non-consecutive numbers
  - Customer orders an item; check if item number is valid
  - Create an array that holds valid item numbers
  - Search the array for an exact match

Searching an Array for an Exact Match (continued -1)

- **Flag**: a variable that indicates whether an event occurred
- Technique for searching an array
  - Set a subscript variable to 0 to start at the first element
  - Initialize a flag variable to false to indicate the desired value has not been found
  - Examine each element in the array
  - If the value matches, set the flag to `True`
  - If the value does not match, increment the subscript and examine the next array element

Searching an Array for an Exact Match (continued -2)

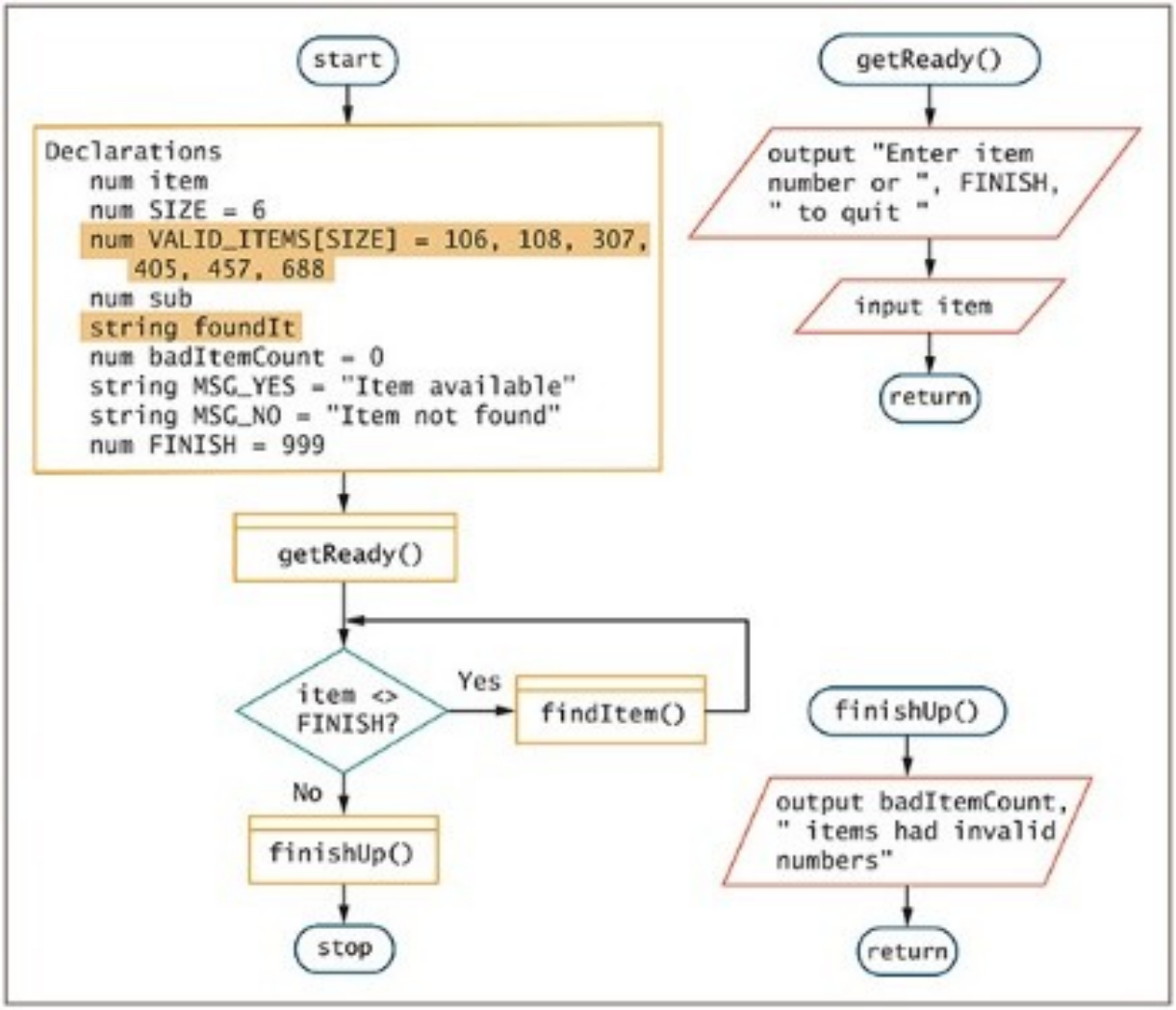


Figure 6-8 Flowchart and pseudocode for program that verifies item availability *(continues)*

Searching an Array for an Exact Match (continued -3)

(continued)

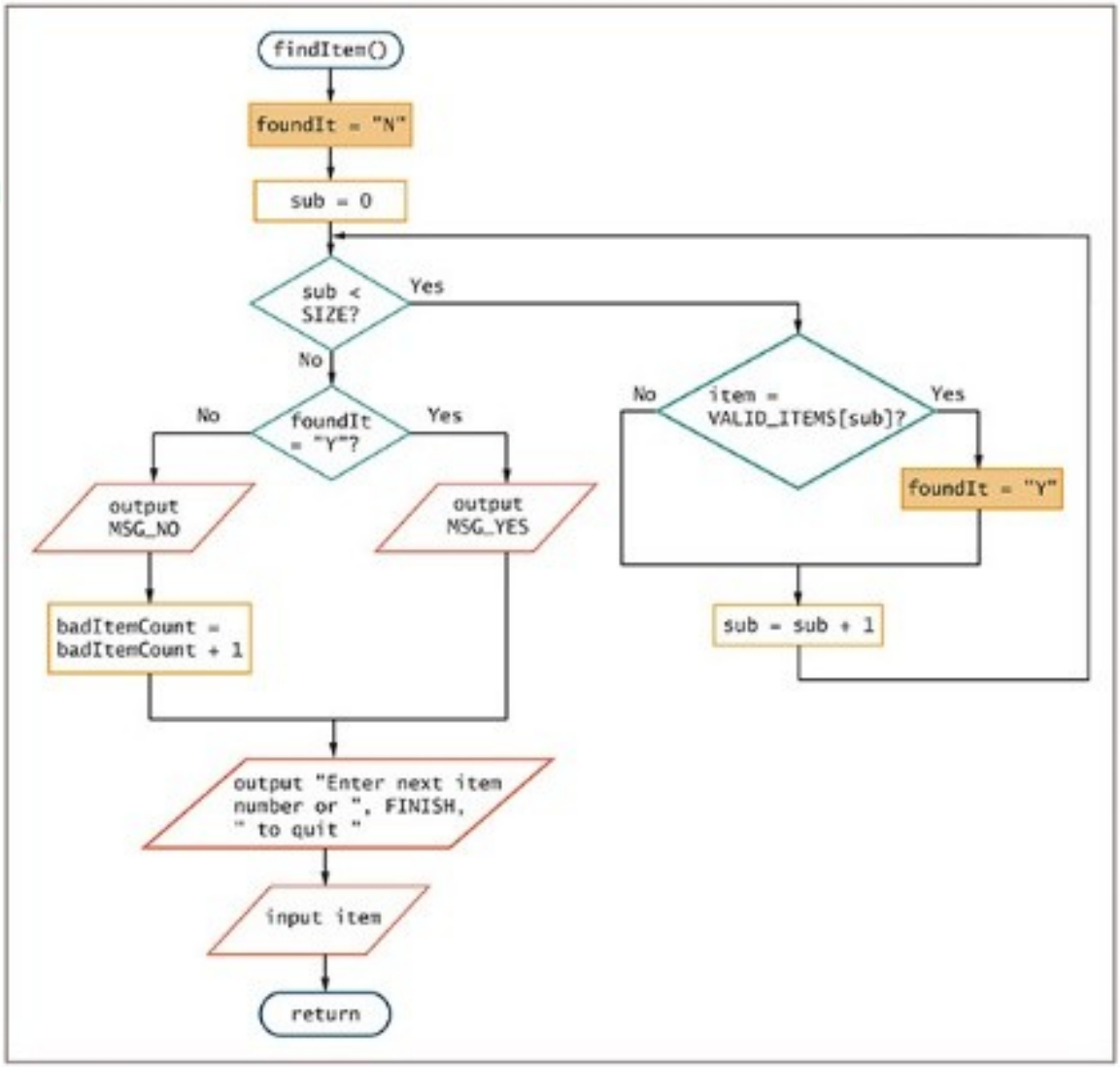


Figure 6-8 Flowchart and pseudocode for program that verifies item availability *(continues)*

Searching an Array for an Exact Match (continued -4)

(continued)

```
start
   Declarations
      num item
      num SIZE = 6
      num VALID_ITEMS[SIZE] = 106, 108, 307,
         405, 457, 688
      num sub
      string foundIt
      num badItemCount = 0
      string MSG_YES = "Item available"
      string MSG_NO = "Item not found"
      num FINISH = 999
   getReady()
   while item <> FINISH
      findItem()
   endwhile
   finishUp()
stop

getReady()
   output "Enter item number or ", FINISH, " to quit "
   input item
return

findItem()
   foundIt = "N"
   sub = 0
   while sub < SIZE
      if item = VALID_ITEMS[sub] then
         foundIt = "Y"
      endif
      sub = sub + 1
   endwhile
   if foundIt = "Y" then
      output MSG_YES
   else
      output MSG_NO
      badItemCount = badItemCount + 1
   endif
   output "Enter next item number or ", FINISH, " to quit "
   input item
return

finishUp()
   output badItemCount, " items had invalid numbers"
return
```

Figure 6-8 Flowchart and pseudocode for program that verifies item availability

Using Parallel Arrays

- Example: mail-order business
  - Two arrays, each with six elements
    - Valid item numbers
    - Valid item prices
  - Each price in the valid item price array is in the same position as the corresponding item in the valid item number array
- **Parallel arrays**
  - Each element in one array is associated with an element in the same relative position in the other array
- Look through the valid item array for the customer's item
  - When a match is found, get the price from the item price array

Using Parallel Arrays (continued -1)

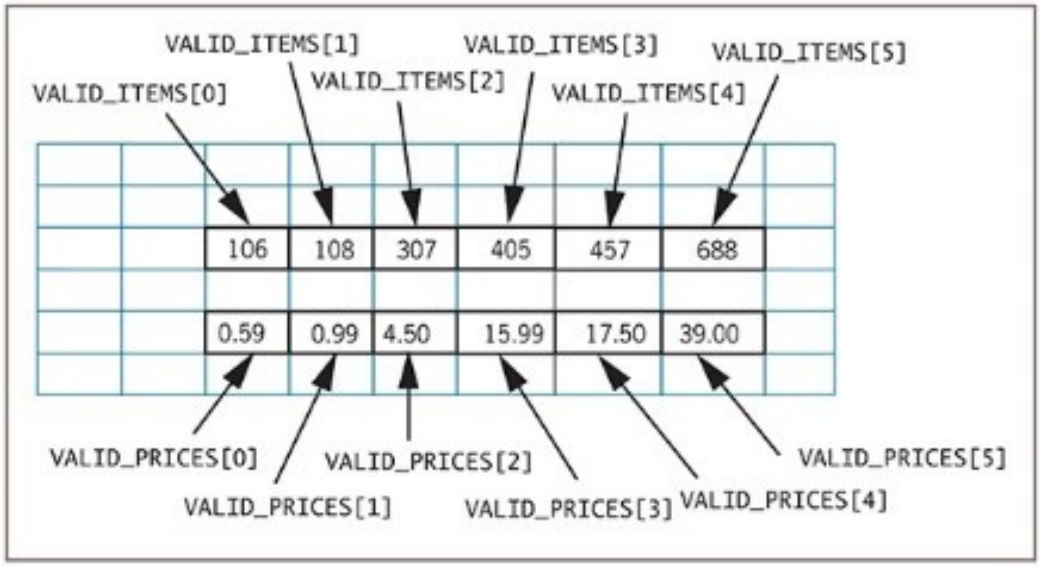


Figure 6-9 Parallel arrays in memory

Using Parallel Arrays (continued -2)

- Use parallel arrays when:
  - Two or more arrays contain related data
  - A subscript relates the arrays
    - Elements at the same position in each array are logically related
- **Indirect relationship**
  - Relationship between an item's number and its price
  - Parallel arrays are very useful

Using Parallel Arrays

(continued -3)

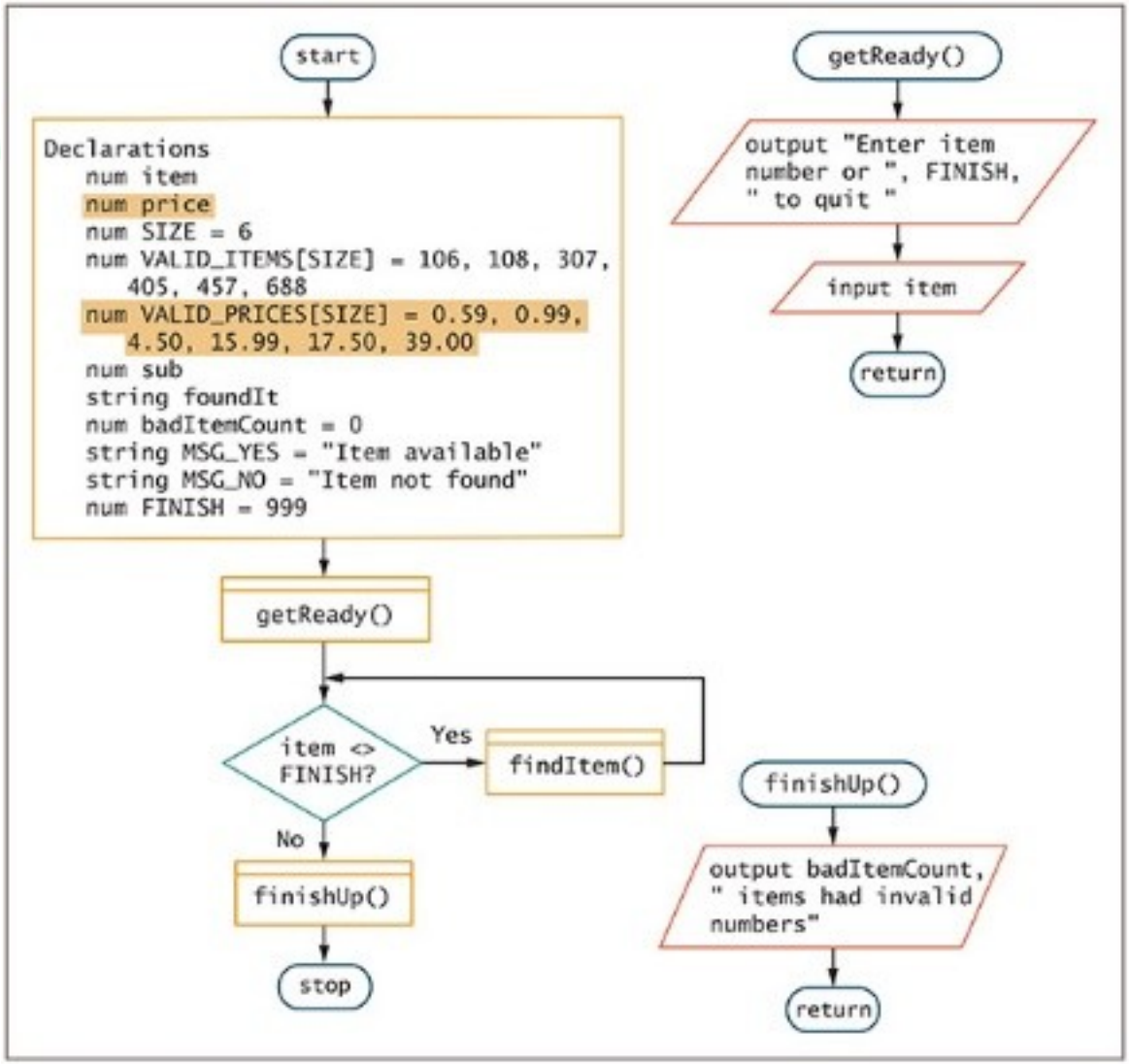


Figure 6-10 Flowchart and pseudocode of program that finds an item price using parallel arrays *(continues)*

Using Parallel Arrays

(continued -4)

(continued)

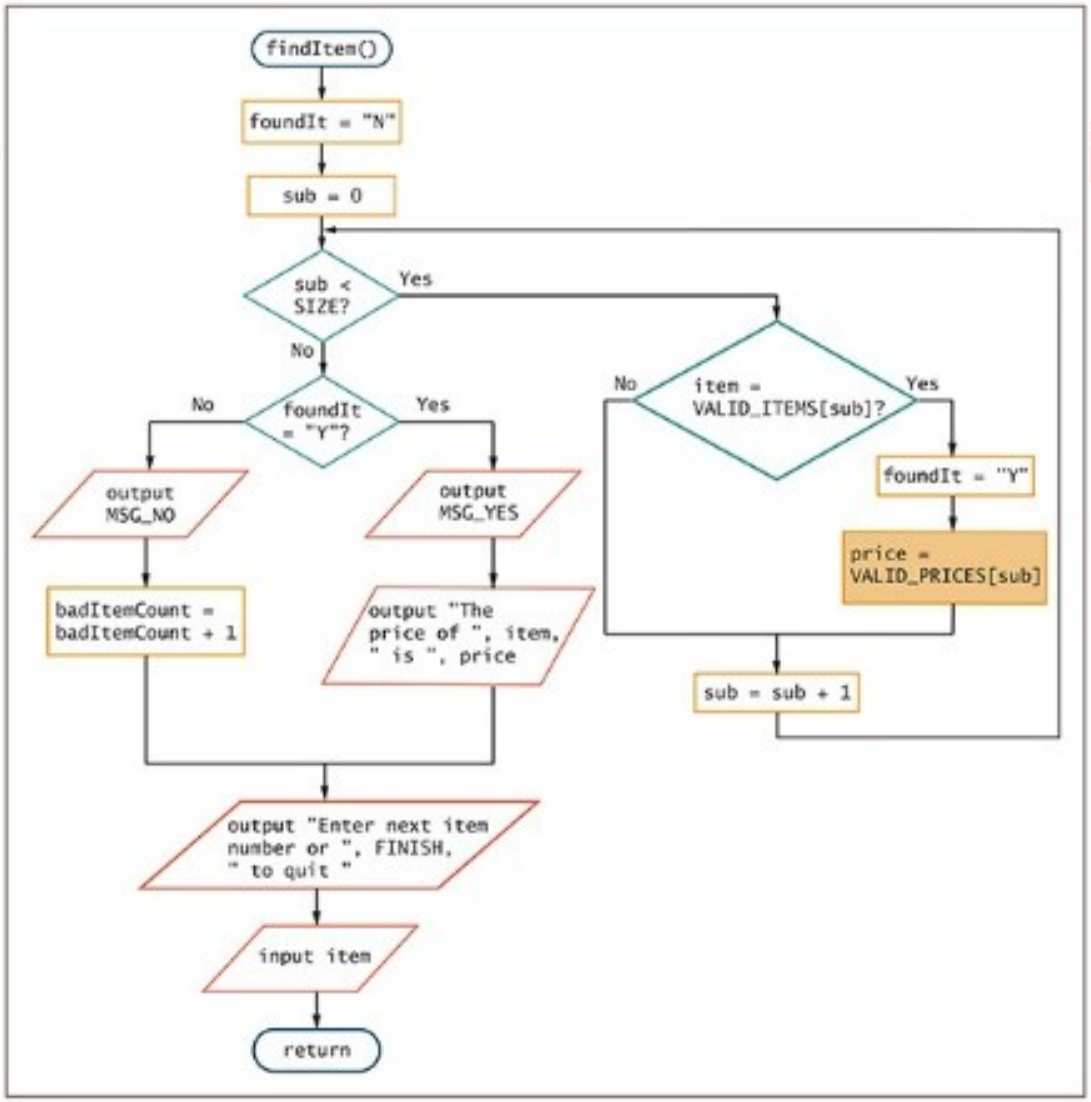


Figure 6-10 Flowchart and pseudocode of program that finds an item price using parallel arrays

(continues)

Using Parallel Arrays

(continued -5)

(continued)

```
start
   Declarations
      num item
      num price
      num SIZE = 6
      num VALID_ITEMS[SIZE] = 106, 108, 307,
         405, 457, 688
      num VALID_PRICES[SIZE] = 0.59, 0.99,
         4.50, 15.99, 17.50, 39.00
      num sub
      string foundIt
      num badItemCount = 0
      string MSG_YES = "Item available"
      string MSG_NO = "Item not found"
      num FINISH = 999
   getReady()
   while item <> FINISH
      findItem()
   endwhile
   finishUp()
stop

getReady()
   output "Enter item number or ", FINISH, " to quit "
   input item
return

findItem()
   foundIt = "N"
   sub = 0
   while sub < SIZE
      if item = VALID_ITEMS[sub] then
         foundIt = "Y"
         price = VALID_PRICES[sub]
      endif
      sub = sub + 1
   endwhile
   if foundIt = "Y" then
      output MSG_YES
      output "The price of ", item, " is ", price
   else
      output MSG_NO
      badItemCount = badItemCount + 1
   endif
   output "Enter next item number or ", FINISH, " to quit "
   input item
return

finishUp()
   output badItemCount, " items had invalid numbers"
return
```

Figure 6-10 Flowchart and pseudocode of program that finds an item price using parallel arrays

Improving Search Efficiency

- The program should stop searching the array when a match is found
- Set a variable to a specific value instead of letting normal processing set it
- Leaving a loop as soon as a match is found improves efficiency
- The larger the array, the more beneficial it becomes to do an early exit

Improving Search Efficiency

(continued -1)

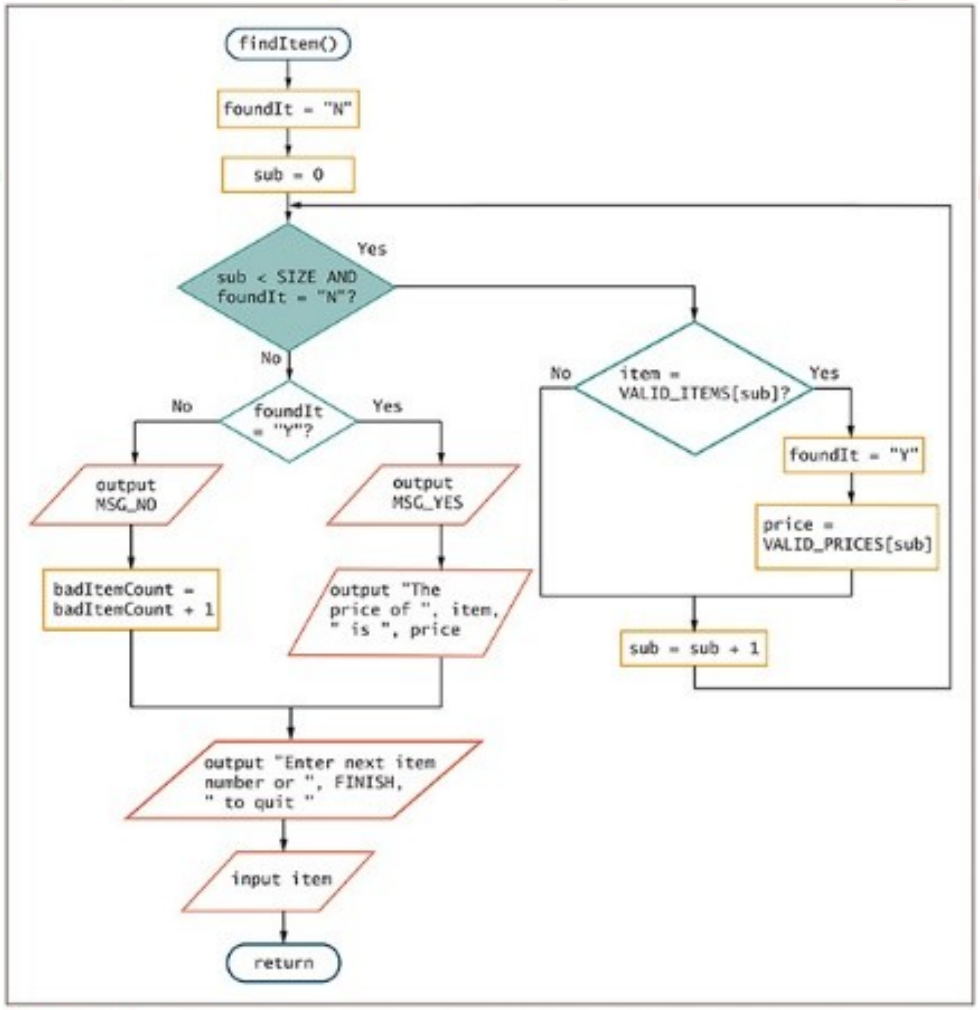


Figure 6-11 Flowchart and pseudocode of the module that finds an item price and exits the loop as soon as it is found (continues)

Improving Search Efficiency

(continued -2)

(continued)

```
findItem()
   foundIt = "N"
   sub = 0
   while sub < SIZE AND foundIt = "N"
      if item = VALID_ITEMS[sub] then
         foundIt = "Y"
         price = VALID_PRICES[sub]
      endif
      sub = sub + 1
   endwhile
   if foundIt = "Y" then
      output MSG_YES
      output "The price of ", item, " is ", price
   else
      output MSG_NO
      badItemCount = badItemCount + 1
   endif
   output "Enter next item number or ", FINISH, " to quit "
   input item
return
```

Figure 6-11 Flowchart and pseudocode of the module that finds an item price and exits the loop as soon as it is found

Searching an Array for a Range Match

- Programmers may want to work with ranges of values in arrays, 1 through 5 or 20 through 30
- Example: mail-order business
  - Read the customer order data; determine the discount based on the quantity ordered
- First approach
  - An array with as many elements as each possible order quantity
  - Store the appropriate discount for each possible order quantity

Searching an Array for a Range Match (continued -1)

- Drawbacks of previous approach
  - Requires a very large array; uses a lot of memory
  - Stores the same value repeatedly
  - How do you know when you have enough elements?
    - Customer can always order more
- Better approach
  - Create two parallel arrays, each with four elements
    - One array has the four discount rates
    - One array has the low end of each quantity range
  - Use a loop to make comparisons

Searching an Array for a Range Match (continued -2)

Quantity	Discount %
0–8	0
9–12	10
13–25	15
26 or more	20

Figure 6-12 Discounts on orders by quantity

```
num DISCOUNTS[76]
 = 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0.10, 0.10, 0.10, 0.10,
   0.15, 0.15, 0.15, 0.15, 0.15,
   0.15, 0.15, 0.15, 0.15, 0.15,
   0.15, 0.15, 0.15,
   0.20, 0.20, 0.20, 0.20, 0.20,
   0.20, 0.20, 0.20, 0.20, 0.20,
   0.20, 0.20, 0.20, 0.20, 0.20,
   0.20, 0.20, 0.20, 0.20, 0.20,
   0.20, 0.20, 0.20, 0.20, 0.20,
   0.20, 0.20, 0.20, 0.20, 0.20,
   0.20, 0.20, 0.20, 0.20, 0.20,
   0.20, 0.20, 0.20, 0.20, 0.20,
   0.20, 0.20, 0.20, 0.20, 0.20
```

Don't Do It
Although this array correctly lists discounts for each quantity, it is repetitious, prone to error, and difficult to use.

Figure 6-13 Usable—but inefficient—discount array

```
num DISCOUNTS[4] =   0, 0.10, 0.15, 0.20
num QUAN_LIMITS[4] = 0,    9,   13,   26
```

Figure 6-14 Parallel arrays to use for determining discount

Searching an Array for a Range Match (continued -3)

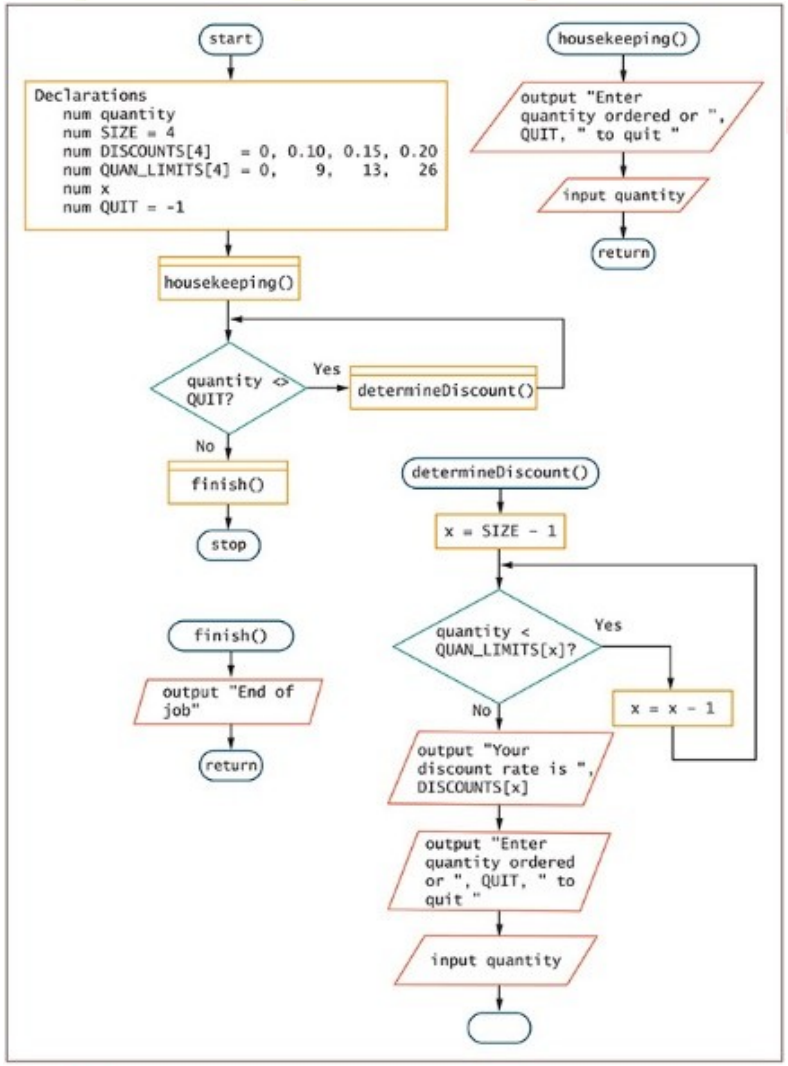


Figure 6-15 Program that determines discount rate *(continues)*

Searching an Array for a Range Match (continued 4)

(continued)

```
start
   Declarations
      num quantity
      num SIZE = 4
      num DISCOUNTS[4] =   0, 0.10, 0.15, 0.20
      num QUAN_LIMITS[4] = 0,    9,   13,   26
      num x
      num QUIT = -1
   housekeeping()
   while quantity <> QUIT
     determineDiscount()
   endwhile
   finish()
stop

housekeeping()
   output "Enter quantity ordered or ", QUIT, " to quit "
   input quantity
return

determineDiscount()
   x = SIZE - 1
   while quantity < QUAN_LIMITS[x]
      x = x - 1
   endwhile
   output "Your discount rate is ", DISCOUNTS[x]
   output "Enter quantity ordered or ", QUIT, " to quit "
   input quantity
return

finish()
   output "End of job"
return
```

Figure 6-15 Program that determines discount rate

Remaining within Array Bounds

- Every array has a finite size
  - Number of elements in the array
  - Number of bytes in the array
- Arrays are composed of elements of the same data type
- Elements of the same data type occupy the same number of bytes in memory
- The number of bytes in an array is always a multiple of the number of array elements
- Access data using a subscript containing a value that accesses memory occupied by the array

Remaining within Array Bounds (continued -1)

- **In bounds**: using a subscript that is within the acceptable range for the array
- **Out of bounds**: using a subscript that is not within the acceptable range for the array
- An invalid array subscript is a logical error
- When an invalid subscript is used:
  - Some languages stop execution and issue an error
  - Other languages access a memory location outside of the array
- The program should prevent bounds errors

Remaining within Array Bounds

(continued -2)

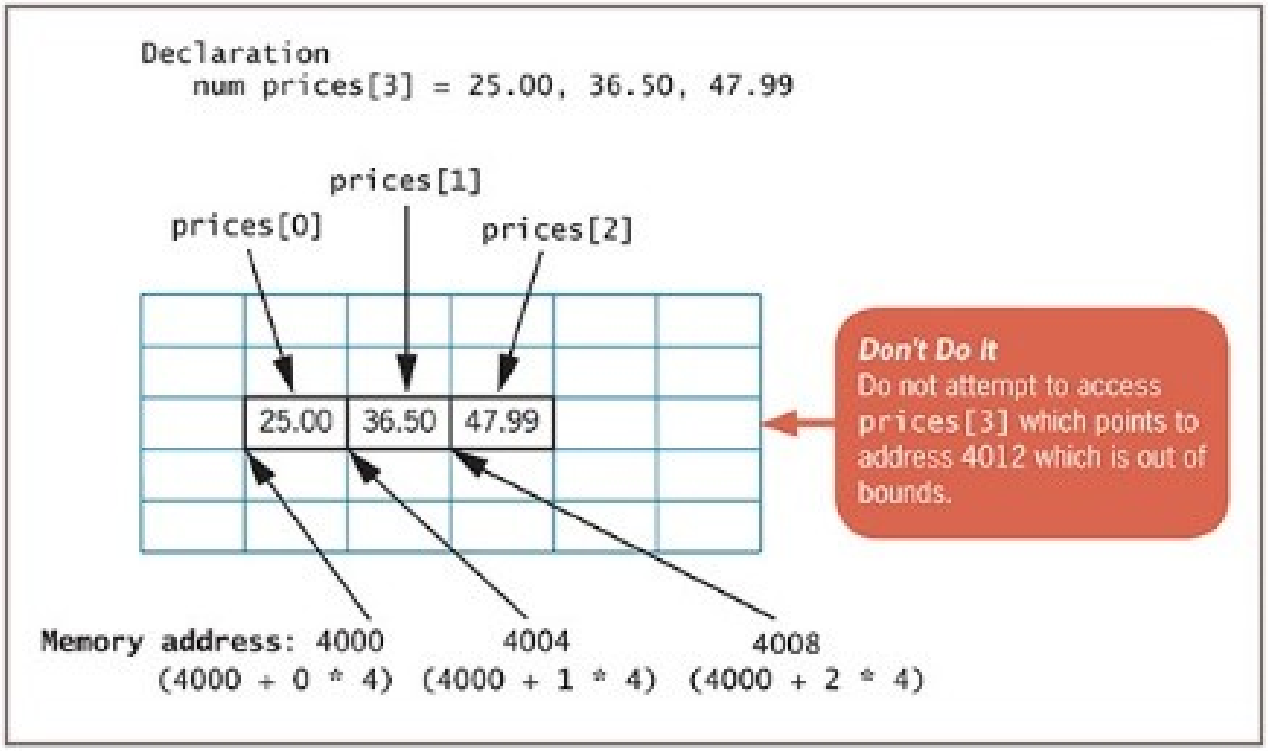


Figure 6-16 An array and its associated memory addresses

Remaining within Array Bounds

(continued -3)

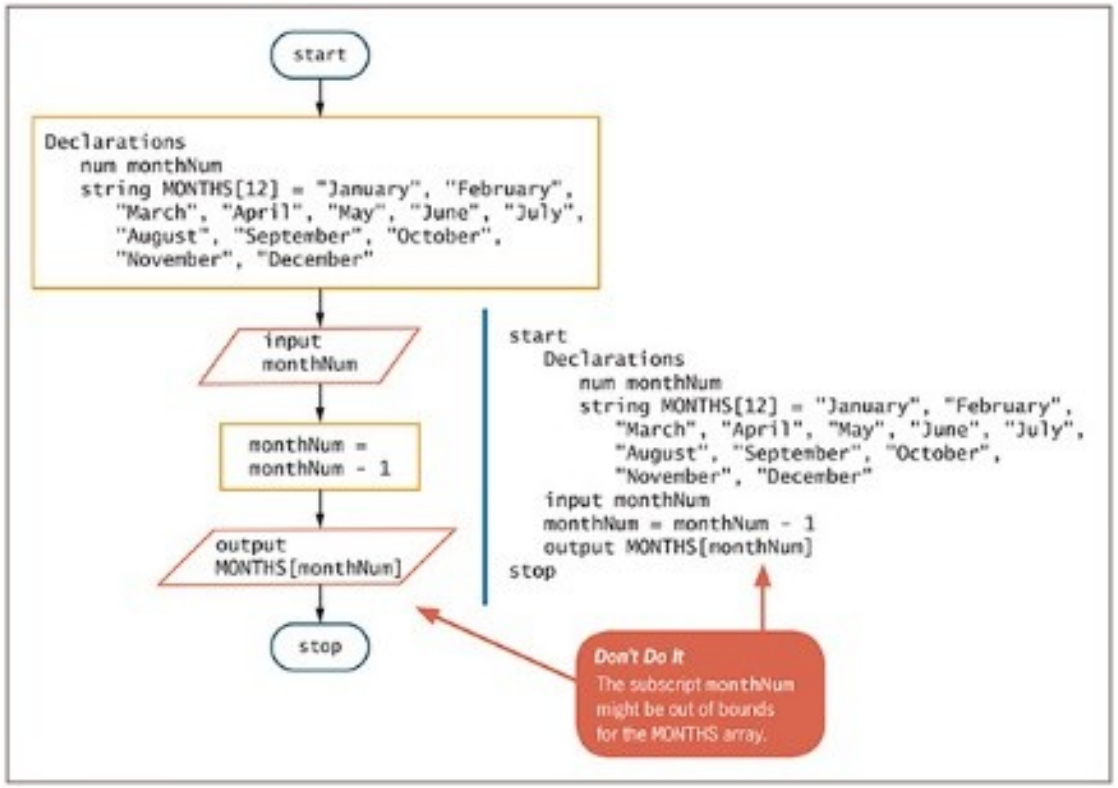


Figure 6-17 Determining the month string from a user's numeric entry

Remaining within Array Bounds (continued -4)

- To improve a program, add a test to ensure the entered subscript is valid
- If entered subscript is not valid:
  - Display an error message and end the program
  - Use a default value
  - Continuously reprompt the user for a new value until it is valid

Using a for Loop to Process an Array

- for loop: a single statement
  - Initializes the loop control variable
  - Compares it to a limit
  - Alters it
- The for loop is especially convenient when there is a need to process every element in the array
- Must stay within array bounds
- Highest usable subscript is one less than the array size

Using a for Loop to Process Arrays (continued -1)

```
start
   Declarations
      num dep
      num SIZE = 5
      string DEPTS[SIZE] = "Accounting", "Personnel",
         "Technical", "Customer Service", "Marketing"
   for dep = 0 to SIZE - 1 step 1
      output DEPTS[dep]
   endfor
stop
```

Figure 6-18 Pseudocode that uses a for loop to display an array of department names

Using a for Loop to Process Arrays (continued -2)

```
start
   Declarations
      num dep
      num SIZE = 5
      num ARRAY_LIMIT = SIZE - 1
      string DEPTS[SIZE] = "Accounting", "Personnel",
         "Technical", "Customer Service", "Marketing"
   for dep = 0 to ARRAY_LIMIT step 1
      output DEPTS[dep]
   endfor
stop
```

Figure 6-19 Pseudocode that uses a more efficient for loop to output department names

Summary

- Array: a named series or list of values in memory
  - Same data type
  - Different subscript
- Use a variable as a subscript to the array to replace multiple nested decisions
- Constants can be used to hold an array's size
- Searching through an array requires
  - Initializing a subscript
  - Using a loop to test each element
  - Setting a flag when a match is found

Summary (continued -1)

- Parallel arrays: each element in one array is associated with the element in a second array
  - Elements in each array have the same relative position
- For range comparisons, store either the low- or high-end value of each range

Summary (continued -2)

- Access data in an array
  - Use a subscript containing a value that accesses memory within the array bounds
- A subscript is out of bounds if it is not within the defined range of acceptable subscripts
- The `for` loop is a convenient tool for working with arrays when processing each element of an array from beginning to end