COURSE TECHNOLOGY
CENGAGE Learning

Programming Logic and Design
Sixth Edition

Chapter 6
Arrays

Objectives

In this chapter, you will learn about:
- Arrays and how they occupy computer memory
- Manipulating an array to replace nested decisions
- Using constants with arrays
- Searching an array
- Using parallel arrays

Objectives (continued)

- Searching an array for a range match
- Remaining within array bounds
- Using a `for` loop to process arrays

Understanding Arrays and How They Occupy Computer Memory

- **Array**
  - Series or list of variables in computer memory
  - All variables share the same name
  - Each variable has a different subscript
- **Subscript** (or **index**)
  - Position number of an item in an array
  - Subscripts are always a sequence of integers

How Arrays Occupy Computer Memory

- Each item has same name and same data type
- **Element**: an item in the array
- Array elements are contiguous in memory
- **Size of the array**: number of elements it will hold

How Arrays Occupy Computer Memory (continued)

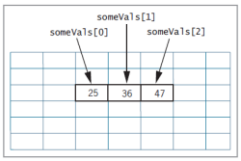

Figure 6-1 Appearance of a three-element array in computer memory

How Arrays Occupy Computer Memory (continued)

- All elements have same group name
  - Individual elements have unique subscript
  - Subscript indicates distance from first element
  - Subscripts are a sequence of integers
- Subscripts placed in parentheses or brackets following group name
  - Syntax depends on programming language

Manipulating an Array to 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 series of decisions
- Application does not scale up to more dependents

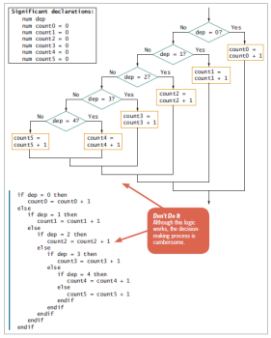

Figure 6-3 Flowchart and pseudocode of decision-making process using a series of decisions—the hard way

Manipulating an Array to Replace Nested Decisions (continued)

- Array reduces number of statements needed
- Six dependent count accumulators redefined as 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

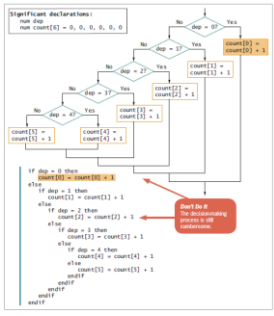

Figure 6-4 Flowchart and pseudocode of decision-making process—but still the hard way

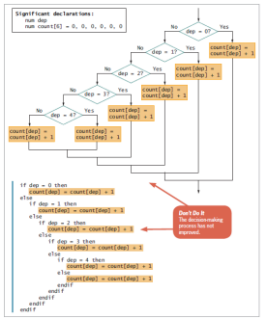

Figure 6-5 Flowchart and pseudocode of decision-making process using an array—but still a hard way

Manipulating an Array to Replace Nested Decisions (continued)

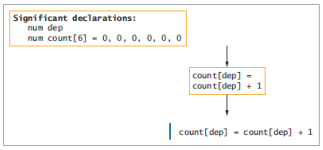

Figure 6-6 Flowchart and pseudocode of efficient decision-making process using an array

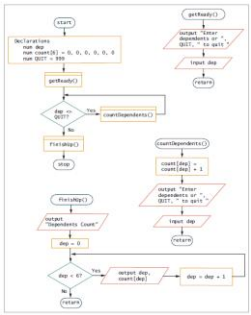

Figure 6-7 Flowchart and pseudocode for Dependents Report program

Manipulating an Array to Replace Nested Decisions (continued)

```
start
   Declarations
      num dep
      num count[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()
   count[dep] = count[dep] + 1
   output "Enter dependents or ", QUIT, " to quit "
   input dep
return

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

Figure 6-7 Flowchart and pseudocode for Dependents Report program (continued)

Using Constants with Arrays

- Use constants in several ways
  - To hold the size of an array
  - As the array values
  - As a subscript

Using a Constant as the Size of an Array

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

- Sometimes must search through an array to find a value
- 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 array for exact match

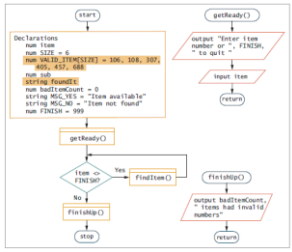

Figure 6-8 Flowchart and pseudocode for program that verifies item availability

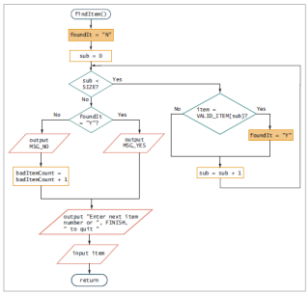

Figure 6-8 Flowchart and pseudocode for program that verifies item availability (continued)

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

Searching an Array (continued)

- **Flag**: 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

Using Parallel Arrays

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

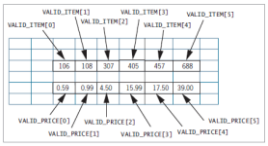

Figure 6-9 Parallel arrays in memory

Using Parallel Arrays

- Use parallel arrays
  - Two or more arrays contain related data
  - A subscript relates the arrays
    - Elements at the same position in each array are logically related

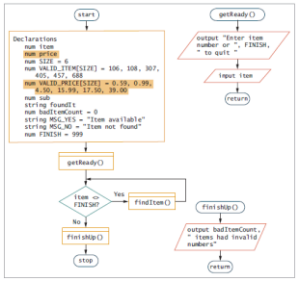

Figure 6-10 Flowchart and pseudocode of program that finds an item's price using parallel arrays

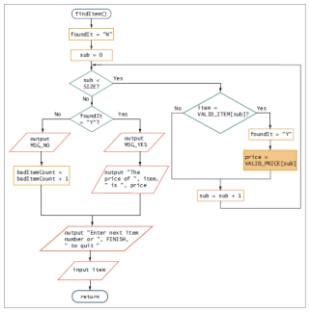

Figure 6-10 Flowchart and pseudocode of program that finds an item's price using parallel arrays (continued)


```
start
   Declarations
      num item
      num price
      num SIZE = 6
      num VALID_ITEM[SIZE] = 106, 108, 307,
         405, 457, 688
      num VALID_PRICE[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_ITEM[sub] then
         foundIt = "Y"
         price = VALID_PRICE[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's price using parallel arrays (continued)

Improving Search Efficiency

- Program should stop searching the array when a match is found
- Setting a variable to a specific value instead of letting normal processing set it
- Improves efficiency
- The larger the array, the better the improvement by doing an early exit

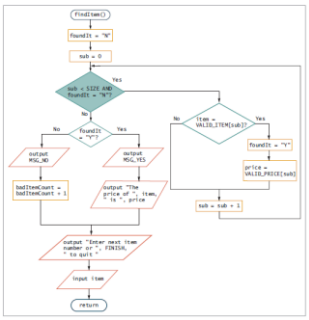

Figure 6-11 Flowchart and pseudocode of the module that finds item price, exiting the loop as soon as it is found

Improving Search Efficiency (continued)

```
findItem()
   foundIt = "N"
   sub = 0
   WHILE sub < SIZE AND foundIt = "N"
      IF item = VALID_ITEM[sub] then
         foundIt = "Y"
         price = VALID_PRICE[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 item price, exiting the loop as soon as it is found (continued)

Searching an Array for a Range Match

- Sometimes programmers want to work with ranges of values in arrays
- Example: mail-order business
  - Read customer order data; determine discount based on quantity ordered
- First approach
  - Array with as many elements as each possible order quantity
  - Store appropriate discount for each possible order quantity

Searching an Array for a Range Match (continued)

```
numeric DISCOUNT[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 is usable, it is repetitious, prone to error, and difficult to use.

Figure 6-13 Usable—but inefficient—discount array

Searching an Array for a Range Match (continued)

- Drawbacks of first approach
  - Requires very large array; uses a lot of memory
  - Stores same value repeatedly
  - How do you know you have enough elements?
    - Customer can always order more
- Better approach
  - Create four discount array elements for each discount rate
  - Parallel array with discount range
- Use loop to make comparisons

Searching an Array for a Range Match (continued)

```
num DISCOUNT[4] =   0, 0.10, 0.15, 0.20
num QUAN_LIMIT[4] = 0,    9,   13,   26
```

Figure 6-14 Parallel arrays to use for determining discount

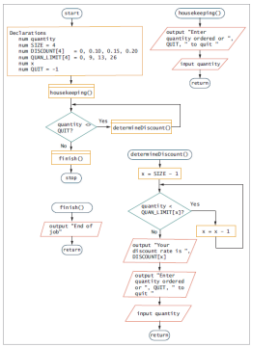

Figure 6-15 Program that determines discount rate

Remaining within Array Bounds

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

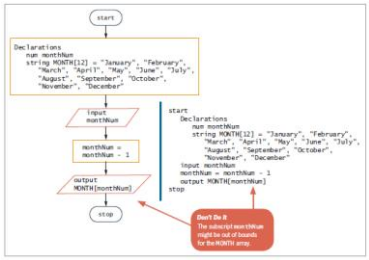

Figure 6-16 Determining the month string from user's numeric entry

Remaining within Array Bounds (continued)

- Program logic assumes every number entered by the user is valid
- When invalid subscript is used:
  - Some languages stop execution and issue an error
  - Other languages access a memory location outside of the array
- Invalid array subscript is a logical error
- **Out of bounds**: using a subscript that is not within the acceptable range for the array
- Program should prevent bounds errors

Using a `for` Loop to Process Arrays

- `for` loop: single statement
  - Initializes loop control variable
  - Compares it to a limit
  - Alters it
- `for` loop especially convenient when working with arrays
  - To process every element
- Must stay within array bounds
- Highest usable subscript is one less than array size

Using a `for` Loop to Process Arrays (continued)

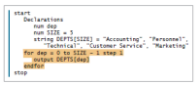

Figure 6-17 Pseudocode that uses a `for` loop to display an array of department names

Using a `for` Loop to Process Arrays (continued)

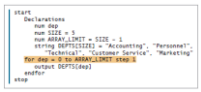

Figure 6-26 Pseudocode that uses a more efficient `for` loop to output month names

Summary

- Array: series or list of variables in memory
  - Same name and type
  - Different subscript
- Use a variable as a subscript to the array to replace multiple nested decisions
- Some array values determined during program execution
  - Other arrays have hard-coded values

Summary (continued)

- Search an array
  - Initialize the subscript
  - Test each array element value in a loop
  - Set a flag when a match is found
- Parallel arrays: each element in one array is associated with the element in second array
  - Elements have same relative position
- For range comparisons, store either the low- or high-end value of each range

Summary (continued)

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