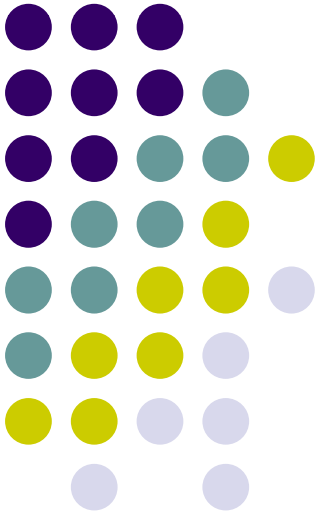
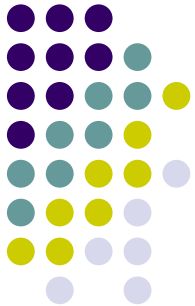
CSI2441: Applications Development

Lecture 5

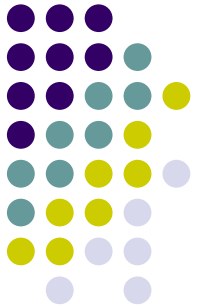
Arrays





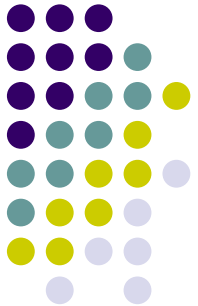
Objectives

- Understand how arrays are used
- Understand how arrays occupy computer memory
- Manipulate an array to replace nested decisions
- Declare and initialize an array
- Declare and initialize constant arrays
- Load array values from a file



Objectives (continued)

- Search an array for an exact match
- Use parallel arrays
- Force subscripts to remain within array bounds
- Improve search efficiency by using an early exit
- Search an array for a range match



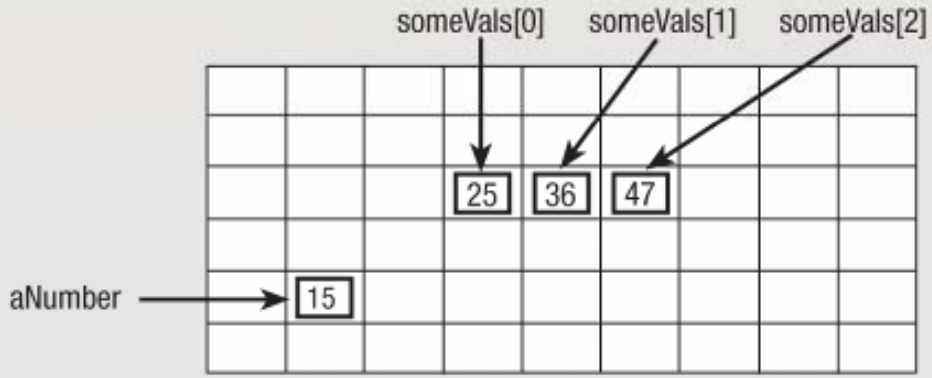
Understanding Arrays

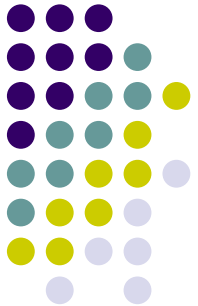
- **Array**:
  - Series or list of variables in computer memory
  - All share the same name
  - Each has a different subscript
- **Subscript** (or **index**):
  - Position number of an item in an array
  - Subscripts are always a sequence of integers

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

FIGURE 8-1: APPEARANCE OF A THREE-ELEMENT ARRAY AND A SINGLE VARIABLE IN COMPUTER MEMORY





How Arrays Occupy Computer Memory (continued)

- Subscript is placed in parentheses or square brackets following the array name (language-dependent)
- **Zero-based array**:
  - First subscript is 0
  - Highest subscript value is 1 less than the array size
- Arrays make programs more efficient and professional

Manipulating an Array to Replace Nested Decisions

- Developing the application:
  - The input file

FIGURE 8-2: FILE DESCRIPTION FOR APARTMENT REQUEST RECORDS

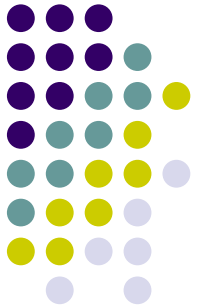
```
APARTMENT INQUIRY FILE DESCRIPTION
File name: APTREQUESTS
FIELD DESCRIPTION      DATA TYPE    COMMENTS
Day of the month       Numeric      1 - 31, day request was made
Bedrooms requested     Numeric      0, 1, 2 or 3 for studio apartment
                                    or number of bedrooms
```

Manipulating an Array to Replace Nested Decisions (continued)

- Developing the application:
  - The desired output

FIGURE 8-3: TYPICAL APARTMENT REQUEST REPORT

```
Apartment Request Report

Bedrooms    Inquiries

   0           91
   1           44
   2           67
   3          102
```

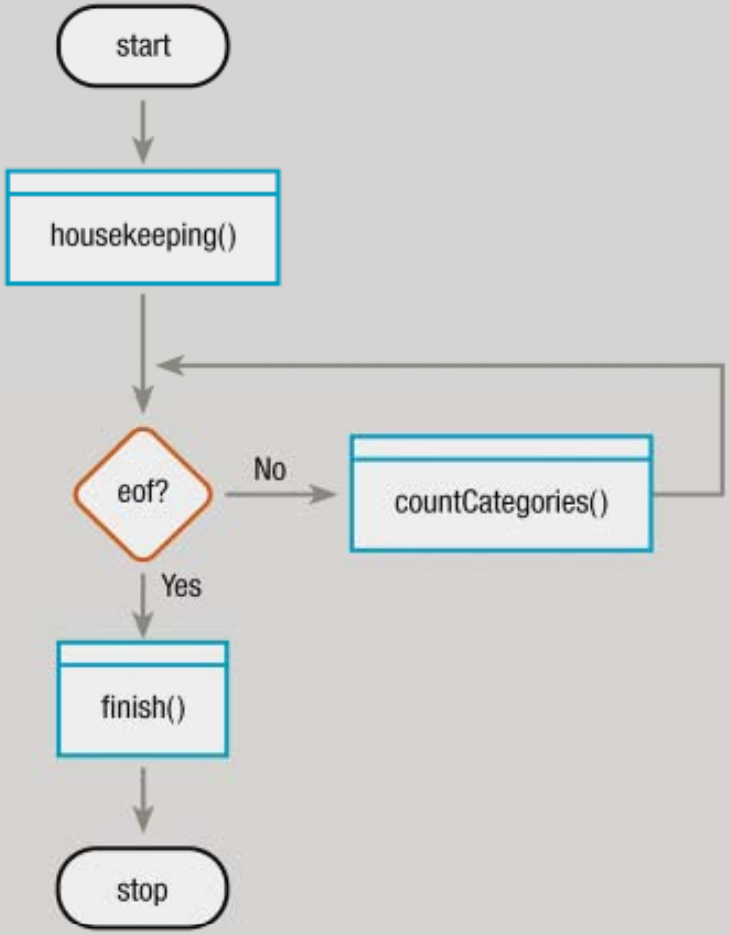


Manipulating an Array to Replace Nested Decisions (continued)

- Report could be created in several ways:
  - If data is sorted, a control break program could be created
  - If unsorted, multiple counters could be used to accumulate the grouping totals
  - Report could be created using arrays to accumulate the grouping totals

Manipulating an Array to Replace Nested Decisions (continued)

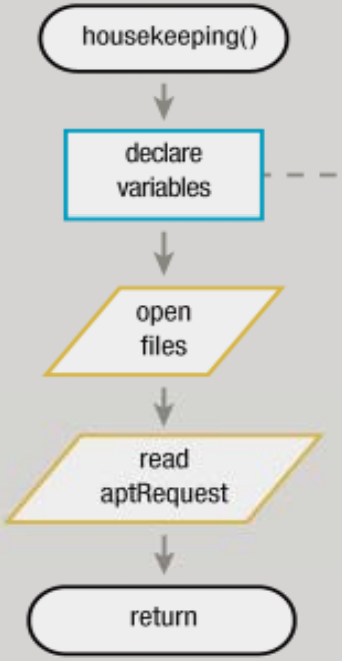
FIGURE 8-4: FLOWCHART AND PSEUDOCODE FOR MAINLINE LOGIC OF APARTMENT REQUEST REPORT PROGRAM




```
start
     perform housekeeping()
     while not eof
          perform countCategories()
     endwhile
     perform finish()
stop
```

Manipulating an Array to Replace Nested Decisions (continued)

FIGURE 8-5: THE `housekeeping()` MODULE FOR THE APARTMENT REQUEST PROGRAM WITHOUT AN ARRAY

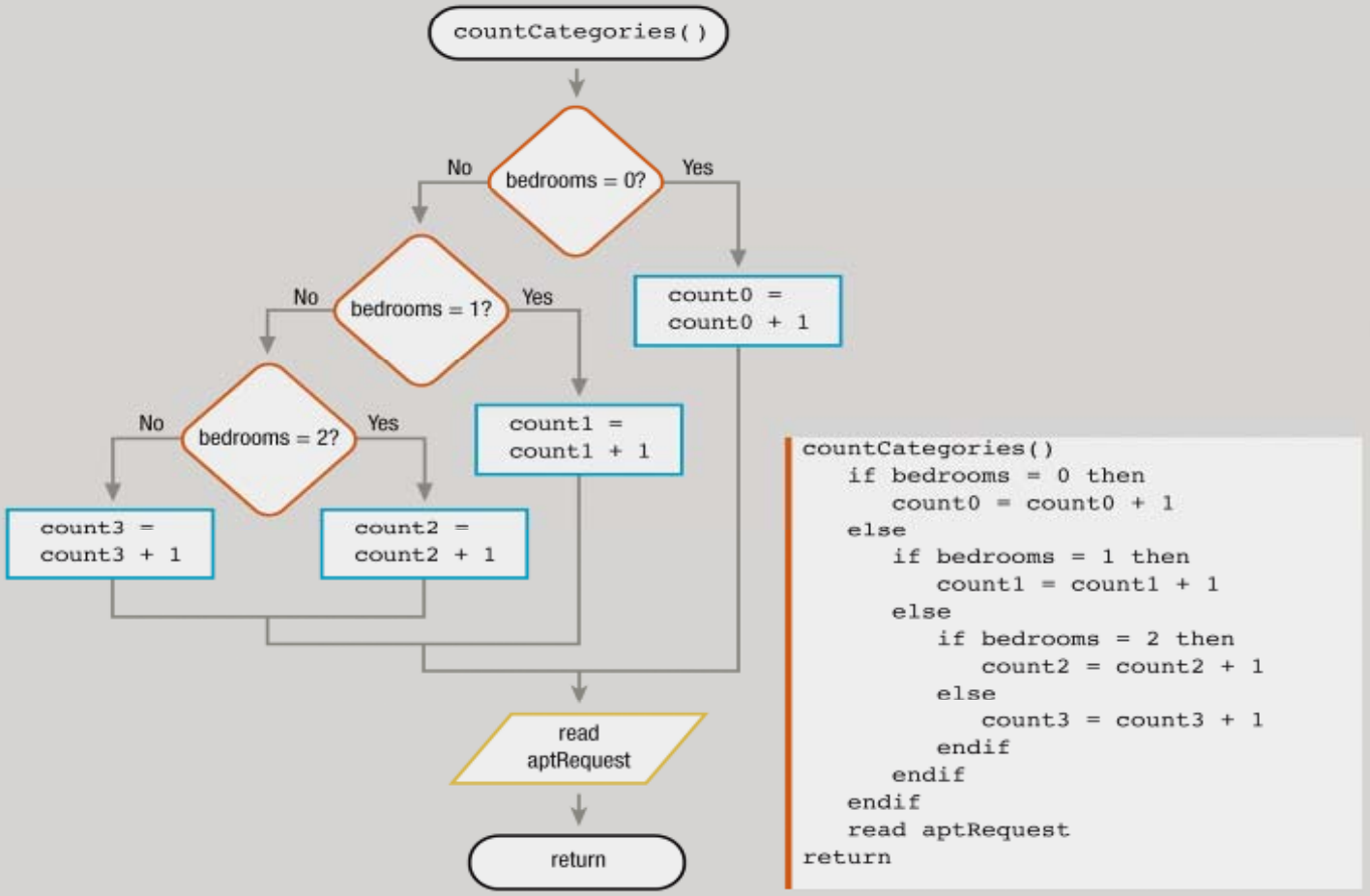



```
aptRequest
   num day
   num bedrooms
char head1 = "Apartment Request Report"
char head2 = "Bedrooms   Inquiries"
num count0 = 0
num count1 = 0
num count2 = 0
num count3 = 0
```

```
housekeeping()
   declare variables
     aptRequest
       num day
       num bedrooms
     char head1 = "Apartment Request Report"
     char head2 = "Bedrooms   Inquiries"
     num count0 = 0
     num count1 = 0
     num count2 = 0
     num count3 = 0
   open files
   read aptRequest
return
```

Manipulating an Array to Replace Nested Decisions (continued)

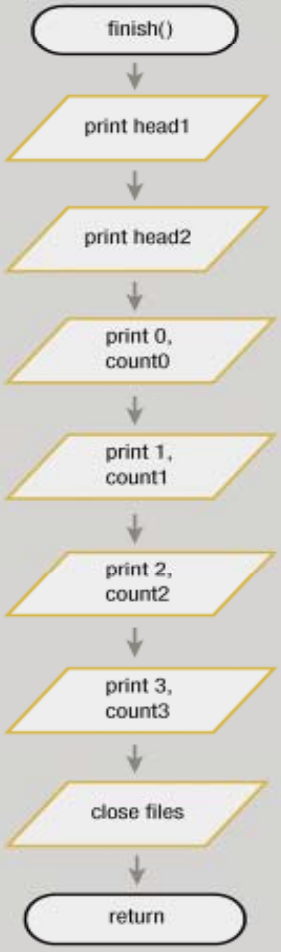
FIGURE 8-6: THE `countCategories()` MODULE FOR THE APARTMENT REQUEST PROGRAM WITHOUT AN ARRAY



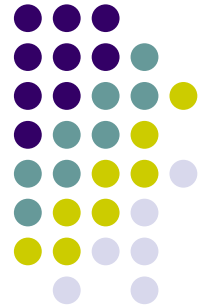

```
countCategories()
   if bedrooms = 0 then
      count0 = count0 + 1
   else
      if bedrooms = 1 then
         count1 = count1 + 1
      else
         if bedrooms = 2 then
            count2 = count2 + 1
         else
            count3 = count3 + 1
         endif
      endif
   endif
   read aptRequest
return
```

Manipulating an Array to Replace Nested Decisions (continued)

FIGURE 8-7: THE `finish()` MODULE FOR THE APARTMENT REQUEST PROGRAM WITHOUT AN ARRAY




```
finish()
   print head1
   print head2
   print 0, count0
   print 1, count1
   print 2, count2
   print 3, count3
   close files
return
```



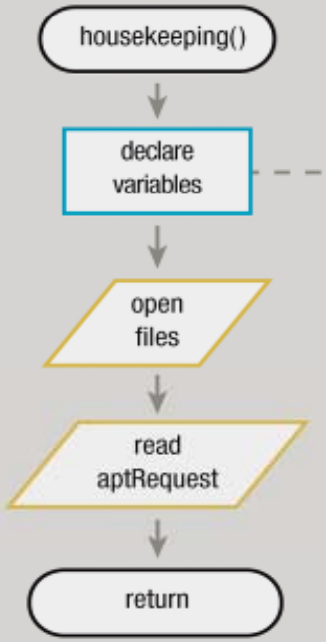
Manipulating an Array to Replace Nested Decisions (continued)

- Using an array
  - Simplifies the program logic
  - Reduces the number of statements needed
- Each element is referenced by name and subscript:

```
count[0]
```

Manipulating an Array to Replace Nested Decisions (continued)

FIGURE 8-8: MODIFIED `housekeeping()` MODULE FOR APARTMENT REQUEST PROGRAM THAT DECLARES AN ARRAY TO COUNT REQUESTS

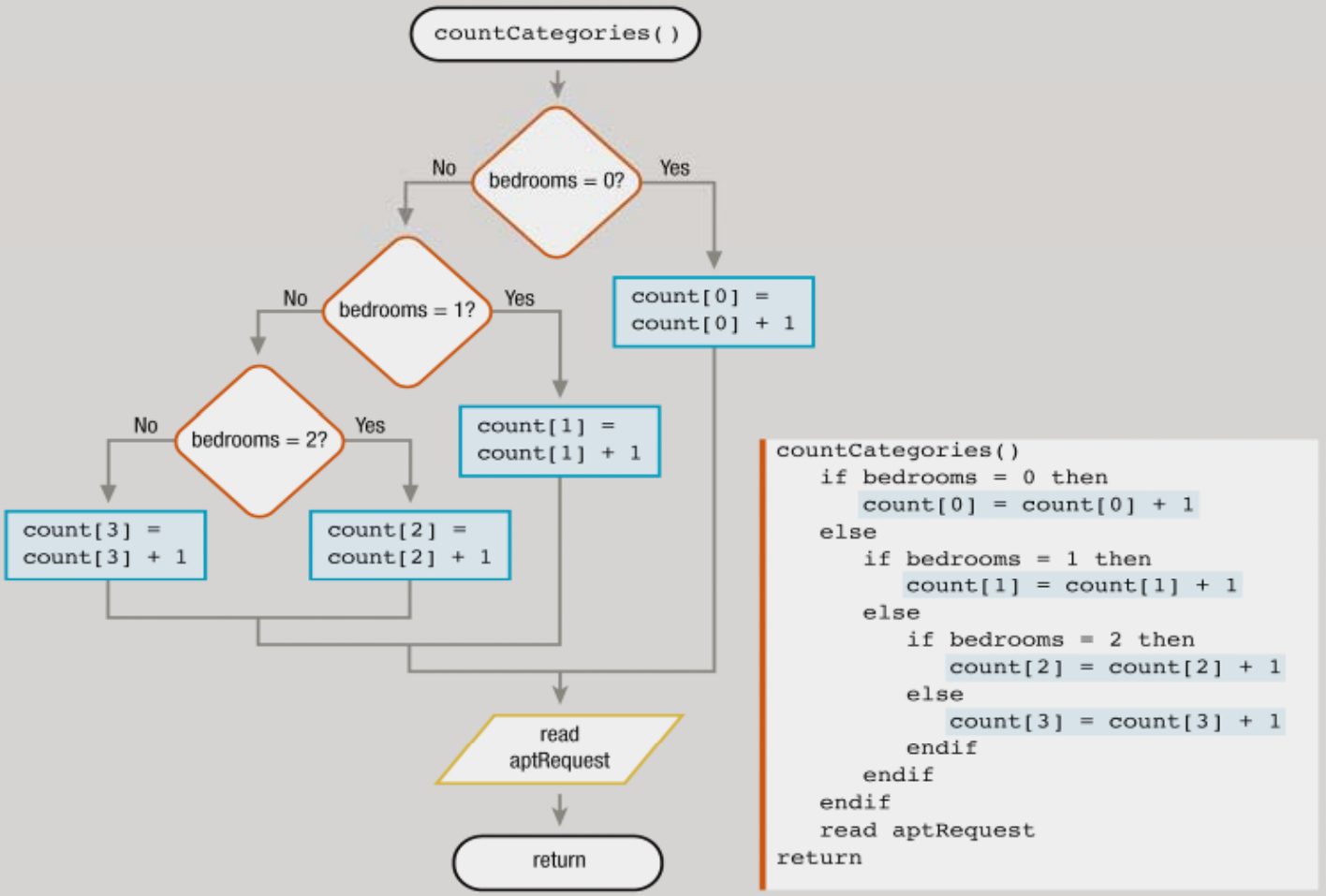



```
aptRequest
   num day
   num bedrooms
char head1 = "Apartment Request Report"
char head2 = "Bedrooms   Inquiries"
num count[0] = 0
num count[1] = 0
num count[2] = 0
num count[3] = 0
```

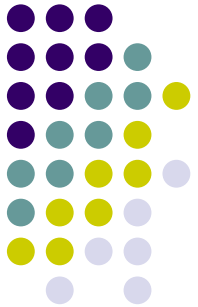
```
housekeeping()
   declare variables
     aptRequest
       num day
       num bedrooms
     char head1 = "Apartment Request Report"
     char head2 = "Bedrooms   Inquiries"
     num count[0] = 0
     num count[1] = 0
     num count[2] = 0
     num count[3] = 0
   open files
   read aptRequest
return
```

Manipulating an Array to Replace Nested Decisions (continued)

FIGURE 8-9: MODIFIED `countCategories()` MODULE THAT USES `count` ARRAY




```
countCategories()
   if bedrooms = 0 then
      count[0] = count[0] + 1
   else
      if bedrooms = 1 then
         count[1] = count[1] + 1
      else
         if bedrooms = 2 then
            count[2] = count[2] + 1
         else
            count[3] = count[3] + 1
         endif
      endif
   endif
   read aptRequest
return
```

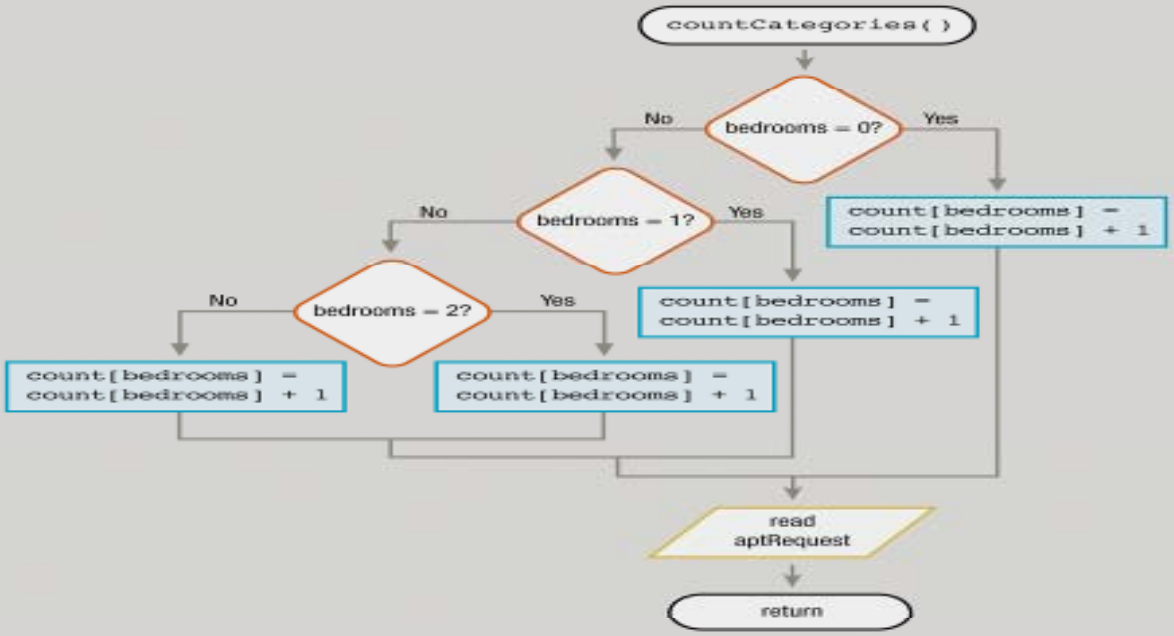


Manipulating an Array to Replace Nested Decisions (continued)

- Benefit: can use a variable as the subscript
- Look for relationships between the array subscript value and the data or processing required
- Application:
  - Number of bedrooms matches the array subscript

Manipulating an Array to Replace Nested Decisions (continued)

FIGURE 8-10: MODIFIED `countCategories()` MODULE USING THE VARIABLE `bedrooms` AS A SUBSCRIPT TO THE `count` ARRAY

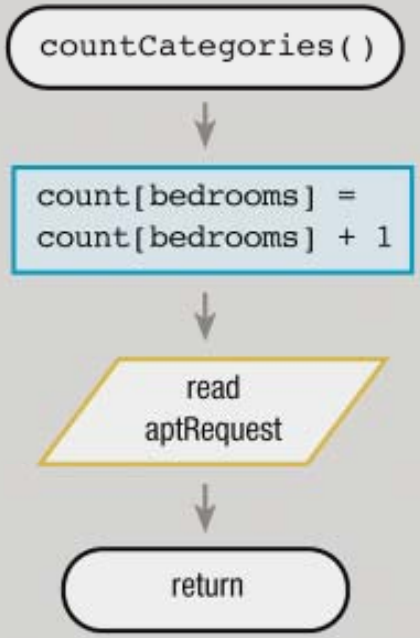



```
countCategories()
    if bedrooms = 0 then
        count[bedrooms] = count[bedrooms] + 1
    else
        if bedrooms = 1 then
            count[bedrooms] = count[bedrooms] + 1
        else
            if bedrooms = 2 then
                count[bedrooms] = count[bedrooms] + 1
            else
                count[bedrooms] = count[bedrooms] + 1
            endif
        endif
    endif
    read aptRequest
return
```

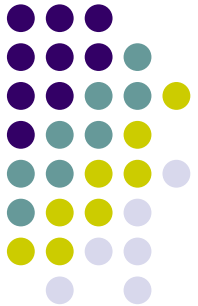
Manipulating an Array to Replace Nested Decisions (continued)

- If the action is the same for all values of the subscript, there is no need to test the value of the subscript

FIGURE 8-11: MODIFIED `countCategories()` MODULE, ELIMINATING NESTED DECISIONS




```
countCategories()
   count[bedrooms] = count[bedrooms] + 1
   read aptRequest
return
```

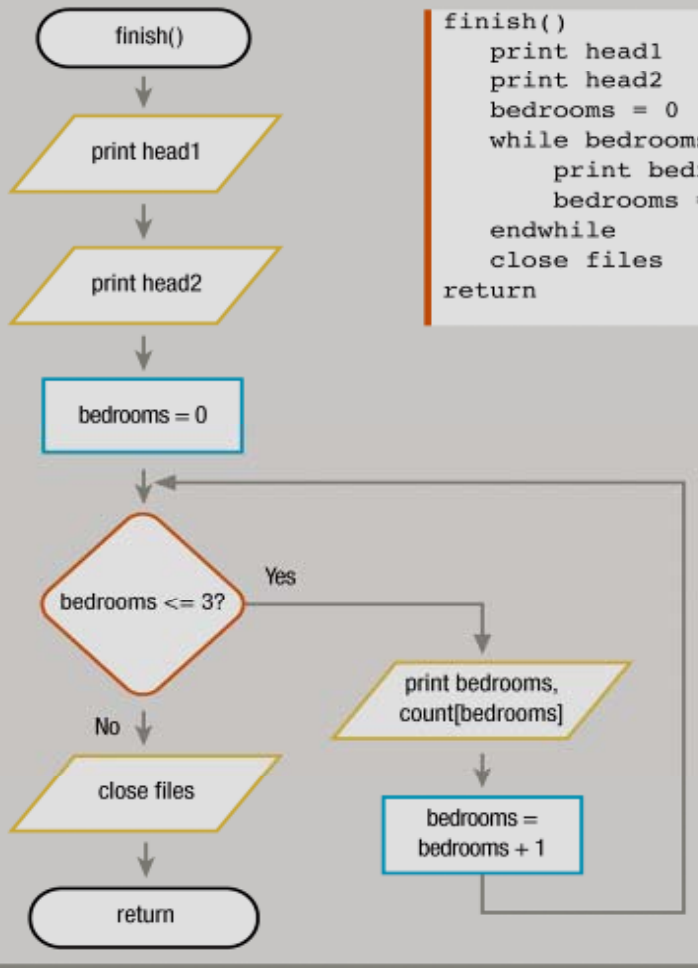


Manipulating an Array to Replace Nested Decisions (continued)

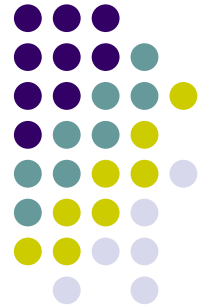
- Use the subscript to control an array printing loop
- Any variable can be used as a subscript for controlling a loop using the array, as long as it is:
  - Numeric with no decimal places
  - Initialized to some value (e.g., 0)
  - Incremented each time the logic passes through the loop (e.g., by 1)

Manipulating an Array to Replace Nested Decisions (continued)

FIGURE 8-12: MODIFIED `finish()` MODULE THAT USES AN ARRAY




```
finish()
   print head1
   print head2
   bedrooms = 0
   while bedrooms <= 3
       print bedrooms, count[bedrooms]
       bedrooms = bedrooms + 1
   endwhile
   close files
return
```

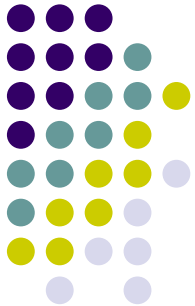


Array Declaration and Initialization

- Array elements do not have to be declared individually
- Put the array size in square brackets in the declaration

FIGURE 8-13: DECLARING A 30-ELEMENT ARRAY NAMED `count` IN SEVERAL COMMON LANGUAGES

Declaration	Programming Language
`DIM COUNT(30)`	BASIC, Visual Basic
`int count[30];`	C#, C++
`int[] count = new int[30];`	Java
`COUNT OCCURS 30 TIMES PICTURE 9999.`	COBOL
`array count [1..30] of integer;`	Pascal

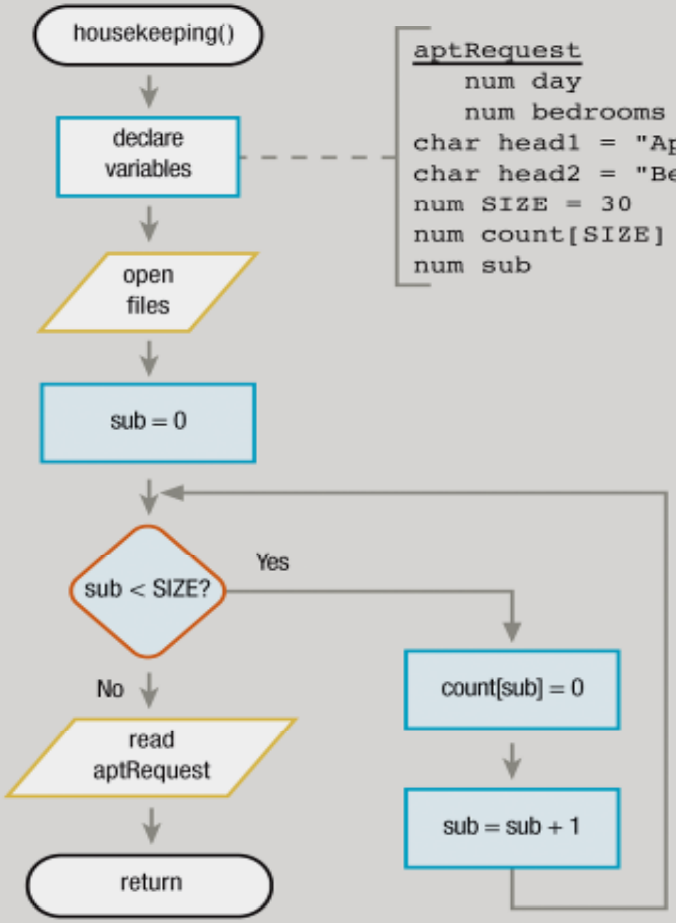


Array Declaration and Initialization (continued)

- Array elements should be initialized
- **Initialization loop**: loop structure that provides initial values to an array
- Use the loop control variable as the array subscript

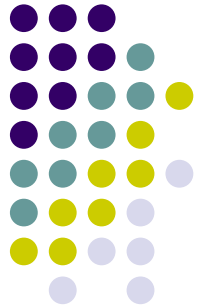
Array Declaration and Initialization (continued)

FIGURE 8-14: A `housekeeping()` MODULE DEMONSTRATING ONE METHOD OF INITIALIZING ARRAY ELEMENTS




```
aptRequest
   num day
   num bedrooms
char head1 = "Apartment Request Report"
char head2 = "Bedrooms   Inquiries"
num SIZE = 30
num count[SIZE]
num sub
```

```
housekeeping()
    declare variables
      aptRequest
        num day
        num bedrooms
      char head1 = "Apartment Request Report"
      char head2 = "Bedrooms   Inquiries"
      num SIZE = 30
      num count[SIZE]
      num sub
    open files
    sub = 0
    while sub < SIZE
        count[sub] = 0
        sub = sub + 1
    endwhile
    read aptRequest
return
```

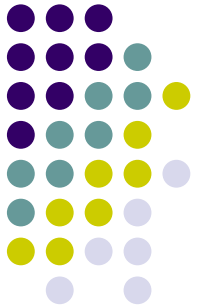


Declaring and Initializing Constant Arrays

- Developing the application:
  - The input file
  - Floor and apartment letter will not change

FIGURE 8-15: TENANT FILE DESCRIPTION

```
TENANT FILE DESCRIPTION
File name: TENANTS
FIELD DESCRIPTION       DATA TYPE    COMMENTS
Tenant name             Character    Full name, first and last
Floor number            Numeric      0 through 4 - 0 is basement
Apartment letter        Character    Single letter - A through F
```

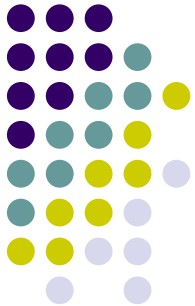


Declaring and Initializing Constant Arrays (continued)

- Developing the application:
  - Need to print rent bills based on the floor of the building

FIGURE 8-16: RENTS BY FLOOR

```
Floor                Rent in $
0 (the basement)        350
1                       400
2                       475
3                       600
4 (the penthouse)      1000
```

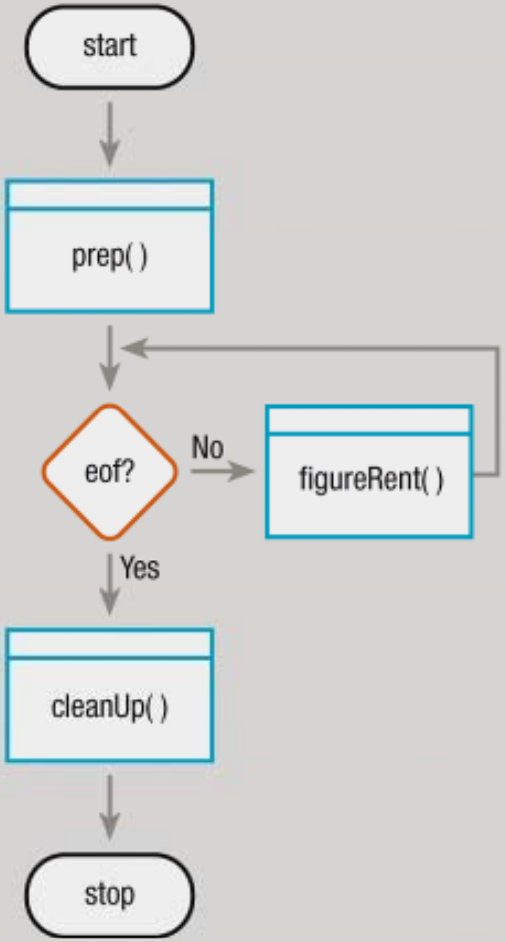


Declaring and Initializing Constant Arrays (continued)

- Use an array to hold the floor and corresponding rent figures
- These array values will be constant
- **Hard coded values**: values assigned directly in program statements

Declaring and Initializing Constant Arrays (continued)

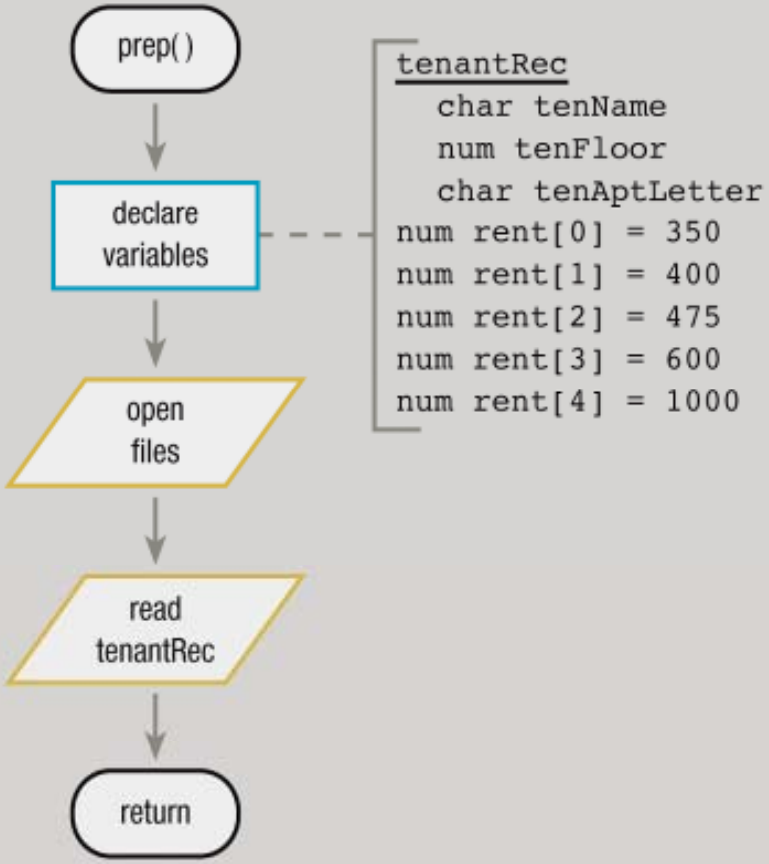
FIGURE 8-17: FLOWCHART AND PSEUDOCODE FOR MAINLINE LOGIC OF RENT PROGRAM




```
start
   perform prep()
   while not eof
      perform figureRent()
   endwhile
   perform cleanUp()
stop
```

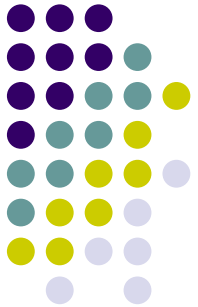
Declaring and Initializing Constant Arrays (continued)

FIGURE 8-18: FLOWCHART AND PSEUDOCODE FOR `prep()` MODULE OF RENT PROGRAM




```
tenantRec
   char tenName
   num tenFloor
   char tenAptLetter
num rent[0] = 350
num rent[1] = 400
num rent[2] = 475
num rent[3] = 600
num rent[4] = 1000
```

```
prep()
      declare variables
      open files
      read tenantRec
return
```



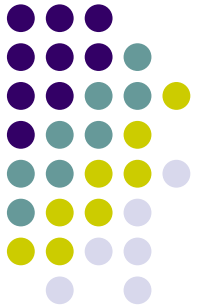
Declaring and Initializing Constant Arrays (continued)

- Developing the application:
  - Choose a variable to be subscript: determine which variable the correct selection depends on
  - Rent depends on the floor, so select **`tenFloor`** variable as the subscript in the **`figureRent()`** module

Declaring and Initializing Constant Arrays (continued)

FIGURE 8-19: FLOWCHART AND PSEUDOCODE FOR THE `figureRent()` MODULE OF THE RENT PROGRAM


```
figureRent()
     print "Dear ", tenName, ", Here is your monthly rent bill"
     print rent[tenFloor]
     read tenantRec
return
```

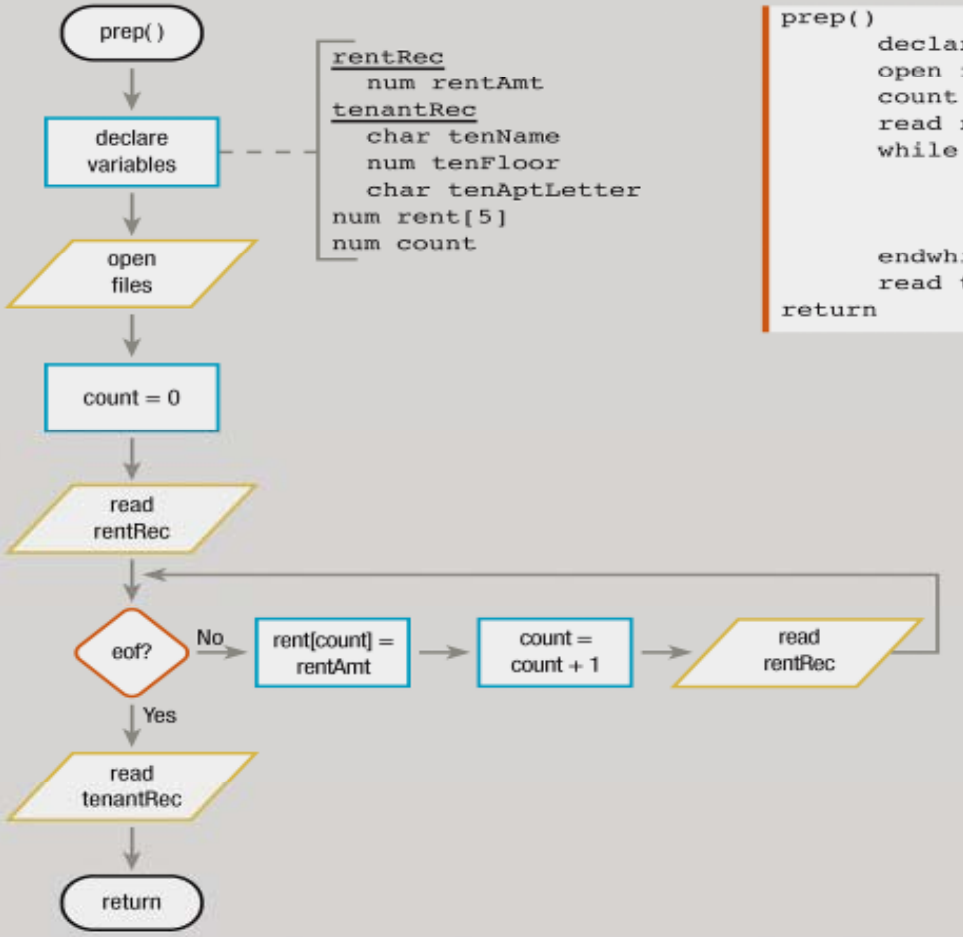


Loading an Array From a File

- Developing the application:
  - If there are many rent values or they change frequently, initialize the **`rent`** array from a file
  - Allows values to change without changing the program
  - In this file, the rent records are stored in order by floor

Loading an Array From a File (continued)

FIGURE 8-21: FLOWCHART AND PSEUDOCODE FOR `prep()` MODULE THAT READS RENT VALUES FROM AN INPUT FILE




```
prep()
        declare variables
        open files
        count = 0
        read rentRec
        while not eof
                rent[count] = rentAmt
                count = count + 1
                read rentRec
        endwhile
        read tenantRec
return
```

Searching for an Exact Match in an Array

- Developing the application:
  - The input file: customer orders
  - Item numbers are not consecutive, with large gaps

FIGURE 8-22: MAIL-ORDER CUSTOMER FILE DESCRIPTION

```
MAIL-ORDER CUSTOMER FILE DESCRIPTION
File name: CUSTREC
FIELD DESCRIPTION      DATA TYPE    COMMENTS
Customer name          Character
Address                Character
Item number            Numeric      A 3-digit number
Quantity               Numeric      A value from 1 through 99
```

Searching for an Exact Match in an Array (continued)

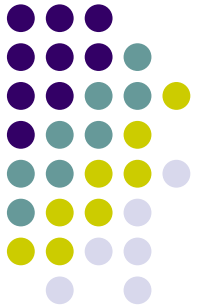
- Developing the application:
  - Create an array with valid item numbers
  - Use a loop to search the array for valid item values

FIGURE 8-23: AVAILABLE ITEMS IN MAIL-ORDER COMPANY

```
ITEM NUMBER
106
108
307
405
457
688
```

FIGURE 8-24: ARRAY OF VALID ITEM NUMBERS

```
num validItem[0] = 106
num validItem[1] = 108
num validItem[2] = 307
num validItem[3] = 405
num validItem[4] = 457
num validItem[5] = 688
```

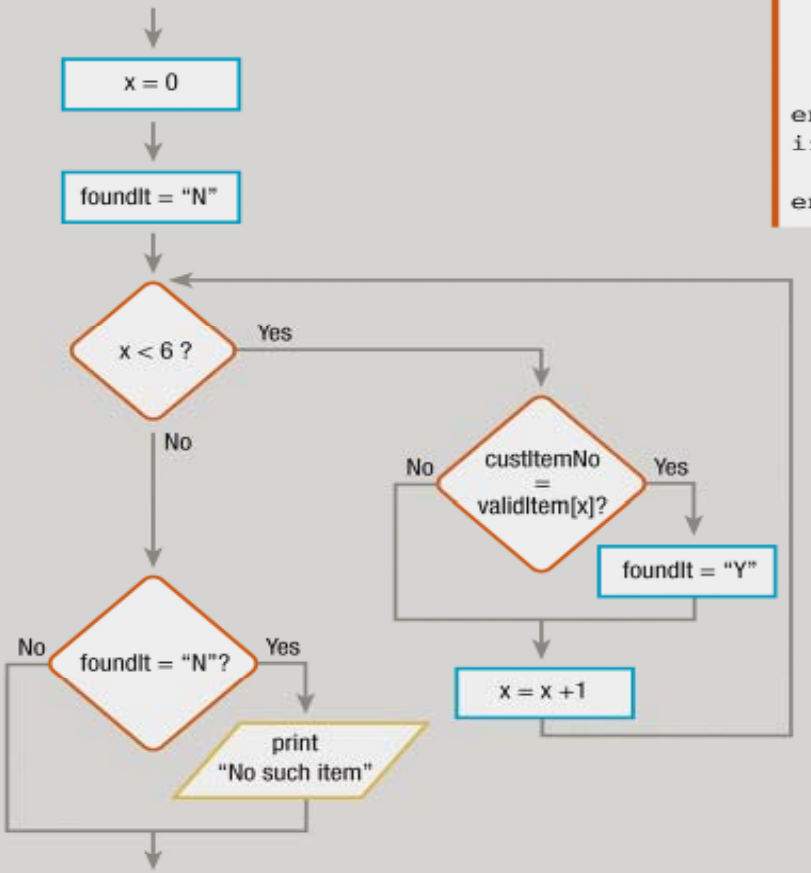


Searching for an Exact Match in an Array (continued)

- **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

Searching for an Exact Match in an Array (continued)

FIGURE 8-25: FLOWCHART AND PSEUDOCODE SEGMENTS FOR FINDING AN EXACT MATCH TO A CUSTOMER ITEM NUMBER




```
x = 0
foundIt = "N"
while x < 6
      if custItemNo = validItem[x] then
            foundIt = "Y"
      endif
      x = x + 1
endwhile
if foundIt = "N" then
     print "No such item"
endif
```

Using Parallel Arrays

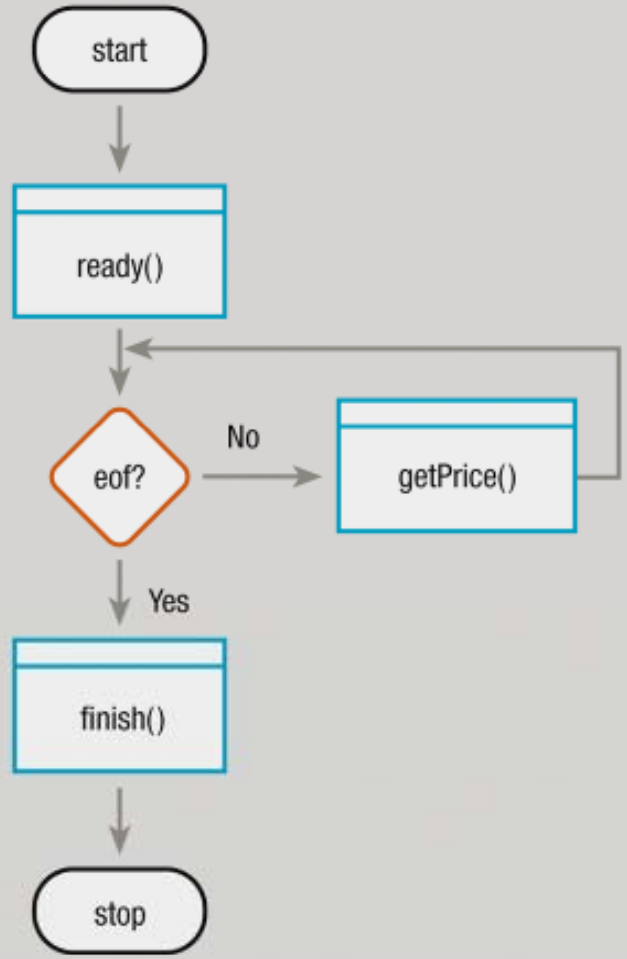
- **Parallel arrays**: two arrays in which each element of one array is associated with the element in the same relative position in the other array
- Developing the application:
  - Use one array for item numbers
  - Use a second array for the item prices
  - When the correct item is found, the price is in the same position in the other array

FIGURE 8-26: AVAILABLE ITEMS WITH PRICES FOR MAIL-ORDER COMPANY

ITEM NUMBER	ITEM PRICE
106	0.59
108	0.99
307	4.50
405	15.99
457	17.50
688	39.00

Using Parallel Arrays (continued)

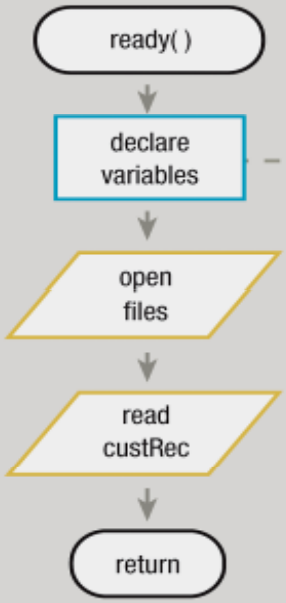
FIGURE 8-27: MAINLINE LOGIC FOR THE PRICE PROGRAM




```
start
      perform ready()
      while not eof
            perform getPrice()
      endwhile
      perform finish()
stop
```

Using Parallel Arrays (continued)

FIGURE 8-28: THE `ready()` MODULE FOR THE PRICE PROGRAM

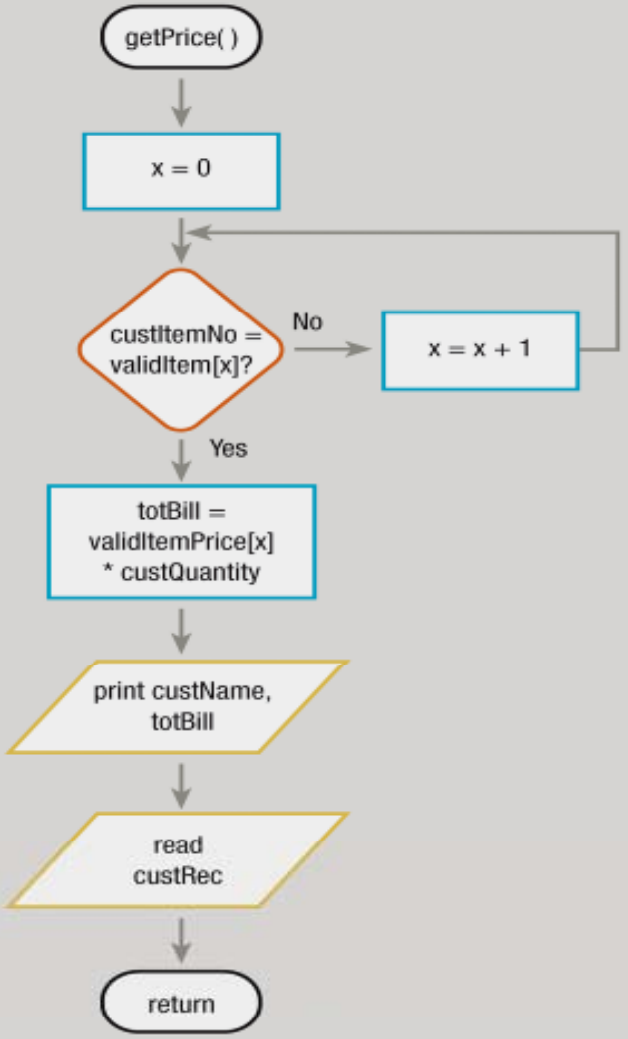



```
custRec
   char custName
   char custAddress
   num custItemNo
   num custQuantity

num validItem[0] = 106
num validItem[1] = 108
num validItem[2] = 307
num validItem[3] = 405
num validItem[4] = 457
num validItem[5] = 688

num validItemPrice[0] = 0.59
num validItemPrice[1] = 0.99
num validItemPrice[2] = 4.50
num validItemPrice[3] = 15.99
num validItemPrice[4] = 17.50
num validItemPrice[5] = 39.00

num totBill
num x
char foundIt
```

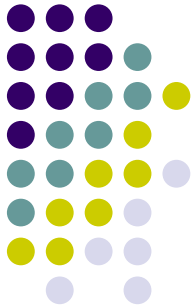
```
ready()
     declare variables
     open files
     read custRec
return
```

Using Parallel Arrays (continued)

FIGURE 8-29: THE `getPrice()` MODULE FOR THE PRICE PROGRAM




```
getPrice()
      x = 0
      while custItemNo not equal to validItem[x]
            x = x + 1
      endwhile
      totBill = validItemPrice[x] * custQuantity
      print custName, totBill
      read custRec
return
```



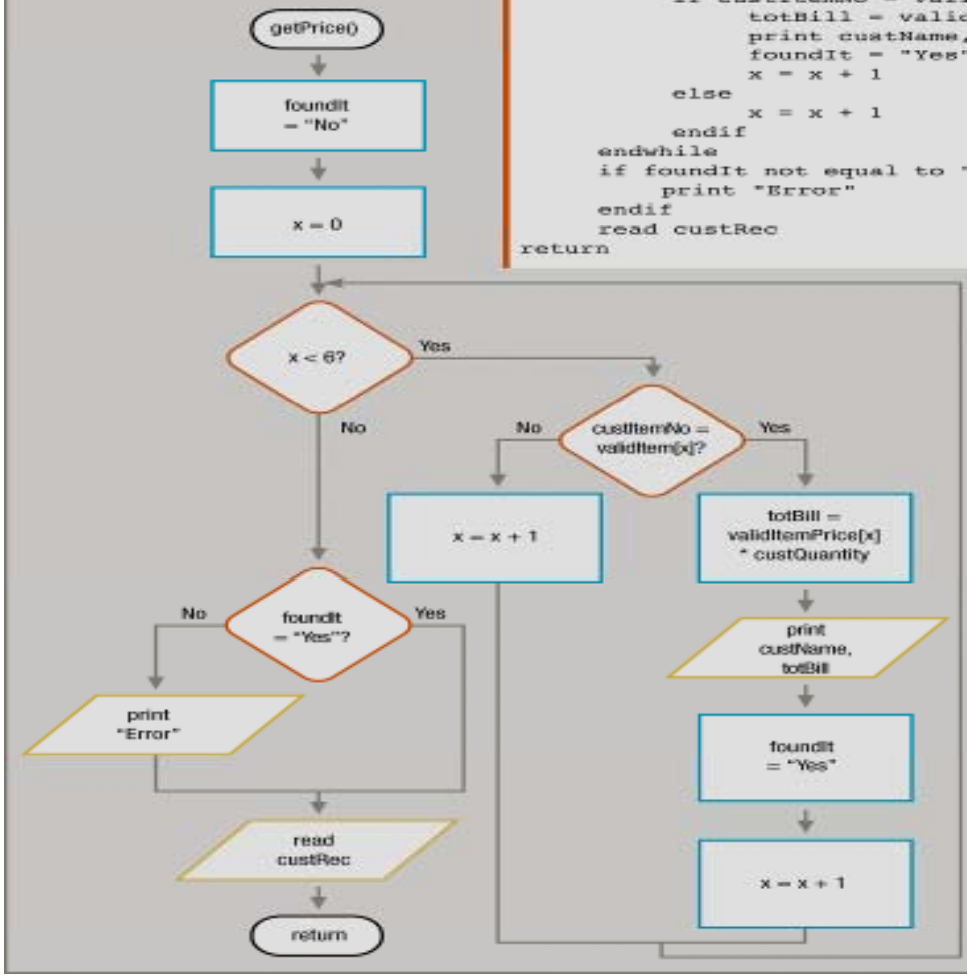
Remaining Within Array Bounds

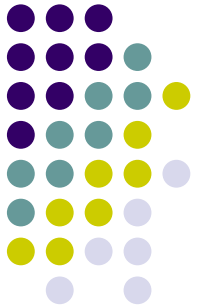
- **Out of bounds**: using a subscript that is not within the acceptable range for the array
- Some languages give an error; others just search through memory beyond the end of the array
- Program should prevent bounds errors
- Developing the application:
  - Subscript should not go beyond 5 because there are only 6 array elements (with subscripts 0 to 5)

Remaining Within Array Bounds (continued)

FIGURE 8-30: THE `getPrice()` MODULE USING THE `foundIt` FLAG

```
getPrice()
      foundIt = "No"
      x = 0
      while x < 6
            if custItemNo = validItem[x] then
                  totBill = validItemPrice[x] * custQuantity
                  print custName, totBill
                  foundIt = "Yes"
                  x = x + 1
            else
                  x = x + 1
            endif
      endwhile
      if foundIt not equal to "Yes" then
            print "Error"
      endif
      read custRec
return
```



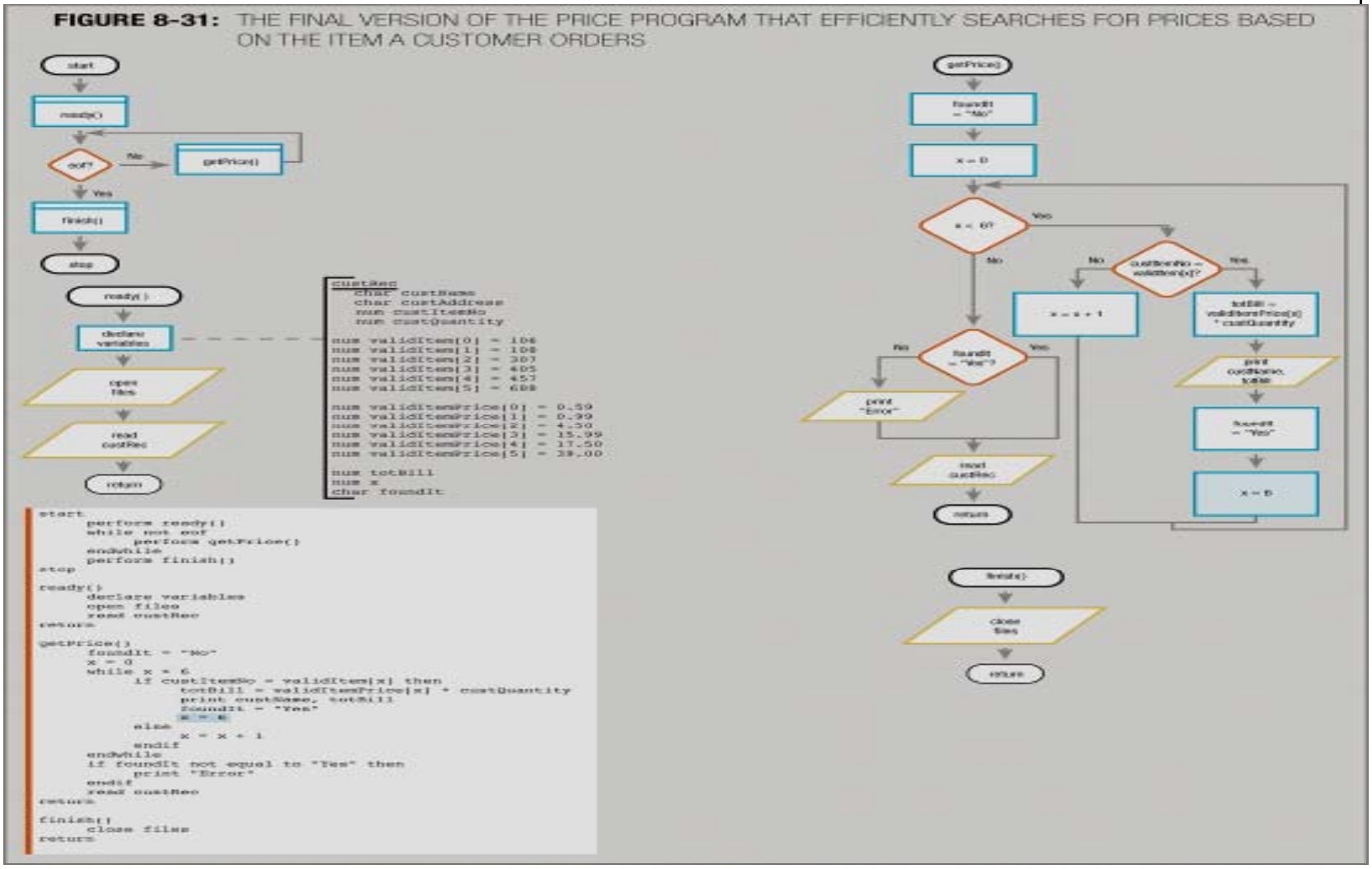


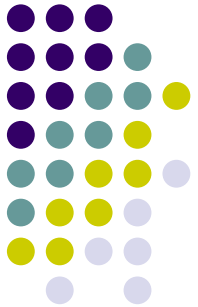
Improving Search Efficiency Using an Early Exit

- To improve efficiency, program should stop searching the array when a match is found
- **Forcing a variable:** setting a variable to a specific value instead of letting normal processing set it
- **Early exit**: leaving a loop as soon as a match is found
- The larger the array, the better the improvement by doing an early exit

Improving Search Efficiency Using an Early Exit (continued)

FIGURE 8-31: THE FINAL VERSION OF THE PRICE PROGRAM THAT EFFICIENTLY SEARCHES FOR PRICES BASED ON THE ITEM A CUSTOMER ORDERS



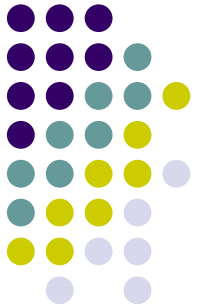


Searching an Array for a Range Match

- **Range of values**: any set of contiguous values
- Developing the application:
  - The input file

FIGURE 8-32: MAIL-ORDER CUSTOMER FILE DESCRIPTION

```
MAIL-ORDER CUSTOMER FILE DESCRIPTION
File name: CUSTREC
FIELD DESCRIPTION      DATA TYPE    COMMENTS
Customer name          Character
Address                Character
Item number            Numeric      A 3-digit number
Quantity               Numeric      A value from 1 through 99
```

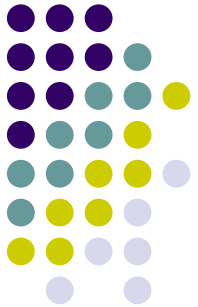


Searching an Array for a Range Match (continued)

- Developing the application:
  - Need to apply a discount for ranges of quantity ordered

FIGURE 8-33: DISCOUNTS ON ORDERS BY QUANTITY

Number of items ordered	Discount %
1-9	0
10-24	10
25-48	15
49 or more	25

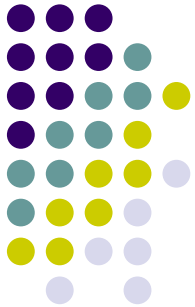


Searching an Array for a Range Match (continued)

- Developing the application: one approach

FIGURE 8-34: USABLE—BUT INEFFICIENT—DISCOUNT ARRAY

```
num discount[0] = 0
num discount[1] = 0
num discount[2] = 0
.
.
num discount[9] = 0
num discount[10] = 10
.
.
num discount[48] = 15
num discount[49] = 25
num discount[50] = 25
.
.
```

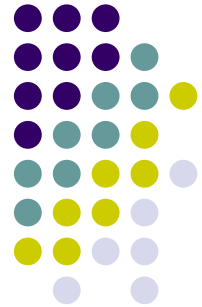


Searching an Array for a Range Match (continued)

- Drawbacks of this approach:
  - Very large array, uses a lot of memory
  - Must store the same values repeatedly
  - What upper limit should be used for array elements?
- A better approach

FIGURE 8-35: SUPERIOR DISCOUNT ARRAY

```
num discount[0] = 0
num discount[1] = 10
num discount[2] = 15
num discount[3] = 25
```



Searching an Array for a Range Match (continued)

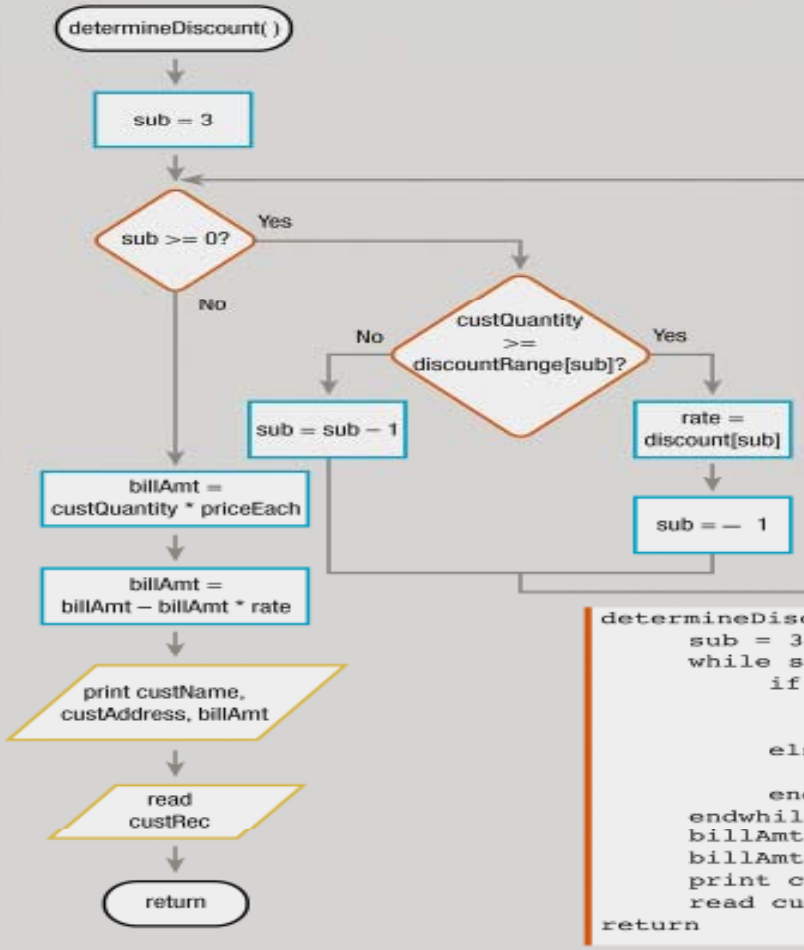
- This approach requires a parallel array to find the appropriate discount level
- Array will hold the low end value of each range
- Start comparing with the last discount array value
  - If quantity is at least as high as the array element value, apply the discount

FIGURE 8-36: THE `discountRange` ARRAY USING LOW END OF EACH DISCOUNT RANGE

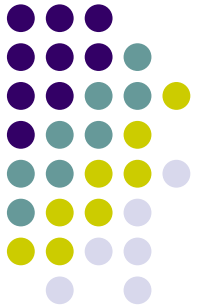
```
num discountRange[0] = 0
num discountRange[1] = 10
num discountRange[2] = 25
num discountRange[3] = 49
```

Searching an Array for a Range Match (continued)

FIGURE 8-37: FLOWCHART AND PSEUDOCODE FOR DISCOUNT DETERMINATION

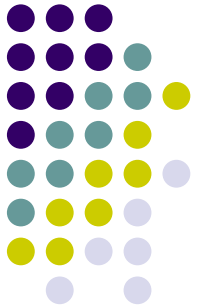



```
determineDiscount( )
      sub = 3
      while sub >= 0
            if custQuantity >= discountRange[sub]
                  rate = discount[sub]
                  sub = – 1
            else
                  sub = sub – 1
            endif
      endwhile
      billAmt = custQuantity * priceEach
      billAmt = billAmt – billAmt * rate
      print custName, custAddress, billAmt
      read custRec
return
```



Summary

- Array: series or list of variables in memory, all with same name and type, but different subscript
- Can use a variable as a subscript to the array to replace multiple nested decisions
- Can declare and initialize all elements in an array with a single statement
- Use a constant array when the array element values are fixed at the beginning of the program
- Can load an array from a file



Summary (continued)

- To search an array, initialize the subscript, then use a loop to test each array element value
- Set a flag when the desired value is found while searching an array
- Parallel arrays: each element in one array is associated with the element in the same relative position in the other array
- Ensure that subscript does not go out of bounds
- For range comparisons, store either the low- or high-end value of each range