CIS1400 – Programming Logic and Technique

Topic 7 → Advanced Data Types

Chapter Topics

8.1 Array Basics

8.2 Sequentially Searching an Array → *later topic*

8.3 Processing the Contents of an Array

8.4 Parallel Arrays

8.5 Two-Dimensional Arrays

8.6 Arrays of Three or More Dimension

8.1 Array Basics

An **array** allows you to store a group of items of the same data type together in memory

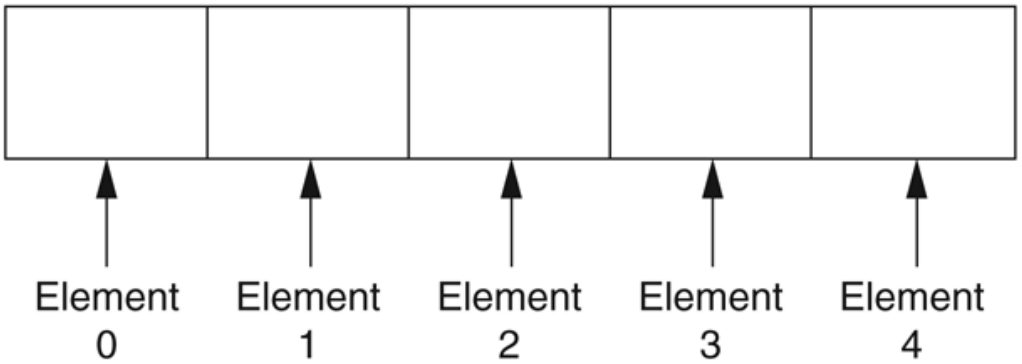
- Why? Instead of creating multiple similar variables such as *employee1*, *employee2*, *employee3* and so on…
- It's more efficient to create just one variable to hold all possible values
  - `Declare String employees[50]`
  - `Declare Integer units[10]`
  - `Declare Real salesAmounts[7]`
- The number in the [ ] is the size of the array
  - In most languages, an array's size cannot be changed while the program is running.

8.1 Array Basics

- The storage locations in an array are **elements**
- Each element of the array has a unique number called a **subscript** that identifies it – the subscript always starts at 0 *(memory offset)*

Figure 8-1 Array subscripts

```
Constant Integer SIZE = 5
Declare Integer numbers[SIZE]
```



8.1 Array Basics

Assigning values can be accessed individually using a subscript…

```
Set numbers[0] = 20
Set numbers[1] = 30
Set numbers[2] = 40
Set numbers[3] = 50
Set numbers[4] = 60
```

Array elements are accessed by name and a subscript, in brackets

But, it is much more efficient to use a Loop to step through the array

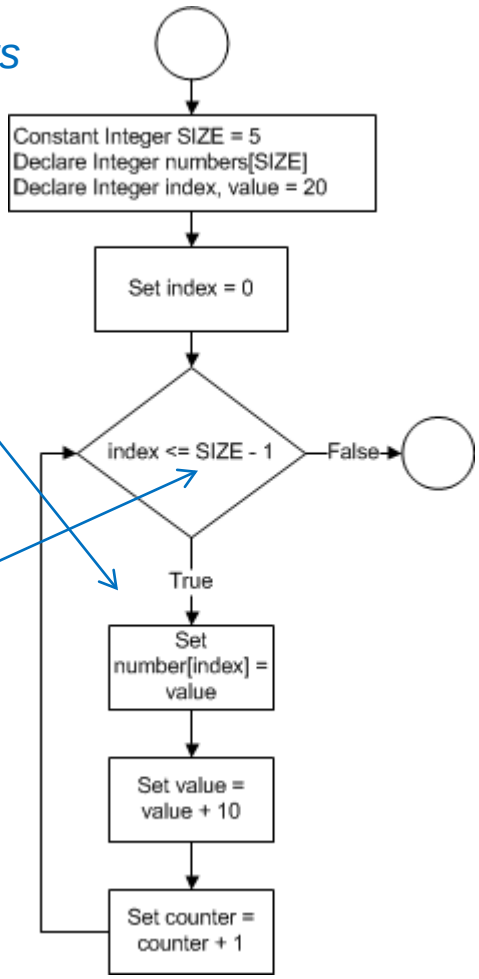
```
Constant Integer SIZE
Integer numbers[SIZE]
```

[0]	[1]	[2]	[3]	[4]
20	30	40	50	60

8.1 Array Basics

Array elements are accessed by name and a subscript, in brackets

```
Constant Integer SIZE = 5
Declare Integer numbers[SIZE]
Declare Integer index, value = 20
For index = 0 To SIZE – 1
   Set numbers[index] = value
   Set value = value + 10
End For
```



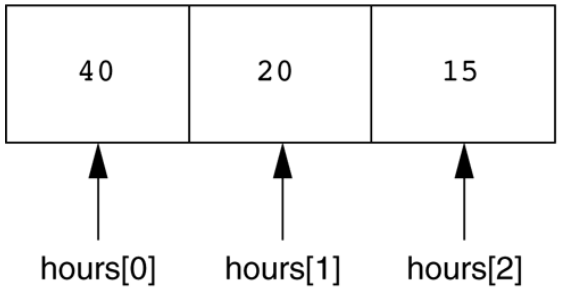
Note how loop counter goes from 0 to maximum array subscript (which is one less than the size)

```
Constant Integer SIZE
Integer numbers[SIZE]
```

[0]	[1]	[2]	[3]	[4]
20	30	40	50	60

8.1 Array Basics

- Assigning array elements through user input
  - Program 8-3




```
Constant Integer SIZE = 3
Declare Integer hours[SIZE]
Declare Integer index
For index = 0 To SIZE – 1
   Display “Enter the hours worked by”
   Display “employee number “, index + 1
   Input hours[index]
End For
For index = 0 To SIZE – 1
   Display hours[index]
End For
```

input

display

8.1 Array Basics

Arrays can be initialized to 0 or specific values in declaration statement

```
Declare String days[7] = “Sunday”,  “Monday”,
  “Tuesday”,  “Wednesday”,  “Thursday”,
  “Friday”,  “Saturday”
```

days

0	*Sunday*
1	*Monday*
2	*Tuesday*
3	*Wednesday*
4	*Thursday*
5	*Friday*
6	*Saturday*

8.1 Array Basics

Array bounds checking should be performed to avoid use of an invalid subscript

```
Set Days[7] = "Saturday"
```

- is invalid because there is no 7 index

- A common error is running a loop one time more than is necessary, exceeding the bounds of the array
  - Usually causes a runtime error (program crash)
- Off-by-one Error
  - Remember: *arrays subscripts start at 0 rather than 1*

```
Constant Integer SIZE = 100
Declare Integer numbers[SIZE]
Declare Integer index
For index = 1 To SIZE – 1
   Set numbers[index] = 0
End For
```

```
Constant Integer SIZE = 100
Declare Integer numbers[SIZE]
Declare Integer index
For index = 0 To SIZE
   Set numbers[index] = 0
End For
```

8.3 Processing the Contents of an Array

Totaling the values in an array and calculating average

Program 8-10

```
Constant Integer SIZE = 5
Declare Real scores[SIZE] = 2.5, 8.3, 6.5, 4.0, 5.2
Declare Real average, total = 0
Declare Integer index
For index = 0 To SIZE - 1
   Set total = total + scores[index]
End For
Set average = total / SIZE
Display "The average of the array elements is ", average
```

total

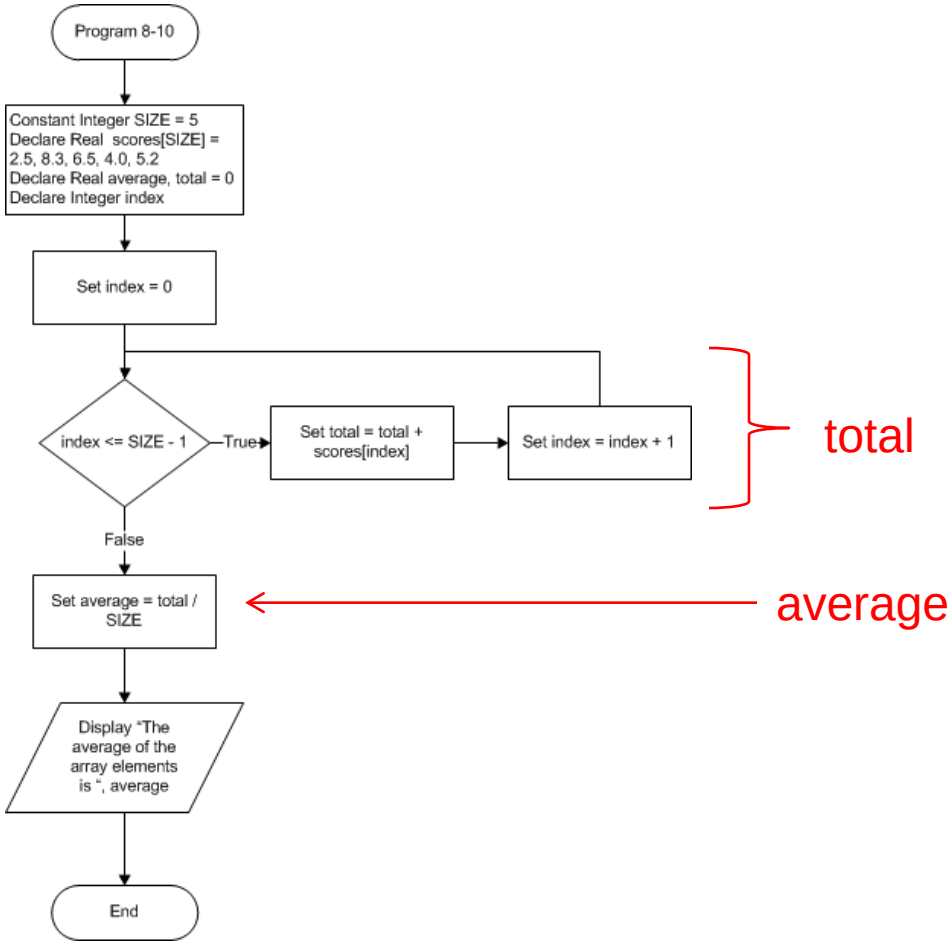
average

Program Output

```
The average of the array elements is 5.3
```

8.3 Processing the Contents of an Array

Totaling the values in an array and calculating average



8.3 Processing the Contents of an Array

Finding the highest & lowest values input

- Create variables to hold the highest and lowest values
- Assign the first input number to the highest and lowest variables
- Use a loop to step through the user input data
  - Lab #4 used sentinel value of -99
- Each iteration, a comparison is made between the user input value to the highest and lowest variables
  - If the element is greater than the highest value, that value is then the assigned to the highest variable
  - If the element is less than the lowest value, that value is then the assigned to the lowest variable

8.3 Processing the Contents of an Array

Finding the highest & lowest values in an array

- The highest
  - Create a variable to hold the highest value
  - Assign the value at element 0 to the highest
  - Use a loop to step through the rest of the elements
    - Based upon array size
  - Each iteration, a comparison is made to the highest variable
    - If the element is greater than the highest value, that value is then the assigned to the highest variable

8.3 Processing the Contents of an Array

Finding the highest & lowest values in an array

- The highest

Program 8-11

```
Constant Integer SIZE = 5
Declare Integer numbers[SIZE] = 8, 1, 12, 6, 2
Declare Integer index
Declare Integer highest
Set highest = numbers[0]
For index = 1 To SIZE – 1
   If numbers[index] > highest Then
      Set highest = numbers[index]
   End If
End For
Display “The highest value in the array is “, highest
```

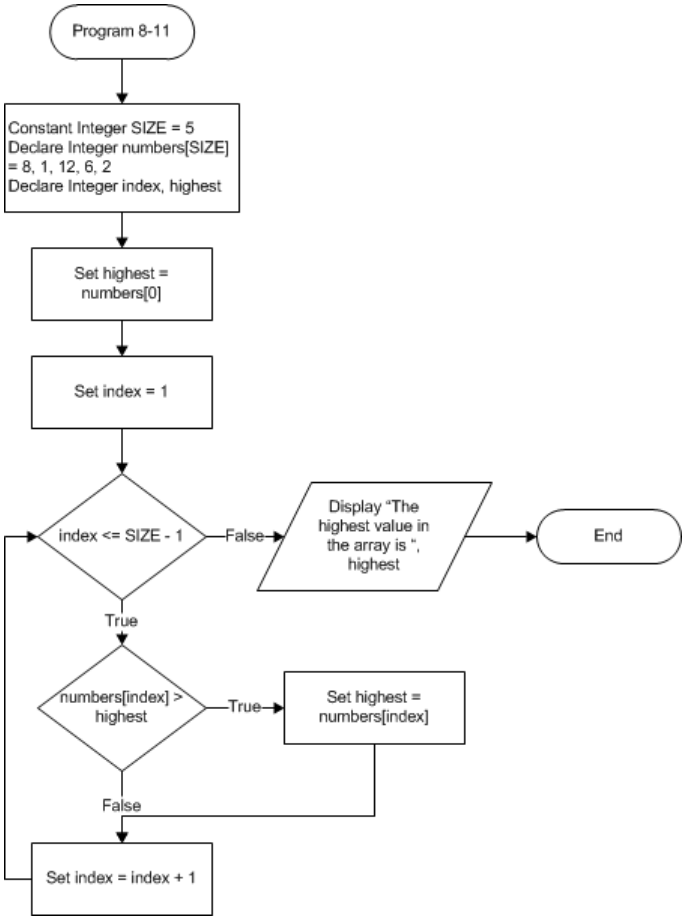
Program Output

```
The highest value in the array is 12
```

8.3 Processing the Contents of an Array

Finding the highest & lowest values in an array

- The highest



8.3 Processing the Contents of an Array

Finding the highest & lowest values in an array

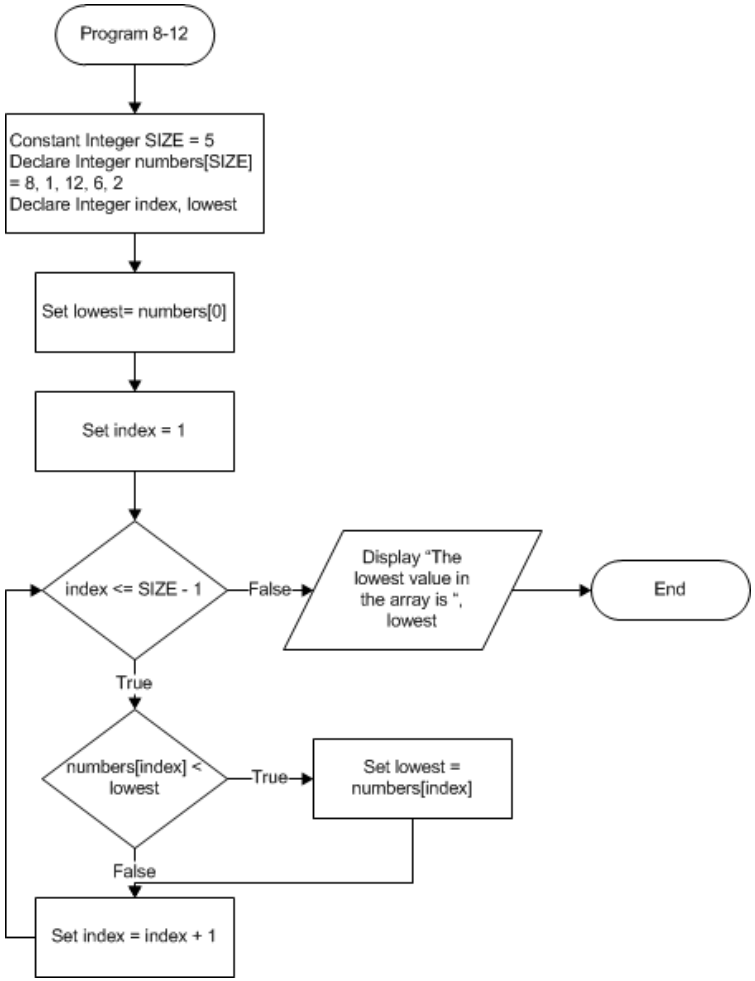
- The lowest
  - Same process, but checks if the element is less than the lowest value

```
Program 8-12
Constant Integer SIZE = 5
Declare Integer numbers[SIZE] = 8, 1, 12, 6, 2
Declare Integer index
Declare Integer lowest
Set lowest = numbers[0]
For index = 1 To SIZE – 1
   If numbers[index] < lowest Then
      Set lowest = numbers[index]
   End If
End For
Display “The lowest value in the array is “, lowest
Program Output
The lowest value in the array is 1
```

8.3 Processing the Contents of an Array

Finding the highest & lowest values in an array

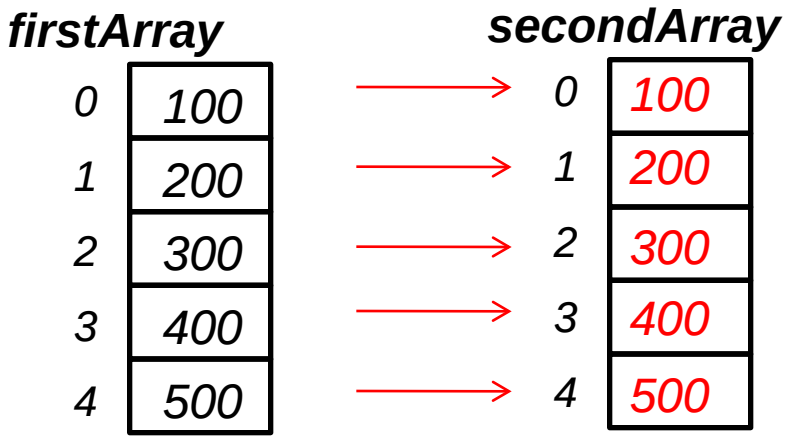
- The lowest

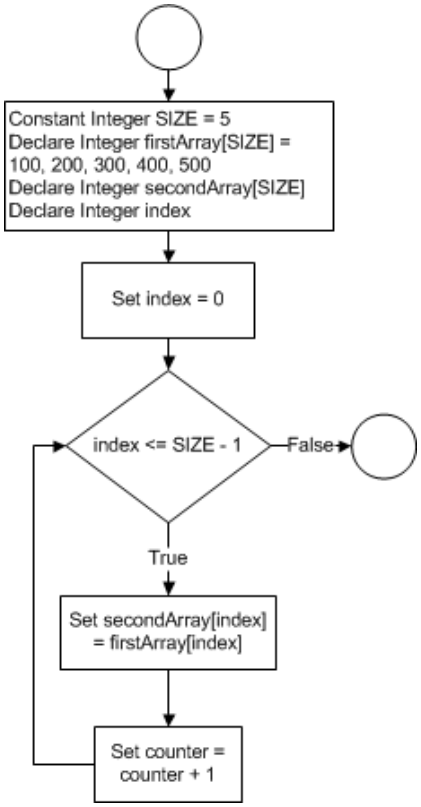


8.3 Processing the Contents of an Array

Copying an array can be done using a loop to copy the individual elements

```
Constant Integer SIZE = 5
Declare Integer firstArray[SIZE] = 100, 200, 300,
  400, 500
Declare Integer secondArray[SIZE]
Declare Integer index
For index = 0 To SIZE – 1
  Set secondArray[index] = firstArray[index]
End For
```





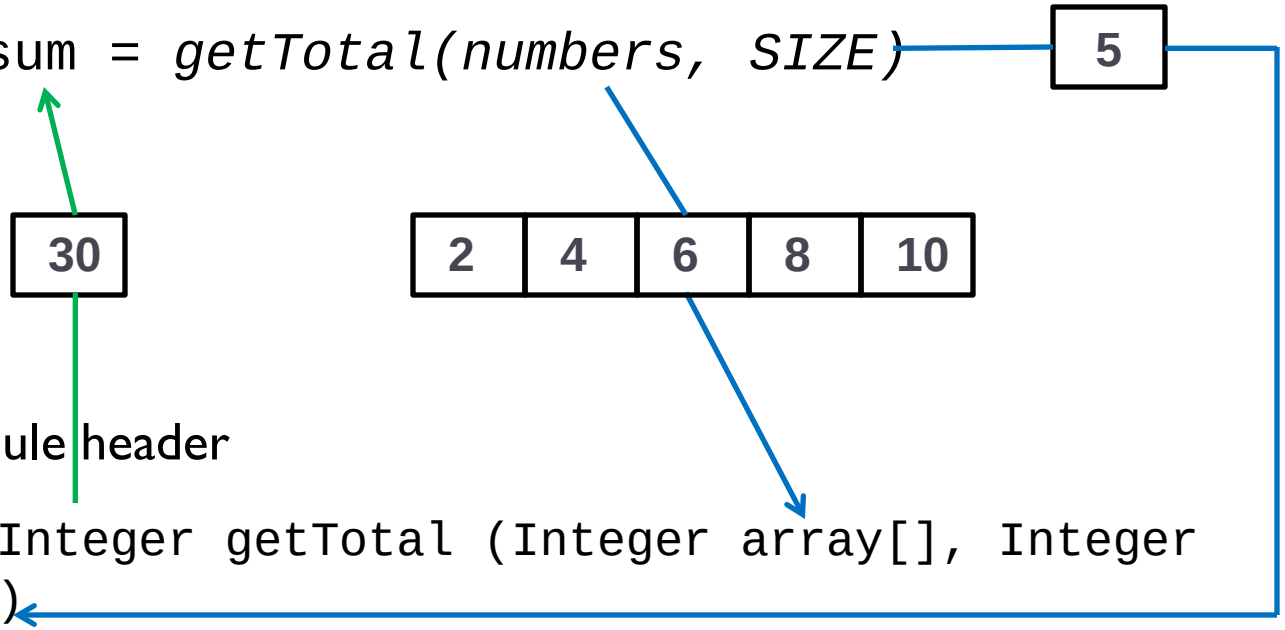
8.3 Processing the Contents of an Array

Passing an Array as an Argument

- Usually must pass the array and the size

The module call

```
Set sum = getTotal(numbers, SIZE)
```



The module header

```
Function Integer getTotal (Integer array[], Integer
arraySize)
```

8.3 Processing the Contents of an Array

Passing an Array as an Argument

```
Program 8-13
Module main()
   Constant Integer SIZE = 5
   Declare Integer numbers[SIZE] = 2, 4, 6, 8, 10
   Declare Integer sum
   Set sum = getTotal(numbers, SIZE)
   Display "The sum of the array elements is ", sum
End Module

Function Integer getTotal(Integer array[], Integer arraySize)
   Declare Integer index
   Declare Integer total = 0
   For index = 0 to arraySize – 1
      Set total = total + array[index]
   End For
   Return total
End Function
```

Program Output

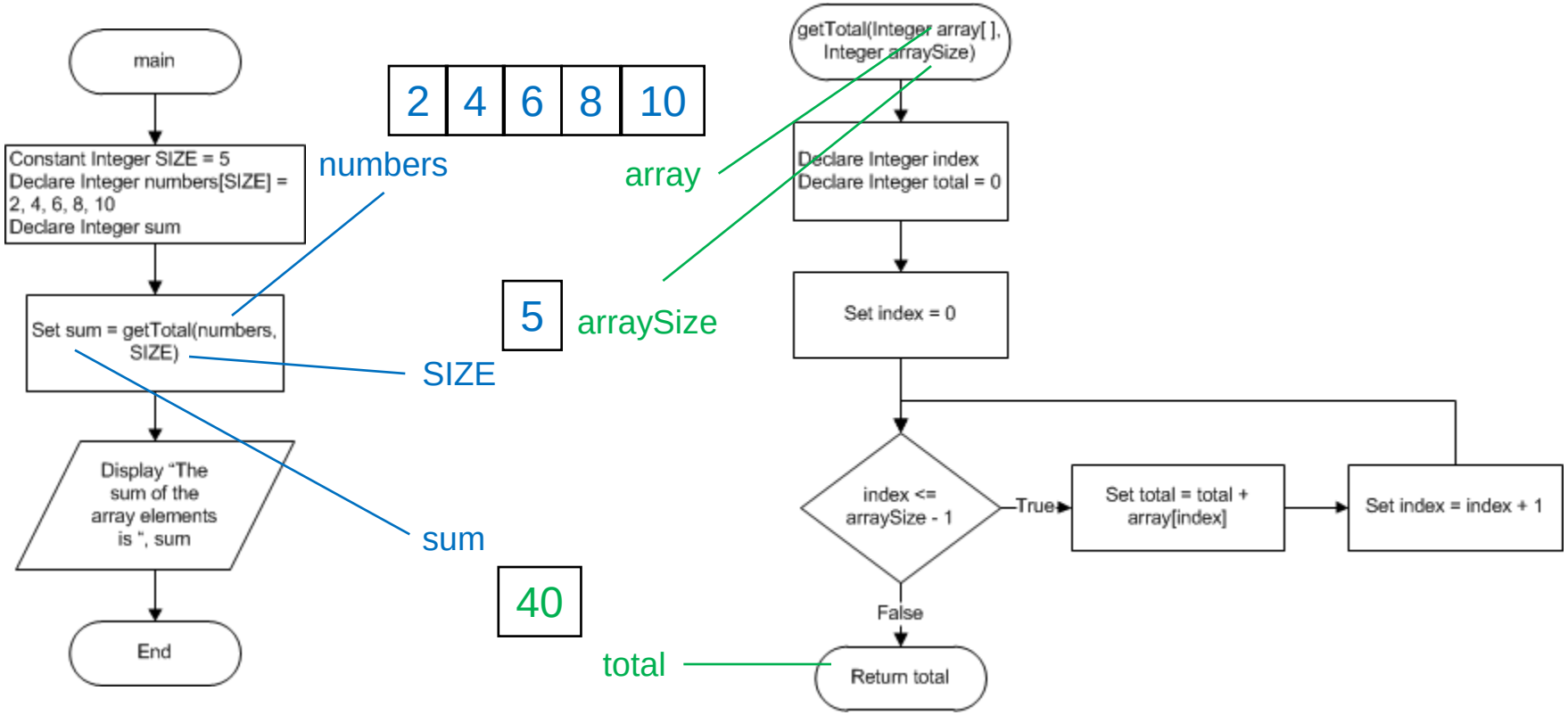
```
The sum of the array elements is 30
```

argument only uses variable name;

parameter uses data type, parameter name, and brackets

8.3 Processing the Contents of an Array

Passing an Array as an Argument

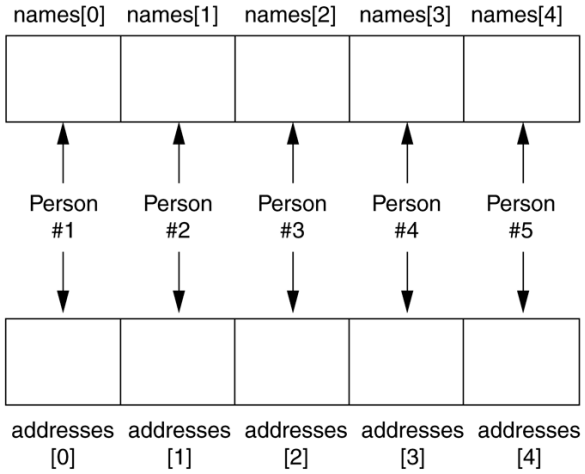


8.4 Parallel Arrays

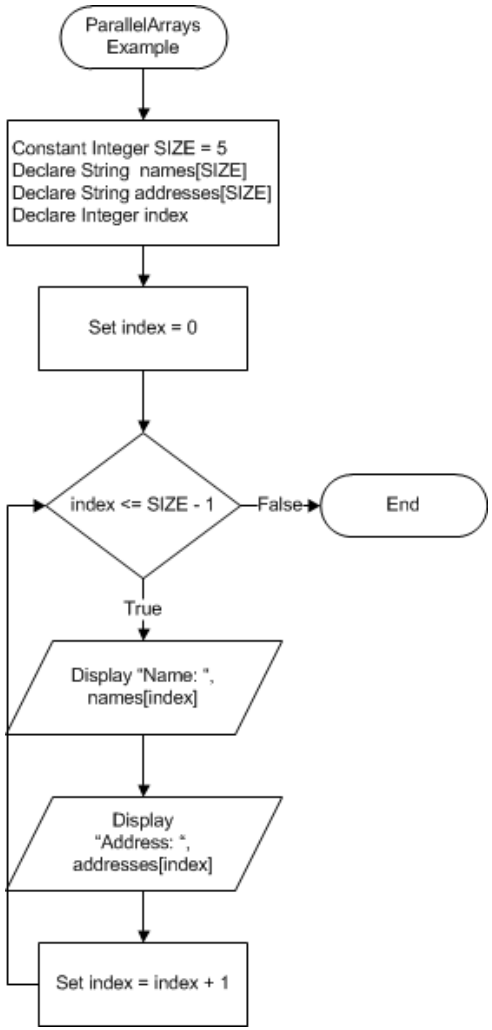
Two or more arrays that hold related data and related elements are accessed with common subscript

- No special terminology in array definition
- Arrays should be of same size
- Arrays need not be of same data type
- Related elements are accessed with a common subscript
  - Stored in same relative location of each array

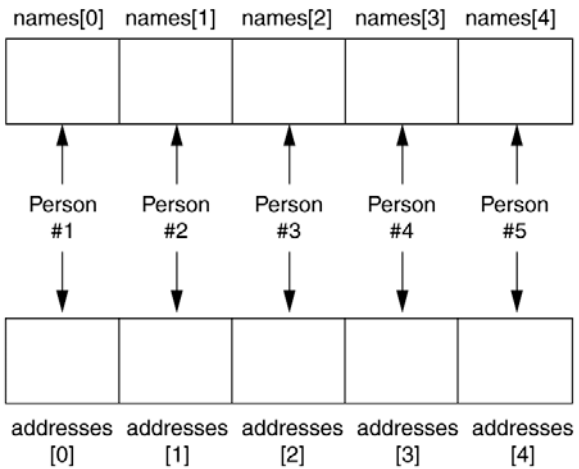
Figure 8-14 The names and addresses arrays



8.4 Parallel Arrays




```
Constant Integer SIZE = 5
Declare String names[SIZE]
Declare String addresses[SIZE]
Declare Integer index
For index = 0 To SIZE – 1
  Display “Name: “, names[index]
  Display “Address: “, addresses[index]
End For
```



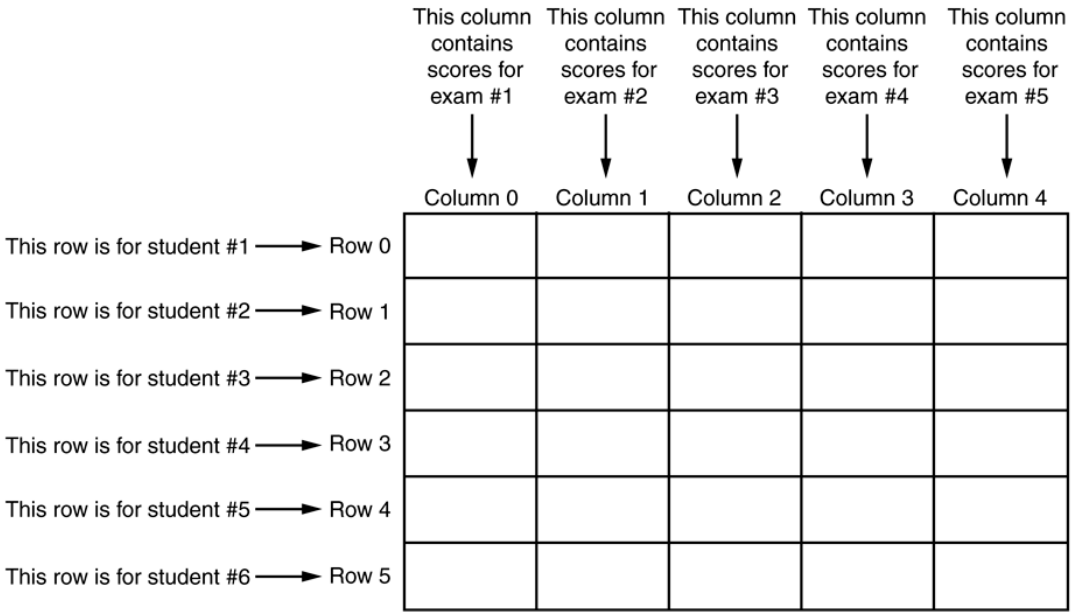
In The Spotlight Using Parallel Arrays (p316)

8.5 Two-Dimensional Arrays

A two-dimensional array is like several identical (data type) arrays put together

- Useful for working with multiple sets of data
- Suppose a teacher has six students who take five tests

Figure 8-17 Two-dimensional array with si rows and five columns



8.5 Two-Dimensional Arrays

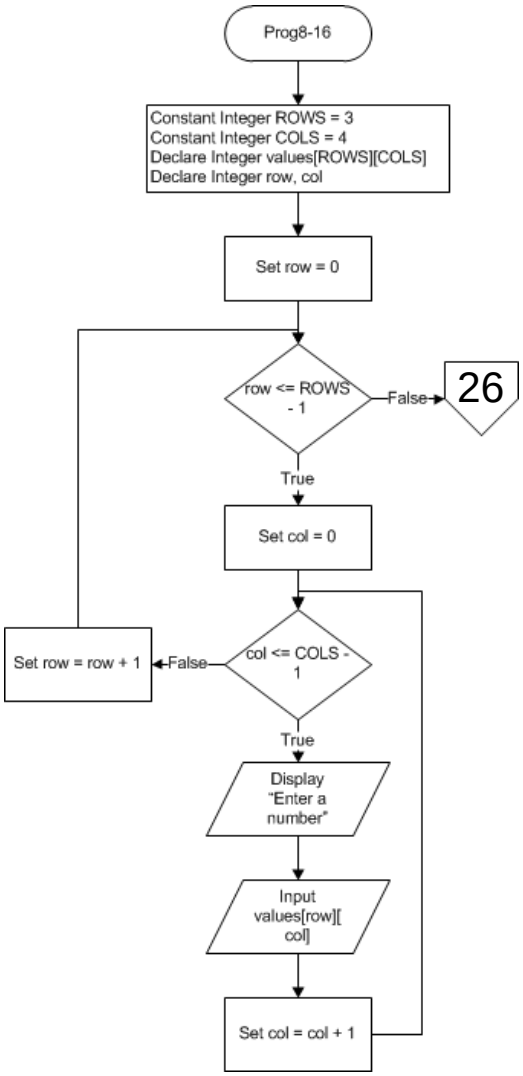
Two size variables are required when declaring

```
Constant Integer ROWS = 3
Constant Integer COLS = 4
Declare Integer values[ROWS][COLS]
Declare Integer row, col
```

Accessing is done with two loops, and both subscripts

- Get values to store in array

```
For row = 0 To ROWS - 1
  For col = 0 To COLS – 1
   Display “Enter a number.”
   Input values[row][col]
  End For
End For
```

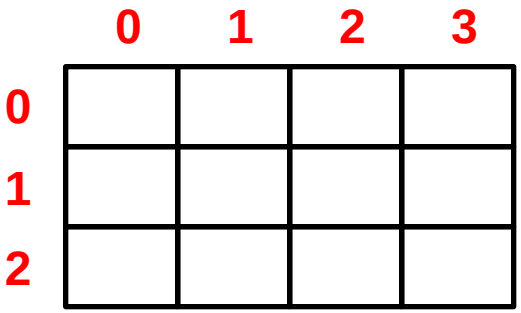


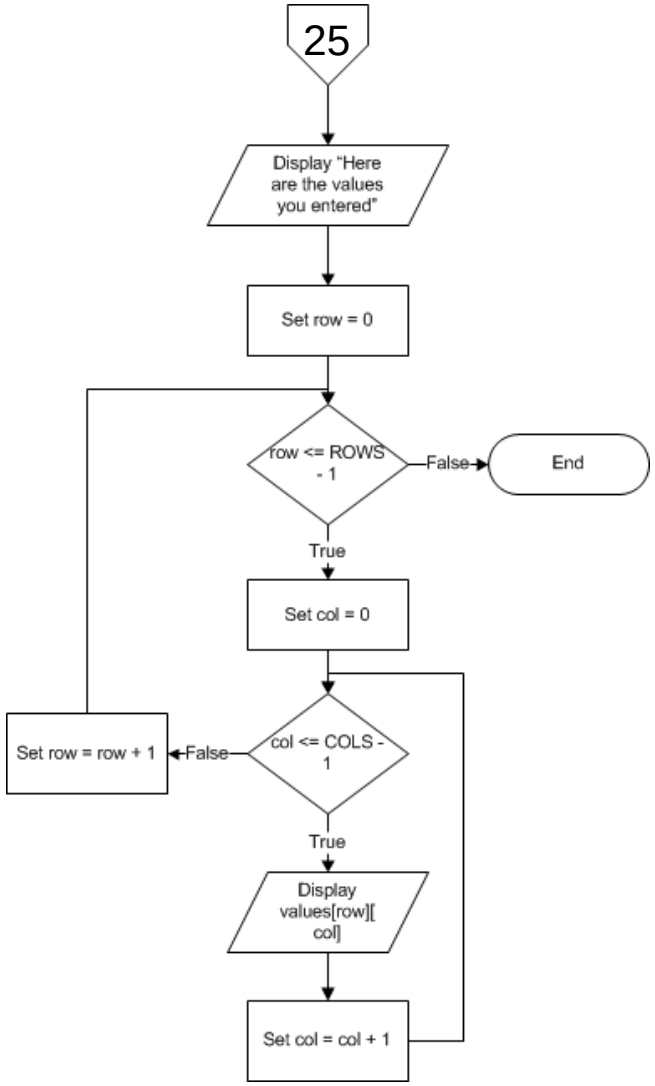
8.5 Two-Dimensional Arrays

Accessing is done with two loops, and both subscripts

- Display values in array

```
Display "Here are the values
  you entered."
For row = 0 To ROWS - 1
  For col = 0 To COLS – 1
   Display values[row][col]
  End For
End For
```





8.6 Arrays of Three or More Dimensions

Arrays can also be three or more dimensions

- Use a bracket for each dimension in declaration

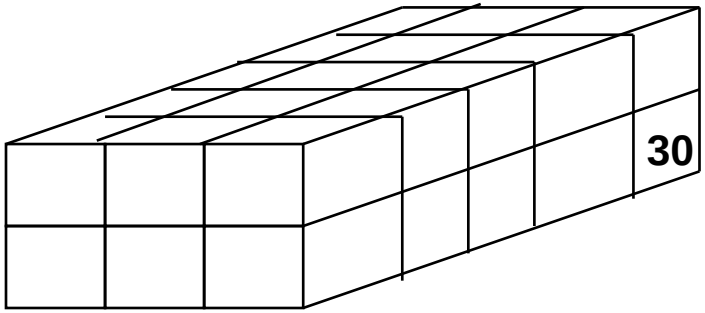
```
Constant Integer ROWS = 2, COLS = 3, DEPTH = 5
Declare Integer values[ROWS][COLS][DEPTH]
```

- Use a bracket for each dimension in accessing one element

```
Set values[1][2][4] = 30
```

- Use a loop for each dimension when accessing all elements

```
For x = 0 To ROWS -1
   For y = 0 To COLS – 1
      For z = 0 To DEPTH - 1
        Display “Enter a number.”
        Input values[x][y][z]
      End For
   End For
End For
```



Chapter Topics

12.1 Introduction

12.2 Character-by-Character Text Processing

12. 1 Introduction

- Functions covered so far operate on strings
  - Chapter 6 Functions

Table 12-1 Common String Functions

Function	Description
`length(string)`	Returns the number of characters in *string*. For example, the expression `length("Test")` would return 4.
`append(string1, string2)`	Returns a string that is created by appending *string2* to the end of *string1*. For example, the expression `append("Hello ", "World")` would return the string `"Hello World"`.
`toUpper(string)`	Returns a string that is an uppercase copy of *string*. For example the expression `toUpper("Test")` would return the string `"TEST"`.
`toLower(string)`	Returns a string that is a lowercase copy of *string*. For example the expression `toLower("TEST")` would return the string `"test"`.
`substring(string, start, end)`	Returns a substring of *string*. The substring is the set of characters starting at the position specified by *start* and ending at the position specified by *end*. (The first character in *string* is at position 0.) For example, the expression `substring("Kevin", 2, 4)` would return the string `"vin"`.
`contains(string1, string2)`	Returns `True` if *string1* contains *string2*. Otherwise it returns `False`. For example the expression `contains("smiley", "mile")` would return `True`, and the expression `contains("Smiley",` [illegible]

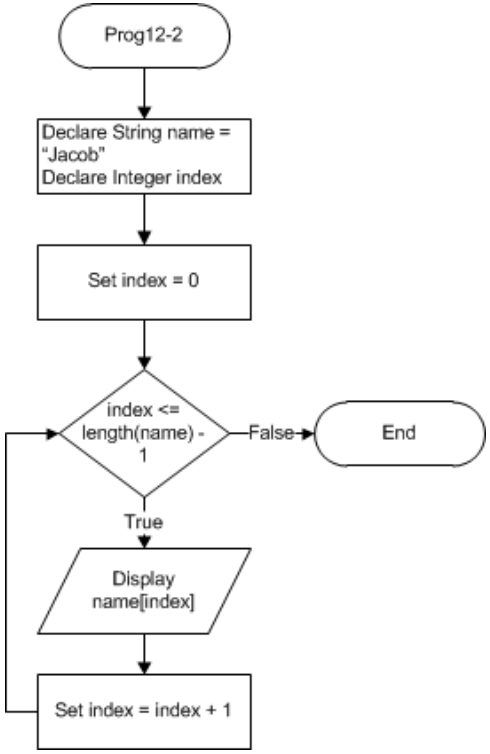
12.2 Character-by-Character Text Processing

- Most languages provide a way to access the individual characters in a string
  - Often this is done with subscript notation, similar to accessing the elements of an array

```
Program 12-2
Module main()
   Declare String name = "Jacob"
   Declare Integer index
   For index = 0 to length(name) - 1
      Display name[index]
   End For
End Module

Program Output
J
a
c
o
b
```

Loop used to step through all characters in string.



12.2 Character-by-Character Text Processing

- Individual characters of a string
  - Can be modified:

```
Declare String str = "Coffee"
str[0] = "T"
Display str
```

Displays “Toffee”

  - Can be tested:

Function	Description
`isDigit(character)`	Returns `True` if *`character`* is a numeric digit, or `False` otherwise.
`isLetter(character)`	Returns `True` if *`character`* is an alphabetic letter or `False` otherwise.
`isLower(character)`	Returns `True` if *`character`* is a lowercase letter or `False` otherwise.
`isUpper(character)`	Returns `True` if *`character`* is an uppercase letter or `False` otherwise.
`isWhiteSpace(character)`	Returns `True` if *`character`* is a whitespace character or `False` otherwise. (A whitespace character is a space, a tab, or a newline.)

In The Spotlight Validating a Password (p480)

12.2 Character-by-Character Text Processing

Program 12-4

```
Module main()
  Declare String str
  Declare Integer index
  Declare Integer upperCaseCount = 0
  Display “Enter a sentence.”
  Input str
  For index = 0 to length(str) - 1
    If isUpper(str[index]) Then
      Set upperCaseCount = upperCaseCount + 1
    End If
  End For
  Display “That string has “, uppercaseCount, “ uppercase letters.”
End Module
```

Program Output

```
Enter a Sentence.
Ms. Jones will arrive TODAY! [Enter]
That string has 7 uppercase letters.
```

12.2 Character-by-Character Text Processing

- Functions for inserting and deleting characters

Function	Description
`insert(string1, position, string2)`	*string1* is a `String`, *position* is an `Integer`, and *string2* is a `String`. The function inserts *string2* into *string1*, beginning at *position*.
`delete(string, start, end)`	*string* is a `String`, *start* is an `Integer`, and *end* is an `Integer`. The function deletes from *string* all of the characters beginning at the position specified by *start*, and ending at the position specified by *end*. The character at the ending position is included in the deletion.

12.2 Character-by-Character Text Processing

- Insert example

```
Declare String str = "New City"
insert(str, 4, "York ")
Display str // displays “New York City”
```

- Delete example

```
Declare String str = "I ate 1000 blueberries!"
delete(str, 8, 9)
Display str // displays “I ate 10 blueberries!”
```