**Chapter 8**

# Arrays

**Coverage:**

This chapter teaches you to use arrays and structures to process data through the use of various loop and table lookup procedures. Single and multi-dimensional arrays are covered. For Each…Next loops are used to process arrays. ListBox controls are also used to process arrays.

**INTRODUCTION**

In this chapter you will develop a project for the VB University Bookstore that enables students to find information about products that are for sale in the bookstore. The initial appearance of the form for the project is shown below.

You will create three versions of this form while studying the concepts taught in this module.

The menu strip control has the following menu items with hot keys and shortcut keys indicated.

|  |  |  |
| --- | --- | --- |
| File  Total Sales by Product  Exit Ctrl-X | Product  Search Ctrl-S  Purchase Ctrl-U  ­-------------------------  Reset Ctrl-R | Help  About |

Text box control (**DescriptionTextBox**, **PriceTextBox**, **QuantityTextBox**, and **TotalDueTextBox**) property settings:

* **ReadOnly** – **True**
* **TabStop** – **False**
* **TextAlign** – **Right** (all but the **DescriptionTextBox**).

**Single-Dimension Arrays**

An **Array** is a **list** **of values** – all values in an array are referenced by using the **same name**.

* An array is like a ListBox without the box where the list of values are all stored in memory.
* Arrays are also called **tables**.
* Arrays are based on the **System.Array** object – a type of **collection** object. Arrays can store almost any type of data such as integer, string, decimal, etc.
* Element – this is the term used to refer to an individual part of an array.
* Each **element** is numbered beginning with the number **zero**. Example, **0**, **1**, **2**, **3**, etc.
* The number referring to an array elementis placed inside parentheses and is called a **subscript** or **index**. This is the general format for an array reference.

**ArrayName(ElementNumber)**

* Example references to array elements:

**'Element 4 of the array named AmountDecimal – remember**

**'that arrays start with 0 so the 4th element is numbered 3  
AmountDecimal(3)**

**'Element 7 of an array named StatesString**

**StatesString(6)**

* An array element can also be referenced by using an index variable – usually each element in a single-dimension array is termed a row so you might name the index variable a name such as **RowInteger**.
* In this example the referenced element depends on the current value of **RowInteger**.

**'The referenced element depends on the value of RowInteger**

**StatesString(RowInteger)**

**Declaring an Array**

Arrays are created in memory by declaring them as you would a variable.

* Generally you declare local arrays with the keyword **Dim** and module-level arrays with the keyword **Private** in order to specify the array scope.
* The number inside parentheses that specifies the number of elements should be an **integer** – if a floating point number is used, VB will round the number to an integer value automatically.
* Numeric array elements default to a value of zero; string array elements default to a value of the empty string.
* There are several different correct syntax variations for declaring arrays. The general formats are:

**Dim ArrayName(UpperSubscript) As DataType**

**Dim ArrayName() As DataType = {ListOfValues}**

**Dim ArrayName As DataType() = {ListOfValues}**

* Example declarations:

**'A String array with 26 elements numbered 0 through 25**

**Dim NameString(25) As String**

**'A decimal array of 11 elements numbered 0 through 10**

**Private BalanceDecimal(10) As Decimal**

* This example declares a module-level array and stores values to the array immediately. When storing values immediately to the array, you **cannot** specify the number of elements within the parentheses—VB will determine the number of elements automatically.

**'A private string array with 6 elements numbered 0 through 5**

**Private SchoolString() As String = {"Arts and Sciences", \_**

**"Business", "Nursing", "Engineering", \_**

**"Education", "Pharmacy"}**

|  |  |
| --- | --- |
| Element Number | School Name |
| 0 | Arts and Sciences |
| 1 | Business |
| 2 | Nursing |
| 3 | Engineering |
| 4 | Education |
| 5 | Pharmacy |

**Array Subscript Errors**

A subscript (also referred to as an index) must always reference a valid, existing array element.

If an array contains **10 elements**, and a VB program attempts to reference **element -1** or **element 11**, the reference will be invalid and result in an error message:

**An unhandled exception of type 'System.IndexOutOfRangeException' occurred in Ch08VBUniversity-Version1.exe**

The subscript (index) is out of the allowable range of 0 through 10 for element numbers.

**For Each...Next Loops**

Arrays are often processed in loops. The power of arrays comes with using a variable as the subscript.

* A **For...Next** loop can process an entire array with a few lines of code as shown here, but the code requires you to manipulate the array index.
* This example writes the array contents to the Output window, one school name per line:

**For RowInteger = 0 To NumberOfSchoolsInteger**

**Debug.WriteLine(SchoolString(RowInteger))**

**Next RowInteger**

Another type of count-controlled loop is the **For Each...Next** loop – the advantage of this coding construct is that you don't have to manipulate the index.

* The loop code shown here is equivalent to the one shown above.

**For Each SchoolNameString As String In SchoolString**

**Debug.WriteLine(SchoolNameString )**

**Next SchoolNameString**

Learn these rules about **For Each...Next** loops:

* The **index** **data type** must match the array data type, i.e., for a string array the index should be a string index, for an integer array the index should be an integer, etc.
* This above example declares **SchoolNameString** as a string variable index and then immediately uses it to search the **SchoolString** array.

An array can be used to accumulate values where you would otherwise need to declare many accumulating variables. Sometimes a numeric accumulating array needs to be re-initialized to zero. This loop sets all elements of an integer array named **TotalInteger**  to zero. Remember a numeric array starts out with each element equal to zero, but if the array is used over and over within a program, it may need to be reinitialized.

**For Each IndividualElementInteger As Integer In TotalInteger**

**IndividualElementInteger = 0**

**Next IndividualElementInteger**

**Structures**

A **structure** object allows you to combine fields of related data into a single object – you can combine data of type integer, decimal, string, color, radio button, and other data types. This allows you to create your own, unique data types.

* Structures are **public** by default, but you can also declare **private** structures.
* This example defines a **Product** structure that might be used to track products sold by retail stores.
* Each product has a product identifier, product description, quantity on hand, and price. The **ProductIDString**, **DescriptionString**, **QuantityInteger**, and **PriceDecimal** variables become properties of the **Product** data type.

**Structure Product**

**Dim ProductIDString As String**

**Dim DescriptionString As String**

**Dim QuantityInteger As Integer**

**Dim PriceDecimal As Decimal**

**End Structure**

* Declare variables and arrays as a structure data type just as you would any other data type.

**Dim TextBookProduct As Product**

**Dim InventoryProduct(450000) As Product**

* These assignment statements store values to the properties of the **TextBookProduct** variable and two of the elements of the **InventoryProduct** array.

**TextBookProduct.ProductIDString = "X599"**

**TextBookProduct.DescriptionString = "Intro to CMIS Textbook"**

**TextBookProduct.QuantityInteger = 75**

**TextBookProduct.PriceDecimal = 79.4D**

**InventoryProduct(0).ProductIDString = "A402"**

**InventoryProduct(0).DescriptionString = "History of America Textbook"**

**InventoryProduct(0).QuantityInteger = 10**

**InventoryProduct(0).PriceDecimal = 65.55D**

**InventoryProduct(1).ProductIDString = "A804"**

**InventoryProduct(1).DescriptionString = "College Logo Tshirt"**

**InventoryProduct(1).QuantityInteger = 15**

**InventoryProduct(1).PriceDecimal = 18.99D**

This example defines a **Sale** structure with three properties – the **SaleDecimal** property is an array.

**'Module-level declaration**

**Structure Sale**

**Dim SaleIDString As String**

**Dim SaleDate As Date**

**Dim SaleDecimal() As Decimal**

**End Structure**

* Declaring an array as a structure element requires use of the **ReDim** statement because when an array is initially declared inside of a structure, you are not allowed to specify the number of elements.

**'later in the code**

**Dim SalesInfoSale As Sale**

**'Inside a procedure you would need to redim the array**

**'attribute of the structure**

**ReDim SalesInfoSale.SaleDecimal(6)**

**'Later during processing.**

**SalesInfoSale.SaleIDString = 4**

**SalesInfoSale.SaleDecimal(RowInteger) = TodaysSalesAmountDecimal**

**Table Lookup**

**Table Lookup** – a programming technique to search an array (table).

* Objective: find a **key value** in order to use some other **non-key value** that is associated with the key for some type of processing.

**Analogous Example:** Consider how you look up information in a telephone book.

* The telephone book is actually a very large table with the Name, Address, and Telephone Number columns.
* The **key value** in the telephone book is the **Name**.
* The value you actually want to use is the **Telephone Number**.

Consider the array shown below with four columns: (1) **ProductIDString** stores each product's identifier, (2) **DescriptionString** stores each items corresponding description, (3) **QuantityInteger** stores the quantity on hand for each product, and (4) **PriceDecimal** stores the price of the item

**Tasks:**

* Declare a **Product** structure.
* Declare an array of type **Product** and store data to the array.
* Write a search procedure—search for a product ID value in order to display the corresponding description, quantity, and price.

|  |  |  |  |
| --- | --- | --- | --- |
| **ProductIDString** | **DescriptionString** | **QuantityInteger** | **PriceDecimal** |
| **A402** | **History of America Textbook** | **10** | **65.55** |
| **A804** | **College Logo Tshirt** | **15** | **18.99** |
| **C344** | **College Logo Sweat Pants** | **25** | **25.99** |
| **F554** | **Drinking Mug** | **8** | **5.49** |
| **G302** | **Pencil and Pen Set** | **15** | **35.50** |
| **M302** | **College Logo Sweat Shirt** | **25** | **22.99** |
| **S499** | **Intro to Philosophy Textbook** | **50** | **85.00** |
| **X599** | **Intro to CMIS Textbook** | **75** | **79.40** |

**Version 1 of the Bookstore Application**

**Create or Copy Project**

**Task 1:** Create a new project named **Ch08VBUniversity-Version1** or copy the **Ch08VBUniversity-Start Project** folder from drive Y of the class server.

If you create a new project, build the form for the VB University Bookstore project as shown earlier in this note set and set the form’s **File Name** = **BookStore1.vb**.

* Set the **CharacterCasing** property of the **ProductIDTextBox** control to **Upper** – this will cause any alphabetic characters typed in the TextBox control to automatically convert to upper case.
* Set the **MaxLength** property of the **ProductIDTextBox** control to **4** – this will limit the number of characters that can be typed into the TextBox control.
* Map the **SearchButton** control to the keyboard’s **Enter** key by setting the form’s **AcceptButton** property.
* Define a structure in the general declarations section named **Product**.

**'Public structure to define Product data type**

**Structure Product**

**Dim ProductIDString As String**

**Dim DescriptionString As String**

**Dim QuantityInteger As Integer**

**Dim PriceDecimal As Decimal**

**End Structure**

* Declare a module-level integer variable specifying the maximum number of products in inventory (set its value to 7 to represent 8 different products numbered 0 to 7).
* Declare a module-level array of type **Product** named **InventoryProduct** that will enable processing up to the eight product items in the bookstore – this should be flexible and easily changed.

**'Module-level variables/constants/arrays**

**Private NumberProductsInteger As Integer = 7**

**Private InventoryProduct(NumberProductsInteger) As Product**

**Load Event**

A form's **Load** event can be used to perform initialization tasks that you need to complete whenever a form loads from disk into memory. Example tasks include connecting to a database or storing initial values to an array. The **Load** event fires whenever the form loads the first time.

**Task 2:** Create a Load event for the form as shown below.

* Load initial values into the array elements by coding the form's **Load** procedure event. When you study **Chapter 10**, you will learn how to access **Product** data values from a database table.

**Private Sub BookStore1\_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load**

**InventoryProduct(0).ProductIDString = "A402"**

**InventoryProduct(0).DescriptionString = "History of America Textbook"**

**InventoryProduct(0).QuantityInteger = 10**

**InventoryProduct(0).PriceDecimal = 65.55D**

**InventoryProduct(1).ProductIDString = "A804"**

**InventoryProduct(1).DescriptionString = "College Logo Tshirt"**

**InventoryProduct(1).QuantityInteger = 15**

**InventoryProduct(1).PriceDecimal = 18.99D**

**InventoryProduct(2).ProductIDString = "C344"**

**InventoryProduct(2).DescriptionString = "College Logo Sweat Pants"**

**InventoryProduct(2).QuantityInteger = 25**

**InventoryProduct(2).PriceDecimal = 25.99D**

**InventoryProduct(3).ProductIDString = "F554"**

**InventoryProduct(3).DescriptionString = "Drinking Mug"**

**InventoryProduct(3).QuantityInteger = 8**

**InventoryProduct(3).PriceDecimal = 5.49D**

**InventoryProduct(4).ProductIDString = "G302"**

**InventoryProduct(4).DescriptionString = "Pencil and Pen Set"**

**InventoryProduct(4).QuantityInteger = 15**

**InventoryProduct(4).PriceDecimal = 35.5D**

**InventoryProduct(5).ProductIDString = "M302"**

**InventoryProduct(5).DescriptionString = "College Logo Sweat Shirt"**

**InventoryProduct(5).QuantityInteger = 25**

**InventoryProduct(5).PriceDecimal = 22.99D**

**InventoryProduct(6).ProductIDString = "S499"**

**InventoryProduct(6).DescriptionString = "Intro to Philosophy Textbook"**

**InventoryProduct(6).QuantityInteger = 50**

**InventoryProduct(6).PriceDecimal = 85D**

**InventoryProduct(7).ProductIDString = "X599"**

**InventoryProduct(7).DescriptionString = "Intro to CMIS Textbook"**

**InventoryProduct(7).QuantityInteger = 75**

**InventoryProduct(7).PriceDecimal = 79.4D**

**End Sub**

**Search Button Click Event**

**Task 3:** Code the **Search** Button control's Click event sub procedure to search the array's **ProductIDString** element and, if found, display the associated description, quantity on hand, and price.

* This event sub procedure also handles the Click event for the corresponding **Product-Search** menu item.
* The **ProductIDTextBox.Text** property does not require the **.ToUpper** method because the **CharacterCasing** property of the TextBox control is set to **Upper** already.

**Private Sub SearchButton\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SearchButton.Click, SearchToolStripMenuItem.Click**

**'Search the ProductIDString property of the inventoryProduct**

**'array to see if the value of ProductIDTextBox matches an ID**

**'in the array**

**'Start variables to control the search**

**Dim FoundBoolean As Boolean = False 'Control how long to search**

**Dim RowInteger As Integer = 0 'Current row in the search**

**'Loop to do the search**

**Do Until FoundBoolean = True Or RowInteger > NumberProductsInteger**

**'Compare textBox to array**

**If ProductIDTextBox.Text = InventoryProduct(RowInteger).ProductIDString Then**

**'found a match - display other data to the readonly textboxes**

**DescriptionTextBox.Text = InventoryProduct(RowInteger).DescriptionString**

**QuantityTextBox.Text = InventoryProduct(RowInteger).QuantityInteger.ToString**

**PriceTextBox.Text = InventoryProduct(RowInteger).PriceDecimal.ToString("C2")**

**'change variable to indicate we have a match**

**FoundBoolean = True**

**Else**

**'no match yet**

**RowInteger += 1**

**End If**

**Loop**

**'After the search determine if the ProductID was found**

**If FoundBoolean = False Then 'no match was found**

**'Clear the textbox controls that display product information**

**'except for the ProductID textbox**

**DescriptionTextBox.Clear()**

**QuantityTextBox.Clear()**

**PriceTextBox.Clear()**

**'Display message that the ProductID is not valid**

**MessageBox.Show("Reenter a valid product ID.", "Invalid Identifier", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)**

**ProductIDTextBox.Focus()**

**ProductIDTextBox.SelectAll()**

**End If**

**End Sub**

Test the program.

* Enter a valid product ID and confirm the correct description, quantity on hand and price values display.
* Enter an invalid product ID and confirm the Invalid Product ID message box displays, the text boxes for description, quantity on hand, and price are cleared, and the product ID text box value is selected and highlighted.

**Purchasing a Product**

Values from arrays can be displayed to ListBox controls as part of on-line order processing.

* In the Bookstore application, application users will first search for a product of interest, then click the **Product-Purchase** menu item.
* Display the product purchased to the **PurchaseListBox**.
* Default to the purchase of a quantity of one of the products.
* Accumulate the total value of the order and display this value to the **TotalDueTextBox** control.

**Task 4:** Add a module-level variable to store the total due for an individual bookstore customer (highlighted in yellow).

**'Module-level variables/constants/arrays**

**Private NumberProductsInteger As Integer = 7**

**Private InventoryProduct(NumberProductsInteger) As Product**

**Private TotalDueDecimal As Decimal 'total due for a customer**

**Task 5:** Code the Click event procedure for the **Product-Purchase** menu item.

* + The **If** statement tests for the **DescriptionTextBox** equal to the **empty string** – this means a valid bookstore product was NOT found by a search.
  + When a match has been found, the information to be stored to the ListBox is stored to the **ProductString** variable.
  + The **TotalDueDecimal** variable accumulates the value currently displayed in the **PriceTextBox** control – use of the **Decimal.Parse** **Globalization.NumberStyles.Currency** enumerated value to convert a formatted product price back to decimal for addition to **TotalDueDecimal**.
  + The **TotalDueTextBox** control displayed the updated value of the total amount due.
  + The form can be cleared to prepare for the next item to be purchased.

**Private Sub PurchaseToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles PurchaseToolStripMenuItem.Click**

**'Test to determine if a product was found.**

**If DescriptionTextBox.Text = String.Empty Then**

**'Cannot purchase, product was not found**

**MessageBox.Show("You must select a valid product before purchasing.", "Cannot Purchase", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)**

**ProductIDTextBox.Focus()**

**ProductIDTextBox.SelectAll()**

**Else**

**'Can purchase the product**

**'Build a string to display in the listbox control**

**Dim ProductString As String = ProductIDTextBox.Text & " - " & DescriptionTextBox.Text & " - " & PriceTextBox.Text**

**PurchaseListBox.Items.Add(ProductString)**

**'Accumulate the total value of this customer order**

**'and display it to the output textbox**

**TotalDueDecimal += Decimal.Parse(PriceTextBox.Text, Globalization.NumberStyles.Currency)**

**TotalDueTextBox.Text = TotalDueDecimal.ToString("C2")**

**'Here you can clear the form of product info if you think**

**'that is a good way to do the processing**

**ProductIDTextBox.Clear()**

**DescriptionTextBox.Clear()**

**PriceTextBox.Clear()**

**QuantityTextBox.Clear()**

**ProductIDTextBox.Focus()**

**End If**

**End Sub**

Test the project to determine:

* If the item purchased displays in the ListBox control.
* The total due value accumulates and displays correctly.

**Resetting the Form**

The form must be reset for the next application user (next bookstore customer). This requires clearing all text box controls, clearing the ListBox control, resetting the **TotalDueDecimal** accumulating module-level variable to **zero**, and setting the focus to the **ProductIDTextBox** control. The solution code is straight-forward and is given here.

**Task 6:** Code the **ResetToolStripMenuItem** Click event sub procedure.

**Private Sub ResetToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ResetToolStripMenuItem.Click**

**'Clear all text box controls**

**ProductIDTextBox.Clear()**

**DescriptionTextBox.Clear()**

**PriceTextBox.Clear()**

**QuantityTextBox.Clear()**

**TotalDueTextBox.Clear()**

**'Clear the list box control**

**PurchaseListBox.Items.Clear()**

**'Reset the total due module-level variable to zero**

**TotalDueDecimal = 0**

**'Set the focus to the product ID text box**

**ProductIDTextBox.Focus()**

**End Sub**

**Product – Search Menu item**

This sub procedure illustrates use of the **PerformClick** method (for review purposes) in coding the Click event of the **Product-Search** menu item. You could alternatively alter the **Handles** clause of the **SearchButton\_Click** event sub procedure.

**Task 7:** Code the **SearchToolStripMenuItem** Click event sub procedure

**Private Sub SearchToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SearchToolStripMenuItem.Click**

**'Call the Click event for the Search button control**

**SearchButton.PerformClick()**

**End Sub**

**Help –About Menu**

The **Help-About** menu option is straight-forward and uses a message box to display application information.

**Task 8:** Code **Help-About** menu.

**Private Sub AboutToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles AboutToolStripMenuItem.Click**

**Dim MessageString As String = "Version 1 of the Book Store Project" & ControlChars.NewLine & "Today's date/time: " & Date.Now**

**Dim TitleString As String = "About Version 1"**

**MessageBox.Show(MessageString, TitleString, MessageBoxButtons.OK, MessageBoxIcon.Information)**

**End Sub**

**Exit the Form**

The click event sub procedure for the **File-Exit** menu item is given here:

**Task 8:** Code **File-Exit** menu.

**Private Sub ExitToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ExitToolStripMenuItem.Click**

**'Close the form**

**Me.Close()**

**End Sub**

The remaining menus for the project are coded when you develop Version 2 of the project.

**Version 2 of the Bookstore Application**

**Using ComboBoxes with Arrays**

When the list of items to be stored in an array is small, such as with the bookstore product application, it can be efficient to let the system user select items from a **ComboBox** – a list type control can list all valid product ID codes or it can list items by description.

Advantages include the coding and processing of data is simplified:

* The current value of the **SelectedIndex** property of a ListBox or ComboBox can be used to as the row index to access other properties of a structure.
* Use of the **SelectedIndex** property as an index for an item selected from a list can be used to replace complex lookup code.

You will now modify the VB University Bookstore application to incorporate use of a ComboBox.

**Start New Project**

**Task 1:** Start a new project.

* Name the project **Ch08VBUniversity-Version2**.
* Delete the **Form1** form that is added to the new project by right-clicking **Form1.vb** in the Solution Explorer window.
* Add the form from the **Ch08VBUniversity-Version 1** project:
  + Select the **Project – Add Existing Item** menu.
  + Browse to the Version 1 project folder.
  + Select the **BookStore1.vb** file and click the **Add** button (this will automatically also copy the **BookStore1.designer.vb** and **BookStore1.resx** files – these files can be listed in the Solution Explorer by clicking the Show All Files button and expanding the BookStore1.vb filename).
* Rename **BookStore1.vb** to be **BookStore2.vb** in the Solution Explorer or Properties window.
* In Solution Explorer, double-click **My Project** – set the Startup form drop-down to **BookStore2**.
* Change the title bar (**Text** property) to update the version to **Version 2**.

**Redesign Interface**

**Task 2:** Redesign the interface.

* Replace the **ProductIDTextBox** control with a combo box named **ProductIDComboBox**.
* Set the **DropDownStyle** property = **DropDownList**.
* Add the product ID values to the Items collection of the **ProductIDComboBox** as shown in this figure. The Product ID values are: **A402**, **A804**, **C344**, **F554**, **G302**, **M302**, **S499**, and **X599**.
* Delete both the **Search** button and the **Product-Search** menu item.
* Check the form’s Tab Order and correct as necessary.

**Coding the Project**

**Task 3:** Code the **ProductIDComboBox** **SelectedIndex\_Changed** event sub procedure.

* Delete the **Click** event sub procedures for the **Search** button and **Product-Search** menu item.
* Code the **SelectedIndex\_Changed** event for the **ProductIDComboBox** (see below).
  + Note that the need for a search loop procedure is eliminated because the application user uses the combo box to perform the search.
  + The **SelectedIndex** value of the ComboBox control (highlighted) represents the product row within the **InventoryProduct** structure array.

**Private Sub ProductIDComboBox\_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ProductIDComboBox.SelectedIndexChanged**

**'Store the selectedIndex to variable**

**Dim RowInteger As Integer = ProductIDComboBox.SelectedIndex**

**'Based on RowInteger, display values to TextBox controls**

**'from the array named inventoryProduct**

**DescriptionTextBox.Text = InventoryProduct(RowInteger).DescriptionString**

**QuantityTextBox.Text = InventoryProduct(RowInteger).QuantityInteger.ToString("N0")**

**PriceTextBox.Text = InventoryProduct(RowInteger).PriceDecimal.ToString("C2")**

**End Sub**

**Task 4:** Modify the **Product-Reset** menu item's Click event sub procedure as shown here. Code changes are highlighted in yellow.

**Private Sub ResetToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ResetToolStripMenuItem.Click**

**'Clear the ComboBox and all TextBox controls**

**ProductIDComboBox.Text = String.Empty**

**DescriptionTextBox.Clear()**

**PriceTextBox.Clear()**

**QuantityTextBox.Clear()**

**TotalDueTextBox.Clear()**

**'Clear the list box control**

**PurchaseListBox.Items.Clear()**

**'Reset the total due module-level variable to zero**

**TotalDueDecimal = 0**

**'Set the focus to the ProductIDComboBox**

**ProductIDComboBox.Focus()**

**End Sub**

**Task 5:** Modify the code in the **Product-Purchase** menu item's Click event sub procedure due to the replacement of the **ProductIDTextBox** with the **ProductIDComboBox** control as highlighted in yellow below.

**Private Sub PurchaseToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles PurchaseToolStripMenuItem.Click**

**'Test to determine if a product was found.**

**If DescriptionTextBox.Text = String.Empty Then**

**'Cannot purchase, product was not found**

**MessageBox.Show("You must select a valid product before purchasing.", "Cannot Purchase", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)**

**ProductIDComboBox.Focus()**

**'Remark out the next line**

**'ProductIDTextBox.SelectAll()**

**Else**

**'Can purchase the product**

**'Build a string to display in the listbox control**

**Dim ProductString As String = ProductIDComboBox.Text & " - " & DescriptionTextBox.Text & " - " & PriceTextBox.Text**

**PurchaseListBox.Items.Add(ProductString)**

**'Accumulate the total value of this customer order**

**'and display it to the output textbox**

**TotalDueDecimal += Decimal.Parse(PriceTextBox.Text, Globalization.NumberStyles.Currency)**

**TotalDueTextBox.Text = TotalDueDecimal.ToString("C2")**

**'Here you can clear the form of product info if you think**

**'that is a good way to do the processing**

**ProductIDComboBox.Text = String.Empty**

**DescriptionTextBox.Clear()**

**PriceTextBox.Clear()**

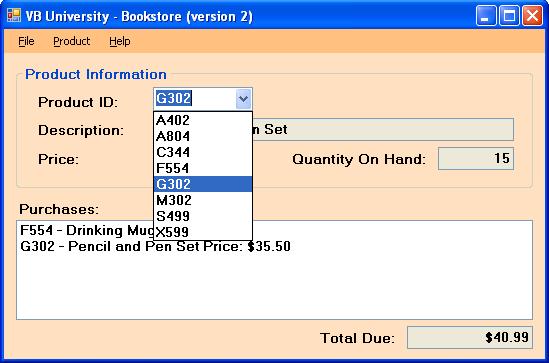
**QuantityTextBox.Clear()**

**ProductIDComboBox.Focus()**

**End If**

**End Sub**

Test the system as before. Ensure that the total due for the purchase of bookstore products accumulates properly. The figure shown below illustrates execution of the revised application.



**Using Array Elements for Accumulators**

You can use arrays as **accumulators**, for example, accumulating the total dollar sales by product for the VB University Bookstore.

**In-Class Exercise**

**Task 6:** Declare an accumulating array.

* A module-level accumulating array can be declared that corresponds to the **InventoryProduct** structure array.

**'Array to store the total sales for each product**

**Private ProductSalesTotalDecimal(NumberProductsInteger) As Decimal**

**Task 7** Add code to the **Product-Purchase** menu item Click event sub procedure.

* As a purchase is made, the value of the purchase is added to the corresponding position within the **ProductSalesTotalDecimal** array.
* This is coded inside the Click event sub procedure for the **Product-Purchase** menu item. The revised sub procedure is shown here with code modifications highlighted in yellow.

**Private Sub PurchaseToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles PurchaseToolStripMenuItem.Click**

**'Test to determine if a product was found.**

**If DescriptionTextBox.Text = String.Empty Then**

**'Cannot purchase, product was not found**

**MessageBox.Show("You must select a valid product before purchasing.", "Cannot Purchase", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)**

**ProductIDComboBox.Focus()**

**ProductIDComboBox.SelectAll()**

**Else**

**'Can purchase the product**

**'Build a string to display in the listbox control**

**Dim ProductString As String = ProductIDComboBox.Text & " - " & DescriptionTextBox.Text & " - " & PriceTextBox.Text**

**PurchaseListBox.Items.Add(ProductString)**

**'Accumulate the total value of this customer order**

**'and display it to the output textbox**

**TotalDueDecimal += Decimal.Parse(PriceTextBox.Text, Globalization.NumberStyles.Currency)**

**TotalDueTextBox.Text = TotalDueDecimal.ToString("C2")**

**'Accumulate total sales by product to an array**

**Dim IndexInteger As Integer = ProductIDComboBox.SelectedIndex**

**ProductSalesTotalDecimal(IndexInteger) += Decimal.Parse(PriceTextBox.Text, Globalization.NumberStyles.Currency)**

**'Here you can clear the form of product info if you think**

**'that is a good way to do the processing**

**ProductIDComboBox.Text = String.Empty**

**DescriptionTextBox.Clear()**

**PriceTextBox.Clear()**

**QuantityTextBox.Clear()**

**ProductIDComboBox.Focus()**

**End If**

**End Sub**

**Task 8:** Code the **File–Total Sales by Product** menu Click event sub procedure.

* Display the total sales by product to the **Visual Basic immediate output window** as a simple means of producing a report.
* This is coded for the **File-Total Sales by Product** menu item.
  + A **For...Next** loop processes both the array and the structure.
  + A string variable (**SalesString**) is used to build the string value to be displayed to the immediate window – the string variable stores the product ID from the structure and corresponding dollar sales value from the accumulating array.

**Private Sub TotalSalesByProductToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles TotalSalesByProductToolStripMenuItem.Click**

**'Display output to immediate window**

**Dim RowInteger As Integer**

**For RowInteger = 0 To NumberProductsInteger**

**'Build string to display**

**Dim SalesString As String = "Product ID: " & InventoryProduct(RowInteger).ProductIDString & " Dollar Sales: " & ProductSalesTotalDecimal(RowInteger).ToString("C2")**

**'Display string to immediate window**

**Debug.WriteLine(SalesString)**

**Next**

**End Sub**

Test the project:

* Add several products and ensure the total for an individual is correct.
* Display output to the Immediate window to check that the accumulated total sales by product is correct for each product.

**Multidimensional Arrays**

A multidimensional array has more than one dimension – we will work with arrays that have both rows and columns to produce a **two-dimensional** table.

An example of an application that uses a two-dimensional table is the use of tax tables for computing personal income taxes to be paid to the U.S. government.

This declaration statement allocates memory for **TaxTableDecimal** as a module-level array assuming there are 150 different income level brackets:

**Private TaxTableDecimal(150,3) As Decimal**

* The **rows** represent the income levels.
* The **columns** represent tax classifications such as single, married, and head of household).
* Remember, numeric arrays are automatically initialized to zero.
* When referencing the **TaxTableDecimal**, the first subscript represents the **row**. The second subscript represents the **column**.
* To display the value of row 20, column 2 to a label, the code is:

**TaxAmountTextBox.Text = TaxTableDecimal(19,1).ToString("C")**

One limitation of the two-dimensional table is that you can only store one type of data in the array – you can work around this to a certain extent by storing data as String that would otherwise be numeric and using Convert methods to convert data from string to Decimal or Integer or another numeric data type to support processing.

**A Two-Dimensional String Table**

This figure shows a table named **StatesString** that stores the names of states of the United States as string data. States are classified in columns according to their land mass size as **Large**, **Middle**, and **Small** sized.

The **StatesString** table is declared as a two-dimensional table and filled with state names as follows:

**Dim StatesString(,) As String = { \_**

**{"Alaska", "Indiana", "R. Island"}, \_**

**{"Texas", "Maryland", "Delaware"}, \_**

**{"California", "Idaho", "Vermont"}, \_**

**{"Florida", "Iowa", "Hawaii"}}**

Note the use of braces, commas, and line-continuation characters.

* The first left brace marks the beginning of the table.
* The next left brace marks the beginning of the first row of data.
* Each element is stored as string within double-quote marks.
* The first right brace marks the end of the first row of data.
* The last right brace marks the end of the table.
* Each element within a row is separated by a comma.
* Each row is separated by a comma from the next row.
* The line-continuation characters make it easier to read the table as each row is on a single line. The table data could be typed as one continuous line of code, but it would be very difficult and messy to read.

If the value **RowInteger** = **2** and **ColumnInteger** = **1**, the state "**Idaho**" is displayed to the text box named **StateTextBox**.

**StateTextBox.Text = StatesString(RowInteger, ColumnInteger)**

This sub procedure illustrates printing the **StatesString** table to the immediate output window for purposes to debugging code with nested **For...Next** loops. Each state will display on a **single** line.

**Private Sub StatesButton\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles statesButton.Click**

**Dim StatesString(,) As String = { \_**

**{"Alaska", "Indiana", "R. Island"}, \_**

**{"Texas", "Maryland", "Delaware"}, \_**

**{"California", "Idaho", "Vermont"}, \_**

**{"Florida", "Iowa", "Hawaii"}}**

**Dim RowInteger As Integer = 0**

**Dim ColumnInteger As Integer = 0**

**'Print to the Immediate Window as an output location**

**For RowInteger = 0 To 3**

**For ColumnInteger = 0 To 2**

**Debug.WriteLine(StatesString(RowInteger, ColumnInteger))**

**Next**

**Next**

**End Sub**

**Lookup Search for Two-Dimensional Tables**

There are several techniques for searching in two dimensions.

1. One approach has the application user enter values into two different TextBoxes.

* One TextBox would represent the **row** index; the other would represent the **column** index.
* If you use this approach, then you must ensure that the values entered for the row and column do not exceed the limitations of the array.
* If the row and column index values are valid, you can use them to directly reference a cell in the array, for example:

**TaxTextBox.Text = TaxTableDecimal(RowInteger, ColumnInteger)**

2. A second approach uses ComboBox controls.

* Here the **SelectedIndex** property of one ComboBox represents the **row** index (income level) and the **SelectedIndex** property of the second ComboBox represents the **column** index (tax classification status).
* Assuming that both ComboBoxes have a selection, the reference for an array cell is:

**Dim RowInteger As Integer = IncomeComboBox.SelectedIndex**

**Dim ColumnInteger As Integer = StatusComboBox.SelectedIndex**

**TaxTextBox.Text = TaxTableDecimal(RowInteger, ColumnInteger)**

3. Another approach requires the application user to search for a value to be found in the array.

* This approach requires coding a nested **For...Next** search or a **Do…Loop** search similar to those techniques shown above.
* With this approach, the application system user enters a value in a TextBox, then a code search procedure searches a specific column in an array to try to find the value that was typed into the TextBox.  If the search succeeds, the code returns a value from a different column of the same array or from column in a different array.

The number of ways that arrays are used is practically limitless, and your ability to manipulate them will improve with practice over time.

**Version 3 of the Bookstore Application**

In this version of the Bookstore Application, a two-dimensional, module-level, string array stores the data – the Product structure is removed from the project.

* Each row represents a product record and each column represents a product field (attribute).
* As with version 1 of the program, application users enter the desired product ID value into a **ProductIDTextBox** control and click the Search button.
* Search procedure code in the Search button control's click event searches the column that stores the product ID value for the two-dimensional array (**column 0** in this case, but it could be any column in a two-dimensional array) for the product ID value entered into the **ProductIDTextBox** control.
* If a match is found, the product's description, quantity on hand, and price will be displayed in the other TextBox controls; otherwise, an error message is displayed.

**Start New Project**

**Task 1:** Start a new project.

* Name the project **Ch08VBUniversity-Version3**.
* Delete the **Form1** form that is added to the new project by right-clicking **Form1.vb** in the Solution Explorer window.
* Add the form from **Ch08VBUniversity-Version1** as follows:
  + Select the **Project – Add Existing Item** menu.
  + Browse to the version 1 project folder.
  + Select the **BookStore1.vb** file and click the **Add** button (this will automatically also copy the **BookStore1.designer.vb** and **BookStore1.resx** files – these files can be listed in the Solution Explorer by clicking the Show All Files button and expanding the BookStore1.vb filename).
* Rename **BookStore1.vb** to be **BookStore3.vb** in the Solution Explorer or Properties window.
* In Solution Explorer, double-click **My Project** – set the Startup form drop-down to **BookStore3**.
* Change the title bar (**Text** property) to update the version to **Version 3**.

**Redesign Interface**

**Task 2:** Redesign the application interface.

* Delete the declarations of the **Product** structure and **InventoryProduct** array.
* Delete the **Load** event.

**Coding the Project**

**Task 3:** Modify the module-level declarations to add an array to store inventory data.

* Create a two-dimensional, module-level string array named **InventoryString** to store the product data as shown below.
  + Do **NOT** enter values for the number of rows and columns in declaring the array – VB will allocate memory properly at runtime.
  + The beginning and end of the data stored to the array are enclosed with a set of **braces** **{ }**.
  + Pay close attention to the punctuation entries of commas, braces, and line-continuation characters (**underscores** with a blank space just before each underscore).
* Add the declaration for the **ProductSalesTotalDecimal** array to store total product sales that was used in Version 2 of the application as shown below.
* Declare two module-level variables named **RowInteger** and **ColumnInteger** to store the row and column values for the search procedure as shown below.
* The module-level variables/constants/arrays declarations should now look like this:

**'Module-level declarations**

**Private NumberProductsInteger As Integer = 7 'number of products**

**Private TotalDueDecimal As Decimal 'total due for a customer**

**'Array to store the total sales for each product**

**Private ProductSalesTotalDecimal(NumberProductsInteger) As Decimal**

**'Module-level variable to store row and column values**

**Private RowInteger, ColumnInteger As Integer**

**'Declare two-dimensional inventory array**

**Private InventoryString(,) As String = { \_**

**{"A402", "History of America Textbook", "2", "$65.55"}, \_**

**{"A804", "College Logo Tshirt", "15", "$18.99"}, \_**

**{"C344", "College Logo Sweat Pants", "25", "$25.99"}, \_**

**{"F554", "Drinking Mug", "8", "$5.49"}, \_**

**{"G302", "Pencil and Pen Set", "15", "$35.50"}, \_**

**{"M302", "College Logo Sweat Shirt", "25", "$22.99"}, \_**

**{"S499", "Intro to Philosophy Textbook", "50", "$85.00"}, \_**

**{"X599", "Intro to CMIS Textbook", "75", "$79.40"}}**

**Task 4:** Modify the **SearchButton** control’s Click event sub procedure.

* Rewrite the code as follows:
  + Column **0** (contains product ID) is searched by varying the **RowInteger** variable from **0** to **NumberProductsInteger**.
  + The search stops when a match is found or the search runs out of rows to examine.
  + When a match between the value in **ProductIDTextBox .Text** and the **InventoryID** in column **0** of the array occurs, the other TextBox controls display values from the same row, columns **1**, **2**, and **3**, respectively.

**Private Sub SearchButton\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SearchButton.Click**

**'Search the ProductIDString property of the InventoryString**

**'array to see if the value of ProductIDTextBox matches an ID**

**'in the array**

**'Start variables to control the search**

**Dim FoundBoolean As Boolean = False 'Control how long to search**

**RowInteger = 0 'Current row in the search**

**ColumnInteger = 0 'Search column zero**

**'Loop to do the search**

**Do Until FoundBoolean = True Or RowInteger > NumberProductsInteger**

**'Compare TextBox to array**

**If ProductIDTextBox.Text = InventoryString(RowInteger, ColumnInteger) Then**

**'found a match - display data to the readonly TextBoxes**

**DescriptionTextBox.Text = InventoryString(RowInteger, 1)**

**QuantityTextBox.Text = InventoryString(RowInteger, 2)**

**PriceTextBox.Text = InventoryString(RowInteger, 3)**

**'change variable to indicate we have a match**

**FoundBoolean = True**

**Else**

**'no match yet**

**RowInteger += 1**

**End If**

**Loop**

**'After the search determine if the ProductID was found**

**If FoundBoolean = False Then 'no match was found**

**'Clear the textbox controls that display product information**

**'except for the ProductIDTextBox**

**DescriptionTextBox.Clear()**

**QuantityTextBox.Clear()**

**PriceTextBox.Clear()**

**'Display message that the ProductID is not valid**

**MessageBox.Show("Reenter a valid product ID.", "Invalid Identifier", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)**

**ProductIDTextBox.Focus()**

**ProductIDTextBox.SelectAll()**

**Else**

**If Convert.ToInt32(QuantityTextBox.Text) = 0 Then**

**PurchaseToolStripMenuItem.Enabled = False**

**Else**

**PurchaseToolStripMenuItem.Enabled = True**

**End If**

**End If**

**End Sub**

**Task 5:** Modify the **Product-Purchase** menu item's Click event sub procedure.

The procedure must accumulate the total sales by referencing the **RowInteger** variable that stores the current row from the search procedure. Modifications are highlighted in yellow.

**Private Sub PurchaseToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles PurchaseToolStripMenuItem.Click**

**'Test to determine if a product was found.**

**If DescriptionTextBox.Text = String.Empty Then**

**'Cannot purchase, product was not found**

**MessageBox.Show("You must select a valid product before purchasing.", "Cannot Purchase", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)**

**ProductIDTextBox.Focus()**

**ProductIDTextBox.SelectAll()**

**Else**

**'Can purchase the product**

**'Build a string to display in the ListBox control**

**Dim ProductString As String = ProductIDTextBox.Text & " - " & DescriptionTextBox.Text & " - " & PriceTextBox.Text**

**PurchaseListBox.Items.Add(ProductString)**

**'Accumulate the total value of this customer order**

**'and display it to the output TextBox**

**TotalDueDecimal += Decimal.Parse(PriceTextBox.Text, Globalization.NumberStyles.Currency)**

**TotalDueTextBox.Text = TotalDueDecimal.ToString("C2")**

**'Accumulate total sales by product**

**ProductSalesTotalDecimal(RowInteger) += Decimal.Parse(PriceTextBox.Text, Globalization.NumberStyles.Currency)**

**'Here you can clear the form of product info if you think**

**'that is a good way to do the processing**

**ProductIDTextBox.Clear()**

**DescriptionTextBox.Clear()**

**PriceTextBox.Clear()**

**QuantityTextBox.Clear()**

**ProductIDTextBox.Focus()**

**End If**

**End Sub**

**Task 6:** Add the **File-Total Sales by Product** menu item's Click event sub procedure (from Version 2) as shown here.

* The product ID value is now referenced from the **InventoryString** two-dimensional array by using **RowInteger** to be the row subscript and zero (0) to be the column subscript. Modifications are highlighted in yellow.

**Private Sub TotalSalesByProductToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles TotalSalesByProductToolStripMenuItem.Click**

**'Use For Next loop to process array and structure**

**Dim RowInteger As Integer**

**For RowInteger = 0 To NumberProductsInteger**

**'Store values to string variable to be added**

**'to the ListBox control**

**Dim SalesString As String = "Product ID: " & InventoryString(RowInteger, 0) & " Dollar Sales: " & ProductSalesTotalDecimal(RowInteger).ToString("C")**

**'Display report to the Immediate Window**

**Debug.WriteLine(SalesString)**

**Next**

**End Sub**

Test the project:

* It should perform exactly as Version 1 – the outer interface looks identical despite the coding modifications and use of the array.

**Possible Project Extensions – Reducing Quantity On Hand**

The project can be enhanced if time allows:

* Add an **About form** to the project and display it from the **Help-About** menu item.
* Update the amount of inventory in the array by reducing the **quantity on hand** value displayed by one whenever a purchase is made.

Reducing the quantity on hand is a relatively straight-forward task. You must decrease the value of the quantity column of the **InventoryString** array each time a purchase is made. However, if the inventory level is zero, then a purchase cannot be made.

The modified **PurchaseToolStripMenuItem\_Click** event sub procedure is shown here.

* A nested **If** statement is added to the **Else** processing branch of the outer If statement. This inner **If** tests the value of the quantity that is stored at location **RowInteger**, **Column 2** to determine if the quantity is greater than **zero** (highlighted in yellow).
* If the quantity is greater than zero, the value is stored from the array to **QuantityInteger** converting the quantity from string to integer. At the same time the quantity is reduced by subtracting **1** from the value.
* The value from **QuantityInteger** is then converted back to string and stored to the array.
* The **Else** branch of the inner **If** statement displays a message box with a message that the product is out of stock.

**Private Sub PurchaseToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles PurchaseToolStripMenuItem.Click**

**'Test to determine if a product was found.**

**If DescriptionTextBox.Text = String.Empty Then**

**'Cannot purchase, product was not found**

**MessageBox.Show("You must select a valid product before purchasing.", "Cannot Purchase", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)**

**ProductIDTextBox.Focus()**

**ProductIDTextBox.SelectAll()**

**Else**

**If Convert.ToInt32(Me.InventoryString(RowInteger, 2)) > 0I Then**

**'Subtract 1 from quantity**

**Dim QuantityInteger As Integer = Convert.ToInt32(Me.InventoryString(RowInteger, 2)) - 1I**

**'Store the quantity back to the array**

**InventoryString(RowInteger, 2) = QuantityInteger.ToString**

**'Can purchase the product**

**'Build a string to display in the ListBox control**

**Dim ProductString As String = ProductIDTextBox.Text & " - " & DescriptionTextBox.Text & " - " & PriceTextBox.Text**

**PurchaseListBox.Items.Add(ProductString)**

**'Accumulate the total value of this customer order**

**'and display it to the output TextBox**

**TotalDueDecimal += Decimal.Parse(PriceTextBox.Text, Globalization.NumberStyles.Currency)**

**TotalDueTextBox.Text = TotalDueDecimal.ToString("C2")**

**'Accumulate total sales by product**

**ProductSalesTotalDecimal(RowInteger) += Decimal.Parse(PriceTextBox.Text, Globalization.NumberStyles.Currency)**

**'Here you can clear the form of product info if you think**

**'that is a good way to do the processing**

**ProductIDTextBox.Clear()**

**DescriptionTextBox.Clear()**

**PriceTextBox.Clear()**

**QuantityTextBox.Clear()**

**ProductIDTextBox.Focus()**

**Else**

**MessageBox.Show("Ask for a raincheck, we are out of that product.", "Out of Stock", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)**

**End If**

**End If**

**End Sub**

Test the application to confirm that the quantity decreases as purchases are made.

**Solutions to In-Class Exercises**

This solution shows the Ch08VBUniversity-Version1 code for the Bookstore1.vb form.

'Project: Ch08VBUniversity-Version1

'D. Bock

'Today's Date

Option Strict On

Public Class BookStore1

'Declare a Product structre

Structure Product

Dim ProductIDString As String

Dim DescriptionString As String

Dim QuantityInteger As Integer

Dim PriceDecimal As Decimal

End Structure

'Module-level declarations

Private NumberProductsInteger As Integer = 7 'number of products

'Array of type Product

Private InventoryProduct(NumberProductsInteger) As Product

Private TotalDueDecimal As Decimal 'total amount for individual customer

Private Sub BookStore1\_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load

InventoryProduct(0).ProductIDString = "A402"

InventoryProduct(0).DescriptionString = "History of America Textbook"

InventoryProduct(0).QuantityInteger = 10

InventoryProduct(0).PriceDecimal = 65.55D

InventoryProduct(1).ProductIDString = "A804"

InventoryProduct(1).DescriptionString = "College Logo Tshirt"

InventoryProduct(1).QuantityInteger = 15

InventoryProduct(1).PriceDecimal = 18.99D

InventoryProduct(2).ProductIDString = "C344"

InventoryProduct(2).DescriptionString = "College Logo Sweat Pants"

InventoryProduct(2).QuantityInteger = 25

InventoryProduct(2).PriceDecimal = 25.99D

InventoryProduct(3).ProductIDString = "F554"

InventoryProduct(3).DescriptionString = "Drinking Mug"

InventoryProduct(3).QuantityInteger = 8

InventoryProduct(3).PriceDecimal = 5.49D

InventoryProduct(4).ProductIDString = "G302"

InventoryProduct(4).DescriptionString = "Pencil and Pen Set"

InventoryProduct(4).QuantityInteger = 15

InventoryProduct(4).PriceDecimal = 35.5D

InventoryProduct(5).ProductIDString = "M302"

InventoryProduct(5).DescriptionString = "College Logo Sweat Shirt"

InventoryProduct(5).QuantityInteger = 25

InventoryProduct(5).PriceDecimal = 22.99D

InventoryProduct(6).ProductIDString = "S499"

InventoryProduct(6).DescriptionString = "Intro to Philosophy Textbook"

InventoryProduct(6).QuantityInteger = 50

InventoryProduct(6).PriceDecimal = 85D

InventoryProduct(7).ProductIDString = "X599"

InventoryProduct(7).DescriptionString = "Intro to CMIS Textbook"

InventoryProduct(7).QuantityInteger = 75

InventoryProduct(7).PriceDecimal = 79.4D

End Sub

Private Sub SearchButton\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SearchButton.Click

'Search the ProductIDString property of the inventoryProduct

'array to see if the value of ProductIDTextBox matches an ID

'in the array

'Start variables to control the search

Dim FoundBoolean As Boolean = False 'Control how long to search

Dim RowInteger As Integer = 0 'Current row in the search

'Loop to do the search

Do Until FoundBoolean = True Or RowInteger > NumberProductsInteger

'Compare textBox to array

If ProductIDTextBox.Text = InventoryProduct(RowInteger).ProductIDString Then

'found a match - display other data to the readonly textboxes

DescriptionTextBox.Text = InventoryProduct(RowInteger).DescriptionString

QuantityTextBox.Text = InventoryProduct(RowInteger).QuantityInteger.ToString

PriceTextBox.Text = InventoryProduct(RowInteger).PriceDecimal.ToString("C2")

'change variable to indicate we have a match

FoundBoolean = True

Else

'no match yet

RowInteger += 1

End If

Loop

'After the search determine if the ProductID was found

If FoundBoolean = False Then 'no match was found

'Clear the textbox controls that display product information

'except for the ProductID textbox

DescriptionTextBox.Clear()

QuantityTextBox.Clear()

PriceTextBox.Clear()

'Display message that the ProductID is not valid

MessageBox.Show("Reenter a valid product ID.", "Invalid Identifier", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)

ProductIDTextBox.Focus()

ProductIDTextBox.SelectAll()

End If

End Sub

Private Sub PurchaseToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles PurchaseToolStripMenuItem.Click

'Test to determine if a product was found.

If DescriptionTextBox.Text = String.Empty Then

'Cannot purchase, product was not found

MessageBox.Show("You must select a valid product before purchasing.", "Cannot Purchase", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)

ProductIDTextBox.Focus()

ProductIDTextBox.SelectAll()

Else

'Can purchase the product

'Build a string to display in the listbox control

Dim ProductString As String = ProductIDTextBox.Text & " - " & DescriptionTextBox.Text & " - " & PriceTextBox.Text

PurchaseListBox.Items.Add(ProductString)

'Accumulate the total value of this customer order

'and display it to the output textbox

TotalDueDecimal += Decimal.Parse(PriceTextBox.Text, Globalization.NumberStyles.Currency)

TotalDueTextBox.Text = TotalDueDecimal.ToString("C2")

'Here you can clear the form of product info if you think

'that is a good way to do the processing

ProductIDTextBox.Clear()

DescriptionTextBox.Clear()

PriceTextBox.Clear()

QuantityTextBox.Clear()

ProductIDTextBox.Focus()

End If

End Sub

Private Sub ResetToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ResetToolStripMenuItem.Click

'Clear all text box controls

ProductIDTextBox.Clear()

DescriptionTextBox.Clear()

PriceTextBox.Clear()

QuantityTextBox.Clear()

TotalDueTextBox.Clear()

'Clear the list box control

PurchaseListBox.Items.Clear()

'Reset the total due module-level variable to zero

TotalDueDecimal = 0

'Set the focus to the product ID text box

ProductIDTextBox.Focus()

End Sub

Private Sub ExitToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ExitToolStripMenuItem.Click

'Exit the form without asking to close

Me.Close()

End Sub

Private Sub SearchToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SearchToolStripMenuItem.Click

'Call the Click event for the Search button control

SearchButton.PerformClick()

End Sub

Private Sub AboutToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles AboutToolStripMenuItem.Click

Dim MessageString As String = "Version 1 of the Book Store Project" & ControlChars.NewLine & "Today's date/time: " & Date.Now

Dim TitleString As String = "About Version 1"

MessageBox.Show(MessageString, TitleString, MessageBoxButtons.OK, MessageBoxIcon.Information)

End Sub

End Class

This solution shows the Ch08VBUniversity-Version2 exercise code of the Bookstore2.vb form.

'Project: Ch08VBUniversity-Version2

'D. Bock

'Today's Date

Option Strict On

Public Class BookStore2

'Declare a Product structre

Structure Product

Dim ProductIDString As String

Dim DescriptionString As String

Dim QuantityInteger As Integer

Dim PriceDecimal As Decimal

End Structure

'Module-level declarations

Private NumberProductsInteger As Integer = 7 'number of products

'Array of type Product

Private InventoryProduct(NumberProductsInteger) As Product

Private TotalDueDecimal As Decimal 'total amount for individual customer

'Array to store the total sales for each product

Private ProductSalesTotalDecimal(NumberProductsInteger) As Decimal

Private Sub BookStore2\_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load

InventoryProduct(0).ProductIDString = "A402"

InventoryProduct(0).DescriptionString = "History of America Textbook"

InventoryProduct(0).QuantityInteger = 10

InventoryProduct(0).PriceDecimal = 65.55D

InventoryProduct(1).ProductIDString = "A804"

InventoryProduct(1).DescriptionString = "College Logo Tshirt"

InventoryProduct(1).QuantityInteger = 15

InventoryProduct(1).PriceDecimal = 18.99D

InventoryProduct(2).ProductIDString = "C344"

InventoryProduct(2).DescriptionString = "College Logo Sweat Pants"

InventoryProduct(2).QuantityInteger = 25

InventoryProduct(2).PriceDecimal = 25.99D

InventoryProduct(3).ProductIDString = "F554"

InventoryProduct(3).DescriptionString = "Drinking Mug"

InventoryProduct(3).QuantityInteger = 8

InventoryProduct(3).PriceDecimal = 5.49D

InventoryProduct(4).ProductIDString = "G302"

InventoryProduct(4).DescriptionString = "Pencil and Pen Set"

InventoryProduct(4).QuantityInteger = 15

InventoryProduct(4).PriceDecimal = 35.5D

InventoryProduct(5).ProductIDString = "M302"

InventoryProduct(5).DescriptionString = "College Logo Sweat Shirt"

InventoryProduct(5).QuantityInteger = 25

InventoryProduct(5).PriceDecimal = 22.99D

InventoryProduct(6).ProductIDString = "S499"

InventoryProduct(6).DescriptionString = "Intro to Philosophy Textbook"

InventoryProduct(6).QuantityInteger = 50

InventoryProduct(6).PriceDecimal = 85D

InventoryProduct(7).ProductIDString = "X599"

InventoryProduct(7).DescriptionString = "Intro to CMIS Textbook"

InventoryProduct(7).QuantityInteger = 75

InventoryProduct(7).PriceDecimal = 79.4D

End Sub

Private Sub PurchaseToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles PurchaseToolStripMenuItem.Click

'Test to determine if a product was found.

If DescriptionTextBox.Text = String.Empty Then

'Cannot purchase, product was not found

MessageBox.Show("You must select a valid product before purchasing.", "Cannot Purchase", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)

ProductIDComboBox.Focus()

ProductIDComboBox.SelectAll()

Else

'Can purchase the product

'Build a string to display in the listbox control

Dim ProductString As String = ProductIDComboBox.Text & " - " & DescriptionTextBox.Text & " - " & PriceTextBox.Text

PurchaseListBox.Items.Add(ProductString)

'Accumulate the total value of this customer order

'and display it to the output textbox

TotalDueDecimal += Decimal.Parse(PriceTextBox.Text, Globalization.NumberStyles.Currency)

TotalDueTextBox.Text = TotalDueDecimal.ToString("C2")

'Accumulate total sales by product to an array

Dim indexInteger As Integer = ProductIDComboBox.SelectedIndex

ProductSalesTotalDecimal(indexInteger) += Decimal.Parse(PriceTextBox.Text, Globalization.NumberStyles.Currency)

'Here you can clear the form of product info if you think

'that is a good way to do the processing

ProductIDComboBox.Text = String.Empty

DescriptionTextBox.Clear()

PriceTextBox.Clear()

QuantityTextBox.Clear()

ProductIDComboBox.Focus()

End If

End Sub

Private Sub ResetToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ResetToolStripMenuItem.Click

'Clear the ComboBox and all TextBox controls

ProductIDComboBox.Text = String.Empty

DescriptionTextBox.Clear()

PriceTextBox.Clear()

QuantityTextBox.Clear()

TotalDueTextBox.Clear()

'Clear the list box control

PurchaseListBox.Items.Clear()

'Reset the total due module-level variable to zero

TotalDueDecimal = 0

'Set the focus to the ProductIDComboBox

ProductIDComboBox.Focus()

End Sub

Private Sub ExitToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ExitToolStripMenuItem.Click

'Exit the form without asking to close

Me.Close()

End Sub

Private Sub ProductIDComboBox\_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ProductIDComboBox.SelectedIndexChanged

'Store the selectedIndex to variable

Dim RowInteger As Integer = ProductIDComboBox.SelectedIndex

'Based on RowInteger, display values to TextBox controls

'from the array named inventoryProduct

DescriptionTextBox.Text = InventoryProduct(RowInteger).DescriptionString

QuantityTextBox.Text = InventoryProduct(RowInteger).QuantityInteger.ToString("N0")

PriceTextBox.Text = InventoryProduct(RowInteger).PriceDecimal.ToString("C2")

End Sub

Private Sub TotalSalesByProductToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles TotalSalesByProductToolStripMenuItem.Click

'Display output to immediate window

Dim RowInteger As Integer

For RowInteger = 0 To NumberProductsInteger

'Build string to display

Dim SalesString As String = "Product ID: " & InventoryProduct(RowInteger).ProductIDString & " Dollar Sales: " & ProductSalesTotalDecimal(RowInteger).ToString("C2")

'Display string to immediate window

Debug.WriteLine(SalesString)

Next

End Sub

Private Sub AboutToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles AboutToolStripMenuItem.Click

Dim MessageString As String = "Version 1 of the Book Store Project" & ControlChars.NewLine & "Today's date/time: " & Date.Now

Dim TitleString As String = "About Version 1"

MessageBox.Show(MessageString, TitleString, MessageBoxButtons.OK, MessageBoxIcon.Information)

End Sub

End Class

This solution shows the Ch08VBUniversity-Version3 exercise code for the Bookstore3.vb form.

'Project: Ch08VBUniversity-Version3

'CS2

'Today's Date

Option Strict On

Public Class BookStore3

'Module-level declarations

Private NumberProductsInteger As Integer = 7 'number of products

Private TotalDueDecimal As Decimal 'total amount for individual customer

'Array to store the total sales for each product

Private ProductSalesTotalDecimal(NumberProductsInteger) As Decimal

'Module-level variable to store row and column values

Private RowInteger, ColumnInteger As Integer

'Declare two-dimensional inventory array

Private InventoryString(,) As String = { \_

{"A402", "History of America Textbook", "2", "$65.55"}, \_

{"A804", "College Logo Tshirt", "15", "$18.99"}, \_

{"C344", "College Logo Sweat Pants", "25", "$25.99"}, \_

{"F554", "Drinking Mug", "8", "$5.49"}, \_

{"G302", "Pencil and Pen Set", "15", "$35.50"}, \_

{"M302", "College Logo Sweat Shirt", "25", "$22.99"}, \_

{"S499", "Intro to Philosophy Textbook", "50", "$85.00"}, \_

{"X599", "Intro to CMIS Textbook", "75", "$79.40"}}

Private Sub SearchButton\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SearchButton.Click, SearchToolStripMenuItem.Click

'Search the ProductIDString property of the InventoryString

'array to see if the value of ProductIDTextBox matches an ID

'in the array

'Start variables to control the search

Dim FoundBoolean As Boolean = False 'Control how long to search

RowInteger = 0 'Current row in the search

ColumnInteger = 0 'Search column zero

'Loop to do the search

Do Until FoundBoolean = True Or RowInteger > NumberProductsInteger

'Compare TextBox to array

If ProductIDTextBox.Text = InventoryString(RowInteger, ColumnInteger) Then

'found a match - display data to the readonly TextBoxes

DescriptionTextBox.Text = InventoryString(RowInteger, 1)

QuantityTextBox.Text = InventoryString(RowInteger, 2)

PriceTextBox.Text = InventoryString(RowInteger, 3)

'change variable to indicate we have a match

FoundBoolean = True

Else

'no match yet

RowInteger += 1

End If

Loop

'After the search determine if the ProductID was found

If FoundBoolean = False Then 'no match was found

'Clear the textbox controls that display product information

'except for the ProductIDTextBox

DescriptionTextBox.Clear()

QuantityTextBox.Clear()

PriceTextBox.Clear()

'Display message that the ProductID is not valid

MessageBox.Show("Reenter a valid product ID.", "Invalid Identifier", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)

ProductIDTextBox.Focus()

ProductIDTextBox.SelectAll()

Else

If Convert.ToInt32(QuantityTextBox.Text) = 0 Then

PurchaseToolStripMenuItem.Enabled = False

Else

PurchaseToolStripMenuItem.Enabled = True

End If

End If

End Sub

Private Sub PurchaseToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles PurchaseToolStripMenuItem.Click

'Test to determine if a product was found.

If DescriptionTextBox.Text = String.Empty Then

'Cannot purchase, product was not found

MessageBox.Show("You must select a valid product before purchasing.", "Cannot Purchase", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)

ProductIDTextBox.Focus()

ProductIDTextBox.SelectAll()

Else

If Convert.ToInt32(Me.InventoryString(RowInteger, 2)) > 0I Then

'Subtract 1 from quantity

Dim QuantityInteger As Integer = Convert.ToInt32(Me.InventoryString(RowInteger, 2)) - 1I

'Store the quantity back to the array

InventoryString(RowInteger, 2) = QuantityInteger.ToString

'Can purchase the product

'Build a string to display in the ListBox control

Dim ProductString As String = ProductIDTextBox.Text & " - " & DescriptionTextBox.Text & " - " & PriceTextBox.Text

PurchaseListBox.Items.Add(ProductString)

'Accumulate the total value of this customer order

'and display it to the output TextBox

TotalDueDecimal += Decimal.Parse(PriceTextBox.Text, Globalization.NumberStyles.Currency)

TotalDueTextBox.Text = TotalDueDecimal.ToString("C2")

'Accumulate total sales by product

ProductSalesTotalDecimal(RowInteger) += Decimal.Parse(PriceTextBox.Text, Globalization.NumberStyles.Currency)

'Here you can clear the form of product info if you think

'that is a good way to do the processing

ProductIDTextBox.Clear()

DescriptionTextBox.Clear()

PriceTextBox.Clear()

QuantityTextBox.Clear()

ProductIDTextBox.Focus()

Else

MessageBox.Show("Ask for a raincheck, we are out of that product.", "Out of Stock", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)

End If

End If

End Sub

Private Sub ResetToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ResetToolStripMenuItem.Click

'Clear all TextBox controls

ProductIDTextBox.Clear()

DescriptionTextBox.Clear()

PriceTextBox.Clear()

QuantityTextBox.Clear()

TotalDueTextBox.Clear()

'Clear the ListBox control

PurchaseListBox.Items.Clear()

'Reset the total due module-level variable to zero

TotalDueDecimal = 0

'Set the focus to the ProductIDTextBox

ProductIDTextBox.Focus()

End Sub

Private Sub ExitToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ExitToolStripMenuItem.Click

'Exit the form without asking to close

Me.Close()

End Sub

Private Sub AboutToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles AboutToolStripMenuItem.Click

Dim MessageString As String = "Version 1 of the Book Store Project" & ControlChars.NewLine & "Today's date/time: " & Date.Now

Dim TitleString As String = "About Version 1"

MessageBox.Show(MessageString, TitleString, MessageBoxButtons.OK, MessageBoxIcon.Information)

End Sub

Private Sub TotalSalesByProductToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles TotalSalesByProductToolStripMenuItem.Click

'Use For Next loop to process array and structure

Dim RowInteger As Integer

For RowInteger = 0 To NumberProductsInteger

'Store values to string variable to be added

'to the ListBox control

Dim SalesString As String = "Product ID: " & InventoryString(RowInteger, 0) & " Dollar Sales: " & ProductSalesTotalDecimal(RowInteger).ToString("C")

'Display report to the Immediate Window

Debug.WriteLine(SalesString)

Next

End Sub

End Class