FIT 100

Lab 11: "What's Your Sign?" with Tables and Procedures

Spring 2002

Reading for Lab 11:

- Chapter 6 of *Computer Programming Fundamentals with Applications in Visual Basic 6.0*
- p. 265-269 (top of page) in Chapter 10 of *Computer Programming Fundamentals with Applications in Visual Basic 6.0*

Computer Programming Fundamentals with Applications in Visual Basic 6.0 is on reserve at Odegaard Undergraduate Library.

Introduction:
Today you will improve your SignFinder, so that is can report more precise errors to the user. For example, it can indicate that June 31st is an incorrect date. You will also work with data structures, specifically tables. In VB tables are called collections. Collections are a way to group together similar items that are important to us and to access them based on an index (number) or a key (string value). Collections are like the arrays you have read about in Chapter 13, but instead of using the index to access a value, a key is used instead.

Objectives:
- Continue to gain familiarity with VB controls and their properties
- To see how a table (Collection) can be populated and used in VB
- To see how a procedure is created and why it might be useful

1. Open Project and Declare a New Variable

A. Continue working with a copy of the code that you created in Lab 10. Copy the Lab10_yourname folder from your Dante account to the Desktop of the local machine.

B. **Rename** it Lab11_yourname. Save this newly named folder back to Dante at the end of lab.

C. Open the project-not the form!- in Visual Basic. Change the caption of the form to Sign Finder 2.

D. Declare a new variable, called **monthName**, of String data type. Add the declaration for this variable at the top of the Code window, under (General) and (Declarations). It should follow the variable declarations already there. This is a **global variable declaration.**

Simple Tables

In this lab we will improve our application by teaching it about the number of days in each month. For example, there are 31 days in January, 28 days in February, etc. How would you represent this information? A simple way to do so is to write a table like this:

Key:month	**Value:**numdays
January	31
February	28
March	31
April	30
May	31
June	30
July	31
August	31
September	30
October	31
November	30
December	31

The left column contains keys: the key determines which line of the table we are interested in. The right column contains values: this is the information we are looking for when given a specific key. For example, if we want to know the number of days in June, then June is our key. First we find the line in the table where the Key is June. Then we look in the Values column on that line to find the desired value: 30.

In this lab, we will keep our life simple. For example, we will deal with just one column for values. We will treat all Februaries as having 28 days. [PS: Different authors and programming environments use a variety of words for what we here call a table, and sometimes use the word table in other ways!]

A table like this is our first example of a "data structure": a way of grouping and organizing data. Data structures form an extremely important part of information technology, both from the theoretical and the practical standpoints.

The lifecycle of a table in Visual Basic consists of the following stages:

- create a table
- initialize it by storing values in it
- query it by looking up values for given keys as needed

We will go through each of these stages in turn.

2. Creating a Table

To create a table, you simply need to declare a variable for that table in a particular way. Let's call our table variable **daysInMonth**. Then the declaration will look like this:

```
Option Explicit
Dim daysInMonth As New Collection
```

The phrase "As New Collection" hints at the fact that a new object of the Collection class is being created. We will use the Collection class provided by Visual Basic to implement our table.

In VB, there are 4 methods that are available to the Collections object. Methods are actions that can be taken by an object. They are noted in VB syntax just as properties are: the name of the object, a dot, then the method.

What are the names of the 4 methods and what do they do?
HINT: Use the Object Browser: **View>Object Browser**
and search for Collections

A. Declare daysInMonth following the existing variable declarations at the top of the Code window, under (General) and (Declarations).

3. Initializing the Table

To initialize the table you need to add to it all the lines that you want to be there. Remember that each line consists of the key and the value. In general, a line is added to the table daysInMonth with the following code:

daysInMonth.Add <value>, <key>

*** ***Remember that the key appears last***, and the value first. ***

Querying the table

After all lines have been added to the table, you can query it by supplying the key you are interested in and asking for the corresponding value. In general, the following code queries the table daysInMonth:

daysInMonth.Item(<key>)

To use the result, assign it to a variable or use it in a conditional. For example:

daysInSeptember = daysInMonth.Item("September")

Now you will start initializing the table daysInMonth to indicate the correct number of days in a month. (You will query it later in this lab.)

Here is one way of doing so:
When the user clicks on a month option button, the line for that month will be added to the table. As a result, when the user clicks on the OK button, the table will already be initialized for the chosen month.

Go to the code that handles the click event for the January button. Add code to do the following:

A. Assign the value "January" to the variable **monthName** (we will need this later);

NOTE You aren't given the actual VB code that is needed. Test your understanding by working it out yourself, and writing it out on a piece of paper *before* trying it on the computer. Check it with someone, perhaps the person sitting next to you, before typing it in. (This technique has a name, by the way: "Peer code review." It is widely used in industry. Of course, on a project for credit this might not be appropriate!)

B. Add a line to **daysInMonth** with key "January" and value 31 (the number of days in January).

C. Do the same for February and March, using the correct day values.

D. Doesn't this sound like repetition? *Case for a procedure!* (But first, save your work!)

4. Procedures

The actions you performed above are a repetition of a general pattern. You will write a procedure to express this pattern. But let's start with some planning.

First, decide what the procedure will do: assign the name of the month to the variable **monthName**, and add a line to the **daysInMonth** table with that name as the key and the corresponding number of days as the value. Then, identify the 4 specifications of the procedure. What are they?

Name: a way to refer to the procedure. Make the name something descriptive of what the procedure does. Let's call it **setMonth.**

Parameters: the input and/or output variables used for this procedure (in our case there are no output variables):

```
(month As String, numDays As Integer)
```

Definition: the instructions or procedure body, performing the actions that we decided upon. For example:

```
monthName = month
daysInMonth.Add numDays, month
```

Declaration: put together the entire package: the procedure's name, parameters and definition are placed in the declaration. In Visual Basic it is written like this:

```
Private Sub setMonth (month As String, numDays As Integer)

    monthName = month
    daysInMonth.Add numDays, month

End Sub
```

A. Plan your own procedure similar to the description above. Choose concise and meaningful names for the procedure and its parameters.

B. Write the procedure declaration at the bottom of the Code window.

When you need to move to that area later, select General from the **Objects** drop-down menu and find the name of the procedure in the **Procedure** drop-down menu.

5. Resume initialization of the daysInMonth table

Calling a Procedure

Let's look back at what you have done. You started to initialize the table by adding lines for each month. Then you noticed that it was a rather repetitive business, so you wrote a procedure to make it easier. Now you should go back and complete table initialization – this time by invoking your procedure.

Recall that in order to call (or invoke) a procedure, you need to specify its name and actual parameters. You could write a procedure call in Visual Basic as follows, assuming it has two parameters:

Call setMonth (<first actual>, <second actual>)

For the procedure that you just wrote, what are the actual parameters going to be?

A. Go to the code that handles the click event for the April button. Add code to invoke your procedure (call it) with appropriate actual parameters. Repeat the same for the remaining months.

B. Now is good time to save your project again, before you start debugging it. Continue saving it periodically, once you get the next addition to it to work.

C. Run your program; click on a couple of month buttons. Then interrupt it by clicking on the **Pause** button. Now check the values of your variables. As you did before, go to the top of the Code window, move your mouse over the variables and verify their values. What should be the value of the **monthName** variable? Ensure that **monthName** actually has that value.

D. This trick does not work for **daysInMonth**. Instead, right-click on it and choose Add Watch; just click OK in the menu that pops up. The **Watch** window should occur at the bottom of Visual Basic with this variable in there. Click on the plus sign and you should see which values have been stored in it. (By the way, what should be values in this case?)

> **NOTE** The Watch window does not show you the keys, it only shows values. Also, do not be concerned with the order in which the values appear – the window shows them in the order they were added.

E. Save your project

6. Improve the check for the number of days

Remember where your program checks that the number of days the user enters is correct? How can we improve this check with the new variables?

A. In the Code window go to the click Event Handler (procedure) for the OK button. Instead of using the number 31 as the maximum allowed number of days in the conditional, insert a query to the **daysInMonth** table. The key for the lookup is the name of the month that the user selected. It is stored in the variable **monthName**.

B. Run your code again. Check it for entering reasonable ***and*** unreasonable dates. Does it work correctly?

C. Improve the message that is printed to the user in case the date is unreasonable. For that, create another error trap called **WrongDay** to handle this particular case, as follows. Leave the current ErrorHandler trap to handle other errors.

   a. At the bottom of this sub, insert the following code (shown in **bold**):

```
WrongDay:
        MsgBox  <you will insert your message here>
Exit Sub

ErrorHandler:
        MsgBox "Please enter a valid date and check a month",
vbOKOnly
        txtUserInput.SetFocus
End Sub
```

   b. In the line with the conditional that checks correctness of the day, replace "Then GoTo ErrorHandler" with "Then GoTo **WrongDay**".

   c. Finally, compose the message that the MsgBox will display. It would say something like "The day you entered is incorrect. There are only 30 days in April." adjusting, of course, to the month that the user selected.

D. Run your program again. Try to enter information that will "break" it. Can you do it?

> **NOTE** What happens if you click on an option button after you have already added the key and value for that month? Most likely you will get an error because the same key cannot be added twice to a collection. Keys must be unique.

E. Create an error trap inside of your setMonth procedure that will, upon error, remove an item from the code book and then allow the procedure to re-add it. You are not given the code-use your past labs and work with error trapping to help you construct the code.

In addition to removing an item from the collection in the error-trap, you will need to use one of the collection methods as well as the **Resume** statement**.** **Resume** simply directs the program to re-execute the procedure (setMonth). Since the error trap has removed the item from the collection, re-executing the procedure will allow your program to add it.

F. After creating the error trap, run your program. Are you able to do the following without error:
   - Look for the sign of a birthday in September
   - Then look for the sign of a birthday in March
   - Look for ANOTHER birthday in September

7. Creating an Executable File:

A. Make **SignFinder** executable when you are finished and happy with it. Save your project and transfer your Lab11 folder to your Dante account.

B. Remember to log off the machine as you leave.

8. Lab Notebook Questions

A. You want to create a table, say **monthNames**, that will hold the names of the months (i.e., the corresponding strings). You want to look up the names using numbers, from 1 to 12. For example, when you query the table with the number 5, you get "May". In your notebook, write VB code to:
   - Create the table,
   - Add the line for May to it,
   - Query it for that line.

B. Suggest a procedure that does some of the above. What parameters would it have?

C. A table is an example of a "data structure:" a way of organizing data. Despite the seeming simplicity, tables are an important data structures concept and can be very powerful in programming.
   - How could you view a telephone book as a table? What are the keys and values?
   - How about Webster's Dictionary? What are the keys and values?
   - Think of another real-world example of a table, and say what its keys and values are.

D. For Lab 13/14 (we will be working on one lab next week), read through the lab logic (not the details) and create and interface and pseudo-code for the program in the same way that the in-class

exercise had you do for conditionals. You will place, in your notebook:

- A sketch of the interface
- Pseudo-code that explains how the program will take user input at various points (you will want to name the objects and events used on your interface sketch) and produces a result: the cost of an ice cream cone.