Handout 2

**VB programming**

The user interface is the first point of contact with the application. A good programmer hides the complexities of the application behind a well-designed interface. In our first meeting we experimented with a few of the UI features. VB offers a wide collection of interface tools. A good UI is like a chassis/body of a Ferrari. It is the richness of the program (power of the engine) beneath the UI layer (under the hood) that eventually makes the application useful to the user.

The syntax of VB is very similar to many other programming languages. Writing efficient code therefore should follow structured programming principles. There are three major principles of structured programming:

- Sequence
- Branching
- Looping

Using these three principles we can write any program!

A program is a collection of instructions executed in sequence where each instruction may be of one of the three types. In addition to these three we also introduce the concept of a procedure or subroutine and function. One could view these as little, independent programs that may be called by other independent programs. We have seen how this works in our previous class. The code attached to a command button (for example, Private Sub cmdClear_Click () … End Sub) is an example of a program. In practice we consider a program to be a collection of these little programs.

Subroutine and functions perform work on the information stored in the computer or passed to it from some source (could be user input or program input). Variables are used to store and transport this information. These variables have to be defined unambiguously for the program to work well. That is, unlike us, the computer program needs to know precisely which variables will hold what type of information. Hence we declare ahead of time what variables to use and what type of information they can hold. Also, we might decide whether these variables are only visible to one procedure (local scope) or function or visible to everyone (global scope). In fact in VB a variable may be defined at the procedure (local) level, form level, module level, or global level! Unfortunately, in VB one could get by without declaring our variables. To ensure that we never do this, we can fix our settings.

**Session 1**

Step 1.

Start up VB if not already up and about. Select Options from the Tools menu. You should see the dialog shown below. If the Editor tab is not active select it. Check the Require Variable Declaration checkbox. Now every time you open a code window, the Option Explicit statement will be included in the declaration section. (Note that this will be true only for projects that you open from now onwards.)
Step 2.
Start a new project. Add two command buttons cmdCommand1 and cmdCommand2. Attach the following code.

```vba
Private Sub cmdCommand1_Click()
    Print "I am in Command1"
    c = 30
End Sub
Private Sub cmdCommand2_Click()
    Print "I am in Command2"
    Print "The value of c is "; c
End Sub
```

Now run the program. You should get an error message informing you that variable is not defined.

Step 3.
Continue working on the project. Go to the code window and select (General) from the Objects box. The term (declarations) appears in the procedures box. Type the line

```
Dim c As Integer
```

Run the program again. The number 30 is printed as the value of c in the second procedure. The variable c is available to both the procedures because it is designed at the form level.
Module and Global level Declarations

Step 1.
Continue with the current project. Delete the code you entered to declare c.

Step 2.
We are going to add another component to our project. Click on the Project menu and choose Add Module. Click on the Open button.

When you do, the code window opens up for this module. Study this window and compare with the Form window. In your Project Explorer you now have an additional item. Save this project.

Step 3.
In the code window for the module add the line

Private c As Integer

Within a module, Dim and Private mean the same. It is recommended that we use Private to ensure that the variable is not global in scope. Variables declared at the Module level are available to all forms and procedures within your project only if the declaration is made Public. Privately declared variables are available only if we omit the explicit declaration feature! The Module will be saved as a BAS file and is available to be used in any other project. Talk about reusability!

Step 4.
Run the program and experiment with different settings.
Step 5.
Continue with project. Add a third command button cmdCommand3 to your Form and add the Form2.Show line to the cmdCommand3_Click () procedure. The button click will show a second form on the screen.

Step 6.
Add a new Form window to the project. To do this select Add Form from the Project menu. You now have two Forms and a Module in your project. Add a command button cmdCommand1 to this new form and code its click procedure thus:

Private Sub cmdCommand1_Click ()
    Print "I am in Form 2"
    Print "The value of c is "; c
End Sub

Step 7.
Run your program. Change the settings and see what happens.

Session 2

Static variables
Visual Basic resets the value (erases) of the variables, irrespective of the level of their scope, once execution leaves the area where it is used. So, if you care about persistence of the value, as in a running total or a counter, you need a different kind of variable. A static variable persists its value between times of use. When execution returns to the static variable the value in the variable is visible to the program. To make a variable persistent, it must be declared static, as in

Static c As integer

Let us see some examples.

Step 1.
Start a new project. Double-click on the Form window, and select the Form_Click () procedure. Put the following code:

Private Sub Form_Click ()
    Dim Anormal
    Static Bstatic

    Anormal = Anormal + 1
    Bstatic = Bstatic + 1
    Cform = Cform + 1
End Sub
Step 2.
Now add Dim Cform in the declarations sections. Run the program repeatedly.

Step 3.
Make changes on your own and see the results. Currently the variables are integers. Declare some as strings.

Subroutines
A subroutine is a mini program. Do the Lesson 1 on page 162. This will get us started on our way at least.

The difference between a subroutine and a function is that a function returns a value via its name. A subroutine returns values through its parameters (simply speaking). In the following examples we see how to use subroutines and functions.

Step 1.
Continue with the Adder project. Open the code window. After the last line in the code window type

`Sub output (result As Variant)`

VB automatically puts in the End Sub and formats the statement you entered. We created a subroutine to display the result.

Step 2.
Change the code in cmdAdd_Click as shown on page 195. Run the program.

Step 3.
Create a function to do addition and another to check if the label caption for lblOutput has changed. Create the Function Sum as shown in Listing 4.24. Create the Function AlreadyAdded as in Listing 4.25.

Step 4.
Modify cmdAdd_Click as shown in Listing 4.26. Run the program.

Discuss the differences between functions and subroutines.

Session 3.
Additional controls
Accelerator Keys: When you type the Caption as &Exit for a command button cmdExit, the letter E will e underlined to indicate that you can use the Alt + E to enable the click.

Msg box functions and constants (pp.239-245).
The MaskEdit control. This control can be used to (1) format screen displays or printed outputs, (2) validate input data, or (3) update data field values. MaskEdit control is an example of ActiveX control.
In order to use an ActiveX control we need to select it from the list of components. The following steps will illustrate how to add an ActiveX control.

**Step 1.**
Select the Project menu and the Components. The Components dialog appears.

**Step 2.**
Select the desired ActiveX component by checking the box next to it. Select as many controls as you need. Click OK when done. The controls are now added to your toolbox and can be used just like any other controls.
Generally, the following ActiveX controls are used often.

**Calendar control**
NextDay, NextMonth, NextWeek, NextYear, PreviousDay, PreviousMonth, PreviousWeek, PreviousYear, Refresh, and Today are some of the common methods.

AfterUpdate, BeforeUpdate, Click, DblClick, KeyDown, KeyPress, KeyUp, NewMonth, and NewYear are some of the events associated with this control.

**CommonDialog control**
ShowSave, ShowColor, ShowFont, ShowPrinter, ShowHelp, ShowOpen are methods of this control.

**RichTextBox control**

**MSChart control**

**Multimedia control**

We want to build the following application now.

**Step 1.**
Add the text boxes, labels, icon, and horizontal lines as shown.
Step 2.
Create masks for each of the following items:

<table>
<thead>
<tr>
<th>EmployeeID</th>
<th>Social security number</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>Upper case state abbreviation</td>
</tr>
<tr>
<td>Zip code</td>
<td>Five numbers followed by four additional numbers</td>
</tr>
<tr>
<td>Home phone</td>
<td>(Area code), three numbers, a dash, and four additional numbers</td>
</tr>
<tr>
<td>Dept. Number</td>
<td>One uppercase letter and three numbers</td>
</tr>
<tr>
<td>Salary/Year</td>
<td>Format as currency, using $##,###.00</td>
</tr>
<tr>
<td>First Day</td>
<td>Format as date, using ##/###/##</td>
</tr>
<tr>
<td>Last Day</td>
<td>Format as date, using ##/###/##</td>
</tr>
</tbody>
</table>

Step 3.
Save the project. Run and try different data input.

**Putting it all together**

Step 1.
Start a new project. We will create an invoice interface. Add text boxes, labels, mask edit controls, and command buttons as shown.
Step 2.
Clicking the Extend button should call a procedure to calculate the extended price. The extended price is then displayed in the extended text box.

Step 3.
Clicking the Total button calls a subtotal procedure, tax calculation procedure, and the grand total procedure. Print will do nothing now. Quit will exit the program.

Save the project and run the program.