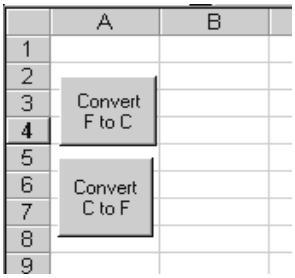
Example # 2 b) : Temperature Converter (MsgBox/ InputBox version)

The previous program is made more interactive by using **Input Boxes** to "input" the data and **Message Boxes** to "output" the calculated temperature. Save Exercise # 2a) under a new name. Delete the **Clear** button and associated code, and delete all titles on the worksheet. Remove any code not shown below; existing code is in *italics*, new code is in **bold**.




```
Option Explicit
 Dim F As Double, C As Double

Private Sub cmdCtoF_Click()
C = InputBox("Enter Temperature in Celsius", "Centigrade to Fahrenheit Conversion")
F = (9 / 5) * C + 32
MsgBox C & " degrees C is equivalent to " & F & " degrees F"
End Sub

Private Sub cmdFtoC_Click()
F = InputBox("Enter Temperature in Fahrenheit", "Fahrenheit to Celsius Conversion")
C = (5 / 9) * (F - 32)
MsgBox F & " degrees F is equivalent to " & C & " degrees C"
End Sub
```

Message Boxes and **Input Boxes** are more fully discussed in Chapter 2.

Message Boxes in general have 3 components: a title, a prompt, and a style. A simplified version has been used in this exercise which includes only the prompt. As will be noted in chapter 2, the style describes both the button type and any symbols added to emphasize the prompt (such as exclamation marks, question marks, critical warning, etc). The simplified version of the message box uses the default condition of presenting only an OK button.

The **Message Box** command in simplified form has the syntax **MsgBox *Prompt*** . Notice that the prompt is not contained inside parentheses nor is an equal sign present. [A **Message Box *Function*** (described in chapter 2) is available when it is necessary to return the choice that the user has made, such as Yes, No, Cancel, etc.]

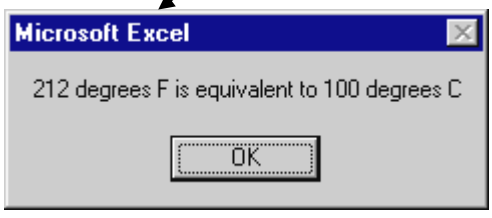
The prompt, and even the title, can contain variable values as demonstrated in this exercise.

MsgBox F & " degrees F is equivalent to " & C & " degrees C"

Prompt

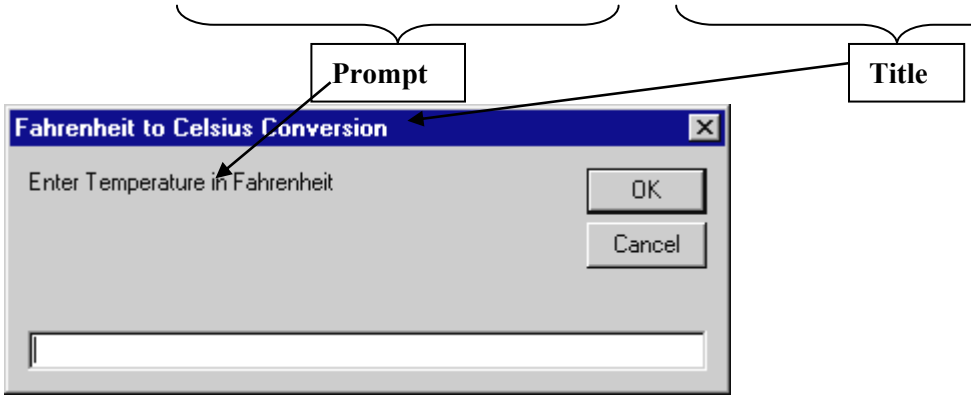
The variable values of **F** and **C** are linked by ampersands [&] to text that is entered inside quotes.

Default title.



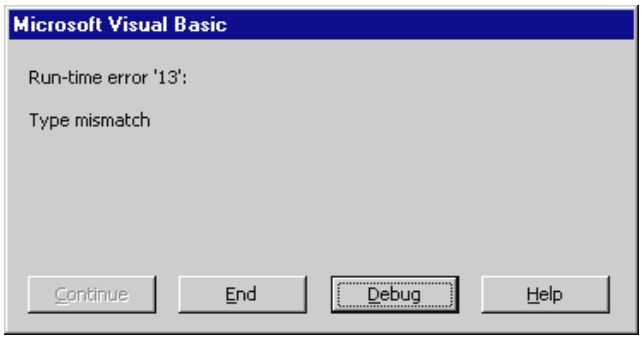
An **Input Box** has 4 components: a title, a prompt, two buttons, and the box where the actual entry is made. The **Input Box** ***function*** statement that is used to call up an **Input Box** has 2 terms in its argument : a Prompt and a Title; the general syntax is **InputBox(Prompt, Title)** where the prompt and title are strings.

F = InputBox("Enter Temperature in Fahrenheit", "Fahrenheit to Celsius Conversion")



If the prompt is long and extends over two or more lines it is convenient to store both the prompt and the title in strings which might appropriately be named **Prompt** and **Title** (refer to **Example # 1**, Chapter 2). Unlike the **Message Box** described next, the user has no control over the type or the number of buttons displayed.

Caution--Input Box Error **:** Since **F** and **C** were originally declared as numeric (**Dim**-med as **Double**) an error message indicating a type mismatch will occur if a non-numeric value is returned by the **Input Box**. This will happen if the user attempts to **Cancel** , or fails to enter a number and closes the **Input Box**, or mis-types and enters a non-numeric value.



Trapping Errors : Although the purpose of explicitly declaring variables is to identify problem values before they are converted into numerical equivalents, the consequence of this approach is to produce an error message that causes the program to "crash". This is not practical or desirable if the program is to be used by someone unable to deal with the crash. One solution is to program around, or trap, the error using an **If** statement (see section 1.6), the **IsNumeric()** function (which detects any non-numeric values), and the **End** command (which terminates execution). In order to bring in the values of **F** and **C** without causing a type mismatch they are declared as **Variants** so that they take on the data type of whatever has been entered.

The following modifications deal with non-numeric data without crashing the program :

```
Option Explicit
Dim F As Variant, C As Variant
Private Sub cmdCtoF_Click()
C = InputBox("Enter Temperature in Celsius", "Centigrade to Fahrenheit Conversion")
If IsNumeric(C) = False Then End
F = (9 / 5) * C + 32
MsgBox C & " degrees C is equivalent to " & F & " degrees F"
End Sub

Private Sub cmdFtoC_Click()
F = InputBox("Enter Temperature in Fahrenheit", "Fahrenheit to Celsius Conversion")
If IsNumeric(F) = False Then End
C = (5 / 9) * (F - 32)
MsgBox F & " degrees F is equivalent to " & C & " degrees C"
End Sub
```

Note the change in data type

IsNumeric() is a Boolean function that returns a **True** value if the argument is numeric, and **False** otherwise.

A more sophisticated modification involves including a message box to inform the user that an error has occurred just before execution is terminated :

```
Dim Prompt as String
.....
Prompt = "A non-numeric value was entered.  Execution will be terminated"
.....
If IsNumeric(C) = False Then MsgBox Prompt: End
```

A colon is used to separate individual statements contained on the same line.

When a lot of values are being read-in it is useful to identify the problem value in the message box.

Unless the program is intended for commercial purposes, the shortcoming here is that a lot of code would be required to check all the values being read-in -- possibly 30 % of the total code for our relatively small applications. Aside from being inconvenient in terms of the extra programming time, the additional code tends to make the program less readable. For the purposes of this course you are advised to "keep it simple" and let problems be identified by **Type mismatch** error messages.

1.6 "If" Statements

An **If** statement checks if a certain **condition** (based on comparison operators such as : = > <) is **True** or **False** and then executes code depending on the outcome. Two types of **If** statements exist :

a *single outcome* **If…Then** statement that executes only one set of code if the decision condition is **True**,

and a *double outcome* **If…Then…Else** that executes *different* code for the two possible **True** or **False** outcomes of the **If** decision condition.

Multiple outcomes can be created by combining (or "**nesting**") **If** statements.

1.6.1 Single Outcome *If* Statements

The single outcome **If** has the syntax : **If** ***condition*** **Then** ***code statement(s)***

All the code must appear on a single line after the **Then** . However, multiple code statements can be included in the single line provided they're separated by colons. In addition, the single line can be made as long as desired by "continuing" it on the line below provided space-underscore continuation characters are used.

The **condition** being tested must evaluate to a Boolean **True** or **False**. When the expression evaluates to **True**, the code to the right of the **Then** is executed .

Examples :

Decision condition

If time > 200 Then Thrust = 0: Drag_Coefficient = 0.05

Single outcome **If** statements must be written on a single line; however, more than one statement can be included in the outcome provided the individual statements are separated by colons.

If Elevator_Weight > 2000 Then Brake = True: Alarm = True

If (X ^ 3 - Y / 2) / Z > X ^ 2 Then X = 2*Y + Z: Worksheets("Sheet1").Range("A2") = X

Calculations can be made in the **If** condition, and many standard operations such as read, write, and **MsgBox** can be included in the outcome statement(s).

If Coefficient_Friction < 0 Then MsgBox "Check data !! Coeff. of Friction is negative"

A **continuation character** (a space followed by an underscore) allows a single line of code (consisting of 4 statements in this case) to be continued onto the next line. It is, however, still considered as a single line.

```
If Years_of_service >= 7 Then Max_Bonus = 15000 :  Paid_sick_days = 12 : _
                                  Vacation_days = 20 : Reserved_parking_space = True
```

An error message will be generated if a single outcome **If** statement is extended onto more than 1 line without a continuation character (that is, if the underscore were eliminated above and the colon retained, or if part of one of the 4 statements were continued on the next line).

Undetected error : A far more serious error occurs if both the colon and the continuation character are omitted. The last 2 statements regarding vacation days and parking would no longer be part of the **Then** outcome, and would always be executed (in this case, though, everyone would benefit !!)

1.6.2 Double Outcome *If* Statements

The double outcome **If** has the syntax :

```
If condition  Then
Code statements
Else
Code statements
End  If
```

The **Else** outcome code is optional .

```
If Years_of_service < 7 Then
    Max_Bonus = 5000
    Paid_sick_days = 8
    Vacation_days = 10
    Reserved_parking_space = False
Else
    Max_Bonus = 15000
    Paid_sick_days = 12
    Vacation_days = 20
    Reserved_parking_space = True
End If
```

Condition is **False**

If the condition **Years_of_service < 7** is **True** the **Then** outcome is selected. Control is transferred to the **End If** once the last statement is executed.

If the condition **Years_of_service < 7** is **False** the **Else** outcome is selected and the program jumps from the first line of the **If** to the **Else**, and finally to the **End If** once all statements have been executed.

This double outcome **If** statement could have been replaced by two single outcome **If**s :

If Years_of_service < 7 Then Max_Bonus = 5000: Paid_sick_days = 8: etc.

If Years_of_service >= 7 Then Max_Bonus = 15000: Paid_sick_days = 12: etc.

For the sake of compactness, simple double outcome **If** statements can be entered on a single line :

If a <= 10 Then b = 1: c = 2 * a Else b = 2: c = 2 * a - b

Example : Suppose you are working in a store that pays $ 10 an hour for the first 40 hours and $ 12 an hour for overtime above 40 hours, with a bonus of $ 200 if you worked 100 hours or more a week (without being caught asleep !). The following code could be used to calculate your pay :

```
Dim Bonus As Double, Total_Hours As Double, Pay As Double, bSleep As Boolean
    Bonus = 200
    Total_Hours = Range("B3")
    bSleep = Cells(2,4)
```

bSleep is a Boolean variable that is **True** if you were caught sleeping, and **False** otherwise.

Condition checked

Double Outcome **If**

```
If Total_Hours <= 40 Then
    Pay = Total_Hours * 10
Else
    Pay = 40 * 10 + (Total_Hours - 40) * 12
End If
```

Code executed if condition is **True**

Code executed if condition is **False**

Single Outcome **If**

Condition checked.

Executed if condition **True**

```
If (Total_Hours >= 100) And (bSleep = False) Then Pay = Pay + Bonus
```

The condition **Total_Hours <= 40** is evaluated as **True** or **False**. If it's **True** then the line **Pay = Total_Hours * 10** is executed and followed by a transfer to the **End If** statement. If the condition is **False** control is transferred to the **Else** statement and the line **Pay = 40 * 10 + (Total_Hours - 40) * 12** is executed, followed by the **End If** statement.

Complex comparisons can be constructed using **comparison operators** (listed in the **Appendix**). For example :

If (A > 5 And A <= 10) Or (B – C < D) Then …….

where **A**, **B**, **C**, and **D** may represent either variables or algebraic expressions.

The first condition cannot be written as **(A > 5 And <= 10) .** The "subject" of the comparison, **A**, must be repeated.

********Although it is good practice to repeat the subject of the comparison, the mathematically familiar operation **If 2 <= X <= 10 Then …** is accepted by **Excel 2003**.

An equal sign sometimes represents a true equality ! It was noted earlier that an equal sign in a computing context is really an assignment operator. An exception occurs with **If** statements in which an equal sign does, in fact, act as a traditional equality operator.

If X = 10 And Y =20 Then …..

1.6.3 Nested *If* Statements

"Nested" Ifs, or **If**s inside other **If**s, are useful when checking exclusive conditions and can be used to create multiple outcome situations.

Example : A university registrar's office needs to create code that selects different response letters based on an applicant's transcript marks. The logical structure might appear as follows :

```
If Average < 60 Then
   ' Send "In your dreams/What were you thinking" letter
Else
   If Average >= 60 And Average < 78 Then
      ' Send polite rejection letter/Reapply after a year at another school.
   Else
      If Average >= 78 And Average <= 88 Then
         ' Warm welcome letter/Send tuition cheques immediately !
      Else
         If Average > 88 And Average <= 98 Then
            ' You definitely qualify for a scholarship/Fill out enclosed forms
         Else
            ' You're brilliant/ Do you want a teaching position ?
         End If
      End If
   End If
End If
```

Indenting and lining up the **If…Else… End If** lines for each condition tested makes the code more readable.

Condition **Average > 98** not tested since it's all that remains.

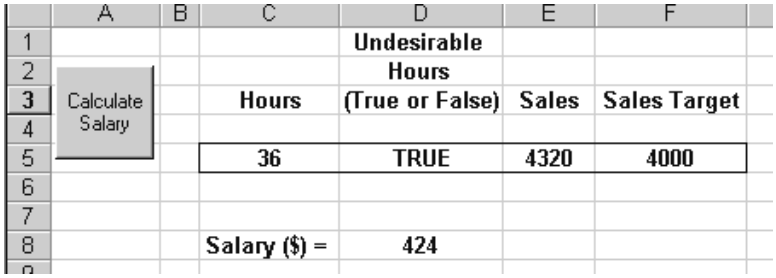
Comments : It is good practice in longer programs to document your code with explanatory comments. It is also sometimes useful to add notes to serve as reminders to change, modify, try, or check something. A line containing a comment or note must be preceded by an apostrophe **'** or the letters **Rem** , which is short for Remark. In the example above comments were used to remind the programmer to add code to produce certain types of letters. **Note :** A comment cannot be added at the end of a line that contains a continuation character.

Nested **If**s can be simplified by using the **ElseIf** statement which eliminates the need for all but the final **End If**.

```
If Average < 60 Then
   ' Send "In your dreams/What were you thinking/Don't waste our time!" letter
ElseIf Average >= 60 And Average < 78 Then
   ' Send polite rejection letter/Reapply after a year at another school.
ElseIf Average >= 78 And Average <= 88 Then
   ' Warm welcome letter/Send tuition cheques immediately !
ElseIf Average > 88 And Average <= 98 Then
   ' You definitely qualify for a scholarship/Fill out enclosed forms
Else
   '  You're brilliant/ Do you want a teaching position ?
End If
```

An alternative to nested **If**s is the **Select Case** command. Refer to **Appendix 23** or a programming manual.

Exercise # 3 : A certain retail store pays sales employees 7.50 $/hour **if** the total hours worked per week are less than or equal to 40 hrs, and 11.50 $/hour for hours worked in excess of 40 hours. The company recognizes that certain work schedules may be undesirable and pays a Salary Bonus of 1.50$ / hr for all hours worked **if** such a situation occurs. Also, a $100 Sales Bonus is paid **if** specific sales targets are achieved : **if** sales exceed $ 4000, **or if** hourly average sales exceed $180. [The latter condition rewards outstanding employees who work for much less than 40 hours and thus would have no chance of reaching the Sales Target.] The bolded conditions above will be represented by **If** statements below.



Underscores are often used to link words in variable names to make them more readable. **Eg. Undesirable_Hours**

```
Option Explicit

Private Sub cmdSalary_Click()
Dim Sales As Double, Hours As Double, Undesirable_Hours As Boolean
Dim Salary As Double, Salary_Bonus As Double
Dim Sales_Bonus As Double, Sales_Target As Double

Sales_Bonus = 0: Salary_Bonus = 0

Hours = Worksheets("Sheet1").Range("C5")
Undesirable_Hours = Worksheets("Sheet1").Range("D5")
Sales = Worksheets("Sheet1").Range("E5")
Sales_Target = Worksheets("Sheet1").Range("F5")

    If Hours <= 40 Then
        Salary = Hours * 7.5
    Else
        Salary = 40 * 7.5 + (Hours - 40) * 11.5
    End If

If Undesirable_Hours = True Then Salary_Bonus = 1.5 * Hours
If Sales > Sales_Target Or Sales / Hours > 180 Then Sales_Bonus = 100

Salary = Salary + Salary_Bonus + Sales_Bonus

Worksheets("Sheet1").Range("D8") = Salary

End Sub
```

Although not absolutely necessary, it is prudent to initialize values that may not be explicitly defined later.

Read-in the data.

Undesirable_Hours is a Boolean variable, which means that it has only two possible values : **True** or **False** (or **1** or **0**)

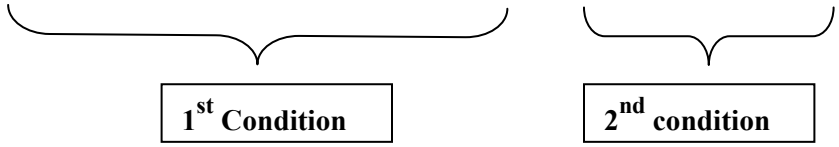
Observe that the calculation of sales per hour can be made within the **If** .

The basic salary is augmented by two possible bonuses. Note the unorthodox algebra.

A double outcome **If** statement is used to decide if the hours worked are less than or equal to 40 in which case the pay rate is set to 7.50 $/hr (the **Then** alternative), or greater than 40 such that the pay rate is 11.50 $/hr (the **Else** alternative). The difference **(Hours – 40)** determines the number of overtime hours to be paid at 11.50 $/hr.

The Sales Bonus condition could be made more stringent by requiring that it be met without overtime being paid using the following statement :

If Sales > Sales_Target And Hours <= 40 Or Sales / Hours > 180 Then Sales_Bonus = 100



[The 40 hour condition is not applied to employees generating in excess of 180 $/hour since it is clearly advantageous to let them work as long as they want. Though a minimum hours condition might be added to ensure that they don't take only a few prime hours .

Either the first condition **or** the second condition must be satisfied to obtain a bonus; hence the appearance of the **Or** operator linking the two conditions. The first condition has two necessary parts that must simultaneously be satisfied; hence the presence of the **And** operator linking the two parts. Brackets are sometimes used to group conditions to make them more readable, especially those linked via **And** operators :

If (Sales > Sales_Target And Hours <= 40) Or (Sales / Hours > 180) Then Sales_Bonus = 100

The original decision statement at the top of the page is of the form : **If A And B Or C Then …** which is ambiguous since we're not certain how it will be evaluated; that is, will the **And** be evaluated first, or will the **Or** . The decision might be interpreted as either : **If A And (B Or C) Then …** or as **If (A And B) Or C Then …** Write a small program to identify how it is evaluated.

1.7 Writing Your Own Code

[This section was written for a group of students who were having unusual difficulty "seeing" how to construct a program based on a problem situation. The process given here is somewhat longer than it needs to be. **If you're having difficulty with the first assignment you may want to read this section, otherwise it's optional**.]

Up until this point the code for the various programs has been provided; the next step it to learn to develop your own code. What is given in this section is one approach to developing a program; you'll no doubt develop your own variations as you gain experience. Each program will require at least one control which is used to activate the program, and the basic program structure given on page 1-1 will be helpful in determining what components will likely be required.

A good starting point is to read the program "requirement" (that is, the problem statement describing what is to be done) and identify which variables will be needed. For example, suppose that the following program requirement was presented :

A certain province has simplified it's income tax system so that there are only 2 tax brackets : for taxable income less than or equal to $ 40,000 the tax rate is 15 % ; for taxable income greater than $ 40,000 the tax rate is 25 % for the portion above $ 40,000 (the first $ 40,000 is taxed at 15 %).

Design a program that reads the value of **rate1** (15 %, or 0.15) and the value of **rate2** (25 %, or 0.25) in from the worksheet. Use an Input Box to enter the value of a person's taxable income and a Message Box to display the amount of tax paid; also write the tax to be paid into a cell on the worksheet.

Clearly, variables describing the two tax rates, the taxable income, and the tax to be paid will be required, so the initial set of variables would be : **rate1**, **rate2**, **Taxable_Income**, and **Tax** . The need for other variables may become apparent later in the program development. Next, the data type of each variable should be identified. Although money and percentages are involved in this application the vast majority of problems in these notes concern physics and engineering applications, so we'll simply declare these four variables as double precision (**Double**). [Despite any personal preferences to round values up or down to the nearest dollar, it's not a good idea to work with **Integer** declarations for **Taxable_Income**, and **Tax** since the maximum value is 32,767; of course a **Long** integer declaration could be used.]

Buttons are the most convenient choice to allow a user to activate subroutines [5] so that a button would be added to the worksheet with a **Name** something like **cmdTax** and an informative **Caption** such as **Calculate Tax**. Double clicking on the button (in design mode) creates the subroutine shell in which the variables are declared with **Dim** statements. The first code entered appears as :

```
Option Explicit

Private Sub cmdTax_Click()
Dim rate1 As Double, rate2 As Double, Taxable_Income As Double, Tax As Double

End Sub
```

Note : if the variables are to be used in more than one subroutine they would have to be declared in the general declarations section under the **Option Explicit** statement.

The next step is to create the code to read-in the various input quantities which in this case are **rate1**, **rate2**, and **Taxable_Income**. Convenient cells are selected for entering the input values keeping in mind the need for appropriate descriptive titles and labels. In this example cells **B2** and **B3** are used to hold the values of **rate1** and **rate2**. Although the problem requirement indicated that the **Taxable_Income** should be entered via an **Input Box**, a useful strategy until you develop confidence is to *keep it simple at first* and add all the "bells and whistles" later. Accordingly, the variable **Taxable_Income** has been set to some convenient test value, **Taxable_Income = 10000**, within the subroutine, and will not be entered via an **Input Box** until later.

```
rate1 = Worksheets("sheet1").Range("B2")
rate2 = Worksheets("sheet1").Range("B3")
Taxable_Income = 10000
```

At this point it is useful to add any output statements realizing, nevertheless, that the actual value of the output has yet to be determined. The value of **Tax** will be written into cell **B5** using **Worksheets ("sheet1").Range("B5") = Tax** . In order to test the program input and output statements a simple but meaningless value of the output variable **Tax** will be constructed from the input values : **Tax = rate1 + rate2 + Taxable_Income** . This statement allows us to also check if the input values are being correctly read-in. The calculation of the output value must, of course, be performed before the write statement is

---

[5] although it might be amusing to dump the final code into, say, a scroll bar control, which would work, but which might end up being a little annoying.

encountered. The code entered so far is thus :

```
Option Explicit

Private Sub cmdTax_Click()
Dim rate1 As Double, rate2 As Double, Taxable_Income As Double, Tax As Double

rate1 = Worksheets("sheet1").Range("B2")
rate2 = Worksheets("sheet1").Range("B3")
Taxable_Income = 10000

Tax = rate1 + rate2 + Taxable_Income

Worksheets("sheet1").Range("B5") = Tax

End Sub
```

Data type declaration statements

Input Read statements

Meaningless test calculation using input data.
Remove once this code is working.

Output write statement

After values for **rate1** and **rate2** have been entered into cells **B2** and **B3** the program can be tested. The value of **Tax** should be 10000**.**40 .

The final, most important, and usually the most difficult step is to construct the actual numerical analysis portion of the program. It is often useful to imagine how you would approach such an analysis without a computer, and then to develop the equivalent procedure in terms of computer code. In the present example you would examine the input value of **Taxable_Income** and decide ***if*** it was less than or equal to $ 40,000, or greater than $ 40,000. This suggests that a double outcome **If** statement might be used to computationally model the decision.

```
If Taxable_Income <= 40000 Then
      do something
   Else
      otherwise do something else
End If
```

Depending on the outcome of the decision, the **Tax** would be calculated in one of two possible ways. If the value of **Taxable_Income** is less than or equal to $ 40,000 the **Tax** is simply the **Taxable_Income** multiplied by **rate1**; mathematically and in terms of proper arithmetic operators **Tax = Taxable_Income*rate1** . Otherwise, if the value of **Taxable_Income** is greater than $ 40,000 , the **Tax** calculation has two terms : the first term is the tax on the $ 40,000 portion of income at **rate1**, while the second term calculates the portion of income above $ 40,000 as **(Taxable_Income – 40000)** and multiples it by **rate2** .

Tax = 40000*rate1 + (Taxable Income - 40000)*rate2

Tax on first $ 40,000

Portion of income above $ 40,000.

[Notice that the use of appropriate variable names not only makes the code more readable but usually simplifies the task of converting the normal mental analysis into the equivalent computer code analysis.]

The two calculation statements are now entered into the **If…Then…Else** :

```
If Taxable_Income <= 40000 Then
      Tax = Taxable_Income * rate1
   Else
      Tax = 40000 * rate1 + (Taxable_Income - 40000) * rate2
End If
```

$ 40,000 **cannot** be written with a comma or dollar sign.

The complete code to this point is :

```
Option Explicit
Private Sub cmdTax_Click()
Dim rate1 As Double, rate2 As Double, Taxable_Income As Double, Tax As Double

rate1 = Worksheets("sheet1").Range("B2")
rate2 = Worksheets("sheet1").Range("B3")
Taxable_Income = 10000

If Taxable_Income <= 40000 Then
      Tax = Taxable_Income * rate1
   Else
      Tax = 40000 * rate1 + (Taxable_Income - 40000) * rate2
End If

Worksheets("sheet1").Range("B5") = Tax
End Sub
```

Replaced by **Input Box**

An **Input Box** can now used to read-in the value of the **Taxable_Income** .

```
Taxable_Income = InputBox("Enter your taxable income", "Taxable Income")
```

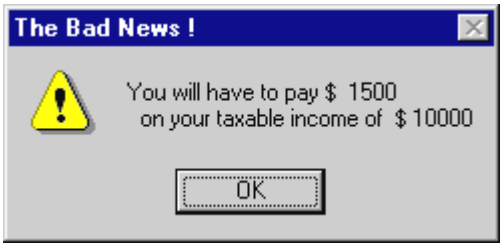
And a **Message Box** is added just before **End Sub** to display the amount of **Tax** to be paid.

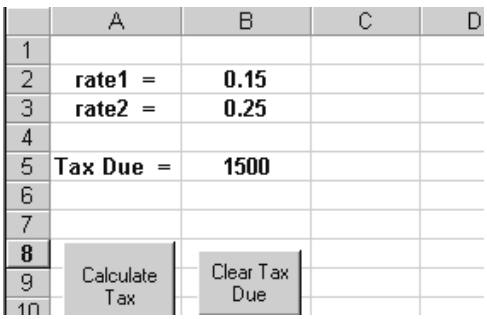
```
MsgBox "You will have to pay $  " & Tax & vbCr & "  on your taxable income of  $ " & _
Taxable_Income, vbExclamation, "The Bad News !"
```

Continuation character

The components of this message box will be considered in chapter 2.

The worksheet and output message box appear as :





1.8 Miscellaneous Tips & Difficulties [Skip on your first read through]

Moving from the Worksheet to the VB Editor : The **VB Editor** can be accessed in five ways : double-clicking on a control in **Design Mode,** via the **VB Editor** icon on the **VB Toolbar**, using **Alt + F11**, via the **VB Editor** icon on the ***taskbar*** at the top or bottom of your screen, or **Tools menu ⇒ Macro ⇒ Visual Basic Editor**. Avoid using the **View Code** icon at the top of the **Tool*box*** since it always seems to produce the extraneous subroutine shell given below :

```
Private Sub Worksheet_SelectionChange(ByVal Target As Excel.Range)
End Sub
```

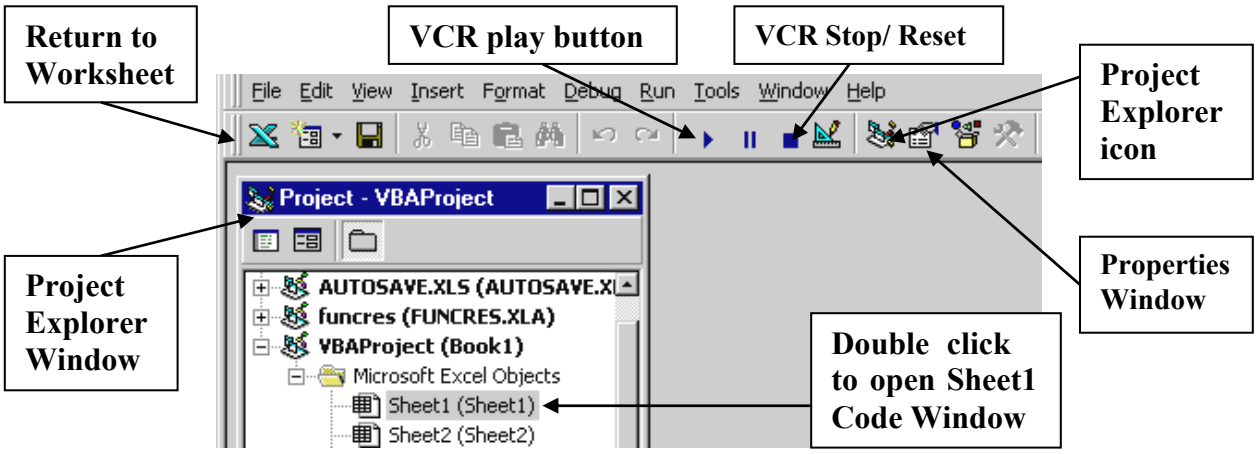
Moving from the VB Editor to the Worksheet : Four ways are available to return to the **Worksheet** : via the **Excel** icon at the left end of the **tool bar** (probably just below the **File menu**), via the **Work-sheet** icon on the ***taskbar*** at the top or bottom of your screen, **Alt + F11** (it toggles you back & forth), **View menu ⇒ Microsoft Excel**.

Grayed-out VB Toolbar : Immediately after a program has been run the **VB Toolbar** will be "grayed-out" (except for the **Design Mode** icon), which means that the **VB Editor icon** cannot be used to access the **Code window**; nor can **Alt + F11** which also isn't working. Fortunately, the **Task Bar icon** above (or below) the worksheet window will work (provided the Editor has already been accessed). Clicking on any worksheet cell, or clicking in and out of **Design Mode** will restore all functions.

Grayed-out icons.

The Program won't run : You may occasionally push a button, but have nothing happen. Consider the following : write statements may have been omitted, you may be caught in an infinite loop (discussed in Chapter 3 -- use **Ctrl + Break**), if an error occurred when the program was last run you may have forgotten to reset the program. To reset the program click in and out of **Design Mode**, or go into the **VB Editor** and press the **VCR Stop** button. If none of the above work, you may have a screen-freeze : press **Ctrl + Alt + Delete** and hope that you'd saved recently !

Empty VB Editor : If the **Sheet1 Code window** is not open try one of the following : enter the **View menu ⇒ Code** (should be the first item) , press **F7** , if the **Project Explorer** is open double-click on **Sheet1(Sheet1)** or click on the **View Code icon** at the top left . *Occasionally the various windows in the **VB Editor** become too large, or attach themselves awkwardly to other windows, refer to **Appendix A.2** at the end of the notes for corrective measures (**Docking & Restoring**).*



Problems [Chapter 1] [Include Clear Buttons with your programs]

1. Write a program that reads the polar form (magnitude & direction) of a vector from two cells on a worksheet, calculates the X and Y components (rectangular form), and writes the components back onto the worksheet. The angle is measured with respect to the positive X axis.

2. Construct a program that converts between Canadian and US dollars. Use a separate button for each of the two possible conversions [as in Example # 2a)]. The program should read the conversion rate (the number of US dollars required to buy 1 $Cdn : use 0.64) from an **Input Box**, and the number of dollars to be converted from the spreadsheet. The conversion rate should be written somewhere on the worksheet. The converted value should be displayed on the worksheet and in a **Message Box** along with the original dollars and conversion rate. [The conversion factor will be 1/0.64 for one of the conversions.] Refer to pg 1-15 & 1-16 for **Input Box** & **Message Box** syntax.

3. Two different expressions describe the magnitude of the gravitational field (acceleration) due to a solid, spherical object of radius R depending on whether the field point (location "r" – which is measured from the center of the object) is inside the object, or outside the object.

$$g = \frac{G M}{r^2} \quad \text{for} \quad r \geq R \qquad\qquad g = \frac{GM r}{R^3} \quad \text{for} \quad r < R$$

[The expression for r < R can be derived by applying Gauss' Law to a solid, spherical distribution of mass. For the sake of continuity, the two expressions must have the same value at r = R .]

Write a program that calculates values of "g" in the vicinity of the Earth (M = **5.98E+24** kg, R = **6.37E+06** m, G = **6.67259E–11** N m^2/kg^2). The value of "r" should be read in from the worksheet, or via an **Input Box**. The value of "g" should be written back into some cell on the worksheet and displayed in a **Message Box**.

4. A utility company, Hydro-Quebec, provides electrical power at the following rates :

   0.0474 $/kWh for the first 30 kWh per day
   0.0597 $/kWh for the balance of the consumption.

   [Example: a user who consumed 660 kWh in a 12 day period would be billed at 0.0474 $/kWh for the first 30 x 12 = 360 kWh, and at 0.0597 $/kWh for the next 300 kWh.]

   Construct a program that reads in the total consumption (kWh) and number of days from the work-sheet, and writes the cost to the user back on the worksheet. Also display the amount owed in a **Message Box** that includes the total consumption and number of days.

5. The quadratic formula can be used to find the roots of the quadratic equation $ax^2 + bx + c = 0$. Design a program that reads the values a, b, and c from the worksheet, and then writes the roots back onto the worksheet as well as displaying them in a **Message Box** along with the equation being solved. Add a single line **If** statement that detects if $b^2 < 4ac$ and then stops execution with an **End** statement, or use a **GoTo statement** to skip the root calculations.

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

6. **Snell's Law** relates the angle of an incident beam of light (θ_1) travelling in a medium having an index of refraction $\mathbf{n_1}$ to the angle of the refracted beam (θ_2) in a medium with index of refraction $\mathbf{n_2}$ according to : $\sin\theta_2 / \sin\theta_1 = n_1 / n_2$ where the angles are measured from the normal. Create a program that reads in the indices of refraction (refer to any physics textbook; identify the two mediums on the worksheet) and the incident angle from cells on the worksheet (properly labelled). Use trial and error to find the critical angle beyond which total internal reflection occurs (remember that the ray will be bent toward the normal when entering a medium with a higher index of refraction, so that no critical angle exists). The value of the critical angle for the two media should be clearly evident on your worksheet. Include a **Clear** button that clears only the value of the refracted angle, θ_2 . Plot a graph of refracted angle versus incident angle that shows the approach to the critical angle (use 10 points).

> **Accessing Worksheet Functions** : No inverse sine function is built into VBA, but the function can be obtained (or derived) from other built in functions (see Appendix 28). Fortunately, all worksheet functions can be accessed using the either of the following staements (with ASIN( ) as an example) :
> **theta2 = Application.WorksheetFunction.Asin( )** [you need to provide the appropriate argument]
> **theta2 = Excel.WorksheetFunction.Asin( )**

7. The lens/ mirror equation $\frac{1}{d_o} + \frac{1}{d_i} = \frac{1}{f}$ relates the object location d_o , image location d_i , and focal length **f** for convex and concave mirrors and lenses. [Check the sign conventions in any physics textbook.] Design a program that calculates the image location given the focal length and the object location. Determine whether lenses and mirrors must be treated with separate buttons. A more sophisticated version of this problem is given in the problem section of Chapter 2.

8. A Pharmaceutical company produces an antibiotic that is prescribed (in a daily dose) according to the mass of the patient :

   Amount (in mL) = 0.002 mass^2 + 0.0143 mass + 0.14 if mass <= 25 kg

   Amount (in mL) = 0.05 mass + 0.5 if mass > 25 kg

   Construct a program that reads the mass of a patient from an **Input Box** and then returns the dosage in a **Message Box** (along with the mass of the patient).

9. A 1500 kg rocket powered racing car is acted on by a thrust that varies as follows :

time (s)	thrust (N)
$0 \le t < 20$	45,000
$20 \le t < 30$	30,000
$30 \le t < 50$	20,000

   Overall friction is assumed to be constant at 500 N. The force of air resistance (drag) is modeled as $\mathbf{bv^2}$ where "**v**" is velocity; **b** = 0.4 if the velocity is less than or equal to 100 m/s and **b** = 0.5 otherwise. Design a program that reads in the time (less than 50 seconds) and velocity (less than 300 m/s) from the worksheet and then writes the instantaneous acceleration back onto the worksheet. The net force acting on the object is **Fnet = Thrust – drag – friction**. [A much less artificial scenario is given in chapter 6. The data given here is not necessarily realistic].

10. A retail store has a wage structure that pays the following base salary and bonuses :

Hours worked (in a 7 day period)	**rate ($/hr)**
$t \leq 35$	7.50
$35 < t \leq 45$	9.50
greater than 45	11.00

Gross Sales ($)	**Bonus %**
Less than 4,000	5
4,000 to 6,000	7.5
greater than 6,000	10

Base salary example: 50 hours worked gives 35*7.50 + 10*9.5 + 5*11.00. Similarly for the bonus.

Construct a program that reads in the total hours worked and gross sales, calculates base salary and sales bonus, and writes them back on the worksheet. The company owner is interested in examining some performance ratios and also wants the program to calculate sales/hour and sales/salary. The two ratios should be written on the worksheet.

11. A student will have a final physics mark of between 58.0 and 59.5 "boosted" to a 60 % pass if certain criteria are met. The first criterion is that the student must not have had a previous boost in physics, and which will be described by a **Boolean** variable named **Previous_Boost** (a **True** value of **Previous_Boost** indicates that a previous boost had occurred). The second criterion is that the student must have passed at least one of the three term tests with a mark of greater than 65 %. Design a program that reads in a student's final physics mark, their 3 term test marks (out of 100), and a **Boolean** variable indicating whether a previous boost had occurred. The program should produce one of four different **Message Boxes** : an indication that no boost is required, an indication that no boost is possible because the final mark is less than 58 %, an indication that a boost will be made, or an indication that the criteria for a boost have not been met. Hint : the first two message boxes can be placed in single outcome **If** statements together with **End** statements.