2013

VB.NET Functions and Subs: Worked Analysis for a Mortgage Loan App.

Jerome Casey
Technological University Dublin, jerome.casey@tudubln.ie

Recommended Citation

Casey, J. (2013). *VB.NET functions and subs:worked analysis for a mortgage loan app.*Software guide for undergraduate students. Technological University Dublin.

HIGHER CERT/BACHELOR OF TECHNOLOGY – DT036A
VISUAL BASIC PROGRAMMING 1

Functions & Sub Procedures

In this Lecture:

1. Functions and Sub Procedures.
2. The difference between arguments and parameters.
3. Passing a value by Reference or By Value.
4. Using a RichTextBox control to output information to the user via its *.Text* property and *AppendText* method, setting and using tabs (vbTab), moving to a new line (vbCrLf), changing font colour etc.

Structured Programming:

Structured program design requires that problems be broken into smaller problems to be solved one at a time. Visual Basic has two devices, **Sub procedures** and **Function procedures** that are used to break problems into manageable chunks. To distinguish them from event procedures, Sub and Function procedures are referred to as **general procedures** or **methods**. General procedures also:

1. eliminate repetitive code,
2. can be reused in other programs, and
3. allow a team of programmers to work on a single program.

A **Sub procedure** is a part of a program that performs one or more related tasks, has its own name and is written as a separate part of the program. The simplest type of Sub procedure has the form:

```
Private Sub ProcedureName( param1 As Single, param 2 As Integer etc.)
        statement(s) that use param1, param2 etc.
End Sub
```

A Sub procedure is invoked with a statement of the form:

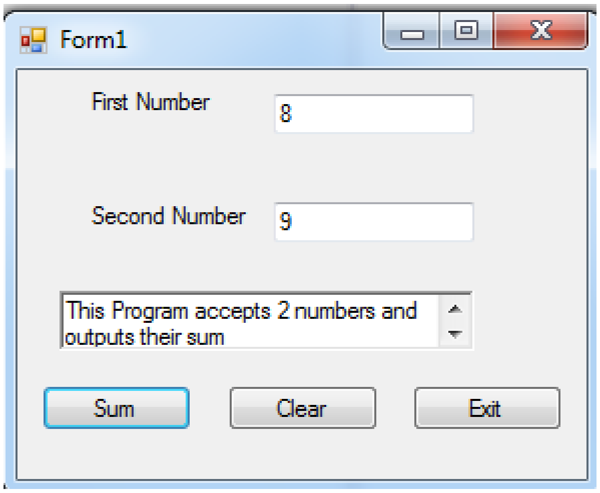
```
        Call ProcedureName(argument1, argument2)
```

The word Call is optional. The rules for naming general procedures are identical to the rules for naming variables. The name chosen for a Sub procedure should describe the task it performs.

Sub procedures make a program easy to read, modify, and debug. The event procedure gives a description of what the program does and the Sub procedures fill in the details. Another benefit of Sub procedures is that they can be called several times during the execution of the program. This feature is especially useful when there are many statements in the Sub procedure.

Program 1: Adding Two Numbers using Sub Procedures

This program is a very simple program to add two numbers and to demonstrate the use of Sub procedures. A later exercise (Amortization) will show how they are used for more substantial programming efforts.



First Number 8

Second Number 9

This Program accepts 2 numbers and outputs their sum

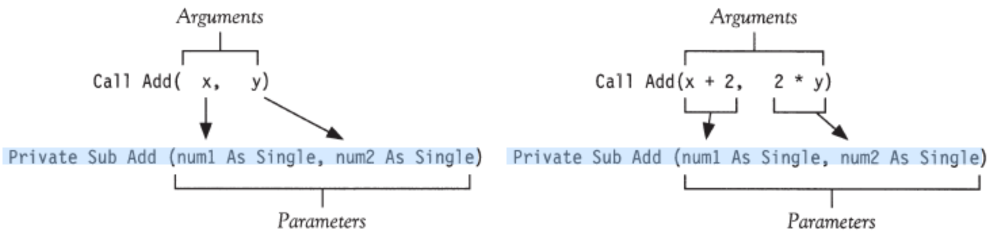
```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        ExplainPurpose()
    End Sub
Private Sub cmdSum_Click(sender As Object, e As EventArgs) Handles cmdAdd.Click
        Dim sngFirst As Single
        Dim sngSecond As Single
        sngFirst = Val(txtFirstNum.Text)
        sngSecond = Val(txtSecondNum.Text)
1.      Add(sngFirst, sngSecond)
End Sub
2. Private Sub Add(sng1 As Single, sng2 As Single)
        rtbOutput.Text = "The Sum is: " & (sng1 + sng2)
End Sub
Private Sub ExplainPurpose()
        rtbOutput.Text = "This Program accepts 2 numbers and outputs their sum "
End Sub
Private Sub cmdClear_Click(sender As Object, e As EventArgs) Handles cmdClear.Click
        txtFirstNum.Text = ""
        txtSecondNum.Text = ""
        rtbOutput.Text = ""
        txtFirstNum.Focus()
End Sub
End Class
```

The program uses two Sub procedures which are shown highlighted.

The statement Add(sngFirst, sngSecond) at 1. causes execution to jump to the Private Sub Add(sng1 As Single, sng2 As Single) statement at 2., which assigns the sngFirst to *sng1* and the sngSecond to *sng2*. After the lines between a Sub procedure statements are executed, execution returns to the line following this call, namely, the End Sub statement in the event procedure.

Arguments and Parameters:

The items appearing in the parentheses of a Call statement are called **arguments**. These should not be confused with parameters, which appear in the heading of a Sub procedure. The variables *num1* and *num2* appearing in the Sub procedure *Add* shown below are called **parameters**. They are merely temporary place holders for the numbers passed to the Sub procedure; their names are not important. The only essentials are their datatype, quantity, and order. In the *Add* Sub procedure shown here, the parameters must be numeric variables and there must be two of them.



```
Call Add( x,   y)
Private Sub Add (num1 As Single, num2 As Single)

Call Add(x + 2,   2 * y)
Private Sub Add (num1 As Single, num2 As Single)
```

Figure shows passing arguments to Parameters of a sub procedure. Arguments can be constants, variables or expressions.

Other datatypes can be passed to a Sub procedure e.g. a String. In this case, the receiving parameter in the Sub procedure must be followed by the declaration As String.

Passing Arguments ByVal or ByRef:

When you pass a value to a procedure you may pass it **ByVal** or **ByRef** (for by value or by reference). The *ByVal* sends a copy of the argument's value to the procedure so that the procedure cannot alter the original value. The *ByRef* sends a reference to the procedure indicating where the argument's value is stored in memory so that the called procedure can alter the argument's original value. You specify how to pass the argument by using the **ByVal** or **ByRef** keyword before the parameter in the procedure header. If you don't specify **ByVal** or **ByRef** then arguments are passed by value by default.

```
Private Sub SelectColor(ByVal IncomingColor As Color)
```

Function Procedures:

Visual Basic has a number of built-in functions that greatly extend its capability. These functions perform such varied tasks as taking the square root of a number Sqr, counting the number of characters in a string Len, and formatting data FormatCurrency. Functions associate with one or more values, called the *input*, and a single value, called the *output*. The function is said to **return** the output value. Often this value is assigned to a variable such as:

```
intCharacters = Len(strSentence)
```

which you can then use in subsequent lines of code. Remember the return value from the MsgBox function? This variable must be of the same datatype as the return value.

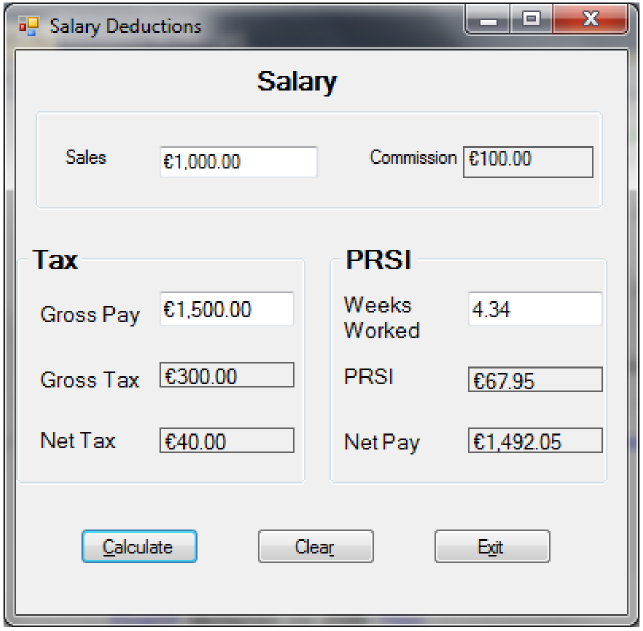
You can also write your own functions that can be called, calculates a value and returns this value to the caller. Thus the main difference in coding a sub procedure and a function procedure is that in the latter you must set up a **return value**. This return value is placed in a variable that VB names with the same name as the function name.

```
Private Function Commission(ByVal decSalesin As Decimal) As Decimal
...... Commission = decSalesin * 0.35
End Function
```

Note: Somewhere in the function you must set the function name to a value.

Program 2: Calculating Commission using a Function

Looking at the Salary program we covered previously, the commission block could be written as a function and called from within the `cmdCalc_Click` event:



Salary Deductions

Salary

Sales €1,000.00

Commission €100.00

Tax

Gross Pay €1,500.00

Gross Tax €300.00

Net Tax €40.00

PRSI

Weeks Worked 4.34

PRSI €67.95

Net Pay €1,492.05

```
    Private Sub cmdCalc_Click(sender As Object  etc.
        ..
        ' call commission function
        decCommission = Commission(decSales)
    End Sub
```

```
    Private Function Commission(decSalesin As Decimal) As Decimal
         If decSalesin <= 1000 Then
             Commission = decSalesin * 0.1
         ElseIf decSalesin <= 1500 Then
             Commission = decSalesin * 0.15
         ElseIf decSalesin <= 2000 Then
             Commission = decSalesin * 0.2
         ElseIf decSalesin <= 2500 Then
             Commission = decSalesin * 0.25
         ElseIf decSalesin <= 3000 Then
             Commission = decSalesin * 0.3
         Else
             Commission = decSalesin * 0.35
         End If
     End Function
```

You can also specify the datatype of the return value by adding the **As** clause after the function name.

Here the function can be called within an expression, in which case it doesn't need the **Call** keyword.

When the function is called the value in *decSales* is passed to the function and assigned to the named argument, *decSalesin*. Within the function process for every reference to *decSalesin* the value of *decSales* is actually used.

Program 3: Loan Analysis to demo Functions, Sub Procedures & Output to RichTextBox
Develop a program to analyze a loan. Assume the loan is repaid in equal monthly payments and interest is compounded monthly. The program should request the amount (principal) of the loan, the annual rate of interest, and the number of years over which the loan is to be repaid. The four options to be provided by command buttons are as follows:

1. Calculate the Monthly Payment. The formula for the monthly payment is:

Monthly Payment = P * r / (1 – (1 + r) ^ (–n)) or $\boxed{Mon.Payment = P * \frac{r}{1-(1+r)^{-n}}}$

where

P is the principal of the loan,

r is the monthly interest rate (annual rate divided by 12) given as a number between 0 (for 0 percent) and 1 (for 100 percent), and

n is the number of months over which the loan is to be repaid.

Since a payment computed in this manner can be expected to include fractions of a cent, the value should be rounded up to the next nearest cent. This corrected payment can be achieved using the formula:

Correct Monthly Payment = Round(Monthly Payment + 0.005, 2)

2. Display an Amortization Schedule, that is, a table showing the balance on the loan at the end of each month for any year over the duration of the loan. Also show how much of each monthly payment goes toward interest and how much is used to repay the principal. Finally, display the total interest paid over the duration of the loan. The balances for successive months are calculated with the formula:

NewBalance = (1 + r) * Oldbal – monPay

where

r is the monthly interest rate (annual rate / 12, a fraction between 0 and 1),

Oldbal is the balance for the preceding month (amount of loan left to be paid), and

monPay is the monthly payment.

3. Show the effect of Changes in the Interest Rate. Display a table giving the monthly payment for each interest rate from 1 percent below to 1 percent above the specified annual rate in steps of one-eighth of a percent.

4. Quit.

Designing the Loan Analysis Program (Hierarchy Chart):

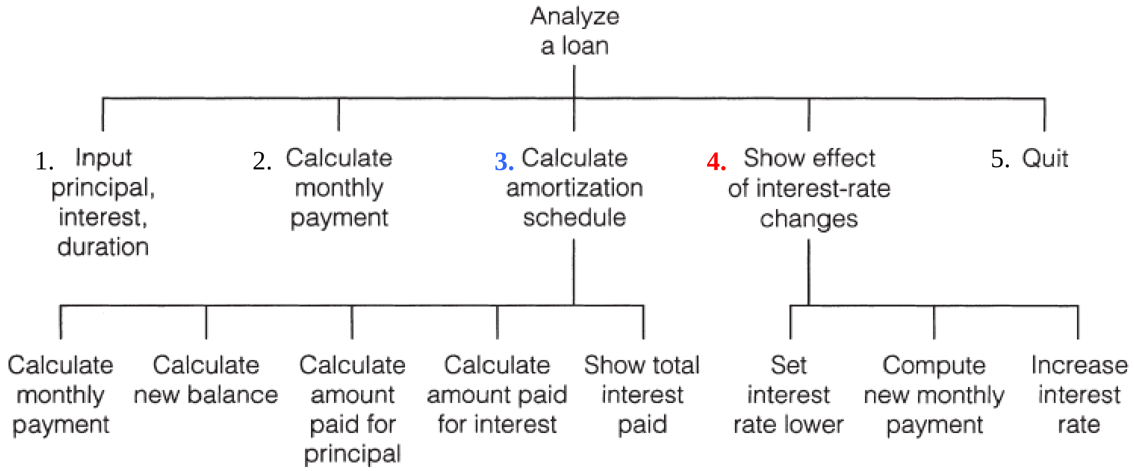
For each of the tasks described in the preceding options 1 to 4, the program must first look at the text boxes to obtain the particulars of the loan to be analyzed. Thus, the first division of the problem is into the following tasks:

1. Input the principal, interest, and duration.
2. Calculate the Monthly Payment.
3. Calculate the Amortization Schedule.
4. Display the effects of Interest Rate Changes.
5. Quit.

Task 1 is a basic input operation and Task 2 involves applying the formula given in Step 1; therefore, these tasks need not be broken down any further. The demanding work of the program is done in Tasks 3 and 4, which can be divided into smaller subtasks.

3. *Calculate Amortization Schedule*. This task involves simulating the loan month by month. First, the monthly payment must be computed. Then, for each month, the new balance must be computed together with a decomposition of the monthly payment into the amount paid for interest and the amount going toward repaying the principal. That is, Task 3 is divided into the following subtasks:
3.1 Calculate monthly payment.
3.2 Calculate new balance.
3.3 Calculate amount of monthly payment for principal.
3.4 Calculate amount of monthly payment for interest.

4. *Display the effects of interest-rate changes*. A table is needed to show the effects of changes in the interest rate on the size of the monthly payment. First, the interest rate is reduced by one percentage point and the new monthly payment is computed. Then the interest rate is increased by regular increments until it reaches one percentage point above the original rate, with new monthly payment amounts computed for each intermediate interest rate. The subtasks for this task are then:
4.1 Reduce the interest rate by 1 percent.
4.2 Calculate the monthly payment.
4.3 Increase the interest rate by 1/8 percent.


Analyze a loan
1. Input principal, interest, duration
2. Calculate monthly payment
3. Calculate amortization schedule
4. Show effect of interest-rate changes
5. Quit
Calculate monthly payment
Calculate new balance
Calculate amount paid for principal
Calculate amount paid for interest
Show total interest paid
Set interest rate lower
Compute new monthly payment
Increase interest rate

Hierarchy Chart for Loan Analysis Program

Pseudocode for the Loan Analysis Program:

```
Calculate Monthly Payment command button:
        INPUT LOAN DATA (Sub procedure InputData)
        COMPUTE MONTHLY PAYMENT (Function MonthlyPayment)
        DISPLAY MONTHLY PAYMENT (Sub procedure ShowMonthlyPayment)


Display Interest Rate Change Table command button:
        INPUT LOAN DATA (Sub procedure InputData)
        DISPLAY INTEREST RATE CHANGE TABLE
           (Sub procedure ShowInterestChanges)
        Decrease annual rate by .01      i.e. 1%
        Do
          Display monthly interest rate
          COMPUTE MONTHLY PAYMENT (Function MonthlyPayment)
          Increase annual rate by .00125    i.e. going up in steps of 0.125%
        Loop Until annual rate > original annual rate + .01   i.e. +1% greater than original


Display Amortization Schedule command button:
        INPUT LOAN DATA (Sub procedure InputData)
        DISPLAY AMORTIZATION SCHEDULE (Sub procedure ShowAmortSched)
        Compute monthly interest rate
        COMPUTE MONTHLY PAYMENT (Function MonthlyPayment)
        Display amortization table
        Display total interest paid
```

Tasks and Their Procedures:

Task	Procedure
1. Input principal, interest, duration.	InputData
2. Calculate monthly payment.	ShowPayment
3. Calculate amortization schedule.	ShowAmortSched
3.1 Calculate monthly payment.	MonthlyPayment
3.2 Calculate new balance.	Balance
3.3 Calculate amount paid for loan.	ShowAmortSched
3.4 Calculate amount paid for interest.	ShowAmortSched
4. Show effect of interest rate changes.	ShowInterestChanges
4.1 Reduce interest rate.	ShowInterestChanges
4.2 Compute new monthly payment.	MonthlyPayment
4.3 Increase interest rate.	ShowInterestChanges

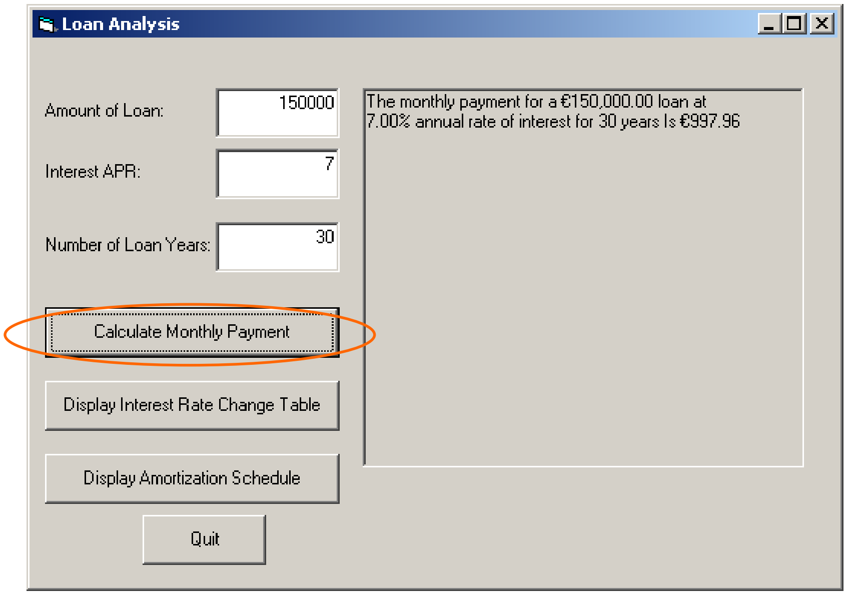
1. Tasks 3.1 and 3.2 are performed by functions. Using functions to compute these quantities simplifies the computations in ShowAmortSched.

2. Since the monthly payment calculation was rounded up to the nearest cent, it is highly likely that the payment needed in the final month to pay off the loan will be less than the normal monthly payment. For this reason, Balance (called from ShowAmortSched) checks if the outstanding balance of the loan (including interest due) is less than the regular monthly payment. If so, it makes appropriate adjustments.

3. The standard formula for computing the monthly payment cannot be used if either:
(i). the interest rate is zero percent or
(ii). the loan duration is zero months.
Although both of these situations do not represent reasonable loan parameters, provisions are made in the function MonthlyPayment so that the program can handle these situations.

Program 3 - Loan Analysis - The Interface and Functionality

The screenshots show the form design as well as the output in the *rtbDisplay* RichTextBox obtained by clicking the respective command buttons for the given data input.


Loan Analysis
Calculate Monthly Payment
Display Interest Rate Change Table
Display Amortization Schedule
Quit

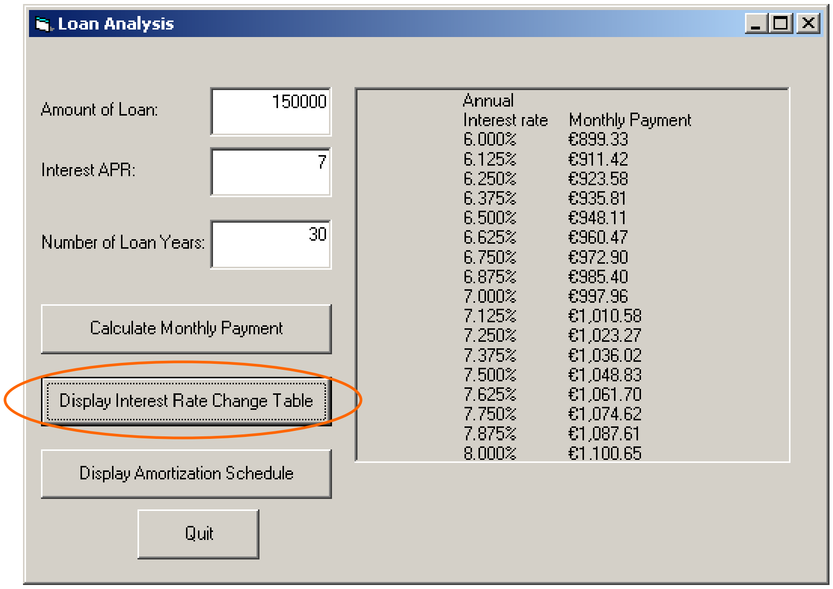
Amount of Loan: 150000

Interest APR: 7

Number of Loan Years: 30

```
The monthly payment for a €150,000.00 loan at
7.00% annual rate of interest for 30 years Is €997.96
```

Monthly Payment for a 30 Year Loan


Loan Analysis
Calculate Monthly Payment
Display Interest Rate Change Table
Display Amortization Schedule
Quit

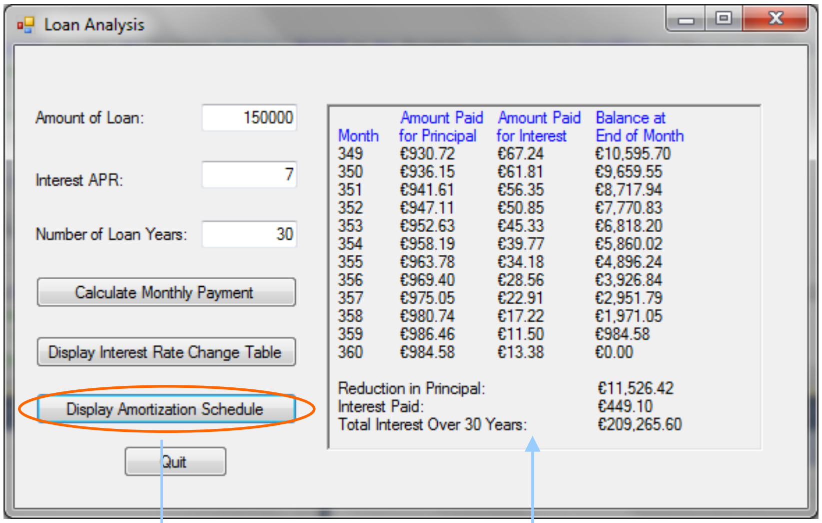
Amount of Loan: 150000

Interest APR: 7

Number of Loan Years: 30

```
Annual
Interest rate   Monthly Payment
6.000%          €899.33
6.125%          €911.42
6.250%          €923.58
6.375%          €935.81
6.500%          €948.11
6.625%          €960.47
6.750%          €972.90
6.875%          €985.40
7.000%          €997.96
7.125%          €1,010.58
7.250%          €1,023.27
7.375%          €1,036.02
7.500%          €1,048.83
7.625%          €1,061.70
7.750%          €1,074.62
7.875%          €1,087.61
8.000%          €1.100.65
```

Interest Rate Change Table for a 30 Year Loan


Loan Analysis
Calculate Monthly Payment
Display Interest Rate Change Table
Display Amortization Schedule
Quit

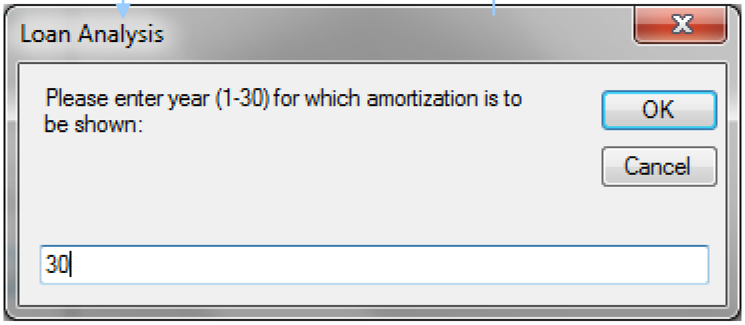
Amount of Loan: 150000

Interest APR: 7

Number of Loan Years: 30

```
         Amount Paid    Amount Paid   Balance at
Month    for Principal  for Interest  End of Month
349      €930.72        €67.24        €10,595.70
350      €936.15        €61.81        €9,659.55
351      €941.61        €56.35        €8,717.94
352      €947.11        €50.85        €7,770.83
353      €952.63        €45.33        €6,818.20
354      €958.19        €39.77        €5,860.02
355      €963.78        €34.18        €4,896.24
356      €969.40        €28.56        €3,926.84
357      €975.05        €22.91        €2,951.79
358      €980.74        €17.22        €1,971.05
359      €986.46        €11.50        €984.58
360      €984.58        €13.38        €0.00

Reduction in Principal:               €11,526.42
Interest Paid:                        €449.10
Total Interest Over 30 Years:         €209,265.60
```

Amortization for Year 30 of the Loan



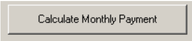
Please enter year (1-30) for which amortization is to be shown:

30

Inputting Year 30 in the Inputbox

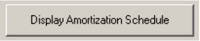
Program-Loan Analysis: The Code

```
Public Class Form1
Dim decPrincipal As Decimal 'Amount of loan
Dim decYearlyRate As Decimal 'Annual rate of interest
Dim intNumMonths As Integer 'Number of months to repay loan

    Private Sub cmdPayment_Click(sender As Object, e As EventArgs) Handles cmdPayment.Click
        Call InputData(decPrincipal, decYearlyRate, intNumMonths)
        Call ShowMonthlyPayment(decPrincipal, decYearlyRate, intNumMonths)
    End Sub
```



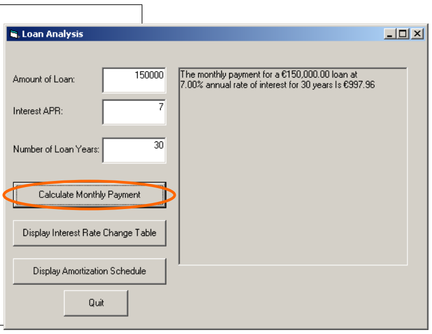
```
    Private Sub cmdRateTable_Click(sender As Object, e As EventArgs) Handles cmdRateTable.Click
        Call InputData(decPrincipal, decYearlyRate, intNumMonths)
        Call ShowInterestChanges(decPrincipal, decYearlyRate, intNumMonths)
    End Sub
```

```
    Private Sub cmdAmort_Click(sender As Object, e As EventArgs) Handles cmdAmort.Click
        Call InputData(decPrincipal, decYearlyRate, intNumMonths)
        Call ShowAmortSched(decPrincipal, decYearlyRate, intNumMonths)
    End Sub

    Private Sub InputData(ByRef decPrincipal As Decimal, ByRef decYearlyRate As Decimal, ByRef intNumMonths As Integer)

        'Input: Pass back by reference 1.the loan amount, 2. yearly rate of interest, and 3. duration in months
        decPrincipal = Val(txtAmt.Text)
        decYearlyRate = Val(txtAPR.Text) / 100  ' convert % taken from textbox to decimal precision value
        intNumMonths = Val(txtYrs.Text) * 12

        rtbDisplay.ReadOnly = True
        rtbDisplay.SelectionTabs = New Integer() {5, 50, 120, 190}
    End Sub
```



Note: Parameters passed *By Reference*

```
    Private Sub ShowMonthlyPayment(decPrincipal As Decimal, decYearlyRate As Decimal, intNumMons As Integer)
        Dim decMonthlyRate As Decimal, strPrincipal As String, strApr As String
        Dim strYrs As String, decPay As Decimal, strPayment As String

        'Display monthly payment amount
        decMonthlyRate = decYearlyRate / 12 'monthly interest rate
        strPrincipal = FormatCurrency(decPrincipal, 2)  'euros with cent
        strApr = FormatNumber(decYearlyRate * 100) 'changing decimal precision to a %, e.g. 0.01 --> 1.00%
        strYrs = FormatNumber(intNumMons / 12, 0)   'convert months back to years
        decPay = MonthlyPayment(decPrincipal, decMonthlyRate, intNumMons)
        strPayment = FormatCurrency(decPay)

        rtbDisplay.Text = "" 'clear RichTextBox of any previous output
        rtbDisplay.Text &= "The monthly payment for a " & strPrincipal & " loan at " & vbCrLf
        rtbDisplay.Text &= strApr & " % annual rate of interest for "
        rtbDisplay.Text &= strYrs & " years Is " & strPayment & "."
    End Sub
```


Loan Analysis
Calculate Monthly Payment
Display Interest Rate Change Table
Display Amortization Schedule
Quit

Amount of Loan: 150000

Interest APR: 7

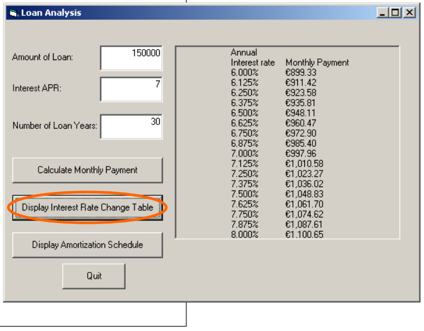
Number of Loan Years: 30

The monthly payment for a €150,000.00 loan at
7.00% annual rate of interest for 30 years Is €997.96

```
Private Function MonthlyPayment(decPrincipal As Decimal, decMonthlyRate As Decimal, intNumMons As Integer) As Decimal

           Dim decPayEst As Decimal
        'the standard formula for computing the monthly payment cannot be used if either
        'the loan duration is zero months or the interest rate is zero percent.
        If intNumMons = 0 Then
            decPayEst = decPrincipal
        ElseIf decMonthlyRate = 0 Then
            decPayEst = decPrincipal / intNumMons
        Else
            decPayEst = decPrincipal * decMonthlyRate / (1 - (1 + decMonthlyRate) ^ (-intNumMons))
        End If
        MonthlyPayment = Math.Round(decPayEst + 0.005, 2) 'round up to the nearest cent
End Function
```

Note: Somewhere in the function you must set the function name to a value. This is the value returned by the function.

```
Private Sub ShowInterestChanges(decPrincipal As Decimal, decYearlyRate As Decimal, intNumMons As Integer)
    Dim decNewRate As Decimal, decMonthlyRate As Decimal, decPMent As Decimal, strPayment As String
    'Display effect of interest changes (from an interest rate of 1% lower up to 1% higher)
    'going up in steps of 0.125%    i.e. 0.00125 in decimal precision
    rtbDisplay.Text = "" 'clear textbox of any previous output

    rtbDisplay.Text &= vbTab & vbTab & "Annual" & vbCrLf
    rtbDisplay.Text &= vbTab & vbTab & "Interest Rate" & vbTab & "Monthly Payment" & vbCrLf
    decNewRate = decYearlyRate - 0.01 ' lower bound is 1% lower than actual rate

    Do
        decMonthlyRate = decNewRate / 12 'monthly rate
        decPMent = MonthlyPayment(decPrincipal, decMonthlyRate, intNumMons)
        strPayment = FormatCurrency(decPMent)
        rtbDisplay.Text &= vbTab & vbTab & FormatPercent(decNewRate, 3) & vbTab & strPayment & vbCrLf
        decNewRate = decNewRate + 0.00125
    Loop Until decNewRate > decYearlyRate + 0.01 ' upper bound is 1% higher than actual rate

End Sub
```



Loan Analysis
Calculate Monthly Payment
Display Interest Rate Change Table
Display Amortization Schedule
Quit

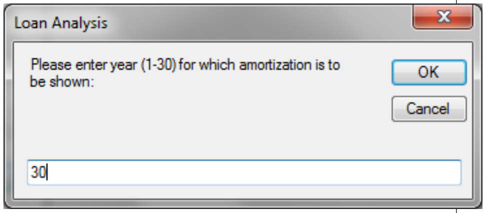
Amount of Loan: 150000

Interest APR: 7

Number of Loan Years: 30

```
Annual
Interest rate   Monthly Payment
6.000%          €899.33
6.125%          €911.42
6.250%          €923.58
6.375%          €935.81
6.500%          €948.11
6.625%          €960.47
6.750%          €972.90
6.875%          €985.40
7.000%          €997.96
7.125%          €1,010.58
7.250%          €1,023.27
7.375%          €1,036.02
7.500%          €1,048.83
7.625%          €1,061.70
7.750%          €1,074.62
7.875%          €1,087.61
8.000%          €1,100.65
```

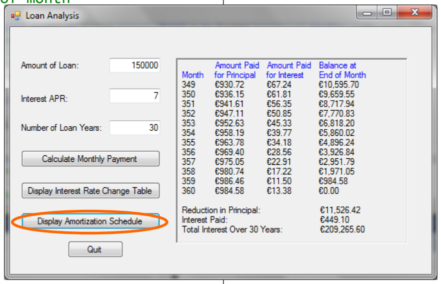
```
Private Sub ShowAmortSched(decPrincipal As Decimal, decYearlyRate As Decimal, intNumMons As Integer)
    Dim strMsg As String, intStartMonth As Integer, decMonthlyRate As Decimal
    Dim decMonPayment As Decimal, decTotalInterest As Decimal
    Dim decYearInterest As Decimal, decOldBalance As Decimal
    Dim intMonthNum As Integer, decNewBalance As Decimal
    Dim decPrincipalPaid As Decimal, decInterestPaid As Decimal
    Dim decReductPrin As Decimal, intLoanYears As Integer

    'Display Amortization Schedule
    strMsg = "Please enter year (1-" & CStr(intNumMons / 12)
    strMsg &= ") for which amortization is to be shown:"
    intStartMonth = 12 * Val(InputBox(strMsg)) - 11
    rtbDisplay.Text = "" 'clear RichTextbox of any previous output
```



Please enter year (1-30) for which amortization is to be shown:

30

```
   'change the attributes of the text that will be appended to the control with the next call to the AppendText method.
   'use the AppendText method if you want to change color of headers. See Page 320 on RichTextBox
    rtbDisplay.SelectionColor = Color.Blue
    rtbDisplay.AppendText(vbTab & vbTab & "Amount Paid " & vbTab & "Amount Paid " & vbTab & "Balance at" & vbCrLf)
    rtbDisplay.SelectionColor = Color.Blue
    rtbDisplay.AppendText(vbTab & "Month" & vbTab & "for Principal" & vbTab & "for Interest" & vbTab & "End of Month" & vbCrLf)

    decMonthlyRate = decYearlyRate / 12 'monthly interest rate
    decMonPayment = MonthlyPayment(decPrincipal, decMonthlyRate, intNumMons)
    decTotalInterest = 0
    decYearInterest = 0
    decOldBalance = decPrincipal

    For intMonthNum = 1 To intNumMons 'calculations done for all months here e.g. if 30 yr loan then intNumMons=360
        decNewBalance = Balance(decMonPayment, decOldBalance, decMonthlyRate)
        decPrincipalPaid = decOldBalance - decNewBalance
        decInterestPaid = decMonPayment - decPrincipalPaid 'rem: monthlyPayment = principal + interest
        decTotalInterest = decTotalInterest + decInterestPaid

        'if block will filter/show only those months for the year specified in the inputbox
        If (intMonthNum >= intStartMonth) And (intMonthNum <= intStartMonth + 11) Then
            rtbDisplay.AppendText(vbTab & FormatNumber(intMonthNum, 0)) ' Month number
            rtbDisplay.AppendText(vbTab & FormatCurrency(decPrincipalPaid)) ' amount paid for principal
            rtbDisplay.AppendText(vbTab & FormatCurrency(decInterestPaid)) ' amount paid for interest
            rtbDisplay.AppendText(vbTab & FormatCurrency(decNewBalance) & vbCrLf)' balance at end of month
            decYearInterest = decYearInterest + decInterestPaid
        End If

        decOldBalance = decNewBalance
    Next intMonthNum
        'rem: monthlyPayment = principal + interest
    decReductPrin = 12 * decMonPayment - decYearInterest
    intLoanYears = intNumMons / 12

    rtbDisplay.AppendText(vbCrLf) 'skip a line
    rtbDisplay.AppendText(vbTab & "Reduction in Principal:") 'i.e. for year specified in inputbox
    rtbDisplay.AppendText(vbTab & vbTab & vbTab & FormatCurrency(decReductPrin) & vbCrLf)

    rtbDisplay.AppendText(vbTab & "Interest Paid:")  'i.e. for year specified in inputbox
    rtbDisplay.AppendText(vbTab & vbTab & vbTab & FormatCurrency(decYearInterest) & vbCrLf)

    rtbDisplay.AppendText(vbTab & "Total Interest Over " & intLoanYears & " Years:")
    'this sentence length crosses a number of tab positions
    rtbDisplay.AppendText(vbTab & vbTab & FormatCurrency(decTotalInterest))
End Sub
```



Loan Analysis
Calculate Monthly Payment
Display Interest Rate Change Table
Display Amortization Schedule
Quit

Amount of Loan: 150000

Interest APR: 7

Number of Loan Years: 30

```
        Amount Paid     Amount Paid     Balance at
Month   for Principal   for Interest    End of Month
349     €930.72         €67.24          €10,595.70
350     €936.15         €61.81          €9,659.55
351     €941.61         €56.35          €8,717.94
352     €947.11         €50.85          €7,770.83
353     €952.63         €45.33          €6,818.20
354     €958.19         €39.77          €5,860.02
355     €963.78         €34.18          €4,896.24
356     €969.40         €28.56          €3,926.84
357     €975.05         €22.91          €2,951.79
358     €980.74         €17.22          €1,971.05
359     €986.46         €11.50          €984.58
360     €984.58         €13.38          €0.00

Reduction in Principal:                 €11,526.42
Interest Paid:                          €449.10
Total Interest Over 30 Years:           €209,265.60
```

```
Private Function Balance(decMonPayment As Decimal, decPrincipal As Decimal,
                                        decMonthlyRate As Decimal) As Decimal
    Dim decNewBal As Decimal 'Compute balance remaining to be paid at the end of the month

     decNewBal = (1 + decMonthlyRate) * decPrincipal

    If decNewBal <= decMonPayment Then   'e.g. the final monthly instalment to be paid
        decMonPayment = decNewBal
        Balance = 0
    Else
        Balance = decNewBal - decMonPayment
    End If
End Function
```

checks if the outstanding balance of the loan (including interest due) is less than the regular monthly payment. e.g. since the monthly payment calculation is rounded up to the nearest cent, it is highly likely that the payment needed in the final month to pay off the loan will be less than the normal monthly payment.