(b) Write a program that uses the XML file from part (a) to display the names, states, and party affiliation of all the senators in a DataGridView ordered by their state. The two senators from each state should be ordered by their first names.

26. The file Top25HR.xml contains statistics for the top 25 home-run hitters of all time in major league baseball. The first nine lines of the file are shown in Fig. 8.12.

(a) Write a program that displays the contents of this file in a DataGridView control in descending order by the number of home runs hit.

(b) Write a program that uses the file Top25HR.xml and creates a CSV file containing the same information.

```
<?xml version='1.0'?>
<!-- This file contains data on the all-time top 25 home -->
<!-- run hitters in major league baseball prior to 2010. -->
<home_run_hitters>
  <player>
    <name>Babe Ruth</name>
    <atBats>8399</atBats>
    <homeRuns>714</homeRuns>
  </player>
```

FIGURE 8.12 Beginning of the file Top25HR.xml.

Solutions to Practice Problems 8.3

1. The problem is with the clause

```
Let pop = st.<population>.Value
```

Since there are no arithmetic operators or numeric conversion functions in the clause, local type inference will interpret *pop* to be a string variable. When the program is run, the first state listed will be Rhode Island, whose population is 998,000. The program will run as intended only if the clause is

```
Let pop = CDbl(st.<population>.Value)
```

2. The line would have to be changed to

```
txtName.Text = query.First.name
```

8.4 A Case Study: Recording Checks and Deposits

The purpose of this section is to take you through the design and implementation of a quality program for personal checkbook management. That a user-friendly checkbook management program can be written in less than four pages of code clearly shows Visual Basic's ability to improve the productivity of programmers. It is easy to imagine an entire finance program, similar to programs that have generated millions of dollars of sales, being written in only a few weeks by using Visual Basic!

Design of the Program

Though many commercial programs are available for personal financial management, they include so many bells and whistles that their original purposes—keeping track of transactions and reporting balances—have become obscured. The program in this section was designed specifically as a checkbook program. It keeps track of expenditures and deposits and produces a report. The program showcases many of the techniques and tools available in Visual Basic.

The general design goals for the program include the following abilities:

- Automatically enter the user's name on each check and deposit slip.
- Automatically provide the next consecutive check or deposit slip number. (The user can override this feature if necessary.)
- Automatically provide the date. (Again, this feature can be overridden.)
- For each check, record the payee, the amount, and optionally a memo.
- For each deposit slip, record the source, the amount, and optionally a memo.
- Display the current balance at all times.
- Produce a report detailing all transactions.

User Interface

With Visual Basic, we can place a replica of a check or deposit slip on the screen and let the user supply the information as if actually filling out a check or deposit slip. Figure 8.13 shows the form in its check mode. The DataGridView control at the bottom of the form will be used to display a report detailing all the transactions. The purposes of the four buttons and the text box above the DataGridView control are obvious.

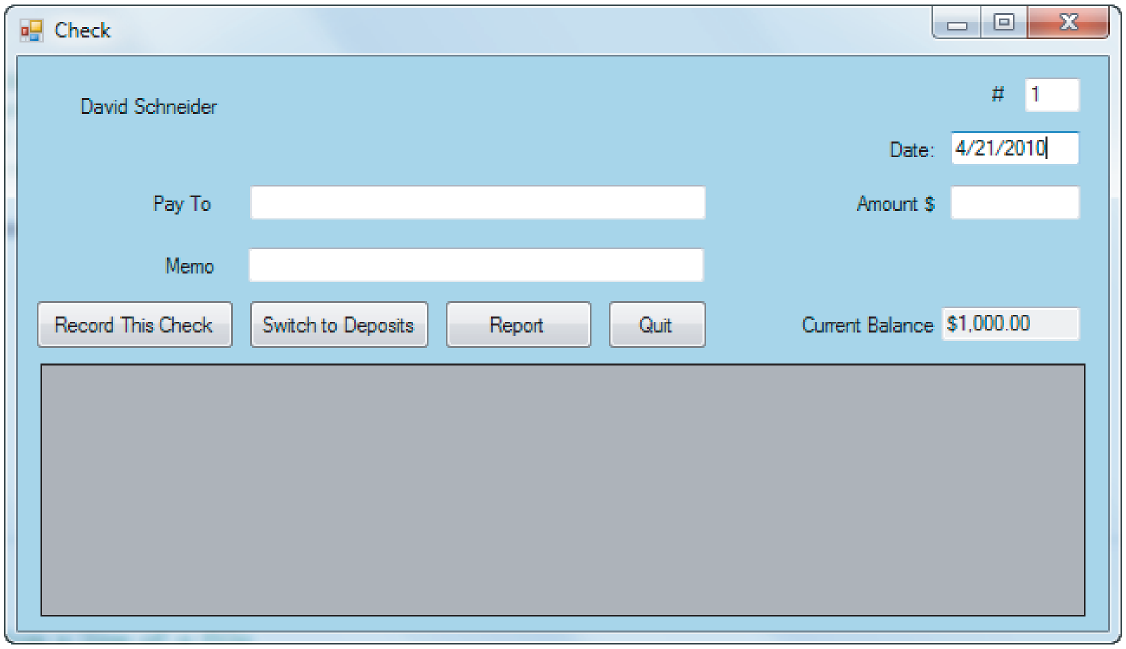


FIGURE 8.13 **Template for entering a check.**

The first time the program is run, the user is asked for his or her name, the starting balance, and the numbers of the first check and deposit slip. Suppose the user's name is David Schneider, the starting balance is $1000, and both the first check number and deposit slip number are 1. Figure 8.13 shows the form after the four pieces of information are entered. The upper part of the form looks like a check. The form has a color of light blue when in check mode. The Date box is automatically set to today's date but can be altered by the user. The user fills in the payee, amount, and optionally a memo. When the user clicks on the *Record This Check* button, the information is written to a text file, the balance is updated, and check number 2 appears.

To record a deposit, the user clicks on the *Switch to Deposits* button. The form then appears as in Fig. 8.14. The form's title bar now reads Deposit Slip, the words Pay To change to Source,

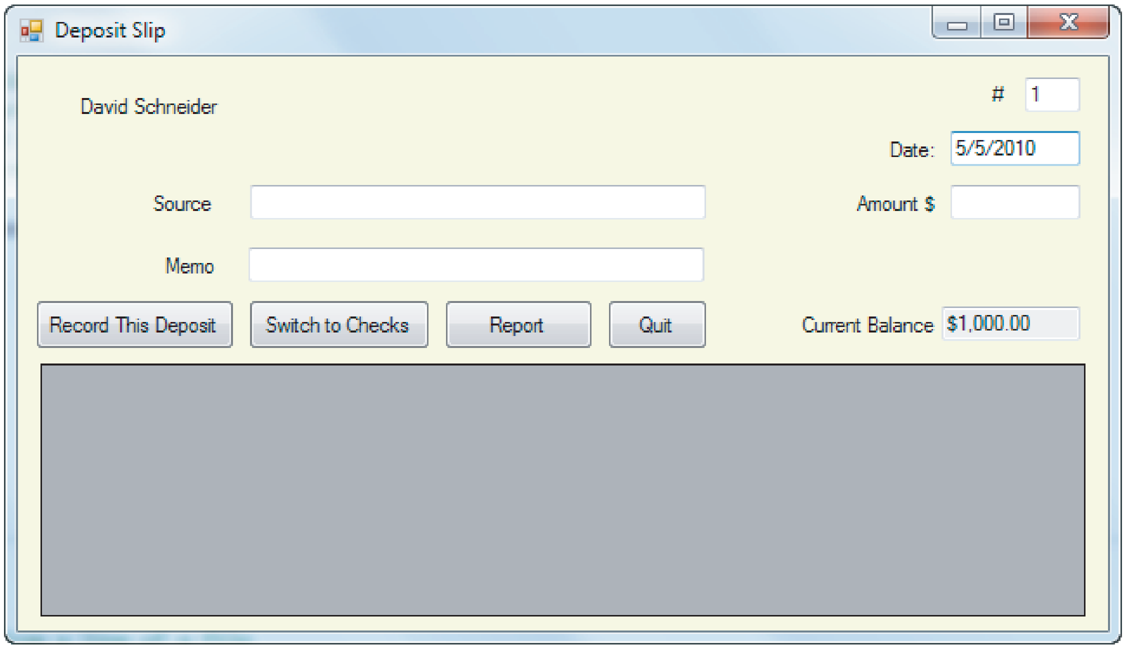
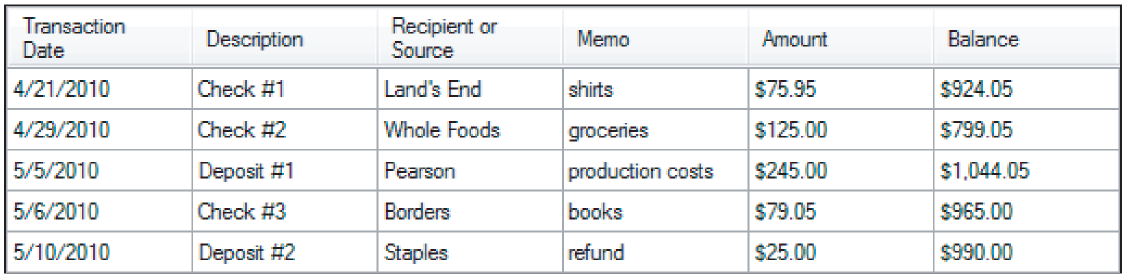


FIGURE 8.14 **Template for entering a deposit.**

and the color of the form changes to light yellow. Also, in the buttons at the bottom of the slip, the words *Check* and *Deposit* are interchanged. A deposit is recorded in much the same way as a check. When the *Report* button is clicked on, a report similar to the one in Fig. 8.15 is displayed in the DataGridView control.



Transaction Date	Description	Recipient or Source	Memo	Amount	Balance
4/21/2010	Check #1	Land's End	shirts	$75.95	$924.05
4/29/2010	Check #2	Whole Foods	groceries	$125.00	$799.05
5/5/2010	Deposit #1	Pearson	production costs	$245.00	$1,044.05
5/6/2010	Check #3	Borders	books	$79.05	$965.00
5/10/2010	Deposit #2	Staples	refund	$25.00	$990.00

FIGURE 8.15 **Sample transaction report.**

The common design for the check and deposit slip allows one set of controls to be used for both items. The text of the label lblName is set to the user's name, while the text of the label lblToFrom will change back and forth between Pay To and Source.

Table 8.3 lists the objects and their initial property settings. Because the program will always begin by displaying the next check, all the text for the labels and the BackColor property of the form could have been set at design time. We chose instead to leave these assignments to the SetupCheck method, which normally is used to switch from deposit entry to check entry but also can be called by the form's Load event procedure to prepare the initial mode (check or deposit) for the form.

The program uses CSV formatted text files named InitialInfo.txt and Transactions.txt. The file InitialInfo.txt consists of a single line containing four comma-delimited pieces of

TABLE 8.3 Objects and initial property settings for the checkbook management program.

Object	Property	Setting
frmAccount		
lblName		
lblNum	Text	#
txtNum		
lblDate	Text	Date:
txtDate		
lblToFrom		
txtToFrom		
lblAmount	Text	Amount $
txtAmount		
lblMemo	Text	Memo
txtMemo		
btnRecord	Text	&Record This Check
btnMode	Text	&Switch to Deposits
btnReport	Text	Re&port
btnQuit	Text	&Quit
lblCurBal	Text	Current Balance
txtBalance	ReadOnly	True
dgvTransactions	RowHeaderVisible	False

information: the name to appear on the check and deposit slips, the starting balance, the number of the first check, and the number of the first deposit slip. The file Transactions.txt contains a line for each transaction—that is, writing a check or making a deposit. Each transaction is recorded as a sequence of eight comma-delimited items: the type of transaction, the contents of txtToFrom, the current balance, the number of the last check, the number of the last deposit slip, the amount of money, the memo, and the date.

Coding the Program

The second row of the hierarchy chart in Fig. 8.16 identifies the different events to which the program must respond. Table 8.4 lists the corresponding event procedures and the general procedures they call.

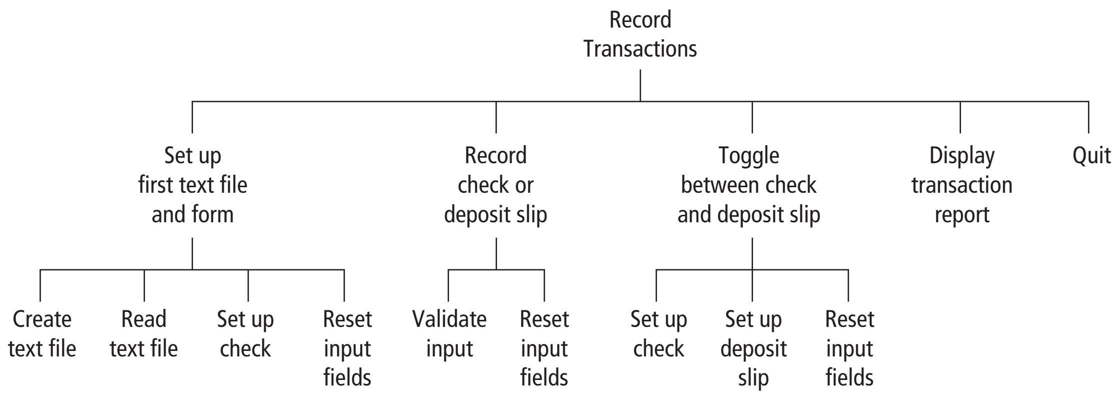


FIGURE 8.16 Hierarchy chart for checkbook management program.

TABLE 8.4 **Tasks and their procedures.**

Task	Procedure
1. Set up first text file and form	frmAccount_Load
1.1 Create first text file	InitializeData
1.2 Read text files	InitializeData
1.3 Set up check	SetupCheck
1.4 Reset input fields	ResetInput
2. Record check or deposit slip	btnRecord_Click
2.1 Validate input	DataValid
2.2 Reset input fields	ResetInput
3. Toggle between check & deposit slip	btnMode_Click
3.1 Set up check	SetupCheck
3.2 Set up deposit slip	SetupDeposit
3.3 Reset input fields	ResetInput
4. Display transaction report	btnReport_Click
5. Quit	btnQuit_Click

Let's examine each event procedure:

1. ***frmAccount_Load*** first calls the InitializeData Sub procedure to process the text file. This procedure first looks to see if the file InitialInfo.txt exists. If it does exist, the procedure uses it (along with possibly the last entry of Transactions.txt) to determine all information needed to proceed. If InitialInfo.txt does not exist, the Sub procedure prompts the user for the name to appear on the checks and deposit slips, the starting balance, and the numbers of the first check and deposit slip and then writes these items to the text file. The event procedure calls the SetupCheck Sub procedure next to set the transaction type to Check and sets the appropriate text and background color for a check. The event procedure then calls ResetInput, which initializes all the text boxes. The InitializeData Sub procedure employs structured exception handling to protect the code from invalid user input.
2. ***btnRecord_Click*** first confirms that the required fields contain valid entries. This is accomplished by calling the function DataValid. If the value returned is True, then btnRecord_Click updates the current balance, opens the text file in append mode, writes eight pieces of data to the file, and then closes the file. When DataValid returns False, the function itself pops up a message box to tell the user where information is needed or invalid. The user must type in the information and then click on the *Record* button again. The DataValid function uses structured exception handling to ensure that the user's input is valid. If either the amount or number field is not a number, the InvalidCastException is thrown. The Catch block handles this exception by displaying an appropriate message asking the user to reenter the information.
3. ***btnMode_Click*** toggles back and forth from a check to a deposit slip. It calls SetupCheck, or its analog SetupDeposit, and then calls ResetInput.
4. ***btnReport_Click*** displays a complete history of all transactions, as shown in Fig. 8.15.
5. ***btnQuit_Click*** ends the program.

```
'class-level named constants and variables
Const INIT_FILE As String = "InitialInfo.txt"
Const TRANS_FILE As String = "Transactions.txt"
'variables used for each entry
Dim isCheck As Boolean
Dim nameOnChk As String   'name to appear on checks and deposit slips
Dim lastCkNum As Integer  'number of last check written
Dim lastDpNum As Integer  'number of last deposit slip written
Dim curBal As Double      'current balance in account

Private Sub frmAccount_Load(...) Handles MyBase.Load
  'Set the class-level variables.
  InitializeData()
  'Set the name and balance labels.
  lblName.Text = nameOnChk
  txtBalance.Text = FormatCurrency(curBal)
  'Set the date field to the current date.
  txtDate.Text = CStr(Today)
  SetupCheck()
  ResetInput()
End Sub

Private Sub InitializeData()
  If IO.File.Exists(INIT_FILE) Then
    Dim data() As String   'holds the data from a line of a file
    Dim initFileContents() As String = IO.File.ReadAllLines(INIT_FILE)
    'Split the single line of INIT_FILE using the delimiter.
    data = initFileContents.First.Split(","c)
    'Load the name to appear on checks, current balance, number of
    'last check written, and number of last deposit slip processed.
    nameOnChk = data(0)
    curBal = CDbl(data(1))
    lastCkNum = CInt(data(2))
    lastDpNum = CInt(data(3))
    'Possibly update numeric values by looking at last record of TRANS_FILE
    If IO.File.Exists(TRANS_FILE) Then
      Dim transFileContents() As String = IO.File.ReadAllLines(TRANS_FILE)
      data = transFileContents.Last.Split(","c)
      curBal = CDbl(data(2))
      lastCkNum = CInt(data(3))
      lastDpNum = CInt(data(4))
    End If
  Else
    'INIT_FILE does not exist, so get initial data from user
    Dim sw As IO.StreamWriter
    nameOnChk = InputBox("Name to appear on checks and deposit slips:")
    Try
      curBal = CDbl(InputBox("Starting Balance:"))
      'get numbers of last check and deposit slip
      lastCkNum = CInt(InputBox("Number of first check:")) - 1
      lastDpNum = CInt(InputBox("Number of first deposit slip:")) - 1
      'The single record in the text file records the name to
      'appear on checks plus the initial data for the account.
      Dim outputLine As String = nameOnChk & "," & curBal & "," &
                                 lastCkNum & "," & lastDpNum
      sw = IO.File.CreateText(INIT_FILE)
      sw.WriteLine(outputLine)
    Catch
      'If a number cannot be converted, then display message and quit.
      MessageBox.Show("Invalid number. Program terminating.", "Error")
      Me.Close()
    Finally
      'Close the writer no matter what happens above.
      sw.Close()
    End Try
  End If
End Sub

Private Sub btnRecord_Click(...) Handles btnRecord.Click
  'Store the input into the transactions file.
  Dim amt As Double
  Dim transType As String
  'store only if all required fields are filled and valid
  If DataValid() Then
    amt = CDbl(txtAmount.Text)
    'adjust balance by amount depending on check or deposit slip mode
    If isCheck Then
      curBal = curBal – amt
      lastCkNum = CInt(txtNum.Text)
      transType = "Check"
    Else
      curBal += amt
      lastDpNum = CInt(txtNum.Text)
      transType = "Deposit"
    End If
    txtBalance.Text = FormatCurrency(curBal)
    'string array contains information to be stored
    Dim transOutput() As String = {transType, txtToFrom.Text,
      CStr(curBal), CStr(lastCkNum), CStr(lastDpNum), CStr(amt),
      txtMemo.Text, txtDate.Text}
    Dim sw As IO.StreamWriter = IO.File.AppendText(TRANS_FILE)
    'append the info to the text file, separated by the delimiter
    sw.WriteLine(Join(transOutput, ","c))
    sw.Close()
    'reset input text boxes to blank for next entry
    ResetInput()
  End If
End Sub

Function DataValid() As Boolean
  'return True if all data are valid, or display a message if not
  Dim errorMessage As String = ""
  'If one of the two essential pieces of information
  'is missing, assign its name to errorMessage.
  If txtToFrom.Text.Trim = "" Then
    If isCheck Then
      errorMessage = "Pay To"
    Else
      errorMessage = "Source"
    End If
    txtToFrom.Focus()
  ElseIf txtAmount.Text.Trim = "" Then
    errorMessage = "Amount"
    txtAmount.Focus()
  End If
  'if no errors yet, then check syntax of the two numerical fields
  If errorMessage = "" Then
    'check syntax of the amount field (Double)
    Try
      If CDbl(txtAmount.Text) <= 0 Then
        errorMessage = "The amount must be greater than zero."
        txtAmount.Focus()
      End If
    Catch exc As InvalidCastException
      errorMessage = "The amount " & txtAmount.Text & " is invalid."
      txtAmount.Focus()
    End Try
  Else
    errorMessage = "The '" & errorMessage & "' field must be filled."
  End If
  'display error message if available
  If errorMessage = "" Then
    'all required data fields have been filled; recording can proceed
    Return True
  Else
    'advise user of invalid data
    MessageBox.Show(errorMessage & " Please try again.")
    Return False
  End If
End Function

Private Sub btnMode_Click(...) Handles btnMode.Click
  'toggle mode between Check and Deposit Slip
  If isCheck Then
    SetupDeposit()
  Else
    SetupCheck()
  End If
  'set fields for next entry
  ResetInput()
End Sub


Sub SetupCheck()
  'prepare form for the entry of a check
  isCheck = True
  Me.Text = "Check"  'set the title bar of the form
  lblToFrom.Text = "Pay To"
  btnRecord.Text = "&Record This Check"
  btnMode.Text = "&Switch to Deposits"
  Me.BackColor = Color.LightBlue
End Sub


Sub SetupDeposit()
  'prepare form for the entry of a deposit
  isCheck = False
  Me.Text = "Deposit Slip"   'sets the title bar of the form
  lblToFrom.Text = "Source"
  btnRecord.Text = "&Record This Deposit"
  btnMode.Text = "&Switch to Checks"
  Me.BackColor = Color.LightYellow
End Sub
```

```
Sub ResetInput()
  'reset all text entry fields except date
  txtToFrom.Clear()
  txtAmount.Clear()
  txtMemo.Clear()
  If isCheck Then
    'make txtNum text box reflect next check number
    txtNum.Text = CStr(lastCkNum + 1)
  Else
    'make txtNum text box reflect next deposit slip number
    txtNum.Text = CStr(lastDpNum + 1)
  End If
  'set focus on To/From text box for the next entry
  txtToFrom.Focus()
End Sub

Private Sub btnReport_Click(...) Handles btnReport.Click
  If IO.File.Exists(TRANS_FILE) Then
    Dim transFileContents() As String = IO.File.ReadAllLines(TRANS_FILE)
    Dim query = From trans In transFileContents
                Let data = trans.Split(","c)
                Let transDate = CDate(data(7))
                Let number = FormNumber(data(0), data(3), data(4))
                Let toFrom = data(1)
                Let Memo = data(6)
                Let Amount = FormatCurrency(data(5))
                Let Balance = FormatCurrency(data(2))
                Select transDate, number, toFrom, Memo, Amount, Balance
    dgvTransactions.DataSource = query.ToList
    dgvTransactions.CurrentCell = Nothing
    dgvTransactions.Columns("transDate").HeaderText = "Transaction Date"
    dgvTransactions.Columns("number").HeaderText = "Description"
    dgvTransactions.Columns("toFrom").HeaderText = "Recipient or Source"
  Else
    MessageBox.Show("There are no transactions to report.")
  End If
End Sub

Function FormNumber(ByVal type As String, ByVal checkNumber As String,
                    ByVal depositNumber As String) As String
  If type = "Check" Then
    Return "Check #" & checkNumber
  Else
    Return "Deposit #" & depositNumber
  End If
End Function

Private Sub btnQuit_Click(...) Handles btnQuit.Click
  Me.Close()     'exit the program
End Sub
```

CHAPTER 8 SUMMARY

1. The *IO.File.WriteAllLines* method copies an array to a text file.
2. When data are stored in text files with the fields of each record separated by commas, LINQ can be used to sort, search, and reorganize the data with a little help from the Split method.