Chapter 3

Variables, Expressions, and Statements

GOALS

After working through this chapter, you will:

- Know how to use the InterCap method to name variables.
- Have used the order of operations to evaluate expressions and to write proper expressions.
- Be able to translate algebraic expressions to Basic.
- Know how to write assignment statements to evaluate expressions and assign the resulting values to variables and properties.
- Know how to set break points, examine values of variables and expressions in the Debug window, and single-step through programs.
- Have used the Timer control to regularly change the display of the system time.
- Understand each of the basic data types used by Visual Basic.
- Have used Dim statements to declare variables.
- Be familiar with the naming convention used for controls.
- Know how to use the SetFocus method.
- Be able to program an application, accepting input from the user, making calculations with assignment statements, and displaying results.
- Know how to use the Click event procedure.

OVERVIEW

In Chapter Two, you experimented with the graphical elements in Visual Basic. You placed objects such as textboxes and command buttons on forms. Then you used the Properties window to change properties of these objects, such as the size of the font or the border of a label. These properties determine the look-and-feel of programs.

Now you are ready to work with code. In this chapter, you will learn how to build code by using:

- *Variables, which work on the same principle as variables in algebra*
- *Expressions, which are combinations of variables and operators (for example, a plus sign)*
- *Assignment statements, which are used to give values to variables and properties of objects*
- *Data types, which define the different types of data used by a program (for example, whole numbers or currency)*
- *Methods, which are used to change the state of controls and to make controls perform actions.*

Along the way, you will learn further good programming practices, such as naming conventions for variables and methods for debugging.

Variables, Expressions, and Statements

Look back at the Play Ball program you built in Chapter 2 (see Figure 2-32). Imagine that you wanted to change the color of the text every five seconds. Or you want the text to flash every hour on the hour.

You need variables and expressions to give these types of instructions to the computer. Specifying any calculation for the computer to perform requires using expressions. Unlike the variables of algebra, the values of Visual Basic variables and expressions are not exclusively numeric. You can use them for dates, times, the results of Yes-or-No choices, letters of the alphabet, and chunks of text ranging in length from a single word to an entire term paper.

Variables

Visual Basic programs are designed to deal with a large amount of data stored in the computer. This data is not entered by you the programmer. Instead, it comes from:

- Users entering information
- Data gathered as the program runs

For example, imagine that you set up a label asking "Do you ride mountain bikes? Y/N". Users would enter "Y" or "N" in the textbox you placed next to the label. Now figure that you want to keep a count of all

the students who ride mountain bikes. You could have the program add 1 to a counter every time it encounters a "Y" user response. The number of students using bikes is data produced by the program, but it is not data that users (or you) enter.

To access this information after it has been stored, it must be named. You cannot give a computer a request such as, "find all the answers to the question I asked in the second row of Form2."

You don't know what the specific information is. After all, you didn't enter it. You don't know how high the count will be for students using bikes, for example. Or imagine this example: you ask users to type their first names in a textbox labeled "FirstName?" You have no idea, when you create the program, what name any user will enter.

What you do know, though, is the category of information. All answers to the question "FirstName?" are first names. Though you don't know how many students use bikes, you do know that your program produces a bike count. You can, therefore, create names for the categories of information: *FirstName*, *BikeCount*. These names are called variables (Figure 3-1).

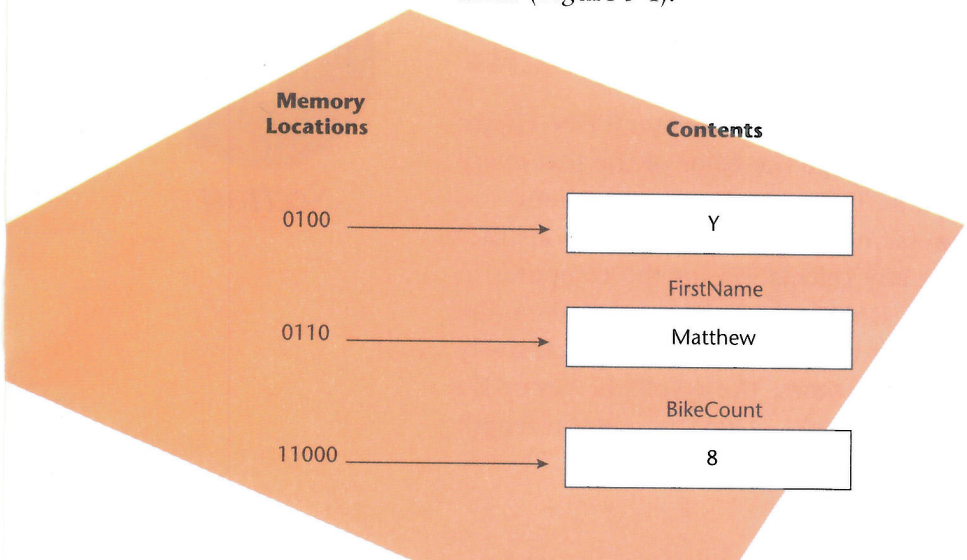


Figure 3-1
Variables as names for categories of information

Using variables, you can refer to the information that is stored as your program runs. And if you can refer to it, you can use it in operations carried out by the program. For example, you may have a variable named Rate. You may want to have the program do something, such as show the user a message on the screen, when the value of Rate reaches a certain number. To build instructions with variables, you use expressions and statements. These are discussed later in this section.

A good variable name should reflect something about the information it represents. If you used the variable name *D* instead of *Rate*, for

example, you would forget what it means before the end of the day. What will you do when you look at the program code a week from now?

There are some rules to follow as you create variable names. These names may be up to forty characters (letters or numerals) long. You should also follow the InterCap method to name variables, using capital letters to show how words or portions of words are put together. A variable representing the speed of a train might be *SpeedOfTrain*, using this naming convention.

Expressions

Expressions are used to calculate values. You build expressions from variables and operators. The variables represent data, and the operators represent the actions performed on the data. For example, if you wanted to calculate distance traveled, you would create an expression like the following:

```
AveSpeed * Hours
```

The star is an operator, joining the two variables. It indicates that you want the values of the two variables to be multiplied by each other. You can use any of the five basic math operations to join variables together in this way.

Addition	+
Subtraction	–
Multiplication	*
Division	/
Exponentiation	^

NOTE:

The symbol used in Visual Basic for exponentiation is the caret. You'll find it as a shifted "6" key on the standard keyboard.

Just as in algebra, you can use more than one operator in a Visual Basic expression. To calculate miles per gallon, for example, you might use:

```
Speed * HoursTraveled/Gallons
```

Which does the program do first, the multiplication or the division? The rules for evaluating Visual Basic expressions are the same as those you use in algebra. You might remember the phrase "Please Excuse My Dear Aunt Sally" from junior high. The letters P, E, M, D, A, and S represent the order of operations:

Parentheses
Exponentiation
Multiplication and Division in order left to right
Addition and Subtraction in order left to right

Statements

The statements of a program are the instructions you write that the computer executes. Writing a computer program means to write a list of instructions.

One of the most common types of statements you will write is an assignment statement. An assignment statement evaluates an expression and assigns the value to a variable. You use the equal sign to show that an assignment is being made. For example, these are all assignment statements:

```
Rate = 0.035
Interest = Principal * Rate * Time
C = Sqr(a^2 + b^2)
Msg = "Enter a value"
Distance = Velocity * Time
```

Each one of the statements above evaluates the expression to the right of the equal sign and assigns that value to the variable on the left side. Every assignment statement you write must be constructed in exactly this way:

variable = expression

An expression used in an assignment statement may be just a single value.

```
Sum = 0
Years = 30
Message = "Press Enter to continue"
```

The expression may also be a combination of values and variables, such as:

```
Cooktime = Pounds * 17
Layers = 2 * Folds
```

NOTE:

Visual Basic uses the equal sign as a symbol of comparison as well as for a symbol of assignment. The context in which it is used shows whether an assignment or a comparison is being made.

Translating Algebraic Expressions

You will often find yourself using Visual Basic programs to solve mathematical problems. These problems may be simple, such as figuring out the time needed for a train to move from point A to point B. Or they may be complicated, such as figuring out the rate of population growth of tadpoles in a pond in spring.

As you write the code to solve these problems, you will need to translate algebraic expressions into Basic expressions. As you have already learned, the concepts are similar, such as the order in which operations are evaluated. There are some differences, however, in how you write these expressions. The six examples here illustrate this point.

EXAMPLE 1

In algebraic notation:

$I = PRT$

where:

I = *interest*
P = *principal*
R = *interest rate*
T = *time*

This simple equation translates easily into Basic.

```
Interest = Principal * Rate * Time
```

EXAMPLE 2

In algebraic notation:

$$F = \frac{9}{5} C + 32$$

where:

F = *degrees Fahrenheit*
C = *degrees Celsius*

In Basic, this translates to:

```
Fahr = 9 * Cel / 5 + 32
```

OR

```
Fahr = 9 / 5 * Cel + 32
```

There are some differences between the two statements in Basic. In the first version, 9 is multiplied first by Cel, then the result is divided by 5 and that result is added to 32. In the second, because Basic strictly adheres to the order of operations, 9 is divided by 5 first, then the result is multiplied by Cel, then that result is added to 32. No parentheses are necessary in the second expression. By the order of operations, multiplication and division are performed in order, left to right.

EXAMPLE 3

In algebraic notation:

$$MPG = \frac{distance}{gallons}$$

This formula calculates the miles per gallon of an automobile by dividing the distance traveled by the number of gallons used. In Basic, you would write:

```
MPG = Distance / Gallons
```

EXAMPLE 4

In algebraic notation:

$$MPG = \frac{speed \times hours}{gallons}$$

The miles per gallon formula from Example 3 has been changed to speed in miles per hour times the number of hours driven in the numerator. The distance traveled by an automobile is the average speed in miles per hour times the number of hours the car travels. In Basic, this translates to:

```
MPG = Speed * HoursTraveled / Gallons
```

EXAMPLE 5

In algebraic notation:

$P = I^2R$

where

P = power in watts
I = current in amperes
R = resistance in ohms

You can express this formula in Basic in two different ways:

```
P = I * I * R
```

OR

```
P = I ^ 2 * R
```

The second version uses the symbol for exponentiation.

EXAMPLE 6

In algebraic notation:

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

This is the Quadratic Formula, the equation used to find the solutions to quadratic equations:

$ax^2 + bx + c = 0.$

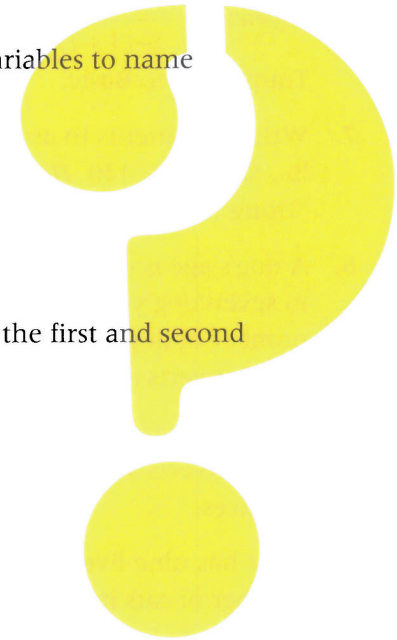
The translation to Basic is a challenge:

```
root1 = (-b + Sqr( b^2 - 4 * a * c )) / ( 2 * a )
root2 = (-b - Sqr( b*b - 4 * a * c )) / ( 2 * a )
```

The minus sign in front of the first *b* is the "unary minus" sign. It shows that the value of the variable *b* is negated. **Sqr(x)** is a built-in function that calculates the square root of the number inside the parentheses. The value of the expression inside the parentheses must not be negative. The function, if sent a negative value, will stop the program with an error. An error that occurs while the program is running is called a run-time error.

A common error when translating this equation is to write *4ac* instead of 4 * *a* * *c*. In algebraic notation, you do not have to include the multiplication symbol; in Basic, you do.

QUESTIONS AND ACTIVITIES



1. Use the InterCap method of naming variables to name the following:
   a) The height of a building
   b) The length of a hose
   c) The time of day
   d) The number of lives
   e) The distance to Racine
2. In the following expressions, what are the first and second operations performed?
   a) 7 * 8 + 5
   b) 5 + 7 * 8
   c) 5 * 7 / 8 + 5
   d) 5 + 7 / 8 ^ 2
   e) 5 * (7 + 8)
   f) (7 + 8) * (5 – 8)
   g) (7 + 8) ^ (5 – 8)

3. Translate the following equations into their equivalent Basic expressions:
   a) Calories per second = ohms x amperes squared x seconds x 0.24
   b) $H = 0.24 \times I \times R \times T$

4. Translate the following expression to Basic. Use parentheses for both the numerator and the denominator. Don't use *H* and *h* for variables. Use *BigH* for *H*, and *Littleh* for *h*:

$$F = \frac{H - h}{H' - h'}$$

5. The sine function in Basic is: **Sin(x)**. The value, *x*, must be expressed in radian form. The cosine function in Basic is **Cos(x)**. Translate the following into Basic:
   a) Take the square root of both sides of the equation, then translate to Basic.

$$c^2 = a^2 + b^2 - 2ab \text{ cosine C}$$

   b) Solve the equation for *a* by multiplying both sides by sinA, then translate.

$$\frac{a}{\sin A} = \frac{b}{\sin B}$$

6. The monthly payment on a mortgage, given the amount borrowed, *B*, the monthly interest rate, *I*, and the total number of payments, *n*:

$$payment = \frac{BI}{1-(1+I)^{-n}}$$

   Translate into Basic.

7. Write statements to assign values to these variables: *Weight* is 230 lb., *Strength* is 140, *Health* is 120, *Potions* is 5, *Spells* is 3, *FirstName* is "Doug".

8. A dog's age is expressed in dog years. A single human year is equal to seven dog years. Write an assignment statement that converts a number of dog years, *DogYrs*, to human years. Write a statement that converts a number of human years, *HumanYrs*, to dog years.

9. Every five tardies is counted as a single absence. Write a statement that converts a student's tardies into the equivalent number of absences.

10. A cat has nine lives. Write a statement that will convert a given number of cats into the equivalent number of lives.

11. The interest paid on a loan for a month is the unpaid balance times the monthly interest rate. Write a statement that assigns a value to the interest paid for a month.

12. Robert Boyle was the first person to perform experiments upon what he called the "springiness of the air." Today, Boyle's law is expressed as: The volume of a gas at constant temperature is inversely proportional to the pressure of the gas. In an "inverse proportion," when one value goes up, the other goes down. In this case, when pressure is increased, the volume is decreased, and when the volume is increased the pressure is decreased (Figure 3-2).

    Normally we express this proportion as:

    Volume times Pressure equals a constant

    This expression is not in the form of a Basic assignment statement.

    a) Solve the equation for *Volume*. Write a Basic statement for volume.
    b) Solve the equation for *Pressure*. Write a statement for pressure.
    c) Solve the equation for the constant value, *K*. Write a statement for the constant.

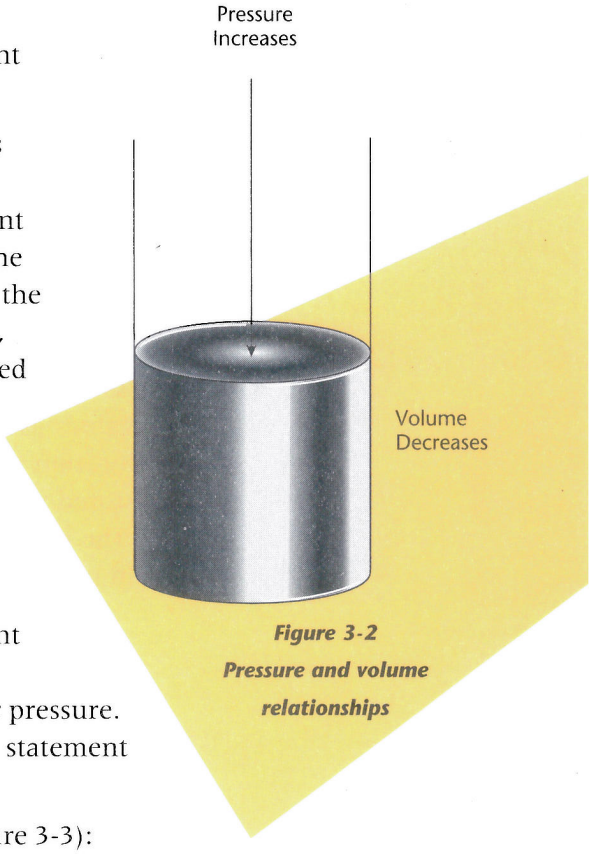


Figure 3-2
Pressure and volume relationships

13. The following finds the horsepower of a gas engine (Figure 3-3):

$$HP = \frac{D^2N}{2.5}$$

    where:
    D = diameter of the cylinder in inches
    N = number of cylinders.

    Rewrite the equation as three assignment statements.

    a) Solve for the horsepower. Write an assignment statement.
    b) Solve for the number of cylinders. Write a statement.
    c) Solve for the diameter of the cylinders. Use the **Sqr(x)** function to calculate the square root of the value. Write a statement

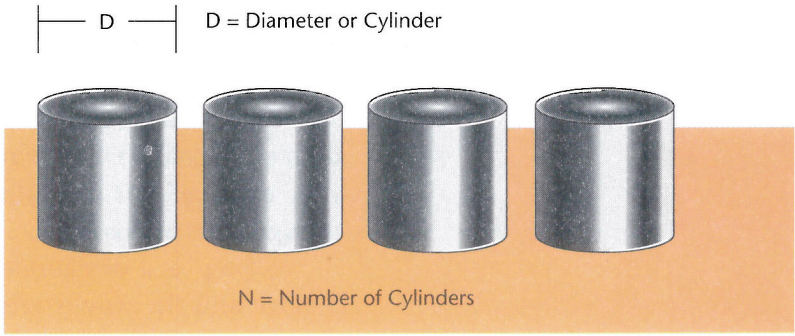


Figure 3-3
Finding the horsepower of an engine

Trying Out Expressions and Statements

This section outlines two simple projects. As you build these projects, you will become used to working with variables, expressions, and statements. At the same time, you will gain experience with the debugging capabilities of Visual Basic.

Variables and Debugging

As opposed to algebraic variables, Visual Basic variables can change values as a program runs. Remember that a variable is a name for a location in which a quantity is stored, and its value is the quantity stored there. In this sense, variables are analogous to phrases such as "the number of oranges in the refrigerator" or "the amount of money in my pocket."

The events of the day can change the amount of money in your pocket. Similarly, the statements of a program can change the contents of a location in which a variable's value is stored. A program is a process that unfolds in time. This dynamic, temporal element is absent from the expressions and equations of algebra, where each variable takes on a value that it retains throughout an entire problem.

A program is a sequence of instructions for the computer to carry out, which programmers specify in its entirety before the program is run. In effect, they build a mental model of how they want and expect the program to behave over time. Then they try to ensure that their sequence of instructions specifies the behavior they have in mind.

Programmers, however, do not always succeed. Sometimes the sequence of instructions they create will be performed in an order they didn't anticipate. Or an expression yields a value different from the one they expected. Or the value of a variable is changed to one that they thought was impossible.

A "bug" is a deviation between the way a program is supposed to behave and the way it actually does. Some bugs are obvious to every user of a program. Others will be apparent only to the program's creator.

The process of eliminating deviations is called "debugging." The most common way to debug a program is to look at the values the program assigns to variables as it runs. If the code is set up correctly, these values should be reasonable and as you expect them to be. For example, imagine that you stop a program, then find a value of 400 assigned to the variable *Age*. This would be an unreasonable value if you were storing human ages. It is a reasonable value for tree ages.

How do you stop a program while it is running so that you can look at variable values? You set break points, which means you mark spots where you want the program to stop executing. When the program is run, it pauses before each highlighted line. During the pause, you can display the values of the variables in the Debug window.

To explore this method of debugging, you will create a form with no controls. You will insert code into the form's Form-Load area, then check the values of variables used in that code.

Generally, you use this area to run code that takes care of housekeeping functions when a form is loaded into memory. You do not need any controls to run the code in this area. Each time a form is loaded, the program automatically executes the commands in the Form-Load section of the form.

Setting Up the Project

Follow these steps to create the project, insert the code, and mark breakpoints:

1. Start Visual Basic. If Visual Basic is already running, select New Project from the File menu. A form opens by default.
2. Select Environment from the Options menu. The Environment Options dialog box opens (Figure 3-4).
3. In the dialog box, set the item Require Variable Declaration to Yes. This means each program will require the declaration of variables. The advantages of this option will be discussed later in the chapter.
4. Click OK.
5. Double-click somewhere on the body of the default form. Visual Basic switches to the Code window for the form. The empty Form-Load subroutine is displayed.
6. In the subroutine, insert the declaration and assignment statements shown in Figure 3-5. Declarations will be described further in the next project.

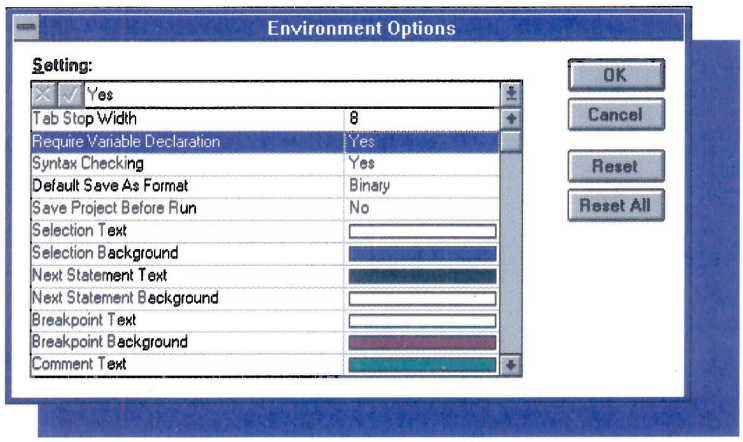
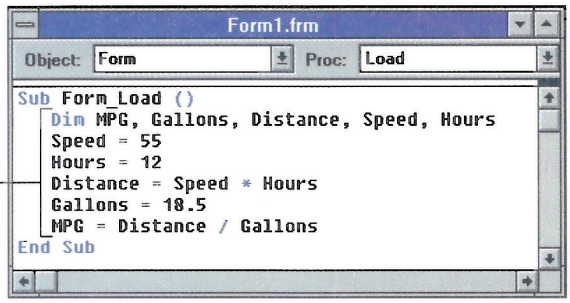


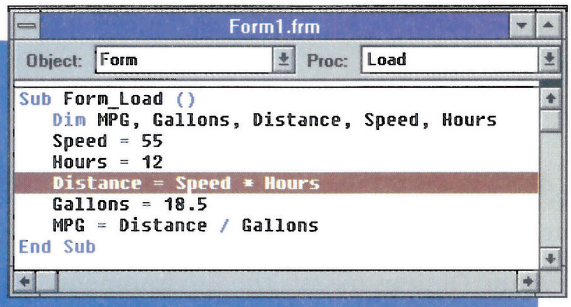
Figure 3-4
The Environment Options dialog box

Figure 3-5
The Code window



7 Click on the line **Distance = Speed * Hours**. Then click on the Break point tool in the Visual Basic toolbar (Figure 3-6).

Figure 3-6
The Code window with breakpoint marked



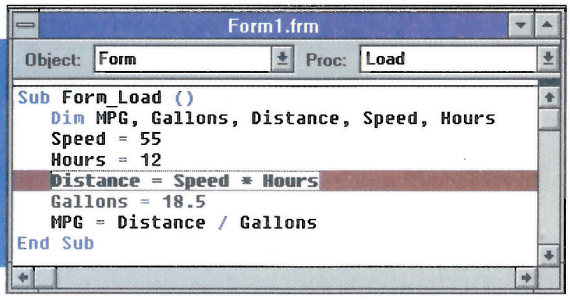
Running the Program

Now run the program:

1 Select Start from the Run menu or click on the Run button on the toolbar to run the program.

The program pauses at the line you marked. The Code window opens again, along with the Debug window (Figure 3-7).

Figure 3-7
Code window, showing break point marked



The break point is now framed. The frame shows the next statement to be executed, which is the break point line of code.

2 Click on the Debug window. If you cannot see the window, select Debug from the Window menu.

3 Use the **Print** statement to check the current value of the variables you added to the Form-Load subroutine. Type into the Debug window (Figure 3-8):

```
Print Speed
Print Hours
Print Distance
```

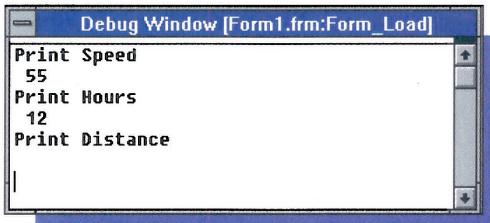


Figure 3-8
Using the Print statement

The caption bar of the Debug window shows the procedure of the form in which the program is halted. The values currently assigned to the variables are also shown. There is no value for *Distance* yet, because the line calculating the distance has not been executed.

4 Type in the Debug window (Figure 3-9):

```
Print Speed * Hours
```

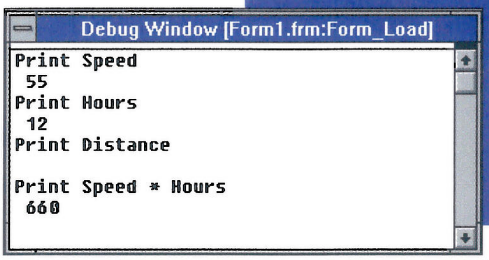


Figure 3-9
The Debug Window

The Debug window allows calculations within the **Print** statement. Still, although the value has been calculated in the **Print** statement entered into the Debug window, the value has not been assigned to *Distance*.

5 Execute the next line of the program by clicking on the Single-Step button in the toolbar (Figure 3-10).

Notice that the frame marking the next statement to be executed has moved down. The line above has now been executed.

6 In the Debug window, type:

```
Print Distance
```

7 Step through the rest of the program. After each step, print each of the variables in the Debug window.

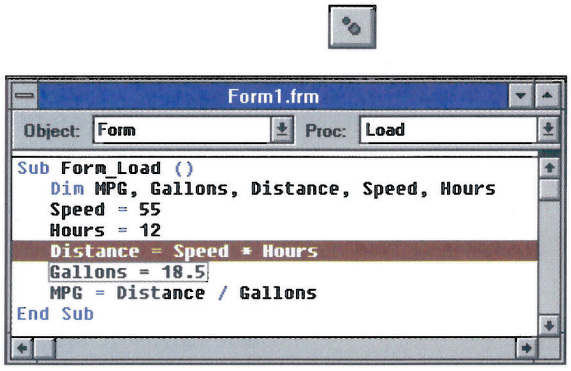


Figure 3-10
Executing the next line of code

In this exercise you will enter the lines of a program to calculate the simple interest earned by $2300 on deposit for two years. Once the program is finished, you set a break point and use the Debug window to examine intermediate values of variables.

To complete the exercise:

1. Select New Project from the File menu.
2. Enter these statements in the Form-Load procedure:

```
Dim Principal, Interest, IntRate, Duration
Duration = 2
Principal = 2300
IntRate = .045
Interest = Principal * IntRate * Duration
```

3. Set a break point on the last line.
4. Run the program.
5. Print the values of the four variables in the Debug window.
6. Single-step one line.
7. Print the value of *Interest*.

Programming a Digital Clock

This project gives you more experience with assignment statements and introduces you to the Timer control. The Timer control lets you perform actions at regular intervals.

If you were writing a game, for example, you could use the Timer to be sure that your animation ran at the same speed on all computers. You would not want the animation to run as fast as possible on all machines. If it did, it would run unplayably fast on powerful machines.

The Timer has only one significant property, the Interval property. This property is loaded with an integer. The Timer counts down from this integer. When the count reaches 0, the control generates a Timer event. You will use this event to set the caption of the digital clock.

You will use an assignment statement to display the current time in the caption of a label. The program takes the current time from a Visual Basic function called **Time**. The value of the Time function is the time of the system clock.

The controls you place on the form will include:

- Command button
- Label
- Timer

You will size the form to fit these three controls.

Figure 3-11 shows the final appearance of the form during the design phase.

Figure 3-12 shows the form's appearance while the program is running.

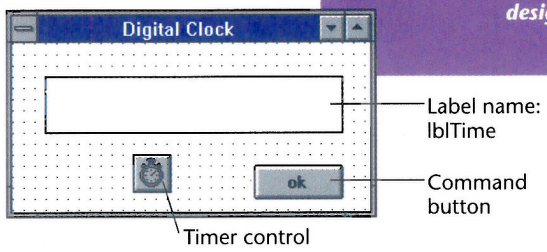


Figure 3-11
Digital Clock project in design mode

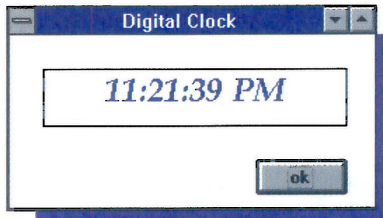


Figure 3-12
Digital Clock project as it is running

PLACING THE OBJECTS

Follow these steps to create the project:

1. Start Visual Basic. If the application is already running, select New Project from the File menu.
2. Double-click on the label icon in the toolbox. Move the label near the top of the form.

3. Put a command button on the form. Double-click on its icon in the toolbox. Move this button to the bottom-right corner of the form.

4. Put a Timer control on the form. Double-click on its icon in the toolbox. Move this button to the bottom of the form.

ALTERING THE PROPERTIES OF OBJECTS

Now you are ready to change the properties of the objects you've placed:

1. Click the label on the form to select it. In the Properties window:
   a) Change the Name to **lblTime**.
   b) Change the caption by deleting the characters.
   c) Change the BorderStyle to 1.
   d) Change the Alignment property to Center. This will center any display within the label.
   e) Change the FontName and the FontSize to something pleasing to your eye.

2 Click on the command button to select it. In the Properties window, change the caption to OK.

3 Click on the Timer control to select it. Set the Interval property to 1000.

4 Click on the body of the form to select it.

a) Change the Caption property to "Digital Clock" (Figure 3-13).

b) Change the BorderStyle to Single-Fixed.

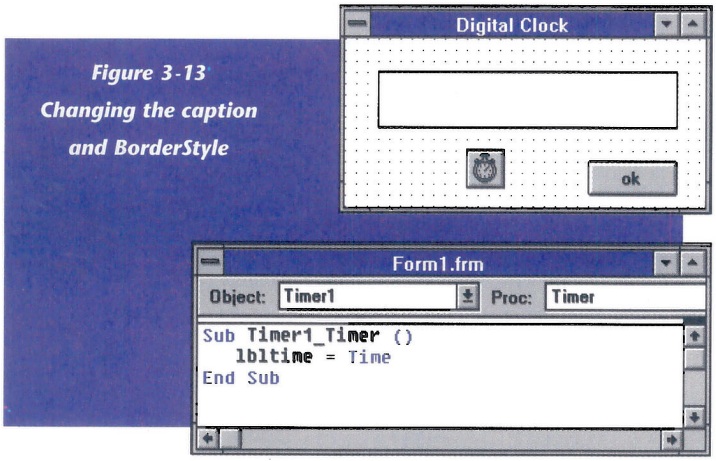


Figure 3-13
Changing the caption and BorderStyle

Figure 3-14
Code window for Timer control

5 The clock requires only one line of code. Double-click on the Timer control to open its Code window (Figure 3-14).

6 Enter the assignment statement shown in Figure 3-14. The function **Time** is the system clock time, and you are using it to update the caption of lblTime.

7 Open the Code window for the command button. In the Click routine, enter the statement: **End**.

8 Run the program.

9 Save the program. Save both the form and the project files.

3
Section

Data Types

Data types are what they sound like: types of data. Think of the different types of data users could enter as they run a program. They could enter a string of text, for example, a percentage, or an amount of money. Visual Basic defines eight different types of data that a user could enter or a program could produce.

Why do you need to know about data types? Defining data types for variables gives you a way to control what a user can enter into a textbox. Imagine, for example, that you want a user to enter a dollar amount in a textbox labeled "Cost of bike?" You do not want the program to accept other types of data, such as text.

First you create a variable to store the data the user enters into this textbox. Then, you can declare the data type of the variable as Currency and you can write some code that displays an error message if the user enters the wrong data type in the textbox.

Looking at the Different Types

The eight different data types include:

- Integer
- Long
- Single
- Double
- Currency
- String
- Variant
- User defined

Some of these data types are briefly introduced in this section. You will use others of these data types in later chapters.

NOTE:

Visual Basic version 4 adds several data types: Byte, Boolean, Date, Object, and an additional Variant type.

INTEGER TYPE

Integers are whole numbers, such as 9 or 100. Many game programs use this data type to represent values such as health points or strength.

You can use this type for whole numbers in the range between -32,768 and +32,767. The difference in the positive and negative limits is due to the way the numbers are represented in memory. Most computers use a scheme called "two's complement binary representation," and because of the way negative numbers are stored, the positive and negative ranges are different.

LONG TYPE

The Long data type is very similar to the Integer type. You use it to represent whole numbers in the range from -2,147,483,648 to +2,147,483,647. Each number of this data type requires twice as much memory to store as a number of the Integer type.

When using long integers, you might be tempted to put in a comma every three digits. While you can format numbers to be displayed with commas, Visual Basic does not recognize numbers with commas in expressions.

SINGLE TYPE

You use the Single data type to represent decimal numbers, which are numbers with a fractional part. Internally, the numbers are stored in something like a binary version of scientific notation.

SCIENTIFIC NOTATION

Scientists and mathematicians represent very small and very large numbers in scientific notation. Like the decimal number system, scientific notation is based on powers of 10. Each positive number is expressed as a decimal number between 0 and 10 times a power of 10.

23,500,000 becomes 2.35 x 10^7

10^7, ten to the seventh power, is 1 followed by 7 zeros: 10,000,000. Thus 2.35 x 10^7 is 2.35 x 10,000,000, or 23,500,000.

Variables of the Single type can represent negative numbers in the range from -3.402823E38 to -1.401298E-45, and positive numbers in the range from 1.401298E-45 to 3.402823E38. The E stands for exponent ,and it means the number that follows is used as a power of ten. The number 3.402823E38 means 3.402823 times 10 raised to the 38th power—1 followed by 38 zeros! Raising 10 to the negative 45 power, as in E-45, means one divided by 10 raised to the 45th power, or 1 over 1 followed by 45 zeros.

1/1,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000

CURRENCY TYPE

You use the Currency data type to represent dollar amounts. A variable of Single type may also be used, but in the conversion from decimal to binary and back, round-off error can occur. This error makes the Single type unsuitable for representation of dollar amounts. The Currency data type avoids round-off errors and maintains values to an accuracy of one hundredth of a cent.

STRING TYPE

The String type is used to represent characters, including:

- Text

- Special characters such as the pound sign (#), the underscore (_), and the tilde (~)
- Digits, 0 through 9

A set of characters contained in double quotes is called a string literal, or string for short. Visual Basic includes two types of variables to hold strings: fixed-length string variables and variable-length string variables. Variable-length string variables can hold strings up to about 65,000 characters long; fixed-length string variables hold only the number of characters you specify when you declare the variable.

Only strings whose length does not exceed that fixed length can be stored in a fixed-length string variable. Though variable-length strings are more common, fixed-length strings are sometimes more appropriate. For example, you may want to gather a user's first and last name for printing on a form with a box for each letter.

VARIANT TYPE

This special type can represent numbers, or strings of any type, as well as times and dates. In exchange for the versatility of using a single type for almost any kind of data, you give up memory space. A variable of the Variant type takes a lot more memory than a variable of the Integer type.

Data type	*Suffix*	*Size*	*Range*
Integer	%	2 bytes	-32,768 to 32,767
Long	&	4 bytes	+/– about 2 billion
Single	!	4 bytes	-3.4E38 to -1.4E-45 for negative values 1.4E-45 to 3.4E38 for positive values
Double	#	8 bytes	-1.8E308 to -4.9E-324 for negative values 4.9E-324 to 1.8E308 for positive values
Currency	@	8 bytes	+/– 922 trillion to 4 decimal places
String	$	variable	0 to about 65,000 bytes
Variant	none	variable	variable

Declaring Variables

Declaring a variable means writing a statement that associates a data type with a variable name. This statement announces the existence of the variable and allocates storage for it.

Once the association is made, any value of the variable will be of the declared data type. To declare a variable in Visual Basic, you use a **Dim** statement:

```
Dim DogYrs As Integer, HumanYears As Integer
Dim a As Integer, b As Integer, c As Integer
Dim chirps As Long
Dim Avogadro As Single
Dim Balance As Currency
Dim LastName As String
Dim Payment As String * 17 (fixed-length string)
```

The Variant data type is the default type for Visual Basic. If you do not declare the type of a variable, Visual Basic declares the variable to be a Variant data type by default.

```
Dim a,b,c
Dim sum, rate, lives
Dim storms, tornadoes
```

Once you have declared a variable as being a specific type, you can use it in an assignment statement. If you place this variable on the left side of an equal sign, it will be assigned the value of the expression on the right. For example:

```
Sum = 34 + 55 + 23
DogYears = 7 * HumanYears
Avogadro = 6.03E23
pi = 3.1415926
rate = 0.08
```

Initializing Variables

You initialize a variable by assigning it a starting value. Before you use a variable in the right side of an assignment statement, you need to initialize it. For example, consider the following:

```
Dim Y As Integer, Z As Integer
Y = 24 + Z
```

Because you haven't assigned a value to *Z* before using it in an assignment statement, Visual Basic gives the variable a value of 0. This may seem convenient but it can lead to trouble, as you can see here:

```
Dim Y As Integer, Zed As Integer
Zed = 48
Y = 24 + Zd
```

You may have hoped the code above would add 48 to 24 and assign the value to *Y*. Since *Zed* is misspelled as *Zd* in the second statement, *Zd* is given the default value of 0. *Y* is assigned the result of 24 + 0, or 24.

OPTION EXPLICIT

Visual Basic gives you a way to avoid this problem that is called Option Explicit. This statement requires that every variable name be declared before it is used. If a variable appears that hasn't been declared, as will be the case if a variable name is misspelled, Visual Basic will flag the use of the undeclared variable as a mistake. Figure 3-15 shows a Code window with the previous lines of sample code, the Option Explicit statement, and the error message that results when the program is run.

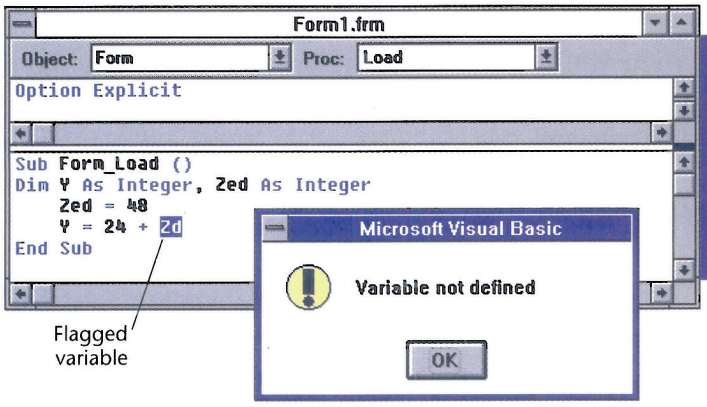


Figure 3-15
Code window and error message

VARIABLES AND THE VENUS SPACE PROBE

Failing to declare variables can have expensive implications. A NASA Venus space probe was lost due to a bug in a FORTRAN program. The bug was ultimately traced to a variable that the programmer thought was an integer. FORTRAN, though, treated it as a floating-point number. FORTRAN lacks mandatory variable declarations and has a default typing scheme whereby a variable's type is determined by the first letter of its name.

USING SUFFIXES TO INDICATE DATA TYPE

In days of yore, Basic variable names often bore suffixes indicating the data type of the variable. The percent sign showed a variable to be of the integer type, Seconds%. Visual Basic allows the same use of suffixes, with additions to cover some new data types. Generally, these suffixes do not make a program easier to read. They are included in the summary only so you may recognize them if you encounter them in code listings.

The Freefall Program

The Freefall program will give you practice working with three commonly used controls: the label, the textbox, and the command button. The user will use the textbox to enter a value into the program. The program uses labels to display a calculated value and to label the display. The command buttons control making the calculation, clearing the form, and quitting the program.

Starting Out

When you start out working on a program, you figure out what the problem is that the program will solve. You then create a plan to solve the problem.

PROBLEM STATEMENT

What is the distance in feet traveled by an object, calculated from the time the object falls? You can create a program to calculate this distance for whatever time period a user enters. The distance can be displayed in either feet or meters. To display in feet, use 32.2 feet per second squared as the acceleration of gravity. To display in meters, use 9.80 meters per second squared for that acceleration.

To calculate distance, the program needs to solve the following equation:

$$\text{Distance} = 0.5 \times \text{Grav} \times \text{Time}^2$$

This equation calculates the distance traveled by an object falling freely with no initial velocity (no starting speed). The variables represent the following (Figure 3-16):

- *Grav* = acceleration of gravity
- *Time* = time the object is falling
- *Distance* = distance that object falls from its starting point

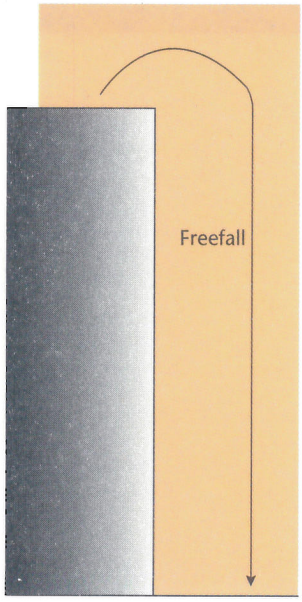


Figure 3-16
A freefalling object

THE PLAN

To solve the problem, you sketch out the form shown in Figure 3-17. You decide to use a textbox for the time, a label for the distance, and three command buttons. The command buttons execute the following actions:

- One button calculates and displays the distance.
- A second clears the textbox and the Distance label. In addition, clicking on this button selects the textbox used for entering the time.

- Clicking on the third command button ends the program.

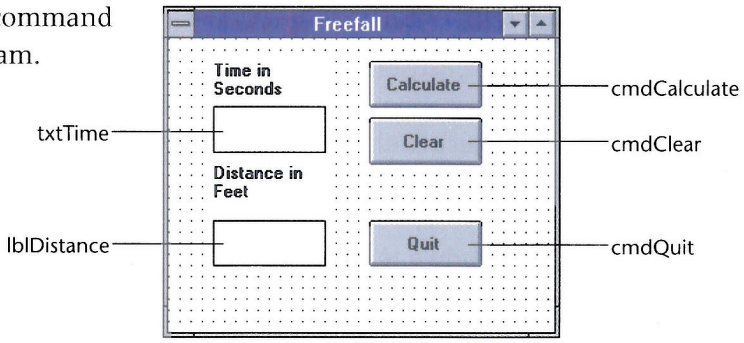


Figure 3-17
The finished Freefall form

Opening a Form and Changing Its Caption

You use the caption at the top of a form to indicate the form's purpose. Each form should have a descriptive name. Start this program by changing the default form's caption to "Freefall".

1. Start Visual Basic.
2. Click on the top line of the form and look at the Properties window. This window lists the properties of the selected object.
3. Select the Caption property and type the word **Freefall** (Figure 3-18).

   This word automatically appears in the edit area near the top of the Properties window. It also appears at the top of the form window (Figure 3-19).

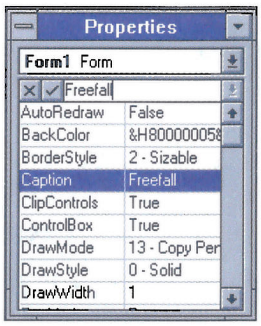


Figure 3-18
New caption for the form

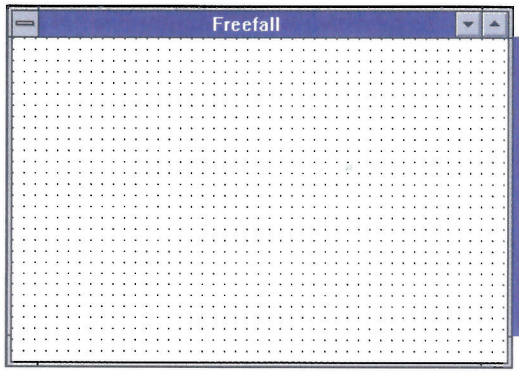


Figure 3-19
Form shown with new caption

NOTE:

The name of the form has not been altered even though the Caption property has been changed. Don't confuse the name and the caption.

Placing Objects

In these steps, you are placing the same types of objects you added to a form in Chapter 2. The form is not only the basis for the program's interaction with the user, it is the container for the program code. The textbox and the command buttons respond to events initiated by the user. These events execute the code that does the work of the program.

TEXTBOX

You need user input in this event-driven program. The user enters a length of time, and the program uses this number to calculate how far the object could fall in that time period. As discussed in Chapter 2, you need to place one or more textboxes on a form if you expect user input. In this case, you need a single textbox.

To add a textbox:

1. Double-click on the textbox icon in the toolbox.
2. Position the box by dragging it to wherever you want it (Figure 3-20).

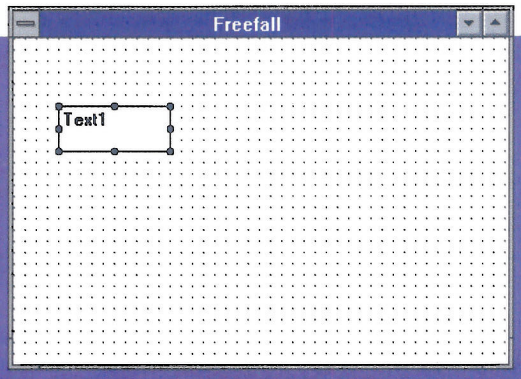


Figure 3-20
Textbox in new position

Figure 3-21
Placing a space in the textbox

3. Select the Text property from the Properties window. Replace the text with a single space by pressing the space key (Figure 3-21).
4. Change the Name property of the box by selecting the appropriate line in the Properties window, then typing txtTime (Figure 3-22).

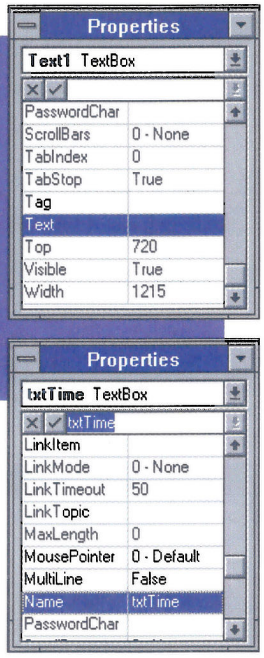


Figure 3-22
Changing the Name property

LABELS

You need two different kinds of labels for the form. One kind of label is a descriptive caption, an explanation of what the user sees on the form. The program uses the other kind of label to display calculated values. In total, you will place three labels on the form.

To add labels:

1. Double-click on the Label icon in the toolbox.

2 In the Properties window, select the Caption property for this first label. Then type a descriptive label for the Time textbox (Figure 3-23).

Change fonts, font sizes, as well as background and foreground colors to suit your fancy.

3 Position this first label by dragging it above the textbox (Figure 3-24).

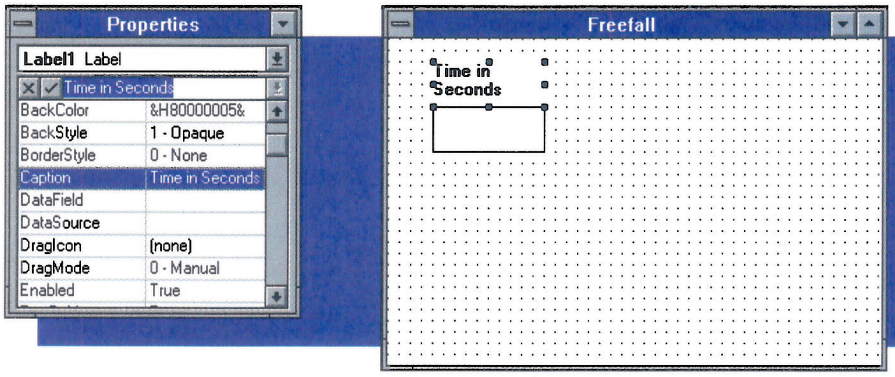


Figure 3-23
Caption for the label of the Time textbox

Figure 3-24
Aligning the label and textbox

4 Now create two more labels. Create one to display the calculated distance and another to be its caption. Change the caption of one label to be a description of the label showing the distance. Change the font, font sizes, as well as the colors to match the descriptive label created in step 2 (Figure 3-25).

5 Delete the text in the caption of the third label by pressing Backspace, then Enter. The caption will change under Program control displaying the distance fallen. Change the BorderStyle property to 1 - Fixed Single (Figure 3-26). The values of this property are available in a list. Double-clicking on the property cycles through the values.

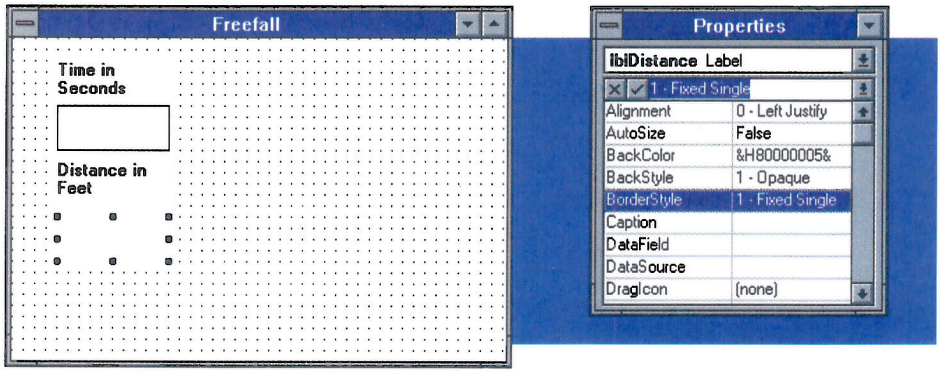


Figure 3-25
Adding more labels

Figure 3-26
Changing the Border Style property

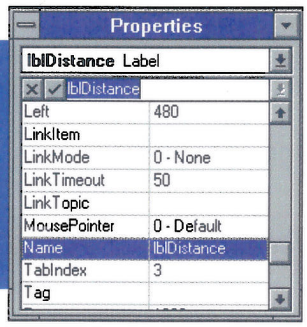


Figure 3-27
Changing the name of the third label

6 Change the name of the third label to **lblDistance** (Figure 3-27).

COMMAND BUTTONS

Adding command buttons to a form allows the user to initiate program actions. As noted in the program plan, you are adding three to the Freefall form.

You can create these buttons one at a time, as noted in the steps below. Or you can try the approach explained in the note.

1 Double-click on the Command Control icon in the toolbox.

2 Drag the first button to the top-right corner of the form (Figure 3-28).

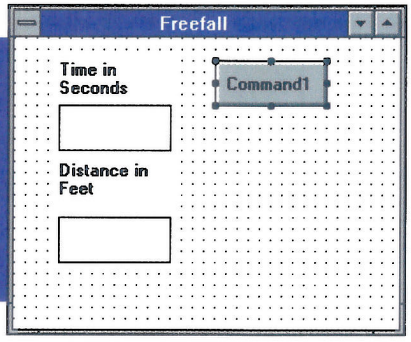
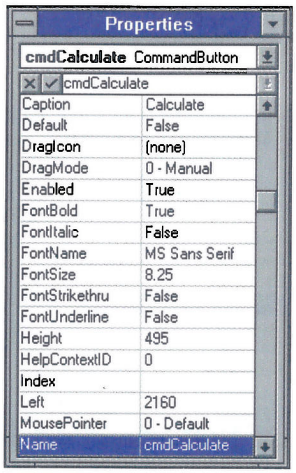


Figure 3-28
Aligning the first Command button

NOTE:

You can speed up these repetitive tasks by creating all three command buttons at once, then changing each property for all three buttons at the same time. To change the captions in all three, select the caption in the first button. Type the desired caption. Select the next button. Visual Basic automatically selects the same property for the new command button. Type your change and select the third button.



3 Change the Caption property of the button to **Calculate**, then change the Name property of the button to **cmdCalculate** (Figure 3-29).

Figure 3-29
Changing properties for the first command button

Here's the form once the changes take effect (Figure 3-30).

4 Create a second command button and drag it below the Calculate button (Figure 3-31). In the Properties window, change the value of the Name property to **cmdClear**, then the value of the Caption property to **Clear**.

5 Create a third command button and drag it below the Clear button. In the Properties window, change the value of the Name property to **cmdQuit**, then the value of the Caption property to **Quit**. Leave some white space between the top two command buttons and the Quit button (Figure 3-32).

The form is designed. It's time to start thinking about the code that will tie everything together.

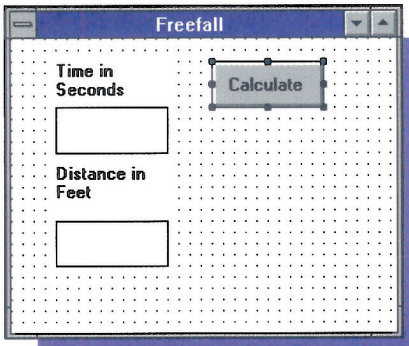


Figure 3-30 Command button with new caption

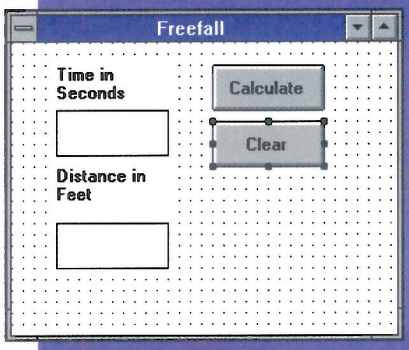


Figure 3-31 Adding a second Command button

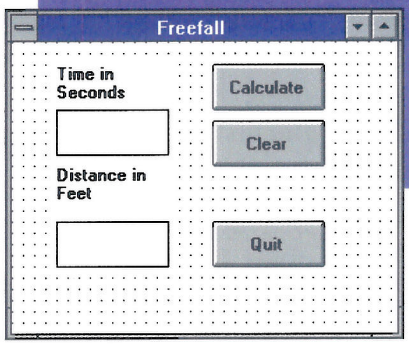


Figure 3-32 Adding a third command button

Writing Command Button Subroutines

The default event procedure for any command button is the Click event. This means that, when a user clicks on the button, the program responds to this event and executes an action. You determine what that action is by writing code in a subroutine. In this program, you want a different action to occur, depending on which button is clicked. When one of the buttons is clicked, for example, distance is calculated. The same event (a user clicking) causes a totally different action to occur with either of the other buttons.

To insert code into the Click subroutine for any of these buttons, you double-click on the button. The Code window for that button opens.

NOTE:

The name of the subroutine is the name of the object with the name of the event appended as a suffix. The empty parentheses after the name of the subroutine indicate that no information is sent into the subroutine by the event.

Visual Basic has already started the code for the routine by providing the top and bottom lines (Figure 3-33).

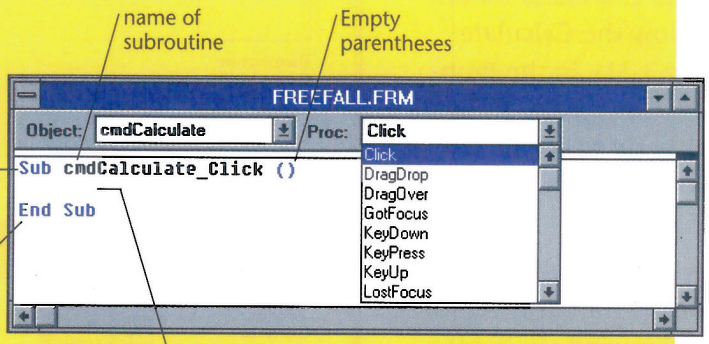


Figure 3-33
Code window showing Click subroutine

CODE FOR THE CALCULATE BUTTON

To write the code for the Calculate button:

1. Double-click on the command button, cmdCalculate, on the form. Pull down the Proc menu on the right. The Procedure menu shows the events to which a command button can respond. You'll be using the default event, Click. Insert the following code between the first and last lines of the Click subroutine:

```
Dim FallTime As Single, Distance As Single
Dim Grav As Single
Grav = 32.2
FallTime = Val(txtTime)
Distance = .5 * Grav * FallTime ^ 2
lblDistance.Caption = Distance
cmdClear.SetFocus
```

The first two of these lines declare the variables. All three variables are declared to be of the Single data type, a type used to represent decimal numbers. The third line assigns the value of 32.2, the acceleration of gravity, to the variable *Grav*.

```
FallTime = Val(txtTime)
```

This line reads the text contained in the box and assigns its value to the variable *FallTime*. However, before the assignment is made, the time, which is entered as a string, is converted to a value. Data entered into a textbox is text data. It is of the String data type. The **Val(x)** command, which is called a function, takes the value of the string, *txtTime*, and con-

verts it to a number. When a textbox name is used and no property is stated, the Text property is assumed.

This line of code is equivalent to:

```
FallTime = Val(txtTime.text)
```

This second version shows how to refer to the properties of objects: state the name of the object, put in a period and follow it with the name of the property. This method is used in programs to change properties of a textbox while the program is running.

The conversion from string to number will fail if the text in the box doesn't represent a number. If **4.5s** is entered, the program will attempt to convert this string to a number, but will fail because of the presence of the letter "s" in the string. The program will halt with a run-time error.

If the program is running within the Visual Basic environment, Visual Basic will pause the program, tell you the error, and highlight the line of code where the problem was encountered. You can fix the problem and either resume or restart the program.

If the text is a proper representation of a number, the conversion will be successful.

```
Distance = .5 * Grav * FallTime ^ 2
lblDistance.Caption = Distance
```

The first of these two lines does the calculation and assigns the value to *Distance*. The second converts the numerical value *Distance* into a string and puts that string into the Caption property of lblDistance. The Caption property is automatically displayed by the label object and the answer appears on the screen. An equivalent statement that doesn't depend on defaults is:

```
lblDistance.Caption = Str$(Distance)
```

In this statement, the **Str$(x)** function converts a number to a string. The string is assigned to *lblDistance.Caption*, which itself represents a string.

The last line of the code uses something not yet mentioned: a method. Once the value for *Distance* has been displayed, the next action is to clear the Input and Output controls (the textbox and the distance label) and select the txtTime box for the next value to be entered. The cmdClear button is intended for this purpose. This line highlights the cmdClear button so that a simple press of the Enter key will execute the cmdClear routine.

METHODS

A method is a built-in subroutine that can be applied to a particular kind of control. The **SetFocus** method is preprogrammed to shift the focus to the object to which it is appended. When a control or textbox has the focus, it receives the user's keyboard input. In fact, the control or window with the focus is, by definition, the one that is the recipient of keyboard input. When a textbox control has the focus, its visual state changes to indicate that it is ready to receive keyboard input.

CODE FOR THE CLEAR BUTTON

The code for the Clear button clears the textbox and the Display Label to prepare for more entries. To create this code:

1 Double-click on the Clear button to open the Code window (Figure 3-34).

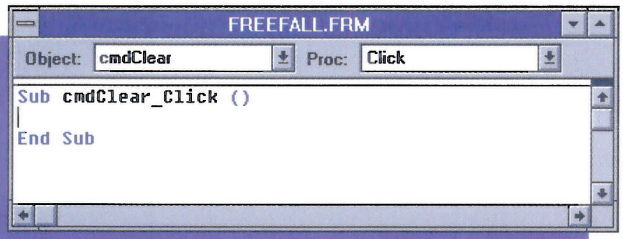


Figure 3-34
Code window for the Clear button

2 Enter the following code between the first and last lines of the Click subroutine:

```
txtTime = ""
lblDistance = ""
txtTime.SetFocus
```

The first two lines again depend on Visual Basic's default to the Text property of a textbox and the Caption property of a label when just their names are used. The Text and Caption properties of the controls are blanked by setting them equal to a string with no characters. The final line highlights the txtTime box so that a user can enter another value, if desired.

CODE FOR THE QUIT BUTTON

When the user clicks on the Quit button, the form closes. To create this code:

1. Double-click on the Quit button to open the Code window.
2. Enter **End** between the first and last lines of the cmdQuit_Click subroutine

GETTING AROUND IN THE CODE WINDOW

After the code is entered, you may want to review or revise it. You can double-click controls to enter the Code window; but once the Code window is open, it is easier to use the Object menu to move from one subroutine to another. Figure 3-35 shows the Code window with the Object drop-down list displayed.

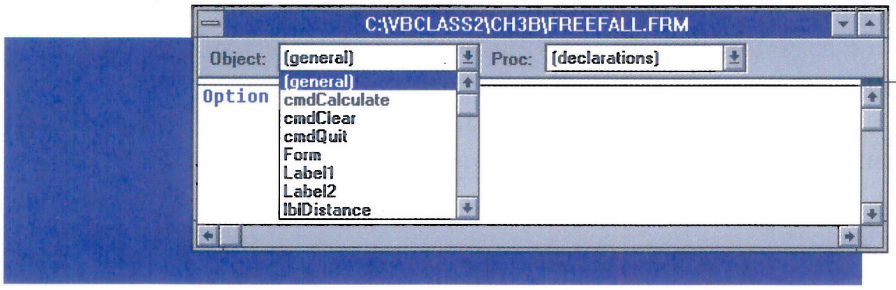


Figure 3-35
Object drop-down list

The Object drop-down list shows the names of each of the objects present on the form, including ones that currently have no code associated with them. The default names of the two descriptive labels appear in the menu.

Selecting the name of an object from the list displays the subroutine for that object. If there is code connected with more than one event procedure, you can display that code by choosing the procedure name from the Proc drop-down list.

When you want to move code from one procedure to another or just compare the code of two procedures, you can display both procedures in the Code window. As you can see in Figure 3-35, there is a narrow bar at the top of the Code window. Use the mouse to drag the bar to the middle of the window. Click in the bottom window and choose the cmdCalculate routine from the Object menu. Figure 3-36 shows the mouse cursor while the separator bar is being dragged.

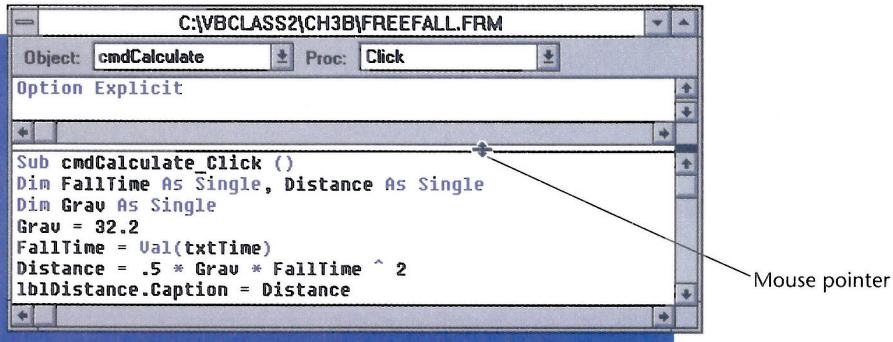


Figure 3-36
Divided Code window

SHIFTING FOCUS

In Windows, clicking an object selects it. Usually, some change in the appearance of the object shows that it has been selected. When an object is selected while the program is running, it is said to "have the focus." You can control the focus of a Windows program both by adding code and by setting the properties of the objects. Start by looking at the properties that control the focus.

The Properties window for a textbox, label control, or a command button contains the following properties:

- TabIndex
- TabStop

The focus is shifted from one object to another when the Tab button is pressed. The order in which the objects are highlighted is determined by the object's TabIndex. The indexes start at 0. The object with TabIndex 0 will receive the focus first when the program is run. By changing the TabIndex value you can change the order in which objects will receive the focus in response to the Tab key (Figure 3-37).

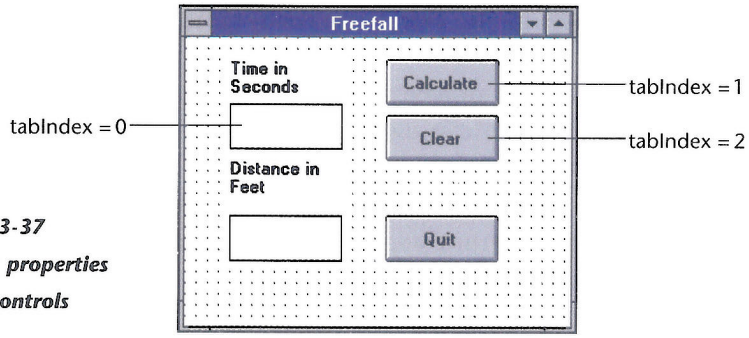


Figure 3-37
The TabIndex properties of three controls

The TabStop value, which can have the values True or False, determines whether or not the focus will stop at an object at all when the Tab button is pressed.

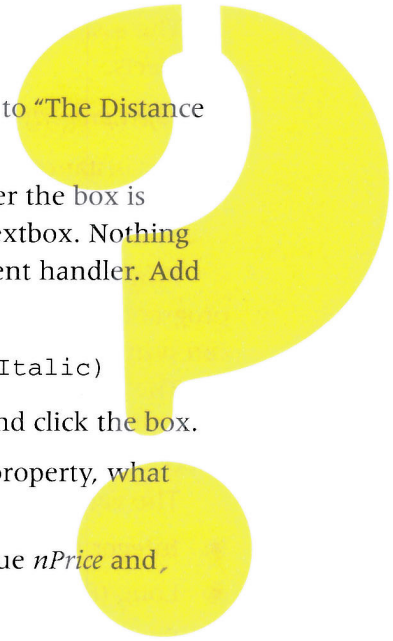
If the tab order is changed by changing the value of the TabIndex, the other objects are automatically adjusted to make room for the reordered values. In our program, it would make sense to set the TabIndex of txtTime to 0, the TabIndex of cmdCalculate to 1, and the TabIndex of cmcClear to 2. After a user enters a value for the time in the txtTime box, pressing the Tab key will shift the focus to the cmdCalculate button. The TabStop property for the other form controls should be set to False.

Finishing the Program

Now you are ready to finish the program:

1. Run the program by pressing F5 or by clicking on the Run button in the toolbar. The Run, Pause, and Stop buttons are designed to resemble buttons on CD players and cassette tape recorders.
2. Stop the program by clicking on the Quit button on the Freefall form or on the Stop button on the Visual Basic toolbar.
3. Save the program by selecting Save Project As from the Visual Basic File menu. Save both the form file and the project file.

QUESTIONS AND ACTIVITIES



1. Change the caption of the Freefall program form to "The Distance Fallen Program".
2. The Click event of a textbox is executed whenever the box is clicked. Run the Freefall program and click the textbox. Nothing happens because there is no code in the Click event handler. Add this line to the txtTime_Click () event handler:

   ```
   txtTime.FontItalic = Not (txtTime.FontItalic)
   ```

   Run the program, enter a value in the textbox, and click the box.
3. When you use the name of a textbox without a property, what property is referenced by default?
4. Use **Val( )** and **Str$( )** to convert *sPrice* to the value *nPrice* and *nQuantity* to the string *sQuantity*.

5. How is the TabIndex property used to control the movement of the cursor?
6. Describe what focus means.
7. What is a method and what does the SetFocus method do?

Variables are names of values contained in memory locations. Variables are named with the InterCap method. You build variable names by joining words or portions of words. Capitalize the first letter of each word or word portion.

You can use these arithmetic operations in Visual Basic: exponentiation (^), multiplication (*), division (/), addition (+), and subtraction (–). You can remember the order in which these operations are executed with "Please Excuse My Dear Aunt Sally:"

- Parentheses
- Exponentiation
- Multiplication and division in order, left to right
- Addition and subtraction in order, left to right

The assignment statement evaluates an expression on the right side of the equal sign and assigns that value to the variable on the left side.

variable = expression

The assignment statement is also used to assign values to properties of objects.

object.property = expression

For instance,

```
txtLastName.FontSize = 24
```

Program break points are set from the toolbar. A break point stops program execution at a particular line. While the program is halted, you can print or change the values of variables from the Debug window.

The Single-Step button in the toolbar executes one line of code with each click. After a line is executed, the program halts, allowing you to check and alter values in the Debug window.

- Currency (@)
- String ($)
- User-defined

The Variant type is capable of representing any of the other types.

The **SetFocus** method, a built-in subroutine, shifts the focus from control to control. **txtTime.SetFocus** shifts the focus to the object named txtTime.

The TabIndex property present in most objects lets you set the order in which the focus will shift from object to object when the Tab key is pressed.

The **Str$(n)** function turns a value into a string.

The **Val(s)** function turns a string into a value.

1. The Material Problem

Write a program to calculate the cost of buying material for a dress. Use a textbox to enter the number of yards of material. Use a label to display the final cost. Use $8.50 as cost per yard of the material. The final cost is the cost per yard times the number of yards. Don't include the dollar sign in the cost per yard.

2. The Material Problem (Part II)

Modify the program above by adding another textbox for the cost per yard. In this program, the user enters both the number of yards and the cost per yard. The final cost is calculated and displayed as above.

3. The Pet Problem

Write a program to calculate and display the cost of buying a pet. Use textboxes to enter values for the following:

a) Purchase price of pet
b) Veterinary fee for checking pet
c) Accessories

Calculate and display the sum in a label as the total cost.

4. The Prom Problem

Write a program to calculate and display the cost of going to the prom. Use textboxes to enter values for the following:

a) Clothing
b) Transportation
c) Flowers
d) Dinner
e) Tickets

Calculate and display the sum in a label as the total cost.

5. The Test Average Problem

Write a program to calculate and display the average of three test scores. Use text boxes to enter values for the three test scores. Calculate the sum of the scores and divide by three to calculate the average. Display the average in a Label.

6. The Semester Average Problem

The semester average is a combination of the two quarter averages and the final exam score. One way to calculate this average is to multiply each quarter average by 2 and find the sum of the three values:

a) quarter 1 x 2
b) quarter 2 x 2
c) final exam

Use text boxes to enter the two quarter averages and the final exam score expressed as a percentage. Divide this result by 5 and display as the Semester average.

7. The Cost-Per-Disk Problem

Write a program to calculate the cost per disk when disks are bought in packages of 12. Use a textbox to enter the price per box. A label displays the cost per box divided by 12.

8. The Universal Gravitation Problem

The force of gravity f generated by two objects of masses m_1 and m_2 is given by the equation:

$$f = G\frac{m_1 m_2}{r^2}$$

where r is the distance between the two masses and G is the universal gravitational constant. The two masses m_1 and m_2 are measured in kilograms (kg) and r is measured in meters (m); G has the value $6.672 \times 10^{-11} N \cdot kg^2/m^2$. The force of gravity, f, is measured in Newtons (N). Write a program using textboxes to enter the two masses and the distance between them, then calculate and display the resulting force of gravity. Make sure your textboxes are labeled properly and the force is displayed in a label.

9. Law of Cosines

The law of cosines lets us calculate the side of a triangle. It states:

$$c^2 = a^2 + b^2 - 2ab\cos C$$

where a,b, and c are the sides of a triangle and C is the angle opposite side c. If we know the lengths of two sides a and b of a triangle, and the measure of the angle C between them, we can calculate the length of the third side c by using the law of cosines and taking the square root of each side of the equation.

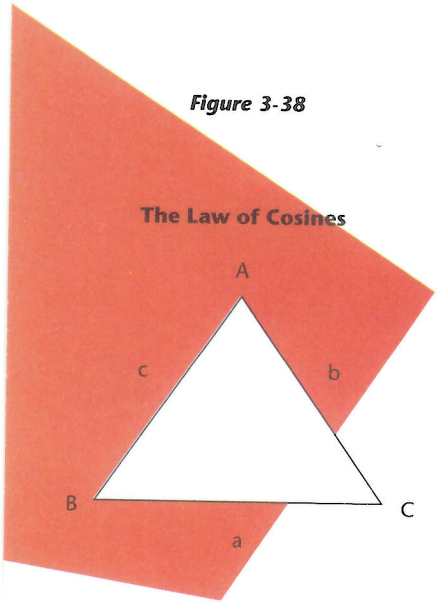


Figure 3-38

Write a program using textboxes to enter values for a, b, and C; the program should calculate *c* and display its value in a label. Arrange the boxes on the form to suggest their relationship to the parts of the triangle they represent. Use the drawing tool to draw the triangle between them.

10. The Drag Race Problem

The distance a race car undergoing a constant acceleration travels over a given period of time is given by the formula:

$s = \frac{1}{2}at^2$

where s is the distance traveled, a is the acceleration, and t is the elapsed time. Write a program that prompts the user to enter the elapsed time in seconds. Assuming the distance traveled is one-quarter mile (1320 feet), calculate and display the acceleration in feet per second squared. Start by solving the equation for a.

11. Concrete

Write a program to calculate the cost of concrete for a sidewalk. Use textboxes to enter the length and width of the sidewalk in feet. Assume the concrete is poured 4 inches thick. Calculate the volume of the sidewalk by multiplying the length times the width times the depth. Make sure the units agree. Display the volume in both cubic feet and cubic yards. Assume the cost of concrete is 60 dollars per cubic yard. Display the cost of the concrete.

12. The Interest Rate Problem

Calculate the interest rate being charged for a loan if the following information is known:

a) Amount borrowed
b) Amount of the monthly payment
c) Number of years of the loan

Assume for this problem that the number of payments per year is 12.

The calculation is done in three parts. Use textboxes to enter the principal, P; the number of years, Y; and the monthly payment, M. Calculate and display the total payments, TP:

$TP = 12 \times Y \times M$

Calculate and display the finance charge, F:

$F = TP - P$

Finally, calculate and display the effective interest rate, R:

$$R = \frac{2 \times 12 \times F}{P(Y \times 12 - 1)}$$

```
End if
x1 = x2
x2 = 0
Then
Else
For x = 1 to 10
Print x; ") My
ame is Joclyn"
nd If
If x1 - x2 = Then
```