Chapter 1 – Here's Some Source Code

InternetBasic is a very easy programming language to learn. But, don't be fooled – it's also a very powerful programming language. Through the course of this book, you will discover why both of these statements are true.

In this chapter, we're going to look at some very simple code. All of the sample programs mentioned in this chapter perform text input/output in the InternetBasic client window. (Other types of screen I/O – such as web pages, Windows controls, and dialog boxes – are explained in later chapters.)

This chapter contains the source code for several sample IB programs. In the first example, every instruction is explained in detail. In the subsequent examples, only the major features are described. By studying the code (including the comments in the source programs), you will see how easy it is to understand this language. All of these samples are contained on the enclosed compact disc. We invite you to study these programs, compile them using the free IB compiler, and run them using the free IB runtime system.

Every programming language has rules. Here's the most important rule for IB programs:

> IB source programs consist of two sections. The first section contains definitions of the data for the program, such as variable names, sizes, and types. The second section contains the executable code.

Here's what this looks like:

Data section
Code section

The data section of the program contains information about the data you want to use in the code section. You need to decide a few things before you define each data field, including these questions:

1. How many characters will be stored in each data field?
2. What names will be chosen for each field?
3. Will this data field be used in one program only, or in a series of programs that are run one after another (such as overlays and subprograms)?

For example, suppose you want to define a data field to store a phone number. Here are some possible answers for the above questions:

1. The phone number field will be 20 characters (an arbitrary value for the sake of this example).
2. Let's name this field PHONE$ (the trailing $ means that it's a string field, not a numeric one)
3. Let's use this field in one program only.

As a result of these choices, we can write the following information in the data section of the source program:

```
LENGTH 20
LOCAL PHONE$
```

The first statement means that we're defining a string field with a length of 20 characters. The second statement means we're defining a field named PHONE$ that will be used in one program only (LOCAL).

(See Chapter 2 for some more information about the data section.)

Once the data section has been written, you can write some executable code. Using the phone number field from above, here are a few examples of IB executable code:

Code	Description
PHONE$ = "(555) 555-5555"	Assign a specific value to PHONE$
INPUT PHONE$	Input a value from the keyboard
PRINT PHONE$	Print (display) the value of PHONE$

Let's look at a complete IB program. The following source program shows the data section and code section for a program that converts between the Fahrenheit and Celsius temperature scales. The data section contains four data fields, and the code section contains instructions that accept input data from the keyboard, process the data, and display the results on the screen.

Here's the source code:

```
! Define the variables

      LENGTH 6.2
      LOCAL Fahrenheit,Celsius

      LENGTH 1
      LOCAL Choice$,Answer$

! Here's the code

Top:  Print (CS)

      Print "This program converts between temperature scales." , @(0,3)

      Print "Enter 1 to convert from Fahrenheit to Celsius, or" , @(0,5)
```

```
      Print "enter 2 to convert from Celsius to Fahrenheit."    , @(0,6)

Choice:
      Print "Your choice?" , @(0,8)
      Input @(14,8),Choice$

      SELECT CASE Choice$

        CASE ""
          STOP

        CASE "1"
          Print "Enter Fahrenheit temperature:" , @(0,10)
          Input @(30,10),Fahrenheit
          Celsius = (Fahrenheit - 32) / 1.8
          Print "Celsius temperature is:",@(0,12) ; Celsius,@(30,12)

        CASE "2"
          Print "Enter Celsius temperature:" , @(0,10)
          Input @(30,10),Celsius
          Fahrenheit = Celsius*1.8 + 32
          Print "Fahrenheit temperature is:",@(0,12) ; Fahrenheit,@(30,12)

        CASE ELSE
          Print "Please enter 1 or 2." , @(0,10)
          GOTO Choice

      ENDSELECT

      Print "Would you like to perform another conversion? (Y/N)" , @(0,14)
      Input @(52,14),Answer$

      Answer$ = UCASE(Answer$)

      IF Answer$ = "Y" THEN
        GOTO Top
      ELSE
        STOP
      ENDIF

END
```

Here's what the screen looks like when this program is run. In this case, the user chose option 1 (to convert from Fahrenheit to Celsius), and then entered 212 as the Fahrenheit temperature. The program converted this value and displayed the answer.

```
This program converts between temperature scales.

Enter 1 to convert from Fahrenheit to Celsius, or
enter 2 to convert from Celsius to Fahrenheit.

Your choice?  1

Enter Fahrenheit temperature:  212.00
```

```
Celsius temperature is:        100.00

Would you like to perform another conversion? (Y/N) __
```

Here's another run of the program. This time the user chose option 2, and then entered a Celsius value of 37. The program converted this to Fahrenheit and displayed the answer.

```
This program converts between temperature scales.

Enter 1 to convert from Fahrenheit to Celsius, or
enter 2 to convert from Celsius to Fahrenheit.

Your choice?  2

Enter Celsius temperature:      37.00

Fahrenheit temperature is:      98.60

Would you like to perform another conversion? (Y/N) __
```

Here's a line-by-line explanation of the temperature conversion program:

```
! Define the variables
```

This is a comment. When the IB compiler sees an exclamation mark (!) it ignores the remainder of the source line. (The only exception to this rule is when an exclamation mark appears inside of quotation marks, in which case the IB compiler treats as part of a character string.)

```
      LENGTH 6.2
```

This is the first line in the data section of the program.

LENGTH 6.2 means that this program is going to have one or more numeric variables with a length of 6 digits and a precision of 2 digits. Such variables can store values from –9999.99 to 9999.99, inclusive.

Note: IB numeric variables can store a maximum length 16 digits and a maximum precision of 15 digits. For example, LENGTH 16.2 defines a 16-digit variable with 2 of those digits to the right of the decimal point.

```
      LOCAL Fahrenheit,Celsius
```

There are the numeric variables, appropriately named Fahrenheit and Celsius. (Since this statement appears after a LENGTH 6.2 statement, these variables are created with a length of 6 digits and precision of 2 digits.)

LOCAL means that these variables are used in this program only. (In later examples we will describe the other classification for variables, which is COMMON.)

```
      LENGTH 1
```

This defines one or more string variables with a size of 1 character.

Note: IB string variables can store a maximum of 254 characters. For example, LENGTH 100 defines a 100-character string variable, while LENGTH 254 defines a 254-character string.

```
      LOCAL Choice$,Answer$
```

The string variables are named Choice$ and Answer$. As with the numeric variables defined above, these variables are defined as LOCAL, meaning they are used in this program only.

```
! Here's the code
```

This is another comment. The IB compiler ignores this line when the program is compiled.

```
Top:  Print (CS)
```

This is the first line in the code section of the program.

This line includes a statement label (Top:) that is referred to by a subsequent statement in the program. The Print statement is one of the output statements in IB. The (CS) control code clears the screen in the IB client window.

```
      Print "This program converts between temperature scales." , @(0,3)

      Print "Enter 1 to convert from Fahrenheit to Celsius, or" , @(0,5)
      Print "enter 2 to convert from Celsius to Fahrenheit."    , @(0,6)
```

These Print statements display text in the IB client window. The first message is displayed starting at column 0, row 3 (Note: column 0, row 0 is the upper, left-hand character position on the IB client screen.) The second and third messages are displayed on rows 5 and 6, respectively.

Note: The positioning parameter is separated from the text by a comma. The space on either side of the comma is optional.

```
Choice:
```

This is a statement label that is referred to by a subsequent statement in this program.

Note: A statement label can appear on a line by itself (as shown here) or on a line with executable code (as shown in the Top statement label earlier in this program).

```
Print "Your choice?" , @(0,8)
```

This statement displays a message (a prompt) starting at column 0, row 8.

```
Input @(14,8),Choice$
```

This statement moves the cursor to column 14, row 8, and creates a one-character space on the screen. This space is an input field where the user types a value. When the user types a value and presses the Enter key, that value is assigned to the string variable named Choice$.

Note: The Input statement creates a space equal to the length of the variable. In this case, the Choice$ variable has a length of 1 character, so the input field is created with a length of 1 character.

```
SELECT CASE Choice$
```

This section of the program evaluates one or more values of Choice$. The first statement in this structure, SELECT CASE Choice$, identifies the variable that is examined in the subsequent statements. The structure ends with the ENDSELECT statement.

```
  CASE ""
    STOP
```

CASE lists a specific value of Choice$ to be evaluated. In these lines of code, the value is null (indicated by the double quotation marks). When a user presses the Enter key at an Input statement without typing a data value, IB sees this as a null string. This program assumes that this event means that the user wants to quit the program, so the STOP statement is executed.

```
  CASE "1"
```

The next four lines in this program are executed only if Choice$ equals “1”. (According to the messages displayed on the screen at the beginning of the program, the user enters “1” to convert from Fahrenheit to Celsius.)

```
    Print "Enter Fahrenheit temperature:" , @(0,10)
    Input @(30,10),Fahrenheit
```

The Print statement displays a message at column 0, row 3. The Input statement creates an input space for the Fahrenheit variable at column 30, row 3. Since this variable has been defined with a length and precision of 6.2, the input field is 8 characters wide (6 for the digits, one for a possible decimal point, and one for a possible minus sign).

```
    Celsius = (Fahrenheit - 32) / 1.8
```

This one is easy. The code converts the Fahrenheit temperature and assigns the answer to the variable named Celsius.

```
    Print "Celsius temperature is:",@(0,12) ; Celsius,@(30,12)
```

Now we'll display a message at column 0, row 12, followed by the Celsius variable at column 30, row 12.

```
  CASE "2"
```

The next four lines in this program are executed only if Choice$ equals "2".

```
    Print "Enter Celsius temperature:" , @(0,10)
    Input @(30,10),Celsius
    Fahrenheit = Celsius*1.8 + 32
    Print "Fahrenheit temperature is:",@(0,12) ; Fahrenheit,@(30,12)
```

This is starting to look familiar. We display a message starting at column 0, row 10, then create an input space at column 30 for the Celsius variable. Next the code converts the Celsius value to Fahrenheit. Finally we display a message and the Fahrenheit value on row 12.

```
  CASE ELSE
    Print "Please enter 1 or 2." , @(0,10)
    GOTO Choice
```

Here's how we deal with the person who typed something other than "1" or "2" at the Input statement. The CASE ELSE statement is located after the other CASE sections, and handles all other values that are entered for Choice$. In our program, we're going to display a warning message starting at column 0, row 10, and then jump back to the statement label Choice (the place where we displayed the first question in this program).

```
ENDSELECT
```

We're done with the SELECT structure.

```
Print "Would you like to perform another conversion? (Y/N)" , @(0,14)
Input @(52,14),Answer$
```

Let's give the user the option to try another temperature conversion. First we'll display a message starting at column 0, row 14. Then we'll create a one-character input field at column 52, row 14 for the Answer$ variable.

```
Answer$ = UCASE(Answer$)
```

The UCASE function turns the user's response into an upper case value. This function lets us evaluate the answer using less code (i.e., we don't need to check for a lower-case response).

```
      IF Answer$ = "Y" THEN
        GOTO Top
      ELSE
        STOP
      ENDIF
```

This is another type of decision structure, one the shows how the IF statement works. The structure starts by testing to see if Answer$ is equal to upper case “Y” (thanks to the UCASE function from above). If so, the program jumps back to the statement label Top (which is the beginning of the code section in this program).

If Answer$ is not equal to “Y” the program stops.

```
END
```

This optional statement tells the IB compiler that it has reached the end of the source program. (Note: This is not an executable statement; it's just a way to tell the compiler to stop compiling our code.)

Now that you've looked at the source code, try running the compiled object program. The program is named SAMPLE1.

Instructions:

From the InternetBasic READY prompt, type SAMPLE1 and press the Enter key.

Take a look at the source program. The source program is named SAMPLE1.MTB.

Instructions:

??? (WinEdit)

Now try compiling this program.

Instructions:

??? (compiling from WinEdit)

You probably noticed that the beginning part of the source program contains a command we haven't discussed yet:

```
! //MTB// Src(SAMPLE1.MTB,BAS) Obj(SAMPLE1,BAS)
```

This is the scripting command for the IB compiler, and it defines the name and location of the source and object programs. In this case, the source program is named SAMPLE1.MTB and the object program is named SAMPLE1. Both files are located on the BAS directory.

Notice that the scripting command starts with an exclamation mark, the comment character in IB. Thus, the compiler does not attempt to compile these lines as part of the program itself. Instead, the compiler reads this lines and, since it is at the beginning of the source code, treats it as a script command.

All of the other IB sample programs in this book contain similar scripting commands. Let's look at a few of those samples.

SAMPLE2

This program does a simple business calculation based on three data values. The break-even point tells a business how many units must be sold in order to break even. The calculation divides the organization's fixed cost (also known as “overhead”) by the difference between the selling price and cost of a single item.

Here's what the screen display looks like for a sample set of data.

```
This program computes the break-even point.

What is the selling price?               5.37
What is the cost?                        4.65
What is the fixed cost (overhead)?   12000.00

The break-even point is:             16667 units


View revenue & expense details? (Y/N) y

Break-even details
Revenue:                             89501.79
Expenses:                            89501.55
Profit:                                   .24


Another calculation? (Y/N)
```

The source program is named SAMPLE2.MTB. Notice that the source program contains several comments that explain the IB statements.

```
! //MTB// Src(SAMPLE2.MTB,BAS) Obj(SAMPLE2,BAS)

! Define the variables

      ! Numeric variables with 2 digits precision

            LENGTH 6.2
            LOCAL Price,Cost

            LENGTH 10.2
            LOCAL FixedCost,Temp,Revenue,Expenses,Profit

      ! Numeric variable with 0 digits precision (i.e., an integer)
```

```
            LENGTH 6.0
            LOCAL BreakEvenPoint

      ! String variable with length of 1

            LENGTH 1
            LOCAL Answer$

! Here's the code

Top:  Print (CS)

      ! Display opening message

            Print "This program computes the break-even point." , @(0,3)

      ! Display prompts and input data

            Print "What is the selling price?" , @(0,5)
            Input @(39,5),Price

            Print "What is the cost?" , @(0,6)
            Input @(39,6),Cost

            Print "What is the fixed cost (overhead)?" , @(0,7)
            Input @(35,7),FixedCost

      ! Calculate the temporary answer

            Temp = FixedCost / (Price - Cost)

      ! If the fractional portion of the temporary answer is more
      ! than zero, increment the break-even point up to the next integer

            IF FPT(Temp) > 0 THEN
               BreakEvenPoint = Temp + 1
            ELSE
               BreakEvenPoint = Temp
            ENDIF

      ! Display the answer

            Print "The break-even point is:" , @(0,9)
            Print BreakEvenPoint , @(36,9) ; "units"

      ! Display prompt and input answer

            Print "View revenue & expense details? (Y/N)" , @(0,12)
            Input @(38,12),Answer$

      ! Convert the answer to upper case

            Answer$ = UCASE(Answer$)

      ! If the answer is "Y" then calcuate and display details
```

```
            IF Answer$ = "Y" THEN
              Revenue = Price * BreakEvenPoint
              Expenses = Cost * BreakEvenPoint + FixedCost
              Profit = Revenue - Expenses
              Print "Break-even details",@(0,14)
              Print "Revenue:",@(0,15) ; Revenue,@(35,15)
              Print "Expenses:",@(0,16) ; Expenses,@(35,16)
              Print "Profit:",@(0,17) ; Profit,@(35,17)
            ENDIF

      ! Display prompt and input answer

            Print "Another calculation? (Y/N)" , @(0,20)
            Input @(28,20),Answer$

      ! Convert the answer to upper case

            Answer$ = UCASE(Answer$)

      ! If the answer is "Y" then go back to the top

            IF Answer$ = "Y" THEN
              GOTO Top
            ELSE
              STOP
            ENDIF

END
```

SAMPLE3

This program plays a number guessing game. The program picks a random number between 1 and 100, and you have to guess the answer. You have up to 16 chances to get the right answer.

Run the program first, and then look at the code.

The source program is named SAMPLE3.MTB. The source program contains several comments that explain the IB statements.

This program contains some IB features that we haven't covered yet. Here's a brief overview:

1. The RND function generates a random number. There are two forms of this function. The RND(1) version “randomizes” the generator, so you'll get a different random number sequence each time you run this program. The RND(0) function generates a random number between 0 and 1. This value is scaled to the appropriate range by a simple formula.
2. The positioning parameter in the Print statement contains a variable for the row number.
3. The STR function converts a number into a string.
4. The STRIP function strips leading and trailing blanks from a string.

5. Many of the CASE statements in this program contain multiple values, separated by semicolons.

Here's the source code:

```
! //MTB// Src(SAMPLE3.MTB,BAS) Obj(SAMPLE3,BAS)

! Define the variables

      ! Numeric variables

            LENGTH 3.0
            LOCAL Number,Guess,Counter,Row

      ! String variables

            LENGTH 3
            LOCAL Number$,Counter$

            LENGTH 1
            LOCAL Answer$

! Here's the code

      Number = RND(1)   ! Randomize

Top:  Print (CS)        ! Clear the screen

      CLEAR             ! Initialize all of the variables

      ! Get random number between 1 and 100

            Number =  RND(0)*100 + 1

      ! Display opening message

            Print "Number Guessing Game" , @(0,0)
            Print "--------------------" , @(0,1)

            Print "I am thinking of a number between 1 and 100." , @(0,3)

      ! Display prompt and input data


            Print "What is your first guess?" , @(0,5)

FirstGuess: Input @(30,5),Guess

            ! Validate the input. If out of range, try again.

            IF Guess LT 1 OR Guess GT 100 THEN
                  GOTO FirstGuess
            ENDIF

            ! Set row number to 5
```

```
            Row = 5

Evaluate:

      ! Increment the counter (keep track of the number of guesses)

            Counter = Counter + 1

      ! If the counter = 16, stop the game

            IF Counter = 16 THEN
              Row = Row + 2
              Print "Sorry, you only get 16 chances.",@(0,Row)

                Number$ = STRIP(STR(Number))    ! Strip blank spaces

              Print " The number was " ; Number$ ; "."
              GOTO PlayAgain
            ENDIF

      ! See if the guess is correct, too high, or too low

            SELECT CASE Guess

GotIt:      ! They got the right number

              CASE IS EQ Number

                Row = Row + 2

      !           Display a different message, depending on the number of
guesses

                SELECT CASE Counter

                  CASE 1
                    Print "Amazing!",@(0,Row)

                  CASE 2;3;4
                    Print "Excellent job!",@(0,Row)

                  CASE 5;6;7
                    Print "Congratulations!",@(0,Row)

                  CASE 8;9;10
                    Print "Not too bad.",@(0,Row)

                  CASE IS GT 10
                    Print "Your guessing skills need some work.",@(0,Row)

                ENDSELECT

      !           Convert number and counter to strings, and strip the
      !           leading and trailing blanks (for display purposes)

                Number$ = STRIP(STR(Number))
                Counter$ = STRIP(STR(Counter))
```

```
      !           Display game-ending message

                Row = Row + 1
                Print "I was thinking of ",@(0,Row) ; Number$
                Print " and you figured it out in " ; Counter$

      !           Check for singular or plural

                IF Counter = 1 THEN
                  Print " guess."
                ELSE
                  Print " guesses."
                ENDIF

      !         Jump to the PlayAgain routine

                GOTO PlayAgain

High: !     The guess is too high

            CASE IS GT Number
              Print Guess,@(50,Row) ; "is too high."

Low:  !     The guess is too low

            CASE IS LT Number
              Print Guess,@(50,Row) ; "is too low."

            ENDSELECT

KeepGoing:  Row = Row + 1
            Print "What is your next guess?" , @(0,Row)

NextGuess:  Input @(30,Row),Guess

            ! Validate the input. If out of range, try again.

            IF Guess LT 1 OR Guess GT 100 THEN
                  GOTO NextGuess
            ENDIF

            GOTO Evaluate

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-

PlayAgain:  ! Display prompt and input answer

                  Row = Row + 2
                  Print "Play again? (Y/N)" , @(0,Row)
                  Input @(18,Row),Answer$

            ! Convert the answer to upper case

                  Answer$ = UCASE(Answer$)
```

```
            ! If the answer is "Y" then go back to the top

                  IF Answer$ = "Y" THEN
                    GOTO Top
                  ELSE
                    STOP
                  ENDIF
END
```

??? Note: This program has been changed. It now contains sound effects.

SAMPLE4

This program displays a list of coin combinations (using nickels, dimes, and quarters) where the total value is exactly $1.00. The program evaluates all combinations, rejecting the ones that don't add up to $1.00.

This program demonstrates two additional IB features:

1. The FOR/NEXT loop counts how many times the loop is executed. (If fact, this program has *nested* FOR/NEXT loops, one for each denomination of coin.)
2. The (Screen) control code sets the size of the IB client window. While the default size is 80 columns by 25 rows, the value can be changed up to a maximum of 255 by 255. In this example, the screen is set to 80 x 40.

The source program is named SAMPLE4.MTB.

```
! //MTB// Src(SAMPLE4.MTB,BAS) Obj(SAMPLE4,BAS)

! Define the variables

      LENGTH 1.0
      LOCAL QUARTERS

      LENGTH 2.0
      LOCAL DIMES,NICKELS,ROW,COUNTER

      LENGTH 3.2
      LOCAL TOTAL

      LENGTH 2
      LOCAL COUNTER$

! Here's the code

      Print (CS)                          ! Clear the screen
      Print (Screen=80,40)                ! Set screen size to 80 x 40

      Print "The following combinations add up to exactly $1.00",@(0,0)

      Print "Nickels",@(0,2);"Dimes",@(10,2);"Quarters",@(20,2)

      ROW = 2                             ! Set row value to 2
```

```
      COUNTER = 0                          ! Initialize the counter

      FOR QUARTERS = 0 TO 4                ! Loop through quarters
        FOR DIMES = 0 TO 10                ! Loop through dimes
          FOR NICKELS = 0 TO 20            ! Loop through nickels

          ! Calculate the total value

          TOTAL = NICKELS*.05 + DIMES*.1 + QUARTERS*.25

          ! If the total value equals $1.00, then display then
          ! number of coins and increment the counter

          IF TOTAL = 1 THEN
            ROW = ROW + 1
            PRINT NICKELS,@(0,ROW);DIMES,@(10,ROW);QUARTERS,@(20,ROW)
            COUNTER = COUNTER + 1
          ENDIF

          NEXT NICKELS
        NEXT DIMES
      NEXT QUARTERS

      ! Display summary message at the bottom of the screen

      ROW = ROW + 2
      COUNTER$ = STRIP(STR(COUNTER))
      Print @(0,ROW);"There are ";COUNTER$;" combinations."

      ! Display ending message and use a "dummy INPUT" to hold it there

      ROW = ROW + 2
      Print @(0,ROW);"Press ENTER to continue..."
      Input @(30,ROW),""
      STOP

END
```

SAMPLE5

This program displays the first 26 numbers in the Fibonacci sequence, where each number is equal to the sum of its two predecessors. The initial values are 0 and 1.

This program also displays the ratio of each number and its immediate predecessor. When you run this program, notice that the ratio converges very quickly to a value of 1.61803. This value is well known in mathematics, nature, architecture and art, and is called the "Golden Mean."

The source code contains several additional IB features:

1. The looping in this program is controlled by a DO loop. In this example, the loop starts with the statement DO UNTIL C > 50000. The loop structure ends with the LOOP statement.

2. This program demonstrates an important point in the way IB handles the division operation. Here's the rule: "The result of a division operation is as precise as the most precise value in the formula."

   The ratio in this program is calculated by dividing one integer by another. In order to calculate a value with 5 digits precision, the formula must contain an extra term with 5 digits of precision. Thus, the formula is 1.00000*C/B.

The source program is named SAMPLE5.MTB.

```
! //MTB// Src(SAMPLE5.MTB,BAS) Obj(SAMPLE5,BAS)

! Define the variables

      LENGTH 2.0
      LOCAL ROW

      LENGTH 5.0
      LOCAL A,B,C

      LENGTH 6.5
      LOCAL RATIO

! Here's the code

      Print (CS)                    ! Clear the screen
      Print (Screen=80,35)          ! Set screen size to 80 x 35

      Print "This program displays numbers in the Fibonacci sequence",@(0,0)
      Print "and the ratio of each number and its preceding value.",@(0,1)

      Print "Number",@(0,3);"Ratio (also known as the 'Golden Mean')",@(10,3)

      A = 0                         ! Set the initial values
      B = 1

      Print A,@(0,4)                ! Print the initial values
      Print B,@(0,5)

      ROW = 5                       ! Set the row counter

DO UNTIL C > 50000                  ! Start loop & set upper limit

      C = A + B                     ! Find the next Fibonacci number

      RATIO = 1.00000*C/B           ! Calculate the ratio C:B
                                    !
                                    ! Note: This formula requires
                                    ! an extra term (1.00000) to
                                    ! force the precision of the
                                    ! result to 5 digits.

      ROW = ROW + 1                 ! Increment the row counter

      Print C,@(0,ROW);RATIO,@(10,ROW)    ! Display the value, ratio
```

```
      A = B                          ! Exchange the values
      B = C

LOOP                                 ! Loop


      ! Display ending message and use a "dummy INPUT" to hold it there

      ROW = ROW + 2
      Print @(0,ROW);"Press ENTER to continue..."
      Input @(30,ROW),""
      STOP

END
```

SAMPLE6

This program shows what happens when you start with one cent and double it every day for a month.

This program contains features shown in earlier examples (screen resizing, FOR/NEXT loop, variable positioning, etc.), but contains one noteworthy feature – a very large numeric variable. The AMOUNT variable is defined with a length and precision of 16.2, meaning that the variable can hold up to 16 digits, two of which are to the right of the decimal point.

The source program is named SAMPLE6.MTB.

```
! //MTB// Src(SAMPLE6.MTB,BAS) Obj(SAMPLE6,BAS)

! Define the variables

      LENGTH 2.0
      LOCAL ROW

      LENGTH 2.0
      LOCAL I

      LENGTH 16.2                                ! That's a big number!!!
      LOCAL AMOUNT

! Here's the code

      Print (CS)                                 ! Clear the screen
      Print (Screen=80,40)                       ! Set screen size to 80 x 40

      Print "If you start with 1 cent and double it every day,",@(0,0)
      Print "how much money will you have within a month?",@(0,1)

      Print "Day",@(0,3);"Amount",@(21,3)

      AMOUNT = .01                               ! Set the initial amount
      ROW = 4                                    ! Set the row counter
```

```
      FOR I = 1 TO 31                              ! Loop from 1 to 31

            Print I,@(0,ROW);AMOUNT,@(10,ROW)      ! Print day #, amount

            AMOUNT = AMOUNT * 2                    ! Double the amount

            ROW = ROW + 1                          ! Increment row counter

      NEXT I                                       ! Loop

      ! Display ending message and use a "dummy INPUT" to hold it there

      ROW = ROW + 1
      Print @(0,ROW);"Press ENTER to continue..."
      Input @(30,ROW),""
      STOP

END
```

MENU1

The final program in this chapter is one that ties together the other examples. This is a menu program. Here's what you'll see when you run this program:

```
Choose one of the following programs
------------------------------------

1 -- Temperature conversion

2 -- Break even calculation

3 -- Number guessing game

4 -- Coin counting example

5 -- Fibonacci numbers and the 'Golden Mean'

6 -- What happens if you double your money every day?

Your choice: ___
```

This program shows how one IB program can run another. The source program contains a SELECT structure, where each valid option is evaluated and carried out with multiple CASE statements. The RUN statement executes the named program. For example, if the user enters a 1, the program executes the following statement:

```
RUN "SAMPLE1"
```

The same reasoning applies for the other options.

The source program is named MENU1.MTB. Here is the code:

```
! //MTB// Src(MENU1.MTB,BAS) Obj(MENU1,BAS)

! Define the variable

      LENGTH 1                          ! One byte
      LOCAL Choice$                     ! Local variable

! Here's the code

Top:  Print (CS)                        ! Clear the screen

      ! Display the menu items

      Print "Choose one of the following programs" , @(0,3)
      Print "-----------------------------------" , @(0,4)
      Print "1 -- Temperature conversion" , @(0,6)
      Print "2 -- Break even calculation" , @(0,8)
      Print "3 -- Number guessing game" , @(0,10)
      Print "4 -- Coin counting example" , @(0,12)
      Print "5 -- Fibonacci numbers and the 'Golden Mean'",@(0,14)
      Print "6 -- What happens if you double your money every day?",@(0,16)

      Print "Your choice:" , @(0,18)     ! Display prompt
      Input @(13,18),Choice$            ! Input choice

      SELECT CASE Choice$               ! Begin case structure

            CASE ""                     ! If input is null,
            STOP                        ! stop the program

            CASE "1"                    ! If input is "1"
            RUN "SAMPLE1"               ! RUN "SAMPLE1"

            CASE "2"                    ! If input is "2"
            RUN "SAMPLE2"               ! RUN "SAMPLE2"

            CASE "3"                    ! If input is "3"
            RUN "SAMPLE3"               ! RUN "SAMPLE3"

            CASE "4"                    ! If input is "4"
            RUN "SAMPLE4"               ! RUN "SAMPLE4"

            CASE "5"                    ! If input is "5"
            RUN "SAMPLE5"               ! RUN "SAMPLE5"

            CASE "6"                    ! If input is "6"
            RUN "SAMPLE6"               ! RUN "SAMPLE6"

            CASE ELSE                   ! Otherwise
            GOTO Top                    ! Go to statement label "Top"

      ENDSELECT                         ! End of case structure

END
```

Notice the CASE ELSE statement towards the end of the SELECT CASE structure. This is an "otherwise" clause, so if the user enters something other than a valid option, the program branches back to the top of the code section and re-displays the menu.

Here's how to connect all of the sample programs in this chapter. Edit the source code for SAMPLE1.MTB through SAMPLE6.MTB. Make the following replacement in all of these programs:

Original statement	Change it to this statement
STOP	RUN "MENU1"

Re-compile the six sample programs. Now run MENU1 and choose one of the six options. The six sample programs now end by running the MENU1 program, which links all of these programs together.

Summary

This chapter provided your first look at InternetBasic source code. While the examples are very simple, they demonstrate quite a few fundamentals of the IB language. Of course, there are more features, more rules, and more sample programs ahead. In the next chapter, we'll take a look at some of the IB rules.