Chapter 1

How Visual Basic Works

In This Chapter

- Looking at the Visual Basic development cycle
- Creating a Visual Basic user interface
- Understanding what BASIC code does

The whole purpose of writing a program is to make your computer do something useful. People often spend thousands of dollars for one of these machines, so it's important that the computer does something more than consume electricity and take up space on a desk.

Before you delve into the world of programming, keep two thoughts in mind. First, anyone can write a program. Programming is just a skill, much like swimming, sailing, or shoplifting. If you've ever taught yourself a new hobby or skill, you can teach yourself how to write a program without an extensive mathematics background or a fancy college degree that puts you in debt for the next ten years.

Second, the key to programming is defining exactly what you want your program to do. Defining what you want your program to do is half the battle (a battle that governments and large corporations routinely lose all the time). The other half of the battle involves taking the time to write your program and making sure it works correctly. (This part of the battle is another place where governments and large corporations fall flat on their faces on a fairly regular basis.)

You can write a program to make your computer do anything you want, short of launching nuclear missiles at your next-door neighbor. (Of course, if you write a program for the Air Force's computers, you may even be able to do that.)

Writing a Visual Basic Program

There is no single correct way to write a program. Theoretically, you can use a million different ways to write a program correctly, just as you can travel from New York to Los Angeles a million different ways. Some people may fly, others may take a train or drive; the more adventurous may walk, hitchhike, or hijack a vehicle. Similarly, you can write the same program a million different ways. However, no matter how you write the program, the result can always be the same.

As a programmer, your job is to write a program that works correctly and is easy to use. If your program doesn't work, nobody can use it (although you often can sell a few thousand copies to unsuspecting individuals first). If your program isn't easy to use, nobody will want to use it, even if it works perfectly.

Testing whether your program works is usually simple enough. If your program is supposed to print mailing labels but erases the computer's hard disk instead, then your program obviously doesn't work correctly.

However, determining whether your program is easy to use is a bit more difficult. What you may consider easy to use may be almost impossible for someone else to understand.

To create programs that everyone can understand how to use, Visual Basic helps you to easily produce windows, pull-down menus, dialog boxes, and command buttons. These features are the same ones found in Windows 95/98/NT programs. Visual Basic helps you write programs that look and act like other programs on the market.

Making your program look and act like an existing program can help others learn your program faster. For example, most people can drive a Toyota or a Ford without any problems; the steering wheel and brakes always look and work the same way, even if the windshield wipers and horn may not. The same goes for programs. Pull-down menus contain a program's numerous commands, and you can always use the mouse to highlight and choose commands or objects. So while each program may work differently, they all look and work in similar ways.

The Visual Basic development cycle

Before writing a Visual Basic program (or any program for that matter), get away from your computer and plan your program using an old-fashioned paper and pencil. After you know what you want your program to accomplish and how you want it to look, then you can start writing your program. Skipping this crucial first step is like building a house without blueprints. You can do it, but it will probably take you longer.

Why programs don't work (Part I)

Writing a program that works 100 percent correctly all the time is mathematically impossible. First of all, if you write a program that works 100 percent correctly today, there's no way you can guarantee that it will work 100 percent correctly on future computer brands, models, processors, and accessories. As a result, you can never guarantee that your program will work correctly on all types of computers unless you exhaustively test every possible computer configuration in the world.

Second, not only do you have to test your program with the latest and greatest products (including the ones invented after you wrote your program), but you also have to consider the virtually infinite number of possibilities that your program must face during everyday use.

For example, your program needs to behave correctly if the user presses any key and then clicks the mouse anywhere on the screen. What if the user clicks the mouse by mistake while tapping a key? What if the user pounds the keyboard in frustration? What if another program happens to interfere with the computer's memory, thus affecting your program? What if . . . (well, you get the idea).

Unless a programmer can plan for an infinite number of possible problems and situations that a program may face during its existence, then writing a program that works 100 percent of the time is impossible.

What's scarier is that this scenario holds true for every computer operating system in the world (such as Windows 95/98/NT). Therefore, you'll always be writing programs to run on an operating system that doesn't work 100 percent correctly either. This situation is like building a house on a foundation of quicksand and then wondering why your house keeps falling apart.

Because no one has an infinite amount of time to test an infinite number of possible problems, computer programs always (yes, always) will have bugs that keep them from working 100 percent correctly. That includes every program you write and every program that Microsoft's millionaire programmers may write. That's why when you write a program, set aside plenty of time for testing so you can kill any potentially fatal bugs before you give your program to someone else.

So the next time you're using a program that doesn't work right, now you'll know that it's not your fault; it's the programmer's fault.

Writing a Visual Basic program requires nine steps — three steps fewer than those required to overcome an addictive habit. The first eight steps are what programmers call the *development cycle*. The ninth step is what programmers call *job security*.

1. Decide what you want the computer to do.
2. Decide how your program will look on the screen. (The appearance of your program is its *user interface*.)

3. Draw your user interface using common parts such as windows, menus, and command buttons. (The parts of a user interface are *objects* or *controls*.)
4. Define the name, color, size, and appearance of each user interface object. (An object's characteristics are its *properties*.)
5. Write instructions in BASIC to make each part of your program do something. (BASIC instructions are *commands*.)
6. Run your program to see if it works.
7. Cry when your program doesn't work perfectly. (Required.)
8. Fix any errors (or *bugs*) in your program.
9. Repeat Steps 6 through 8 over and over again until you get tired of searching for more bugs.

Although you don't have to memorize these nine steps, you do have to follow them. Shortcuts aren't an option. Trying to skip from Step 1 to Step 4 is like trying to start a car by using the gas pedal but forgetting to turn the ignition key. You can try it, but you're not going to get anywhere.

Believe it or not, Step 1 is actually the hardest and most important step of all. After you know exactly what you want your program to do, it's just a matter of finding ways to do it. Persistence and creativity are helpful, as are lots of caffeine-laden beverages and plenty of sleepless nights in front of the computer screen.

Making a neat user interface

The user interface is what someone sees when your program is running. Every program has a user interface in one form or another. Some programs have elaborate, colorful windows, while other programs have a sparse appearance — as if the programmer were afraid that screen phosphor may be in short supply one day.

A Visual Basic user interface consists of forms and objects. A *form* is nothing more than a window that appears on the screen. Most Visual Basic programs have at least one form, although most programs likely will use several forms.

Objects are items that appear on a form (see Figure 1-1), such as a command button, scroll bar, option button, or check box. An object lets the user give commands to your program. If you really wanted, you could create a program with only one form and no objects, but it wouldn't be very useful or interesting.

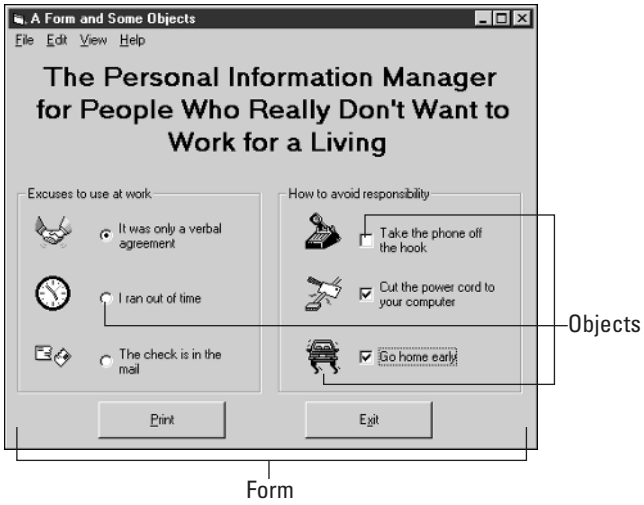

Figure 1-1: A typical form and objects.

Defining properties to make your user interface unique

After you create a form and draw some objects on it, the next step is to define the properties of each form and object. An object's *properties* determine the object's name, color, size, location, and appearance on the screen.

Different objects have different properties. Each time you draw an object on a form, Visual Basic assigns default property values, which define a generic object that no one can really use. If you want to customize an object, you need to define one or more properties for each object that your program uses. Chapter 3 provides a quick introduction to changing an object's properties.

Writing BASIC code

When you're happy with the way your program looks, the next step involves writing BASIC commands (also known as *code*) to make your program actually work. (Don't worry. If you change your mind and want to edit the appearance of your user interface, you can go back and alter it at any time.)

The whole purpose of Visual Basic code is to tell objects on a form what to do when the user does something. For example, if the user clicks on an OK or Cancel command button, nothing happens unless you've written BASIC commands to tell your computer exactly what to do.

Why programs don't work (Part II)

Most programs are written by professional programmers who may have studied programming for years. Does this mean that every program they write will always work? Of course not. This is the world of computers where nothing works right, remember?

Besides the fact that the skill level among professional programmers can vary widely, professional programmers are often called upon to write programs for tasks that they don't understand themselves. For example, programmers may know nothing about accounting yet be hired to write a program to control a bank's electronic-fund-transfer system. Likewise, programmers with no skill or experience in flying may be hired to write a program to control the landing, takeoff, and flight of a 747 jumbo jet. Programmers with no knowledge of medicine may be required to write a program to control a medical instrument that administers doses of radiation to cancer patients. How can you work in a field where you have no experience and still get paid a lot of money? Easy, you become a programmer.

Hiring a programmer with no knowledge of the task she's trying to solve is like hiring a translator to translate Greek into French without that person's knowing how to read or write in either language. Given this paradox in the programming world, is it any wonder that planes crash, banks lose money, and hotels can't keep our reservations straight?

Any time a user presses a key, moves the mouse, or clicks the mouse button, it's called an *event*. Whenever an event occurs, your BASIC commands tell the computer, "Hey stupid, something just happened. Let's do something about it!"

Essentially, writing a Visual Basic program means drawing your user interface and then writing BASIC code to make it work. If you can handle these two steps without losing your mind, you can start writing your very own programs using Visual Basic. Chapter 4 provides a short introduction to writing real-life BASIC code.