1

Fundamentals of Design and Programming – Starting from Scratch

VB Quip *Who cares how it works, just as long as it gives the right answer? –Jeff Scholnik*

That's a rather cavalier approach for anyone to take, especially when computers are involved. Knowledge is power and the more you know, the greater your power. One book isn't enough to give you all-consuming power, especially over a tool as powerful as a computer. It is, however, enough to get you started. One book won't turn you into a nerd that looks at his own shoes when talking, lives on caffeine and stale snack cakes, and would rather hack on a computer than go out on a date. That's just a stereotype perpetuated by teen movies. While it does fit a few people, almost all of the ones I know are smart, articulate, funny, and just all-around bright and inquisitive people. Above all, they're curious. And the best are curious about nearly everything, not just computers. How does it work? What happens when I try this? What if . . . ?

Programming is, above all else, about thinking and problem-solving. If for no other reason, it's useful because it makes you think about thinking and makes you describe how to solve a problem. For the computer to solve a problem, even simple tasks must be explained in great detail. For someone to explain the rules to a computer – in other words, to write a program – takes a great deal of thinking and understanding. Along the way it gives you an appreciation of just how marvelous the machine known as a computer really is, it gives you an understanding of how you think, it makes you think about thinking, it forces you to consider the steps and methods used in problem-solving, and it provides a process for expressing and explaining your thoughts.

The first part of this chapter deals with thinking and problem-solving, the basics of programming. The second part of the chapter familiarizes you with the Visual Basic interface – the nuts and bolts of how to create a program.

What Is Programming?

VB Quip *A computer is essentially a trained squirrel: acting on reflex, thoughtlessly running back and forth and storing away nuts until some other stimulus makes it do something else. –Ted Nelson*

That's closer to the truth than you can imagine and it does an injustice to squirrels. The computer does exactly what you tell it to do, even when that's not what you want it to do. Forget about "computer error" and those million-dollar utility bills. That's not a computer error; it's probably a programmer's error. The person writing the directions for the computer is the one that made the mistake. It's called "human error," and at the heart of most computer problems is a person. Whether we like to admit it, most of the time computer error is merely human error in disguise.

Programming is simply giving the computer directions for completing a task. A *program* is a set of directions for the computer to complete a task. VB.NET takes the directions you write and turns it into a program. It translates your directions into a series of steps for the computer. If there is a mistake in the directions, the mistake gets translated by the computer. Just like a recipe or a set of driving directions, a computer program is only as good as its instructions. Therefore, programming should be about two things: teaching you how to write the directions in the first place, and teaching you how to write directions that won't cause problems. And, of course, the person writing the program is a *programmer*.

You can think of a program as a recipe. A recipe describes, in detail, the steps that need to be taken in order to, say, bake a cake. If the steps aren't correct or they aren't in the correct order, you won't get cake. You might end up with batter because you forgot the "turn the oven on" step. You might end up with a crusted pile of inedible goo because you forgot the "beat 200 times" step. To compound the problem, the directions must be written in a way the computer understands. As painful as it is, you have to learn how to write the directions so the computer can understand them. As yet, the computer cannot understand the directions in the way you want to write them. While VB.NET is much better at understanding your directions than many other languages, it still requires that the directions be written according to some very strict rules.

Basic Tasks

Most computer programs complete the same basic set of tasks and, lucky for you, complete them in the same order. Every program has input, processing, and output. *Input* puts data into the computer. It might be a number or some text. It might be data from a file. Depending on the system and the task, it could be input from a pen, mouse, or keyboard. If attached to sensors, the input could be almost anything, from a temperature sensor to a light meter. It's the stimulus for the computer. *Processing* is the task: In a business program it could be the payroll calculations; in a game, it may move the pieces on a board; in an air conditioning system, it may turn on the fan or turn off the heat. Processing is the work of a program. *Output* is the results, the answers. Often the results are

displayed on the screen, but it could also be a printout or a file. Every program works with data. The data could be the numbers or text used by the program. They could be the hours and wages for the payroll or the temperature for the thermostat. They could be the number of spaceships for your "Alien Invaders" game. These data must be declared before you can use them. *Declarations* tell the computer what type of data you have for your program. Declarations almost always come at the beginning of a task.

In terms of your cake recipe, the declarations are the measuring cups, the measuring spoons, the mixing bowl, and the cake pan. You must make sure you have these before you start. The input is the ingredients, the cake mix, the eggs, the water, and such. The processing is the mixing and the baking – all the steps needed to turn your ingredients into a cake. The output is, well, the cake. The order for these tasks is usually declarations, input, processing, and output. So, in your cake program, you'd declare (or in this case make sure had on hand) the measuring cups and spoons, mixing bowl, and cake pan. Without them you couldn't handle the other tasks. The input would be adding the ingredients. Processing could be described as mixing the ingredients and baking the cake. Those are very general descriptions for a long series of steps, but the analogy fits. You take your input and, through a series of processes, turn it into output. The end result, your output, is a cake. If the steps are in the right order and the directions are clear, you get a cake. If not, well . . .

Almost any problem can be described in terms of declarations, input, processing, and output.

VB Quiz 01

What are the four steps in programming?
Which step comes first?
Describe how programming is like a recipe.
How is programming like a set of directions for a traveler?

Basic Procedures

Nearly every programming task falls into one of three categories: sequence, selection, or repetition. These tasks enable the computer to handle all processing procedures that come along. Each one has a particular utility that makes the computer function effectively. In many respects, these procedures turn the computer into the ideal employee. Think about it. The computer works long hours doing *exactly* what you tell it to do. Once given directions, it performs its task unerringly and with amazing speed and accuracy. It never tires; never asks for the weekend off; never asks for a raise; never takes a potty break; never goes on strike; doesn't need fringe benefits; and won't complain about the lighting, the room temperature, the mess in the break room, or the numbskull in the next cubicle. It is, in short, the perfect slave.

The *sequence* of commands is the order of commands in a program. Sometimes the order of the commands isn't critical, but usually, it is. If the steps aren't in the correct order, you probably won't be able to solve the problem. Sometimes the steps must be exactly in order. Sometimes the steps only need to be in a general order. Washing dishes is a good example. While washing the glasses is done before the plates, and the pots and pans are washed last, the exact order of the individual glasses or plates or pots usually isn't critical to the successful completion of the task.

Selection procedures enable the computer to make decisions. The computer is given two or more sets of directions and a criterion for selecting the correct set of directions. Based on its decision, it selects the appropriate path and follows those directions. Think about the laundry. The steps for washing whites are different from those for colors or delicates. A decision must be made at the start of every load. Depending on the type of load, you might use hot, warm, or cold water. You might add bleach or fabric softener. Depending on your decision, a different set of directions must be followed.

Repetition procedures allow the computer to repeat the same series of steps. Give the computer a set of directions to repeat, tell it when to start and when to end, and it will repeat the same process over and over. It might repeat the same process a specific number of times or it might repeat the process until certain requirements are met. Brushing your teeth is a good example of repetition. You might repeat the up and down process a specific number of times for each tooth. You might repeat the process for a specified amount of time or you simply might do it until you think you're done. The up-and-down brushing process is the repetition.

All programs make use of these procedures and all but the simplest programs are a combination of two or more of them. Your daily routine is filled with tasks that are a series of steps. Some of them are a sequence of steps. Some require decisions and some tasks are repeated. Within each of these tasks, there are probably more, even simpler, tasks. Go back to the sequence of steps for washing the dishes. Within that task is a repetition. You doubtlessly made a series of circular motions while washing a plate or pot. That motion was repeated a certain number of times or until the plate or pot was clean. Life is filled with sequence, selection, and repetition procedures if you stop to look at them.

For example, there are about ten steps to making a pot of coffee. If you skip a step, you get hot water instead of coffee. If you forget a different step, you end up with cold water instead of coffee. If you get the steps out of order, you run the risk of burns or electrocution. If you make a bad decision on how much coffee to add, you end up with a cup of sludge instead of a cup of joe. If you use the wrong number of repetitions when adding water, you won't get espresso. The real trick in a good cup of Java is to get the directions correct and then follow them.

Following Directions

Computers are great at following directions. Unlike teenagers and cats, they do exactly what you tell them to do. All you need to do is provide them with precise directions for the task and provide these directions in a language the computer will understand. That's the programming part. Of course, if the directions are wrong, then the computer makes "mistakes." That's the source of those million-dollar utility bills. Proper planning helps to avoid these problems, but even the best programs can still have mistakes in them. To minimize these mistakes, programmers need to learn the fundamentals for writing programs.

You need to have a plan for developing a set of directions for the computer: a program. This plan is commonly called an *algorithm*. An algorithm is a description of a program. Some algorithms are just a general description of the program. Some are very precise. It just depends on the amount of detail needed for the directions. For a vacation, you'd be perfectly happy with a set of directions that told you to drive to the airport, catch a flight to your destination, hop a cab to your hotel, and then enjoy yourself. That's a good general algorithm, but for each of these, you'd need a more specific algorithm. You'd need specific directions to the airport. You'd need to know the flight, its airline, the gate, and the departure time. Once you arrive, you'd need to be able to get to the hotel and, once you've checked-in, you'd want a list of sights and shows. All of these steps require more precise directions.

For the computer, the description of the steps to solve the problem becomes the algorithm. With it, a programmer decides the sequence of steps, the decisions that must be made, and the steps that need to be repeated. These directions are miniscule, incremental, and precise. On the computer, even the simplest task often takes considerable programming. Processes (thinking) that you have internalized – things that come almost automatically for you – have to be fully described for a program. If asked to give the largest number in a set of three numbers, you could solve that "without thinking." You'd have it completed within a split second of knowing the numbers. The real trick is to be able to describe the process you used to solve the problem and then translate that process into computer directions. You'll see more of this process in Chapter 5 on decisions. For now, it suffices to describe the process in general terms. First you'd declare the space that's needed for data. In this case, it's the numbers you're comparing. The second step is to get the numbers into the computer – the input step in the process. The processing step involves comparing the first number to the second number. Keep the higher of the two. To store it requires a storage space for the data. Then compare that number to the third and keep the higher of those two. Once completed, the computer would "know" the highest number in the list.

Although this description is detailed, it still isn't precise enough for a program. The details are even more exacting. It often takes several commands just for one task and each miniscule detail must be described. The steps in the process are incremental. Each one is spelled out in detail. The directions must be precise and spell out every detail exactly. In this example, you'd need to precisely describe the numbers to store, the numbers to compare, when to compare them, and where to store them. When that is completed, you'd provide precise directions for reporting the results. In short, the program must provide painstaking details. Each baby step is obvious and incremental; if you get the steps in the wrong order, you won't solve the problem. Forget a step and it's all over. And all this is to get the computer to do something you've been able to do "without thinking" since the second grade. Think about that the next time you're reading those "some assembly required" directions and it tells you to put tab A in slot B!

Let's go back to the cake-baking analogy. Your declarations are the utensils you need: the cups, spoons, bowls, and pans; your input is the ingredients; the processing is the mixing and baking; and the output is the finished cake. For us that might be a good enough description, but it wouldn't work for the computer. For the "add the ingredients" step, the computer needs far more direction. It follows the order specified, the sequence, and you must tell it the exact steps in the correct order. The specific amounts to be added and the order to add them must be described. Furthermore, "add an egg" to us means to take it out of its shell and add the inside parts, not the whole egg. That detail needs to be spelled out for the computer, even though we make assumptions for such things. Sometimes this order makes a difference and sometimes not. A good cook knows to put the liquid ingredients in first. That way the dry ingredients won't stick to the bottom of the mixing bowl. A good programmer learns such tricks as well. The mixing requires you to beat the mixture for a specified length of time. That's repetition. The directions call for changing the ingredients or the baking time if you're above a certain elevation. That's a decision and you'll follow one set of directions or another for that part of the process. Baking is one of the last steps in the process. For us, that's a close enough description. For the computer, you'd need to be more precise. If you get the steps out of order, you won't have dessert. And if you're good in the kitchen, you'll know enough to preheat the oven in advance rather than saving the "turn the oven on" step until you're ready to bake. It's safe to say you've never used a recipe or a set of directions quite as precise as those needed by the computer.

It's important to remember two things: (1) the computer does only what you tell it to do and (2) the computer has no intellect. Each command must be given to the computer and each must be in the correct order for the computer to successfully accomplish its task. You wouldn't have to give much thought to averaging four test scores. For the computer, each number needs its own storage location. These must be declared at the start. The program needs four inputs.

It then must add the inputs together to find the total. Once the total is known, it can calculate the average. The last step is to display the average. All of these must be in order and, if one is out of order or forgotten or done incorrectly, you won't find the answer. This simple, little program involves at least a dozen steps, nearly all must be in a specific order, and the correct calculations must be made in order to get an answer that you might be able to calculate in your head. Make just one mistake and it won't work properly. Most of us aren't accustomed to such detail.

The computer has no intellect, no imagination, and no insight. And worst of all, it doesn't have a sense of humor. It does exactly what you tell it to do and not what you want it to do. It makes no assumptions and doesn't correct your mistakes. If you incorrectly tried to find the average for your test scores before you knew the total, you'd back up, correct the mistake, and get it right. The computer can't do that. If your steps are out of order or wrong, the computer doesn't know it. It's only doing what it's told. In that respect, it's the ultimate passive-aggressive machine. The task of the programmer is to understand the program well enough to create a workable solution and then translate that into code the computer can handle.

VB Quiz 02

What three basic procedures are needed to write all computer programs?
Describe these procedures.
What is an algorithm?
Explain why computer programs are miniscule, incremental, and precise.
Describe, in general terms, the process for balancing your checkbook.

Interface/Instructions – The Human/Computer Connections

Developer/User

You must assume two roles to write programs: the developer and the user. The *developer* is the person (or more likely a team) that develops the specifications, designs the program, creates the algorithm, writes the code, and tests the program. The *user* is the person that uses the finished program. Users also have a part in developing the specifications, designing the program, and testing it. At this stage, you're both the developer and the user. You'll design programs and use them, which offers some advantages and some disadvantages. You can see both sides of the program. You'll find out just how hard it is to design and code a program that the average person can use, how much work is involved in programming, and how difficult it is to anticipate the needs of the user. But, on the bright side, you can also design a program that exactly suits your needs.

You'll jump back and forth between programmer and user. When developing the program, you wear the programmer's hat. As a tester or user, you won't be interested in how a program works, only that it *does* work. You'll also run a

program to do debugging. *Debugging* is the process of removing mistakes (*bugs*) from a program. While testing you'll act as a user, but you have a programmer's eye toward how the program runs, what works and what doesn't, and how to make the program work better for the user. Think of the developer as an inventor working on the design of a new widget. The user is the person testing the widget. In most cases, you're both the developer and the tester.

VB Quip *There are only two industries that refer to their customers as "users." –Edward Tufte*

Design Time/Runtime

A developer works in design time and tests in runtime. *Design time* is when the program is developed, the interface is created, and the code is written. *Runtime* is when the program is running. In design time, a developer can add, delete, and modify code and make changes to the program settings. At runtime, the program is running, and changes to the program's design and code cannot be made. However, a programmer can write directions during design time that will change how a program looks and works in runtime. It is important to remember that at design time a developer decides everything that a program will be and can do. At runtime, the program executes the commands given to it at design time.

Form/Code

Most of your development in Visual Basic is done in two windows, the Form window and the Code window. The *Form window* is where the form is designed. That's where you put all the text, pictures, and controls for your program. This is often called the *interface* and is what the user sees when the program runs. The *Code window* is where the code is written. That's where you put the directions for the program. A user never sees the code window. A quick click on a tab switches you back and forth between them. As a developer, you must be familiar with both of these. You'll work with both of them to design and code a program.

Objects/Events

The controls on a form – the text, pictures, buttons, even the form itself – are *objects*. Visual Basic .NET is based on objects and it's far more than just the controls on a form. Objects are covered in much greater detail starting in Chapter 12. Without a knowledge of objects, it is difficult to move beyond the basics of programming. Think of an object as a building block for a program. There will be more on objects later, but for now simply consider them to be the form and all the controls on the form.

Events in a program include the ways a user interacts with the computer. These are things like clicking or double-clicking the mouse, typing in text, or pressing the Enter key. Events trigger a response from the computer. Events include far more than a click or a peck, but these are the most common. As a developer, you write code that runs in response to these events. If the user clicks on a button, the code for that event runs.

A popular term that's bantered about is *object-oriented, event-driven programming*. Throw that into your next conversation. It's a nice buzzword, but it simply means that a program was written with objects and is controlled by user events. Of course, if you don't understand objects and events, the term is meaningless. Suffice to say that most significant programming is done with objects and events.

Code Files/Program

Visual Basic .NET projects have a folder containing several files and other folders that contain files. These files are needed to create your finished program. All of them are used in development. Be very careful when moving or changing these files; one mistake can damage your project. It's best if you manage the project folder and leave the other files intact. Your finished program is an executable file with an .exe extension. This is a standalone program that you can run even if you don't have Visual Studio. To create or modify a program, you need the project folder and its files. These are used to create the executable file. There's more on this in an upcoming section.

What Is a Program?

In Visual Basic .NET, you create a program by developing a project. The project is stored in a folder and contains files and other folders. The finished program is in the bin folder and has an .exe extension. Within the project is at least one form file. A *form* is what the user sees when they run the program. For each form, there is at least one file containing the code for that form. These code files contain the directions for the program. Visual Basic controls the folders and files. As a developer, you are responsible for designing and creating the form and writing the code that makes that form work. The controls on the forms are linked to events. When an event is triggered, it runs the code behind that control. When writing the code, you need to be aware of the input needed from the user, the processing needed to solve the problem, and the expected output the user will get. You manage these resources during design time so the program can accomplish its tasks during runtime.

VB Quiz 03

What two hats are worn when working with Visual Basic .NET?
What is the difference between design time and runtime?

Who sees a form and who works with the code?
What is an object?
What is an event?
How many files make up a program? How many are in a project?

Your First Program – College Tuition

Define the Problem

The first step in any program is to define the problem. For this program, you must determine the cost of your tuition for the semester. To do the calculation by hand, you'd take the number of credits and multiply it by the tuition rate. The product is your tuition. The problem is simple and straightforward; you don't even give it much thought. For the computer, though, it involves numerous steps. You need to declare all the variables, get the user input, make the calculation, and display the output. All the basic steps for a program are incorporated: declarations, input, processing, and output. Nearly every program involves these steps in this order.

One of the hardest parts of programming is defining the problem. However, careful consideration in this step will save you time and effort in the long run. For every program, you must define the variables – the numbers or text needed to solve the problem. You must define the inputs – the data the user provides so you can solve the problem. You must determine the calculations needed to solve the problem. The order in this step is critical. And, finally, you must define the output – the results that solve the problem.

Once you have defined the problem, you can design the form. The form is the screen that the user sees at runtime. Although you may be tempted to do this step first, it's better to define and describe the other steps before you create your form. You can then develop the form and later code your program.

Define Inputs

The input for your program is the numbers or text entered by the user. For now, the input is handled with TextBoxes – more on that shortly. The input is the data for your program – what you need to know to solve the problem. Every program requires input of some kind. For this example, the input is the number of credits. This is a simple problem so we won't worry about undergraduate or graduate tuition rates, in-state or out-of-state tuition rates, or discounts for select groups. The only input the user needs to provide is the number of credits being taken.

Every program requires input. Input enables the user to distinguish one run of a program from every other. Without input from the user, the computer