Visual Basic 2008 Programming

Business Applications with a Design Perspective

Jeffrey J. Tsay

Table of Contents

Chapter 1

Introduction

This book introduces you to Visual Basic (VB) 2008 programming. It is oriented toward business applications, with a design perspective. Many business applications involve data-entry operations. On appearance, data entry appears to be a fairly simple programming task. However, once you start to develop this type of application program, you will encounter many intriguing issues. For example, as soon as you start to design the data-entry screen, you will wonder how to best design the visual interface; thus, you will be faced with interface design issues. In addition, you will have to develop code to validate the data entered by the user because data accuracy is of the utmost importance to any business. Here, you may encounter a lot of duplicate code; thus, you will need to design the code structure to minimize duplication. Business applications also involve data storage and processing; therefore, you will need to deal with data management issues and develop algorithms to process data efficiently. This book is oriented toward handling these issues.

This chapter gives an overview of VB programming. Naturally, when you are learning VB programming, your first question is, "What is Visual Basic?" Section 1.1 provides some explanations. Section 1.2 examines what constitutes a sound application program. This chapter also discusses the steps involved in developing a program. Section 1.3 provides some suggestions on how to use this book. After completing this chapter, you should be able to:

- Explain what Visual Basic is.
- Contrast the operating environment of Visual Basic with that of programs developed using procedure-oriented languages.
- Set forth the criteria for a sound application program.
- Enumerate the steps to develop a program.
- Explain the importance of hands-on experience in learning a programming language.

1.1 What Is Visual Basic?

Visual Basic is a programming language that allows the programmer to design the graphical user interface GUI) visually, typically without involving any code. This feature along with many others makes Visual Basic easy to use to develop applications. To understand the language better, it is desirable to explore how the language evolves first.

Evolution of Visual Basic

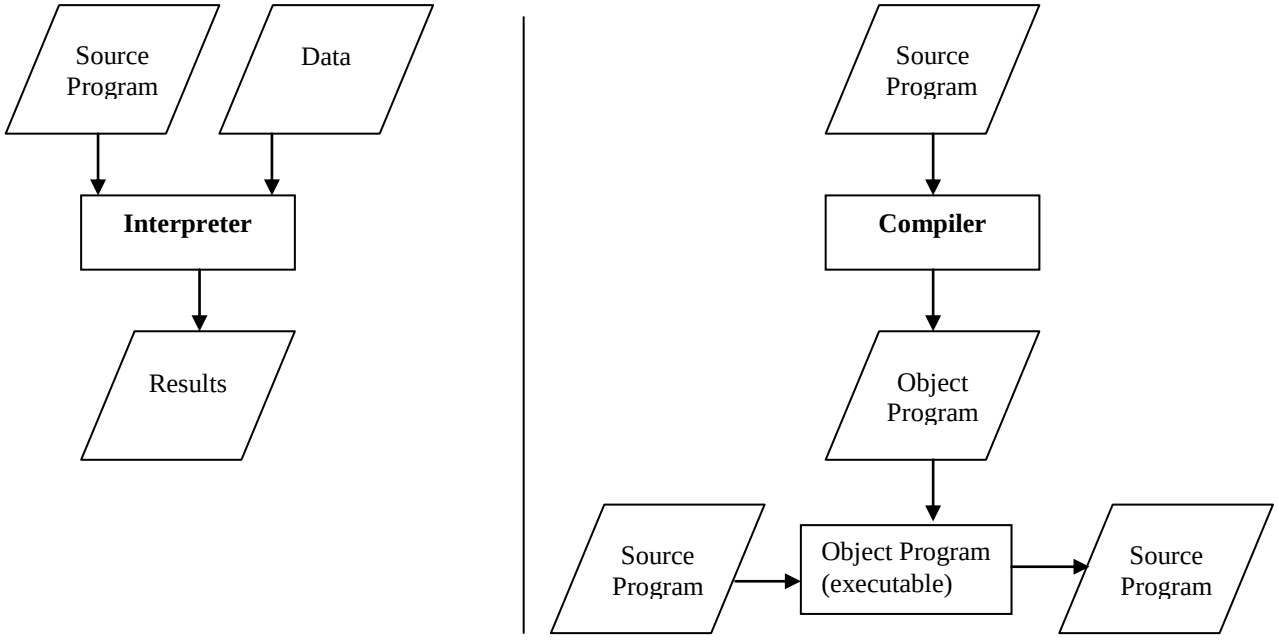
Visual Basic evolves from *BASIC,* an acronym for *B*eginner's *A*ll-Purpose *S*ymbolic *I*nstruction *C*ode, which was created in the 1960s. Its syntax is similar to *FORTRAN*, another programming language that was developed to handle *For*mula *Tran*slation. However, BASIC was used primarily for interactive computing, while FORTRAN was used in the batch-processing environment where each program was run as a job without human interaction or intervention. When microcomputers were introduced, BASIC was the language available in nearly all makes, such as Tandy, Apple, and so on. Its power was fairly

limited because of hardware limitations. When the personal computer (PC) became available, Microsoft introduced GW-Basic with its disk operating system (DOS).

As you may be aware, programs written in BASIC (any version) need to be processed by a *language processor* (a *program* that processes the BASIC program) before their instructions can be understood and executed by the computer. The BASIC language processors up to that point were *interpreters* that read one line of BASIC code at a time, interpreted the code, and carried out the operations called for by the instructions. Because of the overhead associated with interpreting the code, the execution was very slow.

To improve execution speed, BASIC *compilers* were introduced. A version of the BASIC compilers by Microsoft is *Quick Basic*. A compiler is another kind of language processor that translates a *source program*, such as a program written in Quick Basic, into the machine language or some pseudo-code that is fairly close to the machine language. The resulting program is recognized as the *object program*, or *executable*, in modern terminology. Because of the elimination of the overhead of interpretation, the object program runs much faster than a source program under an interpreter. As you can see from the diagram in Figure 1.1, the key difference between an interpreter and a compiler is the output. The interpreter takes the source program along with the data as input and produces the results that the source program calls for. Under this arrangement, each time you need to run the source program, you will need to invoke the interpreter. In contrast, the compiler takes only the source program as input and produces an object program in a lower-level language. The object program must be run along with required data to produce the results. Note that the object program can be run many times with different input, and without being compiled again as long as the source program is not changed.

Figure 1.1
Difference between Interpreter and Compiler



Difference between Interpreter and Compiler: the former produces results of computations; the latter produces an object program, which needs to be loaded into the memory to run in order to obtain results.

Despite its speed, Quick Basic has its drawbacks, especially by today's standards:

- It is a procedure-oriented language. When it runs, the program, instead of the user, dictates the sequence of activities to be carried out. The user is not allowed any flexibility.
- Its interface is text-based instead of graphics-based. The appearance is not attractive. The keyboard is used nearly exclusively to obtain user input or commands. In many cases, text-based input is more susceptible to errors and may not be as efficient as other gadgets that are available in the graphics-based environment, such as check boxes, radio buttons and combo boxes.
- Programs take a long time to develop and are difficult to change, especially when the change involves the visual interface. The programmer needs to track all the details relating to the location, size, and color of all boxes and texts drawn on screen. A minor change can call for painstaking efforts by the programmer to ensure that everything is done correctly.

After the graphic user interface (*GUI*)-based Windows operating system was introduced for PCs, *Visual Basic* (VB) arrived in 1991. The first version of VB had two editions: one for the DOS, and another for the Windows system. As the Windows system gained in popularity, the later versions of VB were designed only for the Windows operating systems. Earlier versions of VB were written for 16-bit operating systems because 32-bit operating systems were not yet available. There were two editions of VB in version 4: one for 16-bit operating systems, such as Windows 3.0) and another for 32-bit operating systems, such as Windows NT and Windows 95. Since version 5, all editions have been made exclusively for 32-bit operating systems. Version 6 expanded several language features and supported ActiveX Data Objects (ADO) for data access. VB.NET was introduced in 2001 as a result of a sweeping overhaul to the language and offers the object-oriented features of modern programming languages. VB 2005 simplified the coding requirements for forms and controls. The current version, VB 2008 provides convenient intelisense hints to the developer while he/she is writing code, making the code writing process much easier and more efficient. The following table briefly highlights the *milestones* of the evolution of VB.

Year	Milestone	Remark
1964	BASIC	For interactive computing in *mainframe* computers
1970's	BASIC	In various microcomputers/PC (BASIC and GW-Basic)
1985	BASIC and Quick Basic	Processed by compilers
1991	Visual Basic	For Windows or for DOS
1992	Visual Basic 2	For Windows only
1993	Visual Basic 3	For Windows only; enhanced with data access capability
1996	Visual Basic 4	For 16 bit or 32 bit Windows systems
1997	Visual Basic 5	For 32 bit Windows systems only
1998	Visual Basic 6	For 32 bit Windows systems only
2001	Visual Basic.NET	Object oriented; sweeping overhaul
2005, 2008	Visual Basic (year)	Further improvement and enhancement; luxurious convenience for developers

Characteristics of Visual Basic

The vocabulary and syntax of VB are derived from BASIC; however, VB differs significantly from BASIC in several respects:

- VB provides a set of *visual objects* (recognized as *controls*) that can be drawn easily onto a window (called a *form*). These controls eliminate the need to develop the code to construct the visual interface. The layout of the windows that contain the controls can be changed easily by dragging and dropping the controls to a new location, without necessitating a change in the code. The process for program development and revision becomes much easier and requires much less time and effort.
- The code structure of VB is *object-oriented*, whereas that of BASIC is *procedure-oriented*. In BASIC, there is no object. Its code usually follows the following syntax:

```
Verb Operand List
```

For example, to display a line of text in BASIC, such as "Display this line", the code will look like the following:

```
Print "Display this line."
```

Although some VB code still retains this form, most of its code is structured around objects. The syntax appears as follows:

```
Object.Method (parameter list)
```

where the so-called method is the action or activity to be carried out.

To display the same line using VB, you would code the following:

```
Console.WriteLine("Display this line.")
```

As you can see, Console is recognized as an object. Its WriteLine method will write the line on the object, which is the console.

Object-oriented coding is easier because the object and the action are identified separately, resulting in a more concise set of vocabulary. In BASIC, you must remember to use Lprint to print on the printer and Print to print onscreen. In VB, you use the same WriteLine method to display (print) on different objects. It is also more consistent with the user's activity in the GUI environment. For example, when you are editing a document with a word processor, you highlight a block of text (identify the object) and then instruct the computer what to do with it (indicate an action such as cut or paste).

Note that an object-oriented language offers many more advantages than merely simplified syntax. One significant advantage is code reusability through *inheritance*, a feature that allows the programmer to extend the functionality from existing code without having to directly modify the original source code. This feature was not available in the previous versions of VB. VB.NET and later versions are a full-fledged object-oriented language that offers many exciting features of modern computer languages.

- VB is an *event-driven* rather than procedure-oriented language. As pointed out before, when a procedure-oriented program runs, it dictates the sequence of operations. This means the programmer must predefine the sequence when developing the program. In addition, changing the sequence of operations requires revising the program. On the other hand, an event-driven program does not dictate the sequence of operations. The user can instruct the computer to perform whatever operations the program is capable of, in any sequence he or she desires. This offers the user flexibility. Any changes in the sequence of operations will not call for revising the

program. In this sense, an event-driven program is easier to develop, and requires fewer revisions.

Event-driven programs, however, present the programmer with a different kind of challenge. In many instances, an activity can be carried out only after some prerequisite actions have been taken or after data are ready. For example, when the user clicks the Send button in an email application, the target email address, the subject line, and the body of message must all be filled in first. When the user initiates the action, there is no guarantee that the prerequisites have been met. As a programmer, you must find ways to ensure that the prerequisites are there.

Visual Basic as a Language and as a Processor

You have learned that VB is a programming language used to write programs to make the computer perform desired tasks. It has its own vocabulary and grammatical rules (syntax). These elements can be combined to form a program, which is the complete set of instructions designed to perform the defined tasks.

You have also learned that the program you write in the VB language will need to be processed by a VB language processor, which is also a computer program. In most instances, the VB language processor is also referred to as Visual Basic. When one states that you can write a program in VB 2008, he is expressing that you can write a VB 2008 program that VB 2008 (the processor) can understand and process. As such, the term Visual Basic can actually mean two different things—the language and the processor—in different contexts, or the language and the processor at the same time.

When you are developing a VB program, you will work with a software program that does more than just process your program. It provides an environment in which you can draw the visual interface, write the code, compile and test the program, and make additional changes. For VB 2008, this processor is recognized as the Visual Studio Integrated Development Environment (IDE), which is actually capable of handling not just VB 2008 but also several other languages. The IDE will be explored further in Chapter 2, "Visual Basic Programming Concepts."

Statement and Code

A VB program can consist of many instructions. An instruction that is complete in meaning and can stand alone like a sentence is recognized as a *statement*. Usually, you will use a line to write a statement. Some statements are very long. In such cases, the statement can be spanned over several lines. The mechanic of spanning a statement over multiple lines is explained in Chapter 2. Statements in a program are collectively recognized as *code*. The following sample code fragment comes from Chapter 7, "Repetition:"

```
For I = 0 To lstNames.SelectedItems.Count - 1
    'Show names selected
    Console.WriteLine(lstNames.SelectedItems(I))
Next I
```

Each line in this code is complete in meaning and, therefore, is a statement.

1.2 Overview of Visual Basic Program Development

Before you get into the details of actually developing a VB program, it is important that you become aware of the criteria for a sound program. These criteria provide benchmarks against which the quality of your programs can be compared, and thus serve as the guide for your application development. In addition, you should also be acquainted with the program development cycle. A good understanding of the cycle will equip you with a step-by-step roadmap to developing your program so that you can be more efficient and effective in carrying out your programming endeavor. This section discusses these two aspects.

Criteria for a Sound Application Program

How do you judge the soundness of an application program? As a programmer, you examine the program in two completely different perspectives: *external* and *internal*. You first inspect the program from an external perspective, and judge it in terms of the application requirements. The criteria include the following:

- *Functionality.* The program must meet the requirements of the application; that is, it must deliver the functionality called for by the application. For example, if an order-processing application needs to update both the customer records and the on-hand quantities of all products ordered, the program cannot be considered complete until it can perform both functions. In short, can the program do what it is expected to do?
- *Efficiency.* The program should also perform the required functions efficiently. The program should minimize the consumption of computer resources, including computer time and storage space. I have seen a general ledger package that takes several minutes of computer time to post 10 transactions from a general journal to a general ledger, almost as fast (slow) as a person can handle the task manually. Such a program obviously fails the efficiency test.
- *User-Friendliness.* The concept of user-friendliness is not new. The ancient concept stipulated that the message to the user be clear, meaningful, and in a friendly tone; however, with the GUI environment, this concept encompasses a much wider variety of user expectations. Briefly, a user-friendly program should do the following:
    - Provide the user with maximum mobility around the user interface.
    - Be consistent in appearance and behavior among different windows.
    - Provide the user with supports in using the program to perform the task; such as providing dialog boxes for lookup and online help.
    - Be flexible in accommodating user's tastes and preferences.
    - Guard the user against errors and mistakes. If not handled carefully in the program, some errors caused by the user can crash the application, resulting in loss of data. The program should properly handle all errors and mistakes by the user to prevent accidental loss of data or causing any inconvenience to the user.

From the perspective of the programmer, the program not only should satisfy the external requirements as outlined above but also should be developed with a set of *internal standards*. The code developed for a program should have the following characteristics:

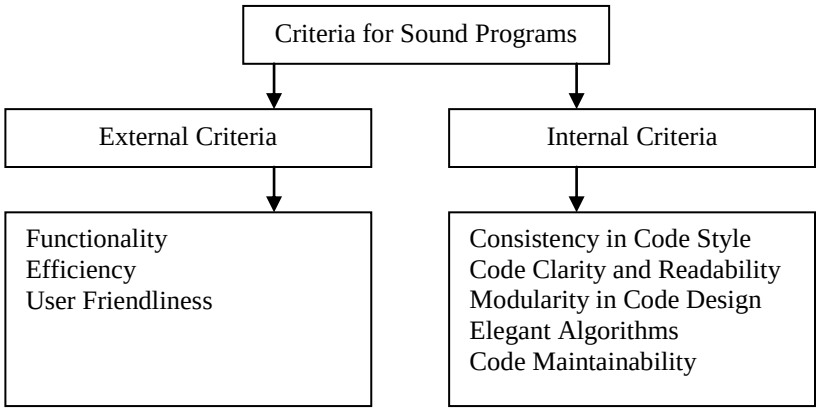
- *Consistency in Coding Style.* There is a set of coding conventions that the programmer should follow when developing code. These conventions include coding mechanics such as properly

indenting blocks of code for visual clues to the code structure and naming objects with predefined prefixes for easy identification of the types of objects involved. Code developed with a consistent style is much easier to read and understand. As an illustration, look again at the code fragment on page 4. It is indented properly, providing a clue that it is a code block. (Note that recent versions of VB IDE automatically indent the code block.)The name *lstNames* also follows the naming convention; it contains a three-letter prefix, lst, indicating the type of object it represents.

- *Code Clarity and Readability.* In addition to the consistent coding style, other factors can also make code clearer and easier to understand. For example, if the names used to represent data in your program are meaningful, your program will become much easier to read and follow. The person who reviews the code can easily associate the names with the data they represent, making the tracing of programming logic much easier. As an example, you also might have inferred that lstNames in the preceding code sample contains names. In many cases, the purpose or meaning of some code may not be clear. Adding comments can make the program easier to understand. In this code example, the lines that start with a single quote (also known as the apostrophe ‘) are comment lines.
- *Modularity in Code Design. Modularity* refers to the design of code structure so that each block of code is isolated from the rest of the program; that is, each block can perform its designated task without depending on the state of other blocks and without being interfered with by the actions of other parts. In this way, each block of code is independent of the others. Program logic and flow of execution are localized and are much easier to trace. Such a code structure is easier to debug, review, and revise.
- *Elegant Algorithms.* An algorithm is a systematic approach to solving a programming problem. It usually involves the iterative application of a group of defined steps to arrive at the solution. An algorithm is said to be elegant when its logic is easy to trace and implement, and is efficient in terms of its speed and storage requirements. Given a fairly complex problem, an infinite number of algorithms can be constructed to solve it. Some are efficient; some are not. For example, the sorting problem (arranging data elements in order) has a simple goal, yet literally hundreds of algorithms exist for sorting. For a fairly large volume of data, the difference in speed between the best and the worst algorithm is a factor of several hundred times. Careful identification and selection of algorithms can make a great difference in the performance of your program.
- *Code Maintainability.* Often, the requirements of a program change over time. A change in the requirements necessitates revision of the program; thus, efforts are involved in maintaining the program. All the internal factors just mentioned have an impact on code maintainability; however, code maintainability is not limited to the aforementioned factors. This is particularly true in VB. Some code by nature is more generally applicable than other code. *General applicability* refers to the attribute of certain code blocks that will not necessitate any revision even when the program requirements are changed. For example, in many cases, you may need to clear the content of the controls (visual objects) in the visual interface. One way to perform this is to refer to the controls by name and assign a certain value to these controls. In VB, you can assign the same value to these controls without referring to their specific names. The latter approach is more generally applicable because no matter how the names of the controls are changed, you can still use the same code to perform the task. The use of the more generally applicable code will undoubtedly result in a program that is much more maintainable. An example is provided in Chapter 7.

The diagram in Figure 1.2 summarizes this discussion.

Figure 1.2
External and internal criteria for sound programs



How are these criteria for sound programs treated in this book? The functionality issue depends on the programmer's understanding of the application requirements. All the other issues are considered throughout the book where applicable. In most cases, your attention is called to why a certain problem is solved with a particular block of code instead of other alternatives. The performance implications of different code structures and algorithms are considered where applicable. The user-friendliness criterion is very important, especially in the context of data-entry screens and user interfaces. Chapter 10, "Special Topics in Data Entry," is devoted in particular to treating this topic thoroughly.

Steps in Program Development

When you are ready to develop an application program, there is a fairly standard set of steps you should follow. Your progress will be much smoother when you observe these steps, which are outlined as follows:

1. *Analyze and Define the Problem.* As discussed previously in this chapter, the first requirement of a sound program is that it must meet the needs of the application. A clear understanding of the problem and goals is the first step in developing the program. Only when the needs and requirements of the program are clearly understood can you determine how the program is to look and act.

Tip
When developing a program, you can save a lot of time, headaches, and effort in revision and rework if you know its exact requirements/specifications at the very beginning. Analyze the problem carefully, and define the requirements clearly before proceeding.

2. *Design the Visual Interface.* Based on your analysis and understanding of the problem, you will be able to design the visual interface for the program. You will start to work with the IDE. You will need to decide what data fields should appear on a form. This process can become quite involved. VB provides various visual objects (controls) that you can use to represent these data fields. It takes a careful analysis to determine which VB control will be the best given the nature of a data field. Chapter 3, "User Interface Design: Visual Basic Controls and Events," provides an analytical framework for this purpose.

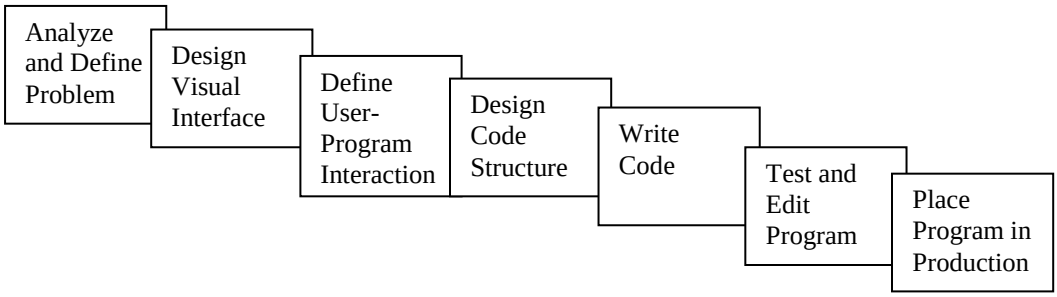
3. *Define User-Program Interaction.* The user interface consists of the *visual aspect* as described earlier in this chapter and the *behavior* in which your program responds to the user's actions and to what happens in the computer internally. You will need to determine what your program should do in detail. The user's actions and system activities are recognized as *events*. User actions that can trigger events include pressing a key, clicking a control, or making a selection from a menu. System activities can also trigger events. Examples of these activities include loading and closing a form. You must first be aware what events will be triggered when each of these actions occurs. Based on how you decide your program should react, you will place the code in the pertinent event to respond accordingly. Some of those commonly seen events are discussed in Chapter 3. Additional events are discussed in other chapters, where the events need to be handled.
4. *Design the Code Structure.* On appearance, the code you will develop to respond to an action should be placed in the event that the action triggers. Your code structure will simply be dictated by the responses you want your program to carry out. In reality, however, it can be much more complex. You will be introduced to various complex situations. Suffice it to say, it pays to analyze the situation thoroughly before writing any code. As your program grows into multiple modules (a module is a code window that contains your code), you will also discover that many code blocks can be shared so the appropriate module(s) in which to place these blocks can become an issue. Your design of the code structure can have far-reaching implications on the maintainability of your code. The importance of this design phase cannot be overemphasized. Some authors maintain that more than 60% of your time and effort should be expended on this phase to produce a sound program. Some of these design issues are discussed in Chapter 10, "Special Topics in Data Entry," Chapter 11, "Menus and Multiple Form Applications," and Chapter 13, "Object-Oriented Programming." The context of the discussions in these chapters pertains to minimizing code duplication, and dealing with the issues associated with large projects.
5. *Write Code.* Based on your design, you will then develop the code to perform the activities that your program requires. In addition to ensuring that the code performs what is called for, you should pay particular attention to your coding style. As mentioned, this includes the mechanical aspects of indenting and following the naming conventions. You will start to learn about this aspect beginning in Chapter 2.
6. *Test and Edit the Program.* Unless a program is very short, it is rare that the program anybody develops will run correctly the first time. The program can have various kinds of errors:
    - ***Syntax errors*** result from the failure to follow the rules to put various code elements together.
    - ***Semantic errors*** result from the difference between what the programmer codes and what the programmer actually means. For example, the programmer may code the statement, `A = B = C` thinking that the three variables will have the same value after its execution; however, the statement actually has quite a different meaning in VB.
    - ***Logic errors*** result from the differences between what the programmer believes a block of code will do and what the program actually does. This type of error is the trickiest and can take days or even weeks to resolve. In most cases, you will test run your program and discover some unexpected problems or results. You will then modify your code to solve the problems you have identified and test it again. You will repeat the process until no problem or unexpected result is encountered.

Testing and editing code constitute most of your effort in the coding activity. To minimize the possibility of encountering mysterious logic errors, it is advisable to break the code into small blocks and test it often. This makes it easier to identify the range of code that causes the error. A smaller number of statements make the error source easier to track down and correct.

7. *Place the Program into Production.* After a program is thoroughly tested, it is ready to be placed in actual use. A program that works with live data and produces real results is called a *production program*. When you are developing and testing the program, you work in the IDE. A program to be placed in production should be the compiled object program (an executable file), which can run without the IDE. Additional explanations of the IDE and the compiled executable file are discussed in Chapter 2.

Figure 1.3 summarizes the steps of program development.

Figure 1.3
Steps of Program Development



When a beginner is working on a programming exercise, some of these steps may be combined, or may not be applicable. For example, the exercise may be so simple that it requires little analysis of the problem or design of code structure. Once you understand the goal, you can start to work on designing the visual interface, writing the code, and testing the program. As you proceed and the problems become more complex, however, it pays to revisit this complete roadmap. You will appreciate the importance of those steps that you initially could do without.

1.3 How to Use This Book

This section offers a few suggestions on how to use this book. It should be emphasized that hands-on practice is the key to successfully learning a programming language. You learn to program by programming, just like you learn to swim by swimming, not by observing. Perhaps, another analogy can help illustrate the point.

Learning the Language: An Analogy

In many ways, learning a computer language is no different from learning a human language to convey ideas. Before you can speak a language fluently, you must build a wide vocabulary. You will also need to know the rules to put the vocabulary together to convey an idea. The sentences must be structured correctly. Without the correct structure, these sentences will be hard to understand. In the case of computer languages, the computer will simply refuse to understand.

It is also important to note that a grammatically correct sentence may not necessarily convey exactly the idea that you have in mind. For example, the two sentences, "You like a dog" and "You are like a dog" are both grammatically correct. But you know how different the reactions can be from your listener. In VB, the statement `A = B = C` may have quite a different meaning from what you may think. (Wait until Chapter 5, "Decision," for the explanation.) You must also choose the right vocabulary to express the right ideas. Furthermore, even if the sentences convey the correct meaning, you may find that there are better ways to convey the same thing; that is, different sentences may have different effectiveness in conveying the same idea.

This discussion suggests that there are four aspects of speaking a language (writing a VB program):

- Vocabulary
- Grammar (syntax rules)
- Semantics (meaning)
- Effectiveness

To express yourself well in a language as a speaker or writer, not just as a listener or reader, you need to build a large vocabulary, follow the syntax rules, clearly understand the meanings of the vocabulary, and find the proper expression to convey the ideas effectively. It takes a lot of practice to learn to speak a new language fluently and convey ideas effectively. At the beginning, even a very simple sentence bears repetition to gain the desired level of familiarity.

Conventions for Syntax Specification

Syntax rules were mentioned in the previous subsection. In this textbook, when introducing new VB features, the rules for code are explained. The following table lists the conventions used in this text.

Syntax Rule	Example	Remark
Regular typed words represent exact words to be used in code.	`Option Strict On`	Every word should be typed as it appears
Italicized letters represent a concept, a placeholder to be replaced with a specific item.	`Dim` *VariableName* `As Integer`	*VariableName* should be replaced with a variable name in actual coding.
Mutually exclusive alternatives are separated by vertical bars ("|").	`ByRef|ByVal`	Either ByRef or ByVal can be specified.
Items specified in brackets are optional.	`[`*ParameterList*`]`	A parameter list can be present or omitted.
Brackets with items separated by vertical bars indicate one item or none can be specified.	`[ByRef|ByVal]`	Either ByRef or ByRef can be specified or omitted.
Braces with items separated by vertical bars indicate one item must be specified.	`{On|Off}`	Either On or Off must be specified.
The plus (+) sign indicates that two keys should be pressed at the same time.	`Ctrl+R`	The Ctrl key and the R key must be pressed simultaneously.

Ellipsis (…) indicates the omission of some parts.	`Dim` *Variable1* `[As` *data type*`][,` *Variable2* `[As` *data type*`]]...`	More can be specified.

Indentation: Four spaces are used for standard code indents. Two-space indents are used for statements that actually appear in one code line but have to span over two PDF lines.	`If A = B Then`
	`  X = Y`
	`End If`
	
	`'This statement is`
	`  actually in one`
	`  code line`
		The If block shows a standard code indent. When one code line has to span over two lines on a PDF page, a two space-indent is use to represent the continuation.

The Case for Hands-On

When you are developing a VB program, you will be the speaker or writer of the language, not just a listener or reader. To be good at VB programming, you will do exactly as you would to learn a new human language. You will learn the vocabulary of VB. You will also learn the rules to combine the elements to form expressions and statements that express what you want the computer to perform. As you progress, you will also discover different ways of getting the same result. In many instances, the different ways may have different performance implications. You will be faced with design choices and will select the more effective and efficient approach.

From learning the vocabulary to making an intelligent design choice, the development of your proficiency takes a lot of hands-on practice, which enables you to:

- Gain familiarity not only with the IDE, but also with the vocabulary and syntax rules of VB without exerting stress on your memory. Many details are involved in each code line. They do not stand out in your memory until you actually write the code. Familiarity will make you much more efficient in handling the same or a similar problem. You will find your second attempt takes much less time than the first one. Familiarity improves your efficiency and enhances your confidence.
- Gain a more in depth understanding of the interrelationship between different parts of the code. You will be able to see the effect of the code you place in different events more readily. This level of understanding allows you to develop and trace programming logic more easily.
- Identify opportunities for code improvement. Some programming problems can appear different in their context but call for the same or similar solution. The first time you encounter the problem, you might just be glad that you have a solution. The second time, you may see some faults of your original solution and attempt to improve on it. In the process, you will explore, experiment, and discover new solutions. As a result, you will gain even more knowledge. You will be able to write programs that perform effectively, much like a speaker delivering an effective speech.

Try This

You have seen how to code to display a line of text on the Console object. Write a line of code in VB to display the following text:

```
Practice makes perfect.
```

How do you feel about your experience with the experiment? As easy as the code may be, you should find that there are details to pay attention to. Only hands-on can provide this additional insight. The solution is given immediately before the summary section.

To repeat, merely reading the text is not enough. You may gain some knowledge of VB by reading, but you still need the hands-on practice to obtain the familiarity, the deep understanding, and the skills for effective delivery. This book provides many devices that facilitate your hands-on practice with VB. To gain that proficiency with VB, do the following:

- Try the examples in each chapter, and ensure that all the results are as expected. In most cases, you can try the examples as you read. As you work with the examples, you become familiar with the code structure and its use.
- Test the code in the "Try This" boxes. These boxes allow you to see the effect of the code and provide you with a deeper understanding. The benefit of these exercises can be immense.
- Complete the "Explore and Discover" exercises. These exercises deal with topics that are not discussed in detail in the text and broaden your working knowledge of VB. They are designed so that you find answers to VB questions in a fun way. They serve to illustrate how you can explore VB on your own. Hopefully, by working with this group of exercises, you will become more adventuresome and daring, willing to try anything without being afraid of encountering an error. After you develop this mental capacity toward VB and the computer, whenever you have an intriguing question, you will be able to devise your own code strategy to test your question, discover the answer, and figure out how and why your code works out the way it does. You will be able to learn a lot on your own.
- Work on the exercises and projects suggested by your instructor—and even more, if you can find time. These assignments give you an opportunity to put together what you learn from the text in a meaningful way. The acid test of your VB programming skill rests in whether you can successfully develop the code to perform the requirements of these assignments. These assignments vary in difficulty and fields of applications. They challenge you in different ways and can be very interesting and intriguing. The ample exercises offer plenty of choices in taking on the challenge.
- Have a thorough understanding of a chapter before proceeding to the next. This is particularly important for the first seven chapters. Each chapter is built on the preceding ones. Together, the first seven chapters form the foundation for the remainder of the book. This also means that *the first few chapters deserve a lot of your study time*. I have had some students who thought that the first few chapters were fairly easy, and thus devoted relatively little time to studying this material. These students had to work twice as hard later just to catch up, while those who worked hard initially had a much easier time and more fun with the later chapters.

Keep this in mind: Programming is more of an art than a science. The more you do, the better and the faster you can program. Hands on. Hands on. And more hands on.

Beyond the Content Coverage

There is so much to learn about VB. It includes many controls and objects that can be used in a wide range of applications. These controls and objects have many features. It takes several books just to document all these features. As a result, it is impossible for a textbook to cover all aspects of this language.

This book, however, provides a special feature that can help equip you to explore, learn, and expand your knowledge in VB on your own. Starting from Chapter 3, each chapter contains several special boxes titled "Look It Up." These boxes show you what types of information on VB you can obtain from the online help file. These boxes also serve as a reminder that a lot of valuable information is available at your fingertips, and are intended to help you build a habit of looking up your questions in the online help file. Follow the instructions and perform all the suggested lookups. You will learn a lot more by just doing this. Better yet, getting familiarized with the help file can be the best resource in your study of VB.

After you become acquainted with the help file, you will be able to appreciate the wealth of information that is readily available. While you are writing your program, the help file is there for your use. It provides many details that textbooks may not have. Above all, it covers *all* the features of VB. If you decide to pursue a topic not covered in this book, you will be able to proceed comfortably by browsing the file for the needed information. Chapter 2 has a section that shows you how to browse the online help file.

Tip

The solution to the "Try This" box in the preceding section is as follows:

```
Console.WriteLine("Practice makes perfect.")
```

Common mistakes include (1) failing to include a dot (.) between Console and WriteLine, (2) failing to enclose the text in a pair of double quotes (not single quotes), and (3) failing to enclose the quoted text in a pair of parentheses.

Summary

- Visual Basic evolves from BASIC, which by the modern standard, has several drawbacks, including inflexibility in accommodating user needs, unattractive user interface, and low maintainability.
- Visual Basic has several advantages:
  - Its visual elements allow easy changes in user interface design.
  - The object-oriented syntax is easier for the programmer to develop the vocabulary and remember the syntax. More importantly, the object-oriented language enhances code reusability.
  - It is event-driven. This allows the user the flexibility to decide the sequence of activities to carry out in performing a task.
- The term, Visual Basic can refer to the language, the software processing a program written in that language, or even both.
- A sound program should possess the quality that meets both the external and internal criteria. The external criteria include functionality, efficiency, and user-friendliness. The internal criteria include consistency in coding style, code clarity and readability, modularity in code design, elegant algorithms, and code maintainability.
- You should follow the standard set of steps in developing application programs. These steps are: (1) analyze and define the problem; (2) design the visual interface; (3) define the user-program interaction; (4) design the code structure; (5) write code; (6) test and edit the program; and (7) place the program into production.

- Learning a programming language is like learning a human language. You learn to build a large vocabulary, follow the syntax rules, understand clearly the meanings of the vocabulary, and find the proper expression to convey the ideas effectively.
- It takes a lot of practice to become a proficient programmer. Hands-on is the only way to develop the necessary skills.
- It is strongly recommended that you do the following in using this book:
    - Try the examples in each chapter.
    - Test the code in the "Try This" boxes.
    - Complete the "Explore and Discover" exercises at the end of each chapter.
    - Work on all assignments suggested by your instructor.
    - Understand each chapter thoroughly before proceeding to the next. This is particularly important for the first seven chapters.
    - Perform all the lookups as suggested in the "Look It Up" boxes.

Review Questions

1-1. What is Visual Basic? Is it a language, or is it a program?

1-2. What is an interpreter? A compiler? Based on the discussion in the text, if both are available (separately) to process a source program, which one will you use? (*Hint:* Your choice should depend on the stage of your program development because one is more convenient but the other is more efficient in execution. Note also that fortunately, you do not have to make such a choice for VB. Chapter 2 explains why.)

1-3. How is Visual Basic different from its predecessor, BASIC?

1-4. What is *inheritance*? How does inheritance enhance code reusability?

1-5. What is an event? How is programming in an event-driven language such Visual Basic different from that in a procedure-oriented language?

1-6. Explain the following terms:

Program
Statement
Code

1-7. Computers are becoming faster in speed and larger in memory and storage size. Why should the programmer still be concerned about program efficiency in speed and in storage usage? (*Hint:* Your computer may be performing more than one task at a time.)

1-8. What does *user-friendliness* mean? What constitutes a user-friendly program?

1-9. Why is it important to observe consistency in coding style?

1-10. What factors can enhance code clarity and readability?

1-11. What does *modularity* mean? Why is it an important consideration in code design?

1-12. How do you judge whether an algorithm is elegant?

1-13. What factors affect the maintainability of a program?

1-14. Enumerate the steps that a programmer should follow in developing a program. What can happen if these steps are not followed?

1-15. Why is hands-on practice important to learning Visual Basic? What benefits can you gain by doing a lot of hands-on practice?

1-16. What benefits can you gain by developing a habit of browsing the online help file?