A Web-Based Introduction to Programming

A Web-Based Introduction to Programming

Essential Algorithms, Syntax, and Control Structures Using PHP and XHTML

Second Edition

Mike O'Kane

Carolina Academic Press

Durham, North Carolina

Library of Congress Cataloging-in-Publication Data

O'Kane, Mike, 1953-

A web-based introduction to programming : essential algorithms, syntax, and control structures using PHP and XHTML / Mike O'Kane. -- 2nd ed.

p. cm.

Includes bibliographical references and index.

ISBN 978-1-59460-844-5 (alk. paper)

1. Computer software--Development. 2. Internet programming. I. Title.

QA76.76.D47O43 2011

005.1--dc22

2010046893

Carolina Academic Press
700 Kent Street
Durham, North Carolina 27701
Telephone (919) 489-7486
Fax (919) 493-5668

www.cap-press.com

Printed in the United States of America.

To my dear mother, with all my love and appreciation

Contents

Preface

The problem I have tried to solve with this textbook is, quite simply, how to effectively introduce general programming concepts to students who have never programmed before. Perhaps like me, you have found yourself frustrated by textbooks that try to cover too much too fast, make inappropriate assumptions about what a student already knows, or take sudden leaps in complexity when providing examples and exercises.

I believe that the purpose of an introductory programming course is to help students gain confidence and develop their understanding of basic logic, syntax, and problem-solving. They do not need to learn all aspects of a language or even learn best practices — these are topics for the next course level. The question is: how to provide the kind of hands-on experience that supports active learning without overwhelming the beginning student with too much syntactical and programmatic detail?

I have tried many approaches over the years before settling on a Web-based approach, using minimal PHP and XHTML code to develop small, interactive Web applications. This approach has proved very successful. Many students take the trouble to report how much they enjoy the course, how much they have learned, and how well the course has served them in subsequent courses and in their professional life. I also hear from many students who tell me that the course positively changed their opinion of programming as a career or subject of interest, which is most gratifying.

Some instructors may be frustrated by my "keep it simple" approach, and may have concerns that my coverage of the PHP and XHTML is insufficient. The book uses a minimal set of XHTML tags and PHP functions and makes use of some arbitrary conventions to keep the focus on basic concepts that are common to most languages. To give a couple of examples: PHP print statements are used rather than echo statements, and these statements always include parentheses and double quotes in order to illustrate text output in a manner consistent with most other languages. And the code examples mostly use a pairing of .html and .php files of the same name to produce simple interactive applications (the .html file provides a form that is processed by the .php file). I have tried to differentiate between these editorial strategies and actual language requirements. The last chapter ("Where to Go From Here") explains which practices are standard and which are particular to the textbook, and also suggests best practices and areas for further study.

Two major topics that are hardly touched in this book are Object Oriented Programming and database/SQL programming. I considered how to incorporate these but came to the conclusion that it was simply too much for a beginning course.

I hope that you will find the book useful for your purposes, and that, if you use the book, you will provide your own feedback and suggestions for the next edition.

Intended Audience

The book is designed to serve:

- Instructors teaching introductory programming, programming logic and design, or Web programming courses, who want a textbook that engages students and provides a solid preparation for subsequent courses, but avoids overwhelming beginners with too much syntactical detail or program complexity.
- Traditional and online students taking a first course in programming, programming logic and design, or Web programming.
- Web designers, graphic artists, technical communicators, and others who find that their work increasingly requires some degree of programming expertise, and need an effective, hands-on introduction.
- Others who wish to learn the basics of programming, either for personal interest, or to explore the possibility of a career in this field.

Approach

The book takes a fairly novel approach, allowing students to learn program logic and design by developing a large number of small Web-based applications. Students love working with the Web, and this approach has other important benefits:

- Important concepts such as client/server design, server-side processing, and interface-driven code modules can be introduced in the form of working applications, and then applied in hands-on exercises.
- Students not only learn the essential control structures and syntax of a programming language, but also learn to use a markup language (and style sheets). This makes sense in today's programming environment where markup and programming are increasingly integrated components of a networked application.
- The material is relevant to students across a range of disciplines: Computer Science, Information Systems, Technical Communications, Network Systems, Digital Media, Web Technologies, Database Programming, and other technology-related fields.
- The focus on hands-on problem-solving and fundamental structures prepare students for next-level, language-specific courses such as PHP, Java or

C++, without replicating a great deal of material, while the syntax covered here is generally consistent with these and other languages.

The book makes use of a programming language (PHP), and a scripting language (HTML), but does not attempt to provide a complete overview of either. Instead, students learn sufficient syntax to convert requirements into working applications using basic programming structures, arithmetic and logical expressions, user interfaces, functions, and data files. The focus remains on basic concepts, logic and design, algorithm development, and common programming procedures. The book provides context throughout, explaining why each topic is important, and referring students to related career paths.

Although the book focuses on Web-based applications, there is NO requirement for a network–based programming environment. The book uses a standalone Apache Web server (the open source xampp distribution provided by the Apache Friends group) that students can install on a USB drive or home computer simply by unzipping a file. As Chapter 2 demonstrates, students can begin programming in HTML and PHP in literally minutes.

Features

Each chapter begins with clearly stated learning outcomes. Each topic is introduced using examples of simple program requirements that are first developed as algorithms and interfaces and then realized in working code. Code statements and control structures are explained step by step.

Different programming topics are treated in separate chapters. Even topics that are commonly combined, such as counting loops and event-controlled loops, have their own chapters so that students have the chance to develop and apply their understanding of each separately.

Each chapter includes quizzes that have been carefully developed to test the student's understanding of the chapter's learning outcomes. The questions have been tested extensively in the classroom.

Three different types of coding exercise are provided at the end of each chapter:

- Fixit exercises provide small programs that include a single error of some kind. These exercises help students improve their problem-solving ability, test their understanding of key concepts, and develop tracing and debugging skills.
- Modify exercises provide working programs that must be modified to perform a somewhat different or additional function. These exercises help students determine how and where to add new code, and test their ability to read and understand existing code.
- Code completion exercises allow students to apply concepts and tools covered in the chapter by developing new applications. These exercises test the student's ability to: understand requirements, develop algorithms, and pro-

duce working code. The code completion exercises follow consistent themes that are developed throughout the book, so that students can more readily appreciate the value of new functionalities that they learn in each chapter.

Templates for each exercise contain partially completed code so students don't waste time typing (and debugging) code that is not relevant to the problem at hand. The templates also help instructors to streamline the grading process.

The textbook CD includes a standalone Web server that can be installed on a fixed or portable drive simply by unzipping a file (so students can bring the software with them to work on computers at any location).

The server installation includes textbook folders that contain all code samples and exercise templates. Students can complete the exercises simply by opening, editing, and saving the appropriate files. Assignments can be turned in simply by zipping and submitting the appropriate chapter folder.

The textbook appendices provide additional learning resources designed to: (a) help individual students with particular needs or interests (for example file/folder management, additional references, and help debugging code); and (b) deliver useful topics not included in the chapters (for example data representation, additional control structures, and multi-dimensional arrays).

The textbook Web site ensures that both students and instructors have access to the most current resources associated with this textbook. The Web site includes all materials found on the CD, and also provides access to additional exercises, test banks, slide presentations, quiz solutions, code solutions, and other instructional resources. The web site can be found at:

http://www.mikeokane.com/textbooks/WebTech/

Changes to the Second Edition

As an instructor myself I know how frustrating it can be to adapt to changes in textbook editions, so I have tried not make significant structural changes in this new edition. Throughout the book I have made minor edits to clarify concepts and procedures where students have seemed to struggle. Chapter 2 provides a much more efficient procedure to install and run the Web server. Examples of flow charts have been added to the chapters that cover control structures. Chapter 11 now has significant new material to further explain the use of associative arrays and Web sessions (note that these additions have significantly extended the length of this chapter and it may require more class time). Chapter 12 provides more extensive discussion of the use of include files, and a more coherent introduction to objects. Chapter 13 has been renamed "Where to Go From Here" and provides additional references to important current technologies. The appendices have also been reviewed and updated. One important language update has been the replacement of the PHP **split**() function with the **explode**() function, since the split() function is now deprecated. The explode() function is identical in the way that it is used.

Chapter Overview

Chapter 1: Introducing Computer Programming. Students learn the relationship between machine language and high-level languages, and review common tasks that computer programs typically perform. The work of a programmer is described, and the software development cycle is explained. The chapter highlights and summarizes significant important design approaches such as algorithm development, interface design, client/server design and object oriented programming. Different programming languages are identified, and the distinction is made between interpreted and compiled languages, and between markup and programming languages. Standalone and network applications are also contrasted.

Chapter 2: Client/Server Applications – Getting Started. This chapter prepares students for the hands-on work they will perform in subsequent chapters. File types and local and Internet addressing schemes are explained. Instructions are provided to install, run, and test the required software. Students are shown how to create, store, and run a number of sample applications in order to become familiar with the process of using a text editor, saving files, running the Web server, and viewing the results in a Web browser.

Chapter 3: Program Design – from Requirements to Algorithms. The general characteristics and requirements of effective instructions are explored, using human and program examples. Students walk through the process of reviewing simple requirements, creating input, processing, and output (IPO) charts, designing the interface, and developing solution algorithms. The chapter introduces sequence, selection and control structures, variables and assignment operations, and arithmetic and logical expressions.

Chapter 4: Basics of Markup – Creating a User Interface in HTML. This chapter explains the significance of data rendering, and provides a brief overview and history of Hypertext Markup Language (HTML), up to the present XHTML implementation. Commonly used HTML tags are explained, and the student is shown how to apply these to create and organize simple Web pages. Cascading style sheets are introduced. Students are shown how to create HTML forms to obtain user input as a first step in developing interactive Web applications. HTML Tables are used to perform simple form layout

Chapter 5: Creating a Working Program – Basics of PHP. This chapter teaches sufficient PHP language syntax to process user input received from HTML forms, perform simple arithmetic, and produce formatted output. In the process, students learn to code arithmetic expressions, use standard operators and functions, create and work with variables, and identify and fix both syntax and logical errors.

Chapter 6: Persistence — Saving and Retrieving Data. This chapter explains the difference between persistent and transient data, and introduces text file processing as well as basic database concepts. Students learn to: open, read, write, and close text files; work

with multiple files; parse lines of data that contain multiple values separated by some kind of delimiter.

Chapter 7: Programs that Choose – Introducing Selection Structures. This chapter introduces selection control structures and demonstrates the use of algorithms to solve problems requiring simple selection. Students learn to use IF and IF..ELSE structures, Boolean expressions, relational operators, truth tables, simple string comparisons, and testing procedures.

Chapter 8: Multiple Selection, Nesting, AND's and OR's. This chapter develops examples from Chapter 7 to handle problems associated with input validation and more complex requirements. Students explore the use of compound Boolean expressions, nested selection structures, chained IF..ELSEIF..ELSE selection structures, and multiple but independent selection structures.

Chapter 9: Programs that Count – Harnessing the Power of Repetition. This chapter introduces loop structures with a focus on count-controlled FOR loops. Students learn how to refer to the counting variable within the loop, and how to use loops to generate tables, crunch numbers, accumulate totals, find highest and lowest values in a series, select values from a file of records, and display bar charts.

Chapter 10: "While NOT End-Of-File" – Introducing Event-Controlled Loops. This chapter introduces WHILE loops and demonstrates the use of the priming read and the standard algorithm to process files of unknown length. The student is shown how WHILE loops can be used to perform various operations on a list of data values, and how a file of records can be processed and searched for specific records or field values.

Chapter 11: Structured Data – Working with Arrays. This chapter introduces numerically-indexed and associative arrays, and shows how arrays can be used to store, access, and update multiple-related values. The use of the FOR loop to process arrays is explained, and various array-processing algorithms are demonstrated. Students learn how to use associative arrays as lookups, and gain a better understanding of the way that data is received from HTML forms. Web sessions are introduced, and students learn how to use session variables to maintain session data between applications. This chapter is longer than most and additional time may be needed.

Chapter 12: Program Modularity – Working with Functions and Objects. This chapter demonstrates the importance of program modularity and introduces functions, include files and objects. Students learn to write their own functions, to build libraries of related functions, and to call functions from different applications as needed. Key concepts and examples of object oriented programming are also introduced in this chapter as an optional topic.

Chapter 13: Where to Go From Here. This last chapter provides a short overview of key concepts and technologies that the students may want to explore after completing this textbook.

The textbook also includes a number of useful appendices as follows:

Appendix A introduces data representation, and shows how binary values can store data for a wide range of purposes.

Appendix B provides an introduction to overview of file and folder management, file addressing schemes (including relative and absolute addresses), and the use of the command line with a list of common DOS and Unix command equivalents.

Appendix C provides help for students wishing to use different Web server installations.

Appendix D provides debugging help for students having trouble identifying and resolving code errors.

Appendix E provides additional material and references for students wishing to learn more about HTML and style sheets.

Appendix F provides additional information regarding PHP data types, and provides a short list of common PHP functions not covered in the book.

Appendix G provides additional coverage of common PHP operators and structures that were omitted from the chapters to avoid overwhelming the beginning student (for example, shortcut operators, the SWITCH statement, DO..WHILE loops, and multidimensional arrays).

Acknowledgments

This textbook could not have been created without the generous help and support of many others. In particular I want to thank my dear wife Constance Humphries for her invaluable technical advice, proof-reading, and daily encouragement and patience! My sincere thanks to Scott Sipe, Beth Hall, and all at Carolina Academic Press for their supportive style, professionalism and experience. Thanks to my fellow instructors at Asheville-Buncombe Technical Community College, especially to Charlie Wallin and Fred Smartt who field-tested the book, made invaluable suggestions and submitted any number of corrections. And thanks to all of those students who have learned with me and sometimes in spite of me as this book evolved in the classroom.

And a huge thank you to Kai 'Oswald' Seidler, Kay Vogelgesang, and all those who have contributed to the Apache Friends Project, and who continue to deliver and support the XAMPP distribution. So many of us owe you our great appreciation for your generosity of spirit!

About the Author

Mike O'Kane holds a master's degree in Systems Science (specializing in Advanced Technology) from Binghamton University. He has twelve years experience teaching computer science courses, including his current position at Asheville Buncombe Technical Community College in North Carolina. He also has extensive practical experience in the use of technology for learning, having worked at IBM as a short-course developer, NC State University as an Instructional Coordinator, and the University of North Carolina system as the first Executive Director of the UNC Teaching and Learning with Technology Collaborative. He has a passion for developing effective instructional content, and learning environments that promote rather than hinder student learning.

A Web-Based Introduction to Programming

Chapter 1

Introducing Computer Programming

Intended Learning Outcomes

After completing this chapter, you should be able to:

- Explain the difference between computers and other machines.
- Describe the purpose of the microprocessor's instruction set.
- Explain the relationship between the instruction set and machine language.
- List some common tasks that computer programs perform.
- Describe what programmers do.
- Summarize the stages of the software development cycle.
- Explain the importance of writing and communications for programmers.
- Explain the relationship between high-level programming languages and machine language.
- Distinguish between the purpose of a compiler and an interpreter.
- Explain the difference between standalone and network applications.
- Explain the difference between programming languages and markup languages.

Introduction

Welcome! If you have never programmed before, this book is for you. By the time you complete the chapters and exercises, you will have a good grasp of the basic logic and design of computer programs. The book is designed to teach common programming syntax and control structures in a manner that will prepare for you for further study in this field. It will also provide you with sufficient expertise to develop small, interac-

tive Web applications, using a combination of the HTML markup language and PHP programming language.

To get started, in this first chapter we will explore the general process of programming and define some important term and practices. For a book that is supposed to be hands-on this chapter is mostly descriptive! Don't be too concerned if some of the topics don't make complete sense yet. Your understanding will deepen as you work through the chapters and develop your own applications

What Is a Computer Program?

A computer is a **programmable machine**. Most machines, such as vacuum cleaners or ceiling fans, are **hard-wired**, designed to perform one task only. But a computer is different. Computers can perform any number of tasks by reading and executing **computer programs**, or **software**. Each computer program contains instructions to direct the computer's operating system and hardware for a specific purpose. The ability to read and execute programs is achievable because each computer contains a microprocessor which includes the computer's **instruction set**. The instruction set defines all of the basic commands that the computer can execute. These basic commands are very low-level activities such as adding numbers, moving a piece of data from one location to another, or comparing values. Each instruction in the instruction set is identified by a number, and so program instructions are actually issued as a list of numeric commands. Your programs work with the computer's operating system to issue commands to the computer to perform whatever task the program is designed to perform. If you've ever wondered why some programs run on one computer but not another, it is because different computers have different microprocessors and different operating systems.

The commands that make up the instruction set constitute the computer's machine language. So a computer program is essentially a set of instructions that tell the microprocessor to execute machine language commands in a particular sequence in order to provide a game, a Web browser, a business application, an email program, or some other useful service.

Although different computer programs serve quite different purposes, all programs share some important characteristics. Here are some common tasks that any computer program might typically perform:

Provide interactive environments for users: programs may use text-based input/output or Graphical User Interfaces (GUI's) to interact with users. Interfaces may include graphics, animations, audio, video, and other multimedia features.

Read and write data: programs may access, create, modify or delete data that is stored in files and databases.

Perform numerical calculations: programs can add, subtract, multiply, divide, and compare numbers, and can combine these operations to perform much more complex calculations.

Perform text-processing operations: programs can validate, convert, search, sort, compare, and replace text, and can construct reports, messages or documents such as this textbook.

Communicate with other programs and devices: programs may exchange data with other programs, cameras, scanners, Web browsers, satellites, cell phones, ATM machines, etc.

Control hardware: programs can control robots, satellites, aircraft, automobiles, printers, and other computers.

A single computer program may perform any combination of these operations. For example a computer game may look up player and game information in a file or database, provide an interactive multimedia environment for the player to play the game, perform numerical calculations, and communicate with programs running on other computers to allow multi-user play over the Internet. Similarly a payroll program may perform text-processing operations (such as validation and conversion), query and modify a database, perform numerical calculations, and send checks to a printer. What this means is that, as a programmer, you will want to know how to write programs that might include any or all of these operations.

What Do Programmers Do?

Programmers write program code, right? Well, actually programmers do a lot more than write code, and many programmers actually write very little code. One of the things that makes programming an attractive career for many men and women is that this work requires an appealing combination of **right-brain** (creative, problem-solving, brain-storming) and **left-brain** (logical, linear, sequential) activities. Another appeal is that there are a variety of career paths within the field, so you if you have a general aptitude you have a good chance of finding work suited to your personal interests. For example if you like to work with people you may find yourself drawn to software design, interface development, usability, or training. If you prefer to work with data, you may prefer server-side development, object design, or database administration.

Let's walk through the major stages in software development. This will clearly demonstrate the range of skills that programmers need in order to be successful in their work. Here we will simply summarize these steps so don't be concerned if this appears a little abstract right now. We will learn how to apply these steps to develop a working application in Chapter 3.

1. Evaluation of Requirements

In order to develop an effective software application we first need to determine the program's requirements. This usually requires careful reading and listening, asking questions, and careful documentation. Understanding and documenting requirements takes time and skill. A common mistake that beginning (and not just beginning) programmers make is to rush this important step in order to begin coding. Rushing the requirements phase can actually be very costly in terms of time, money and even professional relationships. So learn to go slowly and carefully when you are presented with a requirements document or when you first meet with a client. The more time you spend analyzing your program requirements (and asking questions if you are unsure about anything), the more easily the solution will appear. To repeat: the most important skills a programmer needs at this phase are to listen, read, ask questions, document carefully, and communicate effectively with clients, managers, and other programmers.

2. Software Design

Once you have a good idea of the software requirements, it's time to develop the design of your application. This may a one-person job in the case of a small application or may involve a design team. Software design can actually become a career path and some (often the most experienced) programmers spend most of their time evaluating requirements and designing applications that other programmers will then code.

Unless your requirements are quite simple, your application design will most likely consist of multiple code segments or **modules**, each of which can be developed separately. There are many advantages to a modular approach to software. Each module can be developed and tested **separately**, often by different programmers or programming teams. The modules can be developed **concurrently** which speeds up development time. This approach also allows each module to be developed by programmers with the most **suitable skills**. And a modular approach promotes **reusability**: useful modules can be shared by many different applications. Module design includes the design of testing procedures to test that a module will perform as expected when it is coded.

At this time, software designers commonly apply two very successful and widely-used strategies in order to achieve program modularity: **Client/server design** and **Object oriented programming** (**OOP**). These approaches are not exclusive and are both often integrated into the design process. Here is a brief overview of client/server design and OOP:

Client/server design: a client application usually provides an interface to the user, waits for the user to request some action, then calls a server application to process the request and send back a response. When the client application receives the response this is presented to the user and the process repeats. A common example of a client/server program is a Web application, where the Web browser performs as a client, sending the user's requests to a Web server for processing and then displaying the response that is received from the server, so that the user can review the results and de-

cide what to do next. You participate in a client/server interaction every time you use your browser to click a link, type and submit a URL, or submit a Web-based form.

An important aspect of client/server design is that a client application may call any number of server applications as needed, and these applications only execute when they receive a request. A server application may receive requests from many clients, for example a Web server for a major newspaper or online store may receive and respond to thousands of requests from different users every second. Figure 1-1 illustrates a number of clients contacting various servers for different purposes.

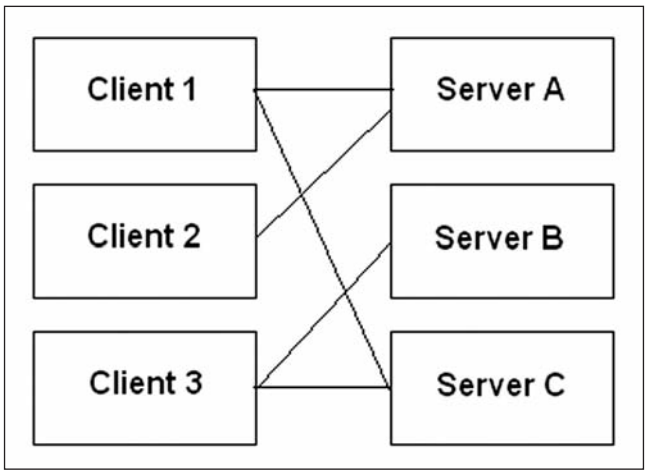

Figure 1-1: Client/server interactions

A useful design tool associated with client-server design is storyboarding where programmers sketch out an application as a series of screens (the user interface). Each screen provides the user with information and provides options to request services from one or more server applications. This is a useful way to consider the application from the point of view of the user, and is helpful in defining the various tasks that the application must perform. For example, Figure 1-2 shows the screens for a very simple client/server application to calculate an employee's pay based on their hours worked and hourly wage.

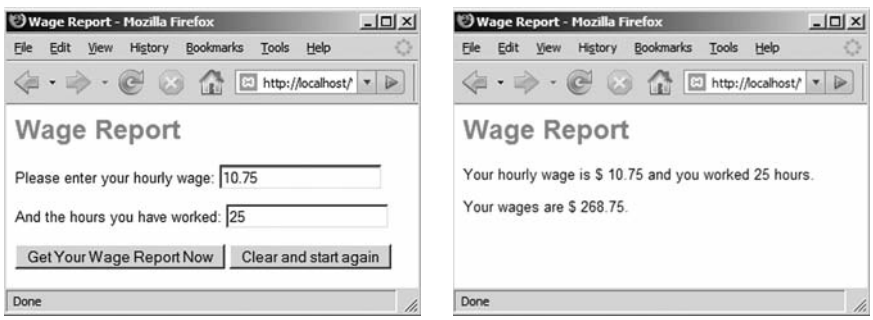

Figure 1-2: A simple client/server interaction

This example provides the user with two Web pages. When you type the URL to request the first page, the Web browser sends a request to the appropriate Web server, which sends back the data to display the page. The first page contains a form. When the clicks the "Get Your Wage Report Now" button, the Web browser sends a second request to the Web server that includes the data that the user has entered into the form. The server executes a program that has been developed especially to process these inputs. This program generates the content for the second page, which the server sends back to the browser for display to the user.

Client/server applications may consist of large number of screens and related interaction. Three primary advantages of client/server design are:

- A single server can deliver the same services to any number of clients.
- Changes to the interfaces and processing instructions for the application can be made on the server with no need to make changes on the client machines
- There is a high level of security since all data and processes are located on a single server.

In this book you will develop many small client/server applications similar to the example described above.

Object Oriented Programming (**OOP**): OOP allows a software designer to develop code independently of any particular software application. Instead code is developed to permit a broad range of **operations** to be performed on a specific **data set** of some kind, and this code can be used by any number of applications as needed. In OOP, the items that comprise the data set are known as **attributes** or **fields**, and the operations are known as **methods**. An **object** consists of a set of attributes along with the methods that are provided to work with the attributes.

To take a simple example, the data set (attributes) associated with an employee's wages might consist of an employee's ID, his or her hourly wage, and the hours that he or she worked. The operations (methods) that can be performed on this data might include obtaining the employee's ID, setting the ID, setting the hours worked, setting the hourly wage, calculating the weekly pay, printing a pay check, etc. Together these attributes and methods define a Wage object. Once the object has been developed, different applications can use it for different purposes. Our Wage object might be used by a program that allows administrative assistants to enter employee wage information, and also by a program that processes weekly checks. The wage entry program might use the methods to set the ID, hours worked and hourly wage of a Wage object, while the pay processing program might use the methods to obtain the employee ID, calculate the pay, and print a paycheck. Figure 1-3 illustrates this example.

What makes object oriented programming especially important is that objects are usually designed and coded for use by any number of programs. The designer of the object is not thinking about any particular use for the data set, but is rather planning for all possible uses. Once an object has been developed in this way, it can be used by many different applications as needed, with no need to duplicate the code.

There are many other advantages to object oriented design. For example, we say that the data within an object is **encapsulated**, meaning that it is only directly accessible by

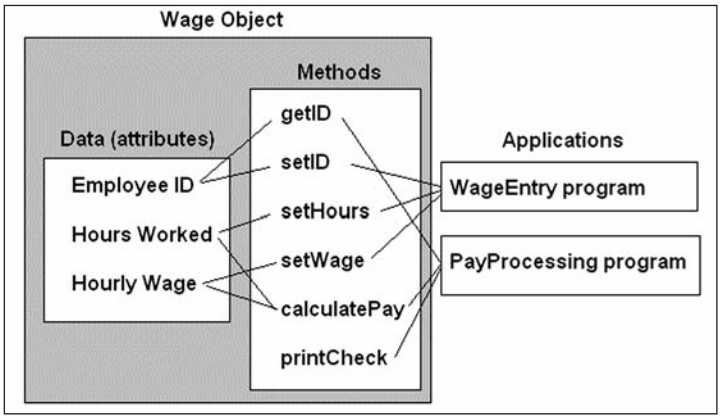

Figure 1-3: An example of Object Oriented Design

the object's own methods. The methods therefore provide an **Application Programming Interface** (**API**) to the data. This approach prevents the inappropriate or incorrect use that could occur if applications could access the data directly.

Objects are defined for many different types of data. As a very different example the data set (attributes) associated with a clickable button that is to be displayed on your computer screen might consist of the button's background color, the text that is to be displayed on the button, the color of the text, whether or not the button is currently active, etc. The operations (methods) that can be performed on a button will include changing the colors, setting the text that the button displays, checking whether the button has been clicked, etc. Together these attributes and methods might constitute a Button object. Once a button object has been defined, any programs that need to display buttons on the screen can create a Button object.

While object oriented programming is not covered in any detail in this book, Chapter 12 provides an introduction to this topic that will help you get started and prepare for further study.

Whether your programs are simple or complex, it helps greatly to stay away from a computer in the early phases of application design! Explore your design ideas using a pen and paper, sticky notes, a white board, even the backs of napkins! If you go to work as a programmer, you will find that this is how software design teams usually work. The reason is simple: using disposable materials prevents you from becoming too invested in any particular approach too quickly and encourages brainstorming and creativity. You will find that sketching out ideas on paper before you start coding will help you think things through and save you time in the long run. Once you have a clear idea of what to do you will be ready to develop your algorithms and application structure.

Important skills for software designers are creative thinking, organization, familiarity with data structures, a background in object oriented programming, writing documentation, and experience with client-server programming and interface design.

3. Algorithm Development

Once you have a general design for your application, it is time to develop **algorithms** for each code module that the application will need. An algorithm is simply a set of clearly written, unambiguous instructions that have been developed to perform a task of any kind . Algorithms are a critical component of software design, often written in some mix of English and programming language syntax (known as **pseudocode**) that programmers can easily understand. You will learn to develop algorithms using pseudocode in Chapter 3. The actual work of coding an application is ideally a fairly straightforward process of converting a carefully developed algorithm into a specific programming language.

The skills required for these activities include careful attention to detail, logical thinking, documentation (writing), and general programming experience.

4. Application Coding

This is the activity that is usually associated with programming! The algorithm for each program module is coded into a programming language. Each module is then carefully tested before the modules are assembled to produce the complete application. The most important skill required for coding is knowledge of the appropriate programming language, but programmers are expected to also have the experience to develop code efficiently, test thoroughly, find and fix errors (debugging) and document the code so that other programmers can refer to the documentation as needed.

Coding, testing, debugging and documenting require patience, thoroughness, and careful attention to detail.

5. Application Testing

Once the application has been assembled, it is time for thorough testing. While the development team may perform many tests for **correctness**, it is also important to test for **usability**. A development team may produce a terrific application that has been thoroughly tested and debugged but turns out to be a disaster when provided to end users. Why? Because the end users find the interface confusing and cannot easily perform the tasks that are most important for their purposes. Usability testing brings the users into the development process and ensures that the needs and concerns of the user are taken into account before the product is distributed. In the case of larger applications, usability testing is often undertaken by usability experts who may observe users working with the product to determine where improvements can be made. In smaller applications, the programmer may simply work with the client to find problems and get feedback.

Testing takes a great deal of patience, attention to detail, and thoroughness. Usability testing requires good listening skills, careful observation, and (if you are the

programmer) humility! It's very easy for any programmer to be so focused on his or her own design that the needs of the user become secondary. If you notice yourself getting impatient with a user who cannot figure out how to use your product, or who wants the software to perform differently, then it's probably time to step back, pay attention to the user's concerns, and reconsider your own design assumptions.

6. User Support, Training, Software Maintenance

So now you have a complete and well-tested application. The development process does not end there! You will also need to develop online or printed manuals and other documentation for use by your end users. You may also need to deliver some form of training. And the software must be maintained. Users will find problems that must be corrected, and suggest additions and improvements that will need continued programming support. User support and training is a career path for those who like to work with non-technical people and who can also communicate effectively with programmers and software designers.

Important skills are the ability to communicate with both technical and non-technical people, the ability to listen and to explain (verbally and in writing) procedures clearly and carefully, patience, and often a sense of humor!

The Software Development Life Cycle

Taken together these activities constitute the software development life cycle:

- Evaluation of requirements
- Application design
- Algorithm development
- Application coding
- Application testing
- User support, training, software maintenance

This is not a precise list—in reality these various stages may not be so neat and sequential, and you will see somewhat different versions of this process in every programming textbook and in every workplace. Development teams may implement these various stages differently. But no matter how these stages are defined, no part of the development cycle should be treated carelessly. As you gain experience as a programmer you will more fully appreciate the special characteristics and importance of every step. And perhaps you can already see how the field of software design attracts people who are both creative and logical, who enjoy using both the left and right sides of their brain equally.

The Importance of Writing and Communicating

Documentation and writing are frequently mentioned as important skills for programmers. Documentation is critical to software development and wherever your own career path takes you in the field of software design and development, you will need to be able to write carefully and communicate effectively. Clients, designers and managers must refer to well-written documents that clearly define requirements, design and code specifications. Everyone involved in the development process must listen carefully and communicate effectively. Programmers must document their working code so that other programmers can easily read and modify it (often the programmer who develops a piece of code will **not** be the programmer who is asked to make changes for the next version). Programmers must often give presentations to clients or to their team. The testing phase also requires extensive documentation that indicates what tests were applied, the results of these tests, and how problems were resolved. Software users will need course manuals and training materials. Lastly all maintenance procedures must be documented so that a complete record is always available regarding the current state and history of the software.

What Are Programming Languages?

We have explored what a computer program **is**, and what programmers **do**, but what about programming **languages**? What is a programming language and why are there so many different languages?

As we have seen, a computer program is basically a sequence of instructions that direct the computer's microprocessor to perform various commands contained with the computer's instruction set. These commands are issued in machine language. Machine language commands are very **low-level** (for example adding two numbers, or copying a value from one memory location to another). It would be extremely time-consuming to write instructions in the 0s and 1s of machine language, and this code would be very error-prone and difficult to debug, maintain, or modify.

Instead of using machine language directly, we develop programs using **high-level programming** languages . Examples of current high-level languages are C++, C#, Java, PHP, Python, and Ruby. Examples of older high-level languages are C, Ada, BASIC, COBOL. Fortran, Pascal, and perl (older does not necessarily mean no longer used—many applications written in older languages are still in widespread use, and programmers are still needed to maintain and even update these programs).

A high-level programming language consists of a set of special words, symbols and operators that a programmer uses to write program instructions. These instructions are often referred to as the program's source code and the process of writing source code is often simply called coding . High-level languages are quite easy for programmers to learn, and applications written in these languages can be developed very quickly and efficiently. However the computer can only understand machine language,

so once a program has been written in a high-level language, the code must then be translated into machine language instructions. There are actually two approaches to translating high-level code to machine language, either by compiling the code or by interpreting the code. The approach depends on the programming language that you are using.

Compilers and Interpreters

Some programming languages are **compiling** languages. This means that the entire source code for a program is converted (or compiled) into an executable file (.exe file) by special software known as a compiler. The .exe file that is produced by the compiler contains the necessary machine language instructions to perform the required task. Once a program has been compiled into an executable file, the source code is no longer required to run the program. End users of the program simply receive the .exe file. The programmers keep the source code in order to perform updates and produce new versions. Usually when you purchase standalone software from a store you are buying an executable program that has been compiled. Two advantages of compiled programs are:

- Compiled programs tend to run faster
- The end user does not have access to the source code (and so cannot change the program).

Since a compiler generates an .exe file containing the machine language instructions for a specific microprocessor and operating system, the source code must be compiled separately for different platforms (Windows, Macintosh, Linux, etc). Also, each time the source code is modified, the new version must be compiled again to produce a new .exe file. The new .exe file must then be distributed to the end users.

Other programming languages are **interpreted** languages. Execution of programs written in interpreted languages is dependent on a special program known as an **interpreter**. An interpreter translates the source code into machine language one instruction at a time. Both the interpreter and the source code are needed **every time that the program is executed** since no executable file is created. One advantage of this approach is that the same source code can be used on any computer. For example computers running Windows, Macintosh or Linux operating systems can each use their own language interpreter to translate the source code into the appropriate machine language for that platform.

A disadvantage of using an interpreter is that in most cases we do not want to deliver the actual source code to the end user of the software. But this approach works very well for network-based programs, such as Web applications, since these programs are not distributed to end users. In these cases, the source code is located on a server computer and executed each time a request is submitted by a client application, such as a Web browser. The source code can be modified quickly and easily with no need to recompile and redistribute the software every time a change is made. PHP is an ex-

ample of an interpreted programming language that is widely used to develop server-based Web applications.

Some languages provide both compiling **and** interpreting options, and some languages actually **combine** compiling with interpreting stages to achieve greater efficiency and platform independence. A notable example is the **Java** language. Java applications are compiled "up to a point" to produce an "almost executable" version in the form of **byte code** that incorporates many of the efficiencies of the compilation process, and then Java interpreters are provided for different platforms so that the same byte code can be distributed for execution on any machine (Windows, Macintosh, Linux, etc.) in order to achieve platform independence.

So Many Languages!

As computer technology evolves, new programming languages are continually developed to take advantage of the latest hardware and software design strategies. For example new languages were developed to implement the functionality of object oriented programming, and to allow client/server application development. There are literally thousands of different programming languages and often a computer programmer is expert in only a few of these. While each programming language has its own special syntax and characteristics, most languages are very similar in their overall characteristics and functionality, and use the same basic logical structures to write instructions. We will learn about these common characteristics as we work through this book. A programmer who is familiar with the general logic of programming, and who has experience coding in one or two languages can usually learn new languages quite quickly.

Standalone and Network Applications

Computer programs can be designed for use on individual machines (as standalone applications), or across networks (as network applications).

A **standalone application** is designed to provide a complete service on the local computer, usually the computer sitting on the user's desk. Standalone applications do not require any network connectivity, interacting only with the computer's operating system and other **utility software** on the local machine. If a new version of the application becomes available, or if updates are required, these must be also installed on every user's computer. Examples of standalone software are traditional word-processing and spreadsheet applications, image-processing software, many games, etc. At this time, most of the programs that you install on your local computer are standalone applications.

Network applications are programs that run partly or entirely on remote computers, linked to the user across a network of some kind. The more traditional network

application was simply installed on a single host computer and then accessed by many users remotely. Each user that signs on to the host computer is provided a user interface to work with the application. An increasingly important type of network application is a **client/server** application which consists of any number of component programs that work together across a network. Some components of a client/server application can be installed on a local computer, and perform as client programs. Other components of a network application are installed on network servers and perform as server programs. The server-based components respond to requests from client components of the application as needed. At the minimum the client component usually provides the user interface that allows the user to submit information and view the results, while the server component does most of the processing.

You are using a client/server application whenever you use your Web browser. The Web browser performs as a local client component, providing your user interface and allowing you to send requests to server programs all over the world. So you are using a client/server application whenever you use your Web browser to obtain information, shop online, or play an online game. Other common examples of client/server applications are ATM's and e-mail programs.

Client/server applications are becoming more and more common because they allow us to obtain services and perform tasks without need for special software on our local computer. We can expect to make increasing use of Web-based client/server applications as bandwdith increases since these require only a local Web browser and an Internet connection, rather than software that must be installed locally and continually updated. For example we are now seeing networked versions of word-processing and spreadsheet applications.

Markup Languages

So far we have discussed the purpose of programming languages. As you now know, the purpose of a programming language is to allow a programmer to write instructions that **process** data. In other words, programming languages are used to perform operations that read or modify existing data or generate new data for some purpose (for example to calculate wages, convert temperatures, or keep track of a game player's score).

Another type of language is a **markup language**. Markup languages are often used in conjunction with programming languages, but have a very distinct purpose. The purpose of a **markup language** is to provide markup instructions (usually in the form of tags) that simply **describe** data or indicate how data is to be **formatted**, or **rendered** (for example to define how data should be displayed on a Web page, or printed in a document). Markup languages are defined for wide range of purposes. For example, your word-processing program uses a markup language to save formatting instructions with your document as you type a report or letter.

The markup language that is used to render data for display in Web browsers is Hypertext Markup Language (HTML). The latest version of HTML has been defined using

a higher-level markup language know as eXtensible Markup Language (XML). XML is a "meta-language", used to define other markup languages using a consistent syntax and framework. Since HTML is now defined by XML, the current version is known as XHTML. We will follow the XHTML standard in this book.

Combining Markup and Programming Languages

In this course, you will learn the basics of programming by developing small, Web-based client/server applications using a programming language known as PHP, one of the most widely used programming languages for this type of application. Since you will use a Web browser to display your program output, you will also learn the basics of the HTML markup language to format your application input and output for display in your browser window.

Summary

A computer is a programmable machine. The computer's microprocessor includes the computer's **instruction set** which defines all of the basic commands that the computer can execute.

Commands to the microprocessor must be issued using the computer's **machine language**. A computer program is a set of machine language instructions that execute a sequence of commands to perform a useful task.

Different programs combine common components to achieve their purpose: interactive environments; read/write operations; numerical operations; text-processing, communication with other programs; control of hardware.

A software designer/developer requires a range of skills that combine creative problem-solving and logical processing. Documentation is an important part of the software process so writing and communication are important skills that are often not associated with this field.

Software development includes a number of stages: evaluation of requirements; application design; algorithm development; application coding; application testing; user support, training, software maintenance

Programs are written in programming languages which may be compiled or interpreted. The code containing the program instructions written in a specific language is known as source code. If the language is a compiler-based language the source code must be converted into an executable version which is then distributed for use. If the language is an interpreted language, the source code itself is distributed and this is then executed one line at a time by a language interpreter.

Some programs are designed to function as standalone applications, which means that they are installed locally and do not need access to other networks. A copy of a standalone application is required for every user. Network applications run over net-

works. A single network application can be installed on one computer and accessed across a network by many users.

An increasingly important type of network application is a client/server application, where client programs on local computers send requests to server programs on remote computers. These requests are processed and results returned to the client. A common example is a Web-based client/server application where a user's Web browser performs as a client to request services from server applications throughout the world.

Markup languages are not the same as programming languages. Programming languages provide instructions to **process** data. Markup languages provide tags to **describe** or format (**render**) data.

In this course you will learn to develop simple Web applications using the PHP programming language and HTML markup language.

Chapter 1 Review Questions

1. A web application is an example of:
   a. Object Oriented Programming
   b. Client/server design
   c. A microprocessor
   d. A standalone application

2. A program that requires the source code each time that it executes is using which method to convert the source code to machine language?
   a. A compiler
   b. An interpreter

3. Which approach is better when evaluating software requirements?
   a. Determine the requirements as quickly as possible in order to move on to the design and coding phases.
   b. Take time to analyze and clarify the application requirements.

4. What kind of thinking activities are most associated with the work of a programmer?
   a. Left brain activities
   b. Right brain activities
   c. Both left AND right brain activities
   d. Neither left NOR right brain activities

5. Which approach to software design focuses on data first?
   a. Storyboarding
   b. Object Oriented Programming
   c. Client/server design
   d. The microprocessor

6. Which language does the computer actually understand when it executes instructions for a program?
   a. Markup language
   b. High-level programming language
   c. Object-Oriented language
   d. Machine language

7. What is the computer's instruction set?
   a. The set of all programming languages that a computer can understand
   b. The set of all software that is currently available on the computer
   c. The basic set of commands that a computer can execute
   d. The rules for using a high-level programming language

8. What is source code?
   a. Programming instructions written in a programming language
   b. Program instructions that have been compiled into machine language
   c. The code used to identify text characters from languages all over the world
   d. The code used to identify memory addresses

9. What does an executable file contain?
   a. Programming instructions written in a programming language
   b. Program instructions that have been compiled into machine language.
   c. Formatted text
   d. An audio image

10. Which term applies to an application model where one program calls another program in order to have some task performed?
    a. Client/server
    b. Standalone program
    c. Instruction set
    d. An algorithm

11. What does an interpreter do?
    a. Reads and executes source code, one line at a time
    b. Converts source code into an executable file.
    c. Sends data from one program to another
    d. Converts and displays text that has been marked up

12. What does a compiler do?
    a. Reads and executes source code, one line at a time
    b. Converts source code into an executable file
    c. Sends data from one program to another
    d. Converts and displays text that has been marked up

13. When you use your Web browser to access information you are working with
    a. A client/server application
    b. A standalone application

14. What is an algorithm?
    a. A set of instructions to meet a requirement of some kind
    b. An executable file
    c. Program instructions that have been compiled into machine language
    d. Programming instructions written in a programming language

15. Which of the following is **not** a feature of modular application design?
    a. Concurrent development of each module
    b. Reusable code
    c. Separate testing of each module
    d. Less time needed to evaluate requirements

16. What category of language is used to describe data?
    a. Markup language
    b. High-level programming language
    c. Object oriented language
    d. Machine language

17. What languages will you learn in this course?
    a. C++ and HTML
    b. XML and Java
    c. PHP and FORTRAN
    d. PHP and HTML

18. Which is the correct order for these stages in the software development life cycle?
    a. application coding, application design, algorithm development
    b. application design, algorithm development, application coding
    c. algorithm development, application design, application coding
    d. algorithm development, application coding, application design

19. Is a Web browser a client application or a server application?
    a. Client
    b. Server

20. In the world of object oriented programming, operations on data are referred to as:
    a. Attributes
    b. Fields
    c. Methods
    d. Actions

Chapter 2

Client/Server Applications — Getting Started

Intended Learning Outcomes

After completing this chapter, you should be able to:

- Explain how Web programs function as client/server applications.
- Identify the content of a file by referring to the file extension.
- Locate files using the Windows addressing scheme.
- Locate files using the Internet (IP) addressing scheme.
- Use the localhost domain to access a standalone Web server.
- Identify the languages we will use in this course.
- Identify the software you will need to complete the hands-on work.
- Install the required software.
- Create, save and open an HTML document that is stored on a local Web server.
- Create, save and run a PHP file that is stored on a local Web server.
- Create, save and run an interactive Web application consisting of an HTML document that includes a form, and a PHP application that processes the form.

Introduction

This chapter will prepare you for the hands-on activities that you will perform as you work through the subsequent chapters of this book. First we will review **client/server** design with a focus on Web-based applications. Next we will look at how files and folders are organized and located, using **local** addresses based on **disk drives**, and **Internet** addresses based on **domain names.** This is important since you will need to be careful

to save files to the correct locations on your disk drive, and then open these files in your Web browser using the correct Internet address.

You will also install the server software that you need to work through the textbook and complete the exercises. This is a very straightforward procedure and you will be able to test that your server is running correctly by running some sample programs.

Once the server has been installed you will have the option to install a text editor, then you will be asked to type in a few small programs, save them, and run them using your Web browser. The code for these programs is provided. The idea is not to learn to develop programs (that comes later), but to simply learn the general process of using an editor to create code, saving your files to the correct location under your Web server, running the server, and then testing that your programs work correctly.

Client/Server Design in Web Applications

In a client/server design, client programs send requests to server programs to perform a task of some kind, just as you might ask someone else to do something for you. The server receives the request from the client and responds appropriately. The server program resides on a networked computer and can respond to hundreds, thousands or millions of client requests coming from any number of computers throughout an organization or from all over the world. Consider an online store that receives requests from many different customers all over the world every minute. An important advantage of client/server design is that new software installations and updates to existing software are performed on servers without requiring changes to the client computers.

Client/server applications are used world-wide across the **Internet**, and are also used to deliver services to members of an organization or company across **intranets** (an intranet is a private network).

A Web application is a familiar example of an Internet-based client/server design. In the case of Web applications, the client program is a Web browser. The Web browser runs on a user's personal or office computer. Each time the user enters a URL, or clicks on a link on a Web page, or clicks a Submit button after entering data into a Web form, the browser sends the user's request. The request is transmitted across the Internet to the appropriate Web server. The Web server receives and processes the request, then sends back a response for the browser to display in the form of a new Web page. Web server s may be located anywhere on the Internet and can accept requests from any client that has Internet access (Figure 2-1). In order to process each request, the server program may communicate with other programs or access databases or files.

This is a very efficient design since it allows the user to obtain all kinds of useful services without installing special software on their local computer (just the Web browser). Instead the user executes programs on remote servers, using an interface de-

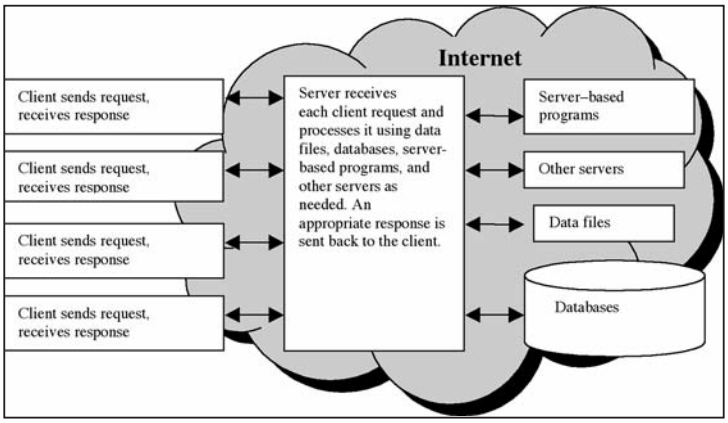

Figure 2-1: Example of client/server design

livered to his or her Web browser window. The server programs can be modified, and new programs added with no need for special installations on the user's computer.

In order to develop Web-based applications we must create files that contain the necessary instructions for our Web pages and programs, and store these files on a Web server. The server can then process the files as needed in response to requests.

In this course we will run a Web server on our **local** computer so that we can develop applications without any need for Internet access. In other words, our Web server will be located on the same computer as our Web browser. We will need to be careful to save our files to the correct locations so that our Web server can find them, and we will also need to provide the correct Web addresses (URL's) for these files when we wish to view them in our Web browser. Let's review how files and folders are organized on disks, and then learn how files can be located using the Windows and Internet addressing schemes.

Working with Files and Folders

Files are used to store data, all kinds of data. A file may contain text, images, videos, word-processing documents, programs, etc., but each file may only contain one type of data. The **file extension** usually indicates the format of the data that is stored in the file. This is very useful since the format indicates what type of program is needed to process the file. For example, a file with a **.jpg** extension contains image data stored using the **jpeg** image format, and can be opened by any image-viewing or image-processing program that can read this format. A file with an **.mp3** extension contains an MP3 audio file that can be handled by any MP3 player. A file with a **.zip** extension contains data that has been compressed using the ZIP compression scheme. To create zip files, or extract data from these files, you will need zip utility software.

Text files contain plain text (characters that can be typed on your keyboard). Text files can be viewed and edited using any text editing software. Files that contain plain text are often saved with a .txt extension. Often however, text files are given special ex-

tensions to indicate the specific purpose of the text that is stored in the file. In this course you will use a number of different extensions for your text files:

- **Plain text** files, using a **.txt** or **.dat** extension, will be used to store simple data for use by your programs. For example you might create a file named **scores.txt** that contains a list of student scores.
- **HTML** files, using an **.html** extension, will be used to store the markup instructions for Web pages. Our Web pages will include forms to allow the user to enter information that will be submitted for processing by our programs. For example you might create a file named wages.html that contains a Web page with a form for the user to submit their hours worked and hourly wage.
- **PHP** files, using a **.php** extension, will be used to store the source code for PHP programs. Our PHP programs will usually receive information from the user or look up data in files in order to perform useful processing operations and generate output. For example you might create a file named **wages.php** that contains the source code to receive wage information from a Web page, then calculate and display the pay.

Since these are all text files we can **create** and **modify** the content of using any text editor. However since these files contain text to serve different purposes, the file extension indicates what software is required to **process** the content of each file. A file with an **.html** extension is usually opened by a **Web browser**, since Web browsers are designed to interpret .html files and display the contents as Web pages. A file with a **.php** extension can only be executed as a set of PHP instructions if it is opened by a **PHP code processor**. A PHP code processor is included with the Web server that you will install as you work through this chapter.

Files are usually organized into **folders** for ease of management, and files and folders are stored on **portable** or **fixed** disks that are accessed through **disk drives** connected to computers. You can access files on disks located in **local** drives that are attached to your personal computer. You can also access files on disks in **remote** disk drives, attached to computers that are connected to your computer through a network such as the Internet. No matter where the drive is located, in order to locate a specific file on a disk, you need to be able to refer to some kind of **file addressing scheme**.

Locating Files and Folders on Computers Running a Windows Operating System

Most often when we want to locate a file or folder using a Windows operating system, we open **My Computer** (or **Windows Explorer**) and point and click our way to the file that we wish to work with. As a programmer, it is important to know that we can also reference a file by providing its unique **file path** or **address**. Every file and folder on a computer has a unique address that is based on its folder location and disk drive spec-

ification. Take a look at Figure 2-2, which displays a screen containing a list of four **folders**.

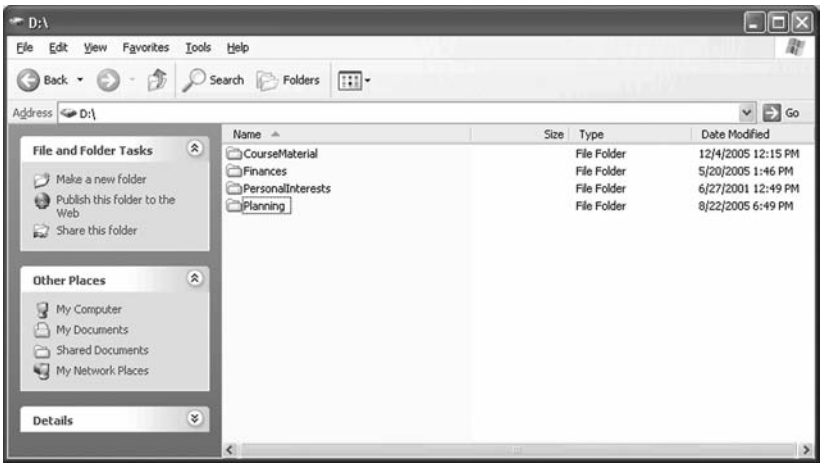

Figure 2-2: Examples of folders on a disk

Note that the address box shows the root address of these folders. They are located on the **D:** drive of the local computer. The address of the folder named **CourseMaterial** is therefore **D:\CourseMaterial**.

Folders can contain any combination of other folders and data files. Let's look inside **CourseMaterial** by double-clicking this folder (Figure 2-3).

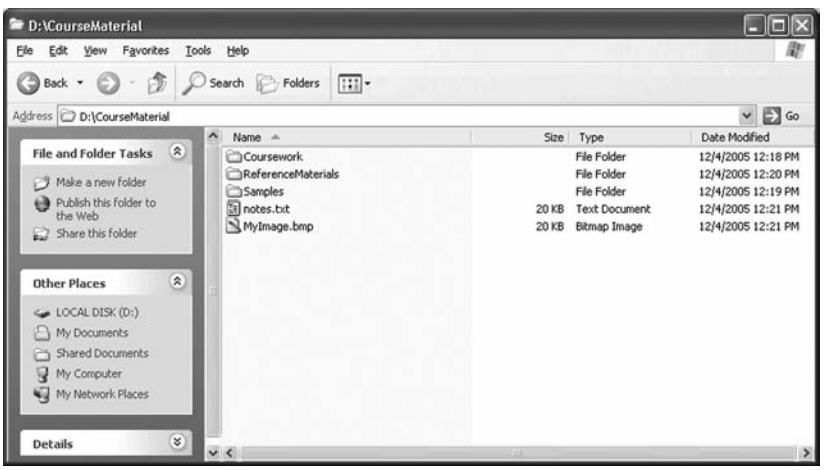

Figure 2-3: Inside the CourseMaterial folder

Notice the address in the address box is now **D:\CourseMaterial**, indicating that we have shifted our location to the CourseMaterial folder. This folder contains three folders (**CourseWork**, **ReferenceMaterials** and **Samples**) and two data files (**notes.txt** and **myImage.bmp**). The file extensions indicate the type of data stored in these two files: **notes.txt** contains plain text, while **myImage.bmp** contains a bitmap image. The com-

plete address of the file named **notes.txt** is **D:\CourseMaterial\notes.txt** and the address of the folder named Coursework is **D:\CourseMaterial\Coursework**.

Now let's open the Coursework folder (Figure 2-4).

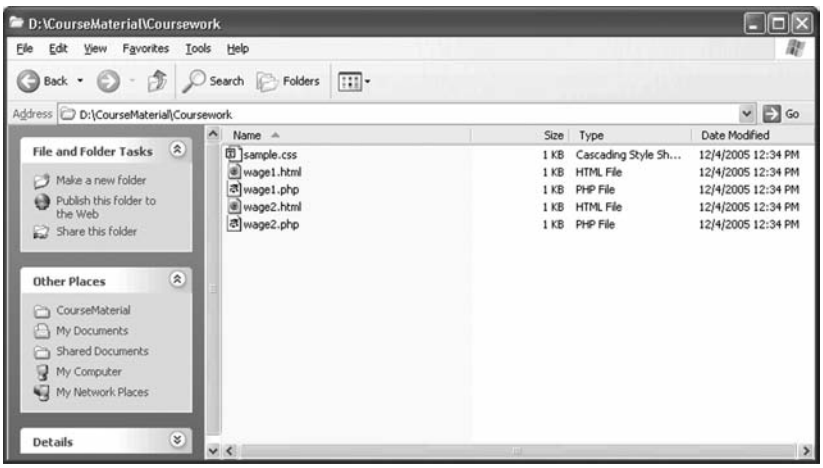

Figure 2-4: Inside the Coursework folder

Note that the address window now displays **D:\CourseMaterial\Coursework**. The Coursework folder contains five files named **sample.css**, **wage1.html**, **wage1.php**, **wage2.html**, and **wage2.php**. You can access any file directly using the file address. For example, instead of navigating to the Coursework folder by clicking through folders as we have done, we can open a file by simply typing the complete file address in the address window. For example, to access the **wage1.html** file we could just type: **D:\CourseMaterial\Coursework\wage1.html**.

Locating Files and Folders on the Internet

The Windows operating system assigns **drive letters** to identify each disk drive attached to your computer. The drive letter forms part of the address of any file stored on one of these disks. However this addressing scheme will not work when we need to locate files on the Internet, which consists of hundreds of thousands of disk drives attached to computers all over the world! An address that begins with **D:** would refer to a different drive on every computer in the network!

Instead all Internet addresses are based on the IP (Internet Protocol) addressing scheme. Each IP address references a specific **folder location** on a specific **disk** attached to a specific **computer** that is performing as a Web server somewhere in the world. Every IP address is unique so it is impossible for a single IP address to refer to two different locations. An example of an IP address is **69.147.83.197** which at the time of writing points to the PHP Group's Web site (this site supports the development and use of the PHP language). Try entering **http://69.147.83.197** in your Web browser's address window.

Imagine typing an IP address every time you needed to connect to a Web server! And how will anyone find our site if we move it to a different IP address? For ease of use and portability, we use **Internet domain names** to represent IP addresses. For example the domain name of the **69.147.83.197** address is **php.net**, so this domain name can be used in place of the IP address. Try typing **http://www.php.net**.

The Internet address of a specific file or folder usually combines a domain name with a file and folder path. For example, **http://www.php.net/license/index.php** is the Web address of a file named **index.php** which is located in the license folder. The exact folder and disk location of the license folder is referenced by the domain name **www.php.net**, which maps to an IP address that references a folder on a disk drive attached to a Web server, somewhere in the world. If these files are moved to a different Web server, the domain name is simply reassigned to point to the new IP address, so the URL does not need to be changed.

A Web address such as **http://www.php.net/license/index.php** is known as a **URL** (**Uniform Resource Locator**). Note that the separator used in URL's is the forward slash **/**, whereas the separator in our Windows addresses used the back slash \. That's because the naming convention for Windows file paths derives from the **DOS** operating system which uses the back slash, whereas the naming convention for Internet addresses derives from the **Unix** operating system which uses the forward slash. Fortunately Windows now also allows you to use the Unix forward slash when typing Windows file paths.

Often we write URL's without specifying a file name at the end. For example, if we were to type **http://www.php.net/license/** into our Web browser's address box, we would actually receive a file even though we did not include a file name. That's because Web servers are configured to add default file names to URL's if none is provided. For example if a Web server is configured to open a file named index.php by default, then the URL **http://www.php.net/license/** would actually access a file in the **license** folder named **index.php** using the address **http://www.php.net/license/index.php**.

For more information about file and folder navigation and addresses, refer to Appendix B.

Working with a Local Web Server

You are going to learn the fundamentals of program logic and design by developing Web applications using our own Web server. In order to work simply and securely you will use special software that allows you to run a Web server on your local computer with no need for Internet access. Although the Web server will be installed locally, in all other respects it will perform exactly as an Internet-based server. The Web server will receive requests from our Web browser, process these requests and respond appropriately (we hope!). Your programs and files will be stored and accessed locally, however, if you were to copy these to a Web server located on the Internet, your applications would perform in exactly the same way across the Web. The required software is easily installed and easy to use.

The IP address of your standalone Web server will be **127.0.0.1** and the domain name will be **localhost**. This is a special non-unique IP address and domain name that allows you to reference your own computer instead of connecting to the **Internet**. So for example, in order to run a program named **myWeb1.php** which is in a folder named **samples**, in a folder named **WebTech** on your local Web server, you would type the URL:

http://localhost/Webtech/samples/myWeb1.php

Actually, if you type that URL into your Web browser right now you will get a message that the page cannot be displayed! That's because you are trying to connect to a Web server that is not actually running, which means that the localhost domain is not available. Once you install and run the Web server (later in this chapter) this URL will work.

What Languages Will I Use?

You will write your Web programs using a combination of two languages:

HTML (Hypertext Markup Language) provides the **markup instructions** that you will use to create and format the Web pages that display the user interface for your Web applications. You will use HTML to display headings, paragraphs, forms, tables, buttons, and images. Since this is a course in logic and design you will not learn everything there is to know about the HTML language. Nevertheless you will learn sufficient HTML to easily extend your skills in subsequent courses or personal research. Everything you learn will be based on current HTML standards, which is known as XHTML.

PHP (**PHP Hypertext Preprocessor**) is a **programming language** that you will use to write server-based programs that process user requests by performing calculations, validating input, making decisions, reading data from files or databases, writing output to files or databases, returning results to the user, etc. This course covers sufficient PHP to teach basic programming logic and design as well as many important aspects of software development. This will prepare you for subsequent programming courses in PHP or other current programming languages.

Figure 2-5 shows the same example of a simple client/server application that you reviewed in Chapter 1.

Figure 2-5: Client/Server Example using HTML and PHP

Let's examine this a application more carefully. These are two Web pages. The first Web page is generated from HTML code located in an **.html** file on a Web server. The HTML code in this file has been designed to display a Web page that contains a heading, a form with two prompts and two input boxes, and two buttons.

When the user enters the information and presses the "Get Your Wage Report Now" button, the browser sends the user input (10.75 and 25 in this case) back to the server with a request to process a file containing PHP code. The code in the PHP file provides instructions to: (1) receive the data submitted from the first Web page; (2) calculate a wage based on this input; (3) generate a new HTML page to display the results. The second Web page is created by this PHP code and the server returns this to the Web browser for display to the user.

HTML and **PHP** are free technologies. You can find extensive information about HTML at **http://www.w3.org/MarkUp/** (the World Wide Web Consortium's HTML home page). You can learn more about PHP at **http://www.php.net/** (the PHP home page).

What Software Will I Need?

To complete your hands-on activities, you will need the following software:

- A **text editor** to create HTML and PHP files.
- A **Web browser** to submit requests to the Web server and display the Web pages that are returned for display.
- A **Web server** that can process requests sent to the localhost domain.

Read the following sections carefully and follow the instructions to install the necessary software for your computer. Once you have the software installed you will be ready to create and test some sample applications.

Installing a Text Editor

You will need a text editor to create and modify your HTML and PHP. You can use any text editor and there are many free editors available for both Windows and Macintosh computers. Some text editors are specifically designed for languages such as HTML and PHP, and contain special features that you will find useful as a programmer, for example code indentation, line numbering, and search and replace functions. Additionally a text editor that recognizes HTML and PHP will automatically apply different colors to special words, tags, and other syntactical elements of your code, which makes it easier to identify typing errors.

Three excellent freeware text editors for Windows are: **Crimson Editor** (an oldie but goody available from **http://www.crimsoneditor.com/**); **Notepad++** (**http://notepad-plus-plus.org/**); and **conTEXT** (**http://www.contexteditor.org/**).

Macintosh users might consider **Textwrangler**, a highly regarded text editor that be found at: **http://www.textwrangler.com**

In all cases installation is simple and for most users the default installation settings will be fine. Take some time to get used to the basic operations to use your editor – you will have a chance to do this later in the chapter when you create some simple applications. Don't try and learn everything — you can explore the full functionality of your editor as you gain experience.

IMPORTANT NOTE: do not use a word-processor (such as MS Word) to create and edit your code. You need to save your code as plain text files.

Installing One or More Web Browsers

You need a Web browser to view the examples and your own programs. Any major browser should be fine and you will already have at least one browser already installed on your computer. Professional developers like to use two or three browsers so that they can test their Web sites more thoroughly. You are encouraged to work with multiple browsers for this reason but this is **not** required for the material in this textbook.

If you want to install additional browsers, **Mozilla Firefox** is an excellent and freely available browser that is available for Windows, Mac or Linux (**http://www.mozilla.com/firefox**). **Apple Safari** is another great browser available for Macintosh and Windows (**http://www.apple.com/safari**). And you might also consider **Google Chrome** (**http://www.google.com/chrome**) for Windows, Mac or Linux. In all cases, installation is simple and for most users the default installation settings will be fine.

Installing Your Web Server

You must also install a standalone Web server that will process your PHP code and deliver Web pages to your browser. The textbook CD contains the **Windows** version of the **XAMPPLite** standalone server along with a document that contains complete installation instructions (installation is simple but be sure to follow the instructions carefully). If you are running Windows be sure to install the Web server from the textbook CD and not from any Web site since the CD version includes the coursework and samples files that are used with the textbook. If you do not have the CD you can download the same version (and the installation instructions) from the textbook Web site at:

http://www.mikeokane.com/textbooks/WebTech/support.php

The CD also contains a document with instructions for **Macintosh** users to download and install the MAMP standalone server. These instructions also tell you how to add

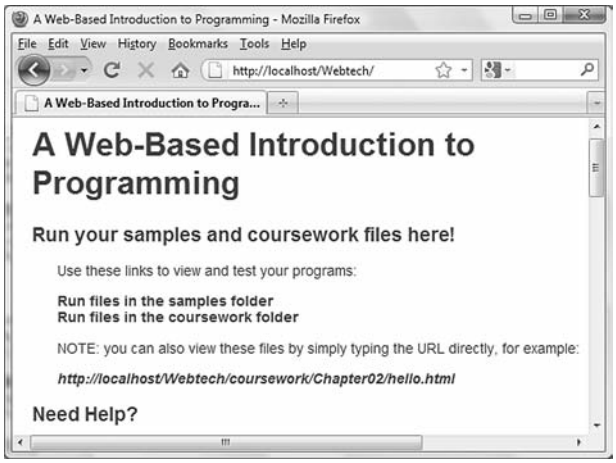

Figure 2-6: http://localhost/Webtech

the **Webtech** folder (also provided on the CD). This folder contains the coursework and samples files that you will use as you work through the textbook. If you do not have the CD. these instructions are also available on the textbook site.

The installation documents also explain how to **run** and **stop** your Web server, and how to test your Web server to be sure that it is working correctly. The material that follows assumes that you have successfully installed and tested your Web server, that you have at least one Web browser, and that you have a text editor to create and edit your program code.

Using Your Web Server

Now let's learn how to use the server to run and test our Web applications! First start the Web server if you have not already done so. Now open any Web browser and type the following URL in the address window:

http://localhost/Webtech

Your installation is configured so that this URL will open a Web page (Figure 2-6) that will help you to use this textbook. If this page displays then your Web server is running successfully. As you will see, the page provides links that make it easy for you to view and work with the samples and coursework files that are discussed in each chapter. Note that your Web server must be running in order for you to view this page (or to view any page with a URL that begins **http://localhost**). If you were to stop your Web server, this URL would no longer work and your browser would report a connection error.

Let's run a simple program from the samples folder. Click the "Run files in the samples folder" link, and then click welcome.html from the list of files that appear.

This should bring up a welcome page with an interactive form for you to complete (Figure 2-7).

Figure 2-7: http://localhost/Webtech/samples/welcome.html

Try completing the form and the press the "Submit the Form" button. If a second welcome screen appears with a response to your submission, then everything is installed correctly and working fine.

Using URL's with Your Web Server

Every file on the Web has a unique URL, or Web address. The **URL** consists of a domain name, followed by the folders and a file name that indicate the location of the particular file on the server. Our local Web server has the domain name **localhost** and this domain name points to the **htdocs** folder which is in the folder that contains your installation).

Earlier you opened the file **welcome.html** from your **samples** folder. Open the file again and this time notice the URL in your browser's address window:

http://localhost/Webtech/samples/welcome.html

All of the URLs of files located on your Web server will begin **http://localhost** because **localhost** is the domain name of your local server. But how does the Web server know where to find the **welcome.html** file in order to send the contents of the file to your Web browser?

The answer is that, by default, the Web server looks in the **htdocs** folder of your Web server installation in response to any URL that begins **http://localhost**. In other words the URL **http://localhost** is associated with the **htdocs** file folder on your drive, and in case you're wondering, the name **htdocs** is a shortened version of "hypertext docu-

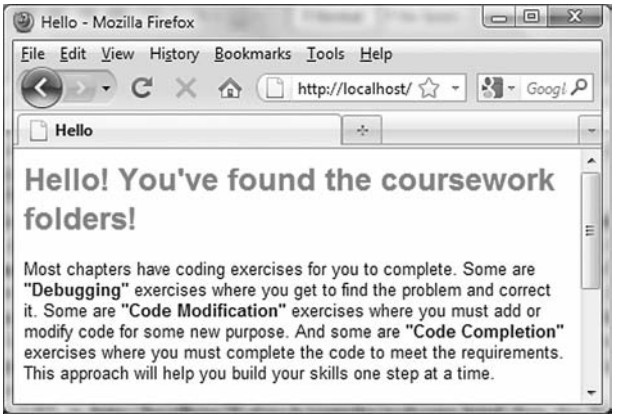

Figure 2-8: http://localhost/Webtech/coursework/Chapter02/hello.html

ments". Use **My Computer** (in Windows) or **Finder** (on a Macintosh) to locate the **htdocs** folder in your Web server installation.

So if the URL is **http://localhost/Webtech/samples/welcome.html** then the Web server will process the file named welcome.html that is located in the folder **htdocs/Webtech/samples**. Use **My Computer** or **Finder** to find this file on your disk — can you find it?

Be sure that you understand this. Each URL that begins **http://localhost/Webtech** refers to a file on your Web server that is located under the **htdocs/Webtech** folder and if you were to change the contents of any file inside this folder and the save your changes, the new version of the file would be displayed if you were to type the URL of the file in your Web browser.

As another example let's look at a file in your **coursework** folder. Type the URL **http://localhost/Webtech** and choose the **coursework** folder. Now click **Chapter02** and choose **hello.html**. This page displays a short introductory message about the kind of work you will do for each chapter (Figure 2-8).

Note that the URL for this file is:

http://localhost/Webtech/coursework/Chapter02/hello.html

Can you use **My Computer** (in Windows) or **Finder** (on a Mac) to locate this file on your disk?

We will keep all our work files in **two** folders under the **htdocs\Webtech** folder. The **samples** folder will contain all of the sample files referenced in the textbook. Try opening some of these programs in your browser now. You will notice that many files are listed in pairs with the same name but two different extensions (**.html** and **.php**). In these cases, click the **.html** files rather than the **.php** files to see what they do (the **.html** files display Web pages with forms that are used to "drive" the PHP programs). The **coursework** folder contains sub-folders for each chapter, and each chapter folder contains the files for your code exercises. If you were to try opening these files you will find

that many of them do not work correctly or generate errors — that's because they contain code that you will complete yourself as you work through the chapters.

To summarize, always remember to first **start** your Web server **before** you attempt to run your programs, and always **stop** the Web server and then **exit the Control Panel** once you have completed your work. The URL to your programs will always begin with **http://localhost/** and this should be followed by the names of any subfolders, followed by the name of the file that you wish to open. To avoid typing the complete URL's, you can just type **http://localhost/WebTech** (Figure 2-6) and then click through the links to open the file you want to view.

Always Use URL's to Run Your Web Applications!

As you probably know, in **Windows** you can often use **My Computer** to not only find a file but also to open the file (on a Macintosh you can use **Finder**). That's because your operating system associates the file type of a file with a default application that can handle that file type. For example, files with **.doc** and **.docx** files are usually associated the **MS Word** application, which is which why MS Word runs and opens the file when you click a file with one of these file types. Your computer usually associates **.htm** and **.html** files with your Web browser, so if you have a file with one of these extensions stored on your computer, and click this files in **My Computer** or **Finder**, the file will be displayed in your Web browser window. The page will display whether or not your Web server is running because you have opened it directly from your file system.

It's important to understand this because in this case you are not using a URL to request the file from your Web server, and that means that your Web server is not processing the file for use by the browser. Your .html pages will still display because these file do not require any special processing. But if you try to submit a form or if you click a .php file you will find a problem because .php files **must** be processed on the Web server before they can be viewed correctly by the browser. Since we are working with a combination of .html and .php files, you **must always** run the Web server and you **must always** use a URL that begins with the **localhost** domain to view your files in your Web browser.

To understand this better, let's try running a Web application without using a URL, just to see what happens.

Use **My Computer** (or **Finder** on a Macintosh) to navigate to the **samples** folder on your drive. Double-click the file named **addTwoNumbers.html**. A Web browser will probably start up, open the file and displays the Web page (Figure 2-9). This page looks fine, but notice the file path that is displayed in the browser's address window, which in Windows will be something like:

file://F:/xampplite/htdocs/Webtech/samples/addTwoNumbers.html

Because we opened the file in Windows, we see the Windows file path in the address window that begins **file://** rather than a URL that begins **http://localhost**. That tells you

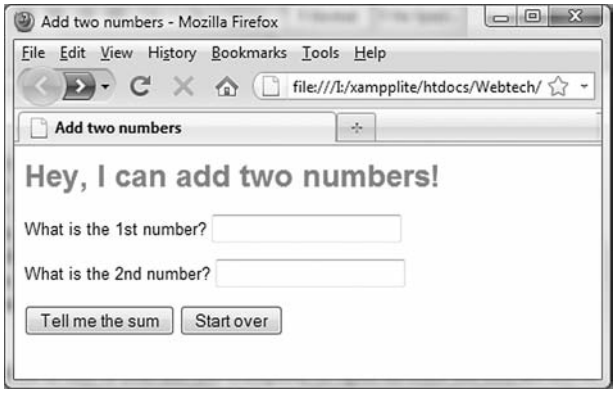

Figure 2-9: Opening addTwoNumbers.html using a Windows file path instead of a URL

that you are not connecting to the file through your Web server, which means you cannot process the form. If you type in two numbers and click the "Tell me the Sum" button, you will see something unexpected (Figure 2-10).

This doesn't make much sense! What is happening is that when the "Tell me the Sum" button was clicked, the Web browser opened a file named **twoNumbers.php** which contains the PHP instructions to process the form. However the browser was simply opening the file and displaying the contents. The browser was not submitting a request to your Web server to open and process the file. Your Web server includes the PHP processor which is necessary to execute the PHP instructions in this file. To access the Web server from your Web browser you **must** use URL's that begin with the domain name **localhost**.

So the correct way to view your.html and.php files is to always first run the Web server (if it is not already running), and then provide the URL to open the appropriate file. For example to correctly run the addTwoNumbers application, use the URL:

http://localhost/WebTech/samples/addTwoNumbers.html

Go ahead and do this to see that the application now works as expected.

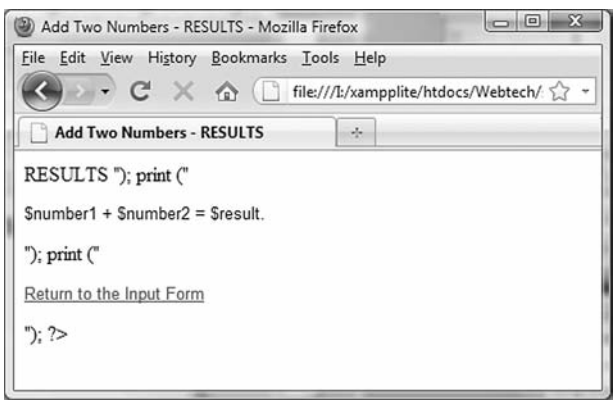

Figure 2-10: Result when opening addTwoNumbers.html without a URL

Where to Save Your Work Files

You will create and edit your HTML and PHP work files in the chapter folders under the **xampplite\htdocs\WebTech\coursework** folder. These files will produce small Web applications. You will use a text editor to develop the code for these files and then save them. It is important that you save these files in the correct location so that the Web server can find them when you type in the URL. Once you are ready to test your applications, be sure that the Web server is running and then use your Web browser to run your programs. The URL for your files will be:

http://localhost/WebTech/coursework/ChapterXX/yyy

where ChapterXX is a chapter number (for example **Chapter02**) and yyy is the name of a specific file, for example **myFirst.html**. To avoid typing the entire URL, you can open just type **http://localhost/WebTech** and then click the links on that page to obtain the appropriate folder and file.

The Importance of Frequent Backups

Always keep a recent backup of your **Webtech** folder on a separate disk (for example on your hard drive at home if you are using a portable disk as your primary workplace, or on a portable disk if your hard drive is your primary workplace. It is good practice to back up your work every time you make major changes, or at least once a week. If your drive crashes, you can reinstall the Web server on a new drive and then copy the backup of your **Webtech** folder into the **htdocs** folder of your new installation. Take your backups seriously — there is nothing worse than losing hours, days, or weeks of hard work.

Creating an HTML Document

In order to get started, you will first create a simple HTML document, store it on your server and then send a request to open the document from your client Web browser. Don't be concerned about understanding this document right now — you will learn about HTML in Chapter 4.

Open **Crimson Editor** or any text editor that you wish to use. Type in the text for **myFirst.html**, but write your name instead of "YOUR NAME", write today's date instead of "TODAY'S DATE", and write something about yourself to replace the words "WRITE ABOUT YOURSELF HERE":

```
<!-- Author:  YOUR NAME
   Date:      TODAY'S DATE
   File:      myFirst.html
   Purpose:   HTML Practice
-->
<html>
<head>
   <title>HTML Example</title>
</head>
<body>

   <h1>My Web Page</h1>

   <p>Hi! My name is <strong>YOUR NAME</strong>. Let me tell you a
   little bit about myself ... </p>

   <p>WRITE ABOUT YOURSELF HERE</p>

</body>
</html>
```

Code Example: myFirst.html

Figure 2-11 shows how your file might look if you are using Crimson Editor as your text editor .

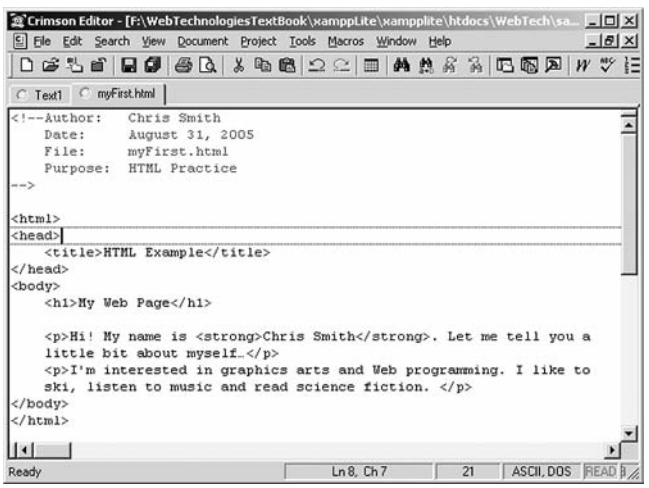

Figure 2-11: Using Crimson Editor

Choose **Save As** from the **File** menu and save the file as follows (your **Save in** address will reflect the actual location of your **xampplite** folder):

Save in: **xampplite\htdocs\WebTech\coursework\Chapter02**
File name: **myFirst.html**
Save as Type: **HTML**

The file has now been stored on the Web server. You can now submit a request to view this file from your Web browser. Be sure your server is running. Now type the following URL in your browser's address box:

http://localhost/WebTech/coursework/Chapter02/myFirst.html

When the browser submits this request, the Web server receives the URL and locates the file (myFirst.html). Since the file has an .html extension, the server simply sends the file contents back to the Web browser for display. Web browsers are designed to read HTML documents and treat any HTML tags as formatting instructions. We will learn more about this in the next chapter. Your HTML page should look similar to the screenshot in Figure 2-12.

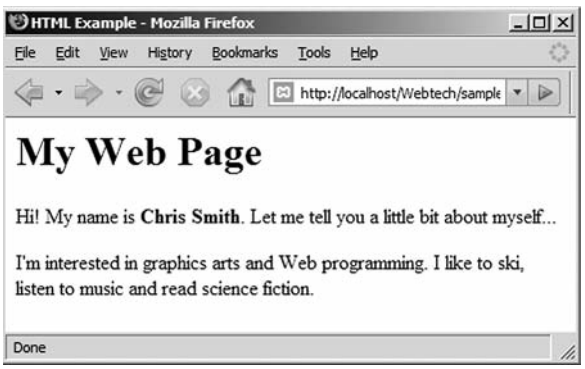

Figure 2-12: myFirst.html screenshot

If the link to **myFirst.html** does not work either your file name is different or the file is not in the correct location, or the Web server is not running.

If you wish to make changes to your document, simply edit your code in your text editor. Be sure to save your changes before viewing the file again and be sure to **refresh** the page in your Web browser otherwise your browser may continue to display the previous version.

Congratulations! You have just created a simple Web page, stored it on your local server, then accessed the page from a client (your Web browser)!

Creating a PHP program

Now let's create a simple PHP program. Don't be concerned about understanding this document right now — you will learn about PHP in Chapter 5. Type the code listing for **myFirst.php** in **Crimson Editor** (or your preferred text editor) exactly as written except type your name instead of "YOUR NAME" and today's date instead of "TODAY'S DATE". Note that you can copy and paste code from **myFirst.html** to save some time:

```
<!-- Author: YOUR NAME
   Date:     TODAY'S DATE
   File:     myFirst.php
   Purpose:  PHP Practice
-->
<html>
<head>
   <title>First PHP Example</title>
</head>
<body>

   <h1>Circle Calculation</h1>

   <?php
      $radius = 15.75;
      $area = pi() * pow ($radius, 2);
      $circumference = 2 * pi() * $radius;

      print("<p>A circle with a radius of $radius has an area of
         $area and a circumference of $circumference.</p>");

      print("<p>That's all that I have been designed to tell
         you!</p>");

   ?>
</body>
</html>
```

Code Example: myFirst.php

Choose **Save As** from the **File** menu and save the file as follows:

Save in: **xampplite\htdocs\WebTech\coursework\Chapter02**
File name: **myFirst.php**
Save as Type: **PHP**

Now view this in your browser by typing the following URL:

http://localhost/WebTech/coursework/Chapter02/myFirst.php

When the browser submits this request, the Web server receives the URL and locates the file (myFirst.php). Since the file has a .php extension, the server runs a PHP processor to execute any PHP code in the file and assemble a new HTML document. Once the PHP has been completely processed, the newly created HTML document is sent back to the Web browser for display. Your page should look something like the screenshot in Figure 2-13. We will learn more about this process in later chapters.

The text may wrap differently depending on the size of your browser window. If the link to **myFirst.php** does not work as expected, you may have used the wrong file name,

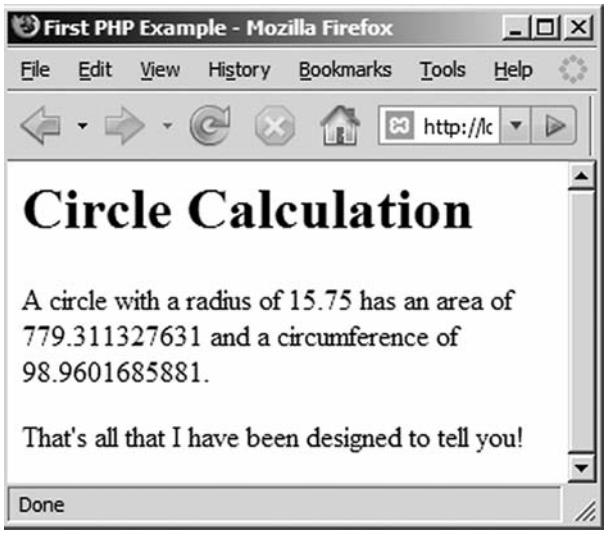

Figure 2-13: myFirst.php screenshot

or saved the files in the wrong location. Or you may have forgotten to start your Web server. You may receive an error message like this:

```
Parse error: parse error, unexpected T_VARIABLE in
F:\xampplite\htdocs\WebTech\coursework\myFirst.php on line 15
```

That means you have a syntax error in your PHP code. Programming languages such as PHP require a very precise syntax. There is a good chance that you may mistype something and your page may display differently (for example one or more numbers may not display as expected). Compare your code carefully with the example and see if you can find the errors. Once again remember to save your changes and remember to refresh the browser window to view your revised program.

Congratulations! You have just created a simple PHP program, stored it on your local Web server, and accessed it from a client (your Web browser)!

Creating an Interactive HTML and PHP Program

That last example displays information concerning a circle with a radius of **15.75**. We could improve the utility of this application by allowing the user to enter **any** radius. Next we will create a new version of this application that consists of two documents. The first (named **circle.html**) will be an HTML document that contains a form so that the user can submit a radius and submit this for processing. The second document (named **circle.php**) will contain a PHP program that receives the radius and calculates and displays the circumference and area of the circle. Here is the code for **circle.html**:

```
<!-- Author: YOUR NAME
   Date:     TODAY'S DATE
   File:     circle.html
   Purpose:  PHP Practice
-->
<html>
<head>
   <title>Circle Calculation</title>
</head>
<body>

   <h1>Circle Calculation</h1>

   <form action="circle.php" method="post">
      <p>What is the radius of the circle?
      <input type="text" size="20" name="radius" /></p>
      <p><input type="submit" value="Tell me the area and
         circumference" /></p>
   </form>
</body>
</html>
```

Code Example: circle.html

Choose Save As from the File menu and save the file as follows (your Save in address will reflect the actual location of your xampplite folder):

Save in: **xampplite\htdocs\WebTech\coursework\Chapter02**
File name: **circle.html**
Save as Type: **HTML**

The file has now been stored on the Web server. You can now submit a request to view this file from your Web browser. Just type the following URL in your browser's address box:

http://localhost/WebTech/coursework/Chapter02/circle.html

If you do this you the document will display. Assuming that you typed everything correctly, you will see that it contains will see a Web page with a form (Figure 2-14).

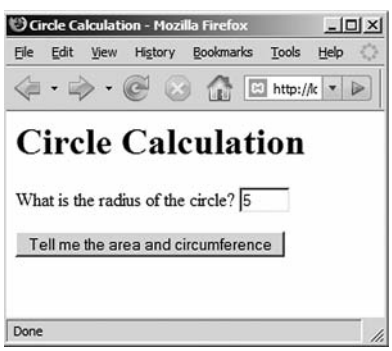

Figure 2-14: circle.html screenshot

If you enter a radius in to the text box and click the "Tell me the area and circumference" button, you will get an error message, similar to that shown in Figure 2-15.

Figure 2-15: Connecting to a page that does not exist

That's because the form on this Web page is designed to send the radius to a program named **circle.php** in order for it to be processed. The problem is that we haven't created the **circle.php** program yet! So let's do that right now!

Here is the code for **circle.php**:

```
<!-- Author: YOUR NAME
   Date:     TODAY'S DATE
   File:     circle.php
   Purpose:  PHP Practice
-->
<html>
<head>
   <title> Circle Calculation</title>
</head>
<body>

   <h1>Circle Calculation</h1>

   <?php
      $radius = $_POST['radius'];
      $area = pi() * pow ($radius, 2);
      $circumference = 2 * pi() * $radius;

      print("<p>A circle with a radius of $radius has an area of
         $area and a circumference of $circumference.</p>");
   ?>
   <p><a href="circle.html">Calculate another circle?</a></p>

</body>
</html>
```

Code Example: circle.php

Choose **Save As** from the **File** menu and save the file as follows:

Save in: **xampplite\htdocs\WebTech\coursework\Chapter02**
File name: **circle.php**
Save as Type: **PHP**

Now you should be able to use your form correctly. Open your Web document again:

http://localhost/WebTech/coursework/Chapter02/circle.html

Type a radius into the text box and click the "Tell me the area and circumference" button. This time you should see a new page that displays the area and circumference of a circle with the radius that you submitted (Figure 2-16).

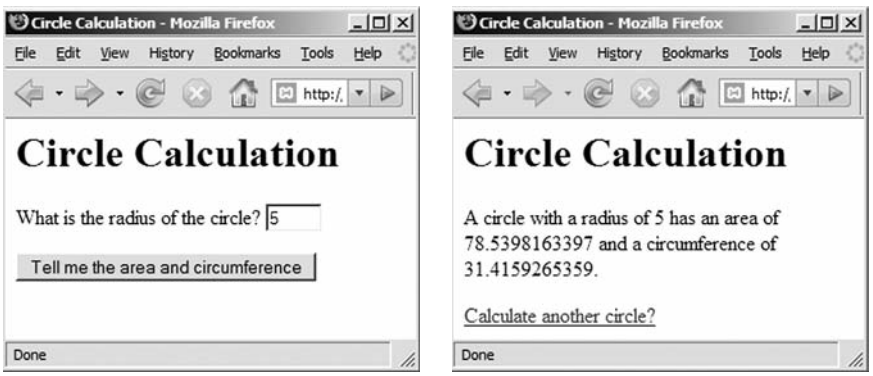

Figure 2-16: circle.html and circle.php screenshots

Note that you can click the "Calculate another circle?" link to return to the first page.

Congratulations! You have now created a simple Web application that: provides an input form, processes the input, and displays the result.

Do not be concerned about how these documents actually work at this point. The purpose of the current exercise is to give you practice using a text editor to type HTML and PHP code, saving your documents to the correct location on your disk, making corrections as needed, and viewing your applications in a Web browser using the correct URL. In the following chapters you will learn how to design and create Web applications that include HTML pages with forms and PHP programs that process these forms.

Summary

Client/Server applications are designed so that client-based software such as Web browsers can submit requests to server-based programs such as Web servers for processing. This makes it easy for large numbers of client programs to make use of common services. Another advantage is that new software installations and updates to existing software are performed on the server without requiring changes to the client computers. Client/server applications are used world-wide across the **Internet**, and are also used to deliver services on private **intranets**.

Web-based applications are developed by creating the required code and storing this on a Web server. Clients can submit requests to the server to process these files and return the results. When a server receives a request for an **.html** file, it simply locates and opens the file and returns the document for display by the Web browser. When a server receives a request for a **.php** file, it first processes the PHP instructions in the file and then returns the new HTML document that is generated. The Web server must be running in order for a client to submit requests.

Files usually include file extensions to indicate the data format that is stored in the file. Files can only be processed by programs designed to work with the file format. Text files can be opened by any text editing program, and sometimes contain text that requires specialized processing. In this course we will work with text files using three file extensions. Files with a **.txt** extension will be used to store data such as student scores or wage information. Files with an **.html** extension will contain HTML markup code for display in a Web browser. Files with a **.php** extension will contain PHP code that is processed by a PHP processor running on a Web server.

Under the Windows addressing scheme, file/folder locations are based on a drive letter to represent a specific disk drive, for example:

C:\xampplite\htdocs\WebTech\coursework\Chapter02\circle.html

Under the Internet addressing scheme, file/folder locations are based on an Internet domain name, for example:

http://localhost/WebTech/coursework/Chapter02/circle.html

The domain name is mapped to an IP address. In this case the IP address for the **localhost** domain is **127.0.0.1**

In this chapter you learned the basic steps to install, start and stop a local Web server. You also learned to create and modify HTML and PHP code using a text editor, and then view the results in a Web browser.

When you use the Web server, the URL's of your documents will always begin:

http://localhost/

This URL points to the **xampplite/htdocs** folder which is located wherever you installed the Web server on your disk. Be sure that the Web server is running before using this URL. Always be sure to use the Internet address when you attempt to view your .html and .php files. Do not use the Windows address, otherwise your .php programs will not be processed.

Don't be concerned about the actual content of the HTML and PHP code yet. The purpose of these exercises is to learn the procedures. You are now ready to learn how to design and code your own Web-based, client/server programs using HTML and PHP.

Chapter 2 Review Questions

1. Consider the following address: **D:\CourseMaterial\Coursework\wage1.html**
   Which statement is true?
   a. The file is stored in a folder named wage1.html
   b. The file is stored in a folder named Coursework
   c. The file is stored in a folder named CourseMaterial
   d. The file is stored in a folder named D:
   e. This address does not specify a file

2. Consider the following address: **D:\CourseMaterial\Coursework\wage1.html**
   What is the name of the file?
   a. wage1.html
   b. Coursework
   c. CourseMaterial
   d. D:
   e. This address does not specify a file

3. Consider the following address: **D:\CourseMaterial\Coursework\wage1.html**
   Where is the CourseMaterial folder located?
   a. Inside wage1.html
   b. Inside the Coursework folder
   c. On the D: drive
   d. It is not possible to tell from this address
   e. The address is incorrect

4. What type of address is this? D:\CourseMaterial\Coursework\wage1.html
   a. Internet address
   b. Microsoft Windows address

5. What is wrong with the following URL?
   http:\\www.w3.org\Markup\Guide\Style.html
   a. Internet addresses must use the forward slash / as a separator
   b. The file name is in the wrong location
   c. The domain name is the wrong location
   d. The file name is missing
   e. The drive letter is missing

6. What is wrong with the following URL?
   http://www.w3.org/Style.html/Markup/Guide/
   a. Internet addresses must use the back slash \ as a separator
   b. The file name is in the wrong location
   c. The domain name is the wrong location
   d. The file name is missing
   e. The drive letter is missing

7. Which component of a client/server application processes a PHP file?
   a. Client
   b. Server

8. What does HTML stand for?
   a. Highly Technical Markup Language
   b. Host Translated Markup Language
   c. Hypertext Markup Language
   d. Hands-on Technical Markup Language
   e. Hyper Transitional Markup Language

9. What is HTML?
   a. A markup language used to provide formatting instructions for text
   b. A programming language used to process input, perform calculations and other operations, and generate output
   c. An addressing scheme for URL's
   d. A name for a button on a user interface
   e. A form used to validate user input

10. What is PHP?
    a. A markup language used to provide formatting instructions for text
    b. A programming language used to process input, perform calculations and other operations, and generate output
    c. An addressing scheme for URL's
    d. A name for a button on a user interface
    e. A form used to validate user input

11. What is the domain name of the Internet address that you will use to access Web pages delivered by your standalone server?
    a. localhost
    b. www.w3.org
    c. WebTech
    d. samples
    e. welcome.html

12. Where should you save the html and php files that you create for this course?
    a. Anywhere on your disk is fine
    b. In the correct Chapter folder under xampplite\htdocs\WebTech\coursework\
    c. In the correct Chapter folder under xampplite\WebTech\htdocs\coursework\
    d. In the correct Chapter folder under xampplite\WebTech\coursework\
    e. In the correct Chapter folder under xampplite\coursework\

13. Which folder is divided into chapters?
    a. samples folder
    b. coursework folder

14. Which one of the following files can be found in your samples folder?
    a. gettingStarted.html
    b. gettingStarted.php
    c. courseWebSite.html
    d. quoteGenerator.html
    e. quoteGenerator.php

15. In your samples folder there is a file named tempConverter1.php. What should you do to run this program and find out what it does?
    a. Open the file using Windows Explorer
    b. Run your local Web server and then type the url **http://localhost/WebTech/samples/tempConverter1.php** in your browser window.
    c. Run your local Web server and type http://localhost/WebTech/ in your browser window, then click on samples and then click on **tempConverter1.php.**
    d. Either of the last two procedures will work but not the first one

16. In your samples folder, what happens if you run quoteGenerator.php in your browser window?
    a. The browser displays the same quote every time you run it.
    b. The browser displays a different quote each time you run it.
    c. The browser displays a different presidential quote each time you run it.
    d. The browser asks you to input a quote.

17. Which of the following file types does NOT appear in the samples folder?
    a. .bmp
    b. .html
    c. .jpg
    d. .php
    e. .txt

18. What is the correct URL for your myFirst.php document?
    a. http://localhost/WebTech/coursework/Chapter02/myFirst.php
    b. http://WebTech/coursework/Chapter02/myFirst.php
    c. http://localhost/coursework/Chapter02/myFirst.php
    d. http://WebTech/Chapter02/myFirst.php
    e. http://localhost/myFirst.php

19. In the samples folder, if you open welcome.html and submit the information but leave the first name and last name boxes blank, what does the resulting Web page display (among other things)?
    a. ERROR — INPUT IS MISSING!
    b. ERROR — YOU MUST ENTER YOUR FIRST NAME AND LAST NAME!
    c. Welcome!
    d. Welcome Whoever You Are!
    e. Welcome! You must enter your first name and last name!

20. In the samples folder, what is the difference between wage1.html and wage2.html?
    a. wage1.html allows you to input your hours worked and hourly wage but wage2.html does not
    b. wage2.html allows you to input your hours worked and hourly wage but wage1.html does not
    c. wage1.html allows you to input your name but wage2.html does not wage2.html allows you to input your name but wage1.html does not
    d. There is no difference between wage1.html and wage2.html

Code Exercises

The exercises for this chapter are simply intended to ensure that (a) you have your software installed and working correctly, and (b) you are comfortable with the process of creating, editing and running your Web applications.

1. First be sure that you have installed your Web server. Now run the XAMPP Control Panel and start your server. If you have any problems when you do this, first review the steps to install and run your Web server before you assume there is a problem with your installation. Refer to the **Installation Problems** guide on the textbook Web site at:

   http://www.mikeokane.com/textbooks/WebTech/support.php

2. With your Web server started, open a Web browser and type the URL:

   http://localhost/Webtech/samples/artGallery.html

   You should see a Web page that displays a "Welcome to the Art Gallery" heading and allows you to choose an artist from a drop down list. If you do not see this and receive an error message instead, first be sure that you typed the URL correctly. If you still receive an error, then (a) your Webtech folder was not included in your installation (use **My Computer** to ensure that this folder is located in your **xampplite/htdocs** folder), or (b) your server is not running (the server may not be running if you also get an error message when you type the URL **http://localhost** in your browser address window), or (c) your server was not installed correctly (this may be the case if you received an error message when you started the server in the XAMPP Control Panel). Make a careful note of all error messages and anything else that might be useful, then refer to the **Installation Problems** guide on the textbook Web site for help.

3. Assuming that the artGallery page is displayed, select an artist and then click the "Show me an Artwork" button. An artwork should now be displayed along with some additional information. If you do not see this, check the URL in your Web browser's address window. If the URL begins with **file://** then you are not using the right URL and you are not connecting to the Web server. Go back and use the correct URL. Remember that, to connect to your Web server, your URLs must **always** begin **http://** and in this case the URL should be:

   http://localhost/Webtech/samples/artGallery.html

4. If the artGallery application performed successfully then you are ready to work. To become familiar with the procedure that we will follow throughout this textbook, use your text editor as directed in this chapter to create **myFirst.html**, **myFirst.php**, **circle.html**, and **circle.php** and save these files in the **Chapter02** folder of your coursework folder if you have not already done this. Test each of these by running your Web server, opening a Web browser, and typing the appropriate URLs. You may need to fix some of your code but eventually each program should run as described in the chapter. **Do not bypass this exercise**. You need to be comfortable creating and running your applications and using your Web server in order to work through this book.