Preface

If you want to take up **programming** seriously, you've come across **the right book**. For real! This is the book with which you can make your first steps in programming. It will give a flying start to your long journey into learning modern programming languages and software development technologies. This book teaches the **fundamental principles and concepts of programming**, which have not changed significantly in the past 15 years.

Do not hesitate to read this book even if C# is not the language you would like to pursue. Whatever language you move on to, the knowledge we will give you here will stick, because this book will teach you to think like programmers. We will show you and teach you **how to write programs for solving practical algorithmic problems**, form the skills in you to come up with (and implement) algorithms, and use various data structures.

As improbable as it might seem to you, the basic principles of writing computer programs have not changed all that much in the past 15 years. Programming languages change, technologies get modernized, integrated development environments get more and more advanced but **the fundamental principles of programming remain the same**. When beginners learn to think algorithmically, and then learn to divide a problem instinctively into a series of steps to solve it, as well as when they learn to select the appropriate data structures and write high-quality programming code that is when they become programmers. Once you acquire these skills, you can easily learn new languages and various technologies – like Web programming, HTML5 and JavaScript, mobile development, databases and SQL, XML, REST, ASP.NET, Java EE, Python, Ruby and hundreds more.

About the Book

This book is designed specifically to teach you to think like a programmer and the C# language is just a tool that can be replaced by any other modern programming languages, such as Java, C++, PHP or Python. **This is a book on programming, not a book on C#!**

Please Excuse Us for the Bugs in the Translation!

This book was originally **written in Bulgarian** language by a large team of volunteer software engineers and later **translated into English**. None of the authors, translators, editors and the other contributors is a native English speaker so you might find many mistakes and imprecise translation. **Please, excuse us!** Over 70 people have participated in this project (mostly Bulgarians): authors, editors, translators, correctors, bug submitters, etc. and

still the quality could be improved. The entire team congratulates you on your choice to read this book and we believe the content in it is more important that the small mistakes and inaccuracies you might find. Enjoy!

Who Is This Book Aimed At?

This book is **best suited for beginners**. It is intended for anyone who so far has not engaged seriously in programming and would like to begin doing it. This book starts from scratch and introduces you step by step into the fundamentals of programming. It won't teach you absolutely everything you might need for becoming a software engineer and working at a software company, but it will lay the groundwork on which you can build up technological knowledge and skills, and through them you will be able to turn programming into your profession.

If you've never written a computer program, don't worry. There is always a first time. In this book we will **teach you how to program from scratch**. We do not expect any previous knowledge or abilities. All you need is some basic computer literacy and a desire to take up programming. The rest you will learn from the book.

If you can already write simple programs or if you have studied programming at school or in college, or you've coded with friends, **do not assume you know everything**! Read this book and you'll become aware of how many things you've missed. This book is indeed for beginners, but it **teaches concepts** and skills that even experienced professional programmers lack. Software companies are riddled with a shocking amount of self-taught amateurs who, despite having programmed on a salary for years, have no grasp of the fundamentals of programming and have no idea what a hash table is, how polymorphism works and how to work with bitwise operations. Don't be like them! Learn the **basics of programming first** and then the technologies. Otherwise you risk having your programming skills crippled, more or less, for years, if not for life.

If, on the other hand, you have programming experience, examine this book in details and see if you are familiar with all subjects we have covered, in order to decide whether it is for you or not. Take a close look especially at the chapters "Data Structures and Algorithms Complexity", "Object-Oriented Programming Principles", "Methodology of Problem Solving" and "High-Quality Programming Code". It is very likely that, even if you have several years of experience, you might not be able to work well with **data structures**; you might not be able to evaluate the **complexity of an algorithm**; you might not have mastered in depth the concepts of **object-oriented programming** (including UML and design patterns); and you might not be acquainted with the best practices for writing **high-quality programming code**. These are very important topics that are not covered in all books on programming, so don't skip them!

Previous Knowledge Is Not Required!

In this book **we do not expect any previous programming knowledge** from the readers. It is not necessary for you to have studied information technology or computer science, in order to read and comprehend the book content. The book **starts from scratch** and gradually gets you involved in programming. All technical terms you will come across will have been explained beforehand and it is not necessary for you to know them from other sources. If you don't know what a compiler, debugger, integrated development environment, variable, array, loop, console, string, data structure, algorithm, algorithm complexity, class or object are, don't be alarmed. From this book, you will learn all these terms and many more and gradually get accustomed to using them constantly in your everyday work. **Just read the book consistently and do the exercises.**

Certainly, if, after all, you do have prior knowledge in computer science and information technologies, they will by all means be of use to you. If, at university, you major in the field of computer science or if you study information technology at school, this will only help you, but it is not a must. If you major in tourism, law or other discipline that has little in common with computer technology, **you could still become a good programmer**, as long as you have the desire. The software industry is full of good developers without a computer science or related degree.

It is expected for you **to have basic computer literacy**, since we would not be explaining what a file, hard disk and network adapter is, nor how to move the mouse or how to write on a keyboard. We expect you to know how to work with a computer and how to use the Internet.

It is recommended that the readers have at least some basic knowledge of **English**. The entire documentation you will be using every day and almost all of the websites on programming you would be reading at all times are in English. In the profession of a programmer, **English is absolutely essential**. The sooner you learn it, the better. We hope that you already speak English; otherwise how do you read this text?

Make no illusion you can become a programmer without learning even a little English! This is simply a naive expectation. If you don't speak English, complete a course of some sort and then start reading technical literature, make note of any unfamiliar words and learn them. You will see for yourselves that Technical English is easy to learn and it doesn't take much time.

What Is the Scope of This Book?

This book covers the **fundamentals of programming**. It will teach you how to define and use variables, how to work with primitive data structures (such as numbers), how to organize logical statements, conditional statements and

loops, how to print on the console, how to use arrays, how to work with numeral systems, how to define and use methods, and how to create and use objects. Along with the **basic programming knowledge**, this book will help you understand more **complicated concepts** such as string processing, exception handling, using complex data structures (like trees and hash tables), working with text files, defining custom classes and working with LINQ queries. The concepts of object-oriented programming (OOP) – an established approach in modern software development – will be covered in depth. Finally, you'll be faced with the practices for **writing high-quality programs** and solving real-world programming problems. This book presents a complete methodology for solving programming problems, as well as algorithmic problems in general, and shows how to implement it with a few sample subjects and programming exams. This is something you will not find in any other book on programming!

What Will This Book Not Teach You?

This book will not award you **the profession "software engineer"**! This book won't teach you how to use the entire .NET platform, how to work with databases, how to create dynamic web sites and develop mobile applications, how to create window-based graphical user interface (GUI) and rich Internet applications (RIA). You won't learn how to develop complex software applications and systems like Skype, Firefox, MS Word or social networks like Facebook and retail sites like Amazon.com. And no other single book will. These kinds of projects require **many, many years of work** and experience and the knowledge in this book is just a wonderful beginning for the future programmer geek.

From this book, you won't learn software engineering, team work and you won't be able to prepare for working on real projects in a software company. In order to learn all of this, you will need a few more books and extra courses, but do not regret the time you will spend on this book. You are making the right choice by **starting with the fundamentals of programming** rather than directly with Web development, mobile applications and databases. This gives you the opportunity to **become a master programmer** who has in-depth knowledge of programming and technology. After you acquire the fundamentals of programming, it will become much easier for you to read and learn databases and web applications, and you will understand what you read much easier and in greater depth rather than if you directly begin learning SQL, ASP.NET, AJAX, XAML or WinRT.

Some of your colleagues directly begin programming with Web or mobile applications and databases without knowing what an array, a list or hash table is. Do not envy them! They have set out to do it the hard way, backwards. They will learn to make low-quality websites with PHP and MySQL, but they will find it **infinitely difficult to become real professionals**. You, too, will learn web technologies and databases, but before you take them up, **learn how to program!** This is much more important. Learning one

technology or another is very easy once you know the basics, when you can think algorithmically and you know how to tackle programming problems.

Starting to program with web applications or/and databases is just as incorrect as studying up a foreign language from some classical novel rather than from the alphabet and a textbook for beginners. It is not impossible, but if you lack the basics, it is much more difficult. It is highly-probable that you would end up lacking vital fundamental knowledge and being the laughing-stock of your colleagues/peers.

How Is the Information Presented?

Despite the large number of authors, co-authors and editors, we have done our best to make the style of the book similar in all chapters and highly comprehensible. The content is presented in a **well-structured** manner; it is broken up into many titles and subtitles, which make its reception easy and looking up information in the text quick.

The present book is **written by programmers for programmers**. The authors are active software developers, colleagues with genuine experience in both software development and training future programmers. Due to this, the quality of the content presentation is at a very good level, as you will see for yourself.

All authors are distinctly aware that the **sample source code** is one of the most important things in a book on programming. Due to this very reason, the text is accompanied with many, many examples, illustrations and figures.

When every chapter is written by a different author, there is no way to completely avoid differences in the style of speech and the quality of chapters. Some authors put a lot of work (for months) and a lot of efforts to **make their chapters perfect**. Others could not invest too much effort and that is why some chapters are not as good as the best ones. Last but not least, the **experience** of the authors varies – some have been programming professionally for 2-3 years, while others – for 15 years. This affects the quality, no doubt, but we assure you that **every chapter has been reviewed** and meets the quality standards of Svetlin Nakov and his team.

C# and .NET Framework

This book is about **programming**. It is intended to teach you to think as a programmer, to write code, to think in data structures and algorithms and to solve problems.

We use **C#** and **Microsoft .NET Framework** (the platform behind C#) only as means for writing programming code and we do not scrutinize the language's specifics. This same book can be found in versions for other languages like Java and C++, but the differences are not very significant.

Nevertheless, let's give a short account of C# (pronounced "see sharp").

C# is a modern programming language for development of software applications.

If the words "C#" and ".NET Framework" are unknown to you, you'll learn in details about them and their connection in the next chapter. Now let's explain briefly what C#, .NET, .NET Framework, CLR and the other technologies related to C# are.

The C# Programming Language

C# is a **modern object-oriented, general-purpose programming language**, created and developed by Microsoft together with the .NET platform. There is highly diverse software developed with C# and on the .NET platform: office applications, web applications, websites, desktop applications, mobile applications, games and many others.

C# is a **high-level language** that is similar to Java and C++ and, to some extent, languages like Delphi, VB.NET and C. All C# programs are object-oriented. They consist of a set of definitions in classes that contain methods and the methods contain the program logic – the instructions which the computer executes. You will find out more details on what a class, a method and C# programs are in the next chapter.

Nowadays C# is **one of the most popular programming languages**. It is used by millions of developers worldwide. Because C# is developed by Microsoft as part of their modern platform for development and execution of applications, the .NET Framework, the language is widely spread among Microsoft-oriented companies, organizations and individual developers. For better or for worse, as of this book writing, the **C# language** and the **.NET platform** are maintained and **managed entirely by Microsoft** and are not open to third parties. Because of this, all other large software corporations like IBM, Oracle and SAP base their solutions on the Java platform and use Java as their primary language for developing their own software products.

Unlike C# and the .NET Framework, the **Java language and platform are open-source** projects that an entire community of software companies, organizations and individual developers take part in. The standards, the specifications and all the new features in the world of Java are developed by workgroups formed out of the entire Java community, rather than a single company (as the case of C# and .NET Framework).

The C# language is distributed together with a special environment on which it is executed, called the **Common Language Runtime (CLR)**. This environment is part of the platform .NET Framework, which includes CLR, a bundle of standard libraries providing basic functionality, compilers, debuggers and other development tools. Thanks to the framework CLR programs are portable and, once written they can function with little or no changes on various hardware platforms and operating systems. C# programs

are most commonly run on MS Windows, but the .NET Framework and CLR also support mobile phones and other portable devices based on Windows Mobile, Windows Phone and Windows 8. C# programs can still be run under Linux, FreeBSD, iOS, Android, MacOS X and other operating systems through the free .NET Framework implementation **Mono**, which, however, is not officially supported by Microsoft.

The Microsoft .NET Framework

The C# language is not distributed as a standalone product – it is a part of the Microsoft .NET Framework platform (pronounced "Microsoft dot net framework"). **.NET Framework** generally consists of an environment for the development and execution of programs, written in C# or some other language, compatible with .NET (like VB.NET, Managed C++, J# or F#). It consists of:

- the .NET programming **languages** (C#, VB.NET and others);
- an environment for the execution of managed code (**CLR**), which executes C# programs in a controlled manner;
- a set of **development tools**, such as the **csc** compiler, which turns C# programs into intermediate code (called MSIL) that the CLR can understand;
- a set of **standard libraries**, like **ADO.NET**, which allow access to databases (such as MS SQL Server or MySQL) and **WCF** which connects applications through standard communication frameworks and protocols like HTTP, REST, JSON, SOAP and TCP sockets.

The .NET Framework is part of every modern Windows distribution and is available in different versions. The latest version can be downloaded and installed from Microsoft's website. As of this book's publishing, the latest version of the **.NET Framework is 4.5**. Windows Vista includes out-of-the-box .NET Framework 2.0, Windows 7 – .NET 3.5 and Windows 8 – .NET 4.5.

Why C#?

There are many reasons why we chose C# for our book. It is a modern programming language, widely spread, **used by millions of programmers** around the entire world. At the same time C# is a very simple and **easy to learn** (unlike C and C++). It is natural to start with a language that is suitable for beginners while still widely used in the industry by many large companies, making it one of the **most popular programming languages** nowadays.

C# or Java?

Although this can be extensively discussed, it is commonly acknowledged that **Java is the most serious competitor to C#**. We will not make a comparison between Java and C#, because C# is undisputedly the better,

more powerful, richer and just better engineered. But, for the purposes of this book, we have to emphasize that any modern programming language will be sufficient to learn programming and algorithms. We chose C#, because it is **easier to learn** and is distributed with highly convenient, free integrated development environment (e.g. Visual C# Express Edition). Those who prefer Java can prefer to use the Java version of this book, which can be found here: www.introprogramming.info.

Why Not PHP?

With regards to programing languages popularity, besides C# and Java, another widely used language is **PHP**. It is suitable for developing small web sites and web applications, but it gives rise to serious difficulties when implementing large and complicated software systems. In the software industry PHP is used first and foremost **for small projects**, because it can easily lead developers into writing code that is bad, disorganized and hard to maintain, making it inconvenient for more substantial projects. This subject is also debatable, but it is commonly accepted that, because of its antiquated concepts and origins it is built on and because of various evolutionary reasons, **PHP is a language that tends towards low-quality programming**, writing bad code and creating hard to maintain software. PHP is a procedural language in concept and although it supports the paradigms of modern object-oriented programming, most PHP programmers write procedurally. PHP is known as the language of "code monkeys" in the software engineering profession, because most PHP programmers write **terrifyingly low-quality code**. Because of the tendency to write low-quality, badly structured and badly organized programming code, the entire concept of the PHP language and platform is considered wrong and serious companies (like Microsoft, Google, SAP, Oracle and their partners) avoid it. Due to this reason, if you want to become a serious software engineer, start with C# or Java and **avoid PHP** (as much as possible).

Certainly, **PHP has its uses in the world of programming** (for example creating a blog with WordPress, a small web site with Joomla or Drupal, or a discussion board with PhpBB), but the entire PHP platform is **not well-organized** and engineered for large systems like .NET and Java. When it comes to non-web-based applications and large industrial projects, PHP is not by a long shot among the available options. Lots and lots of experience is necessary to use PHP correctly and to develop high-quality professional projects with it. PHP developers usually learn from tutorials, articles and low-quality books and pick up bad practices and habits, which then are hard to eradicate. Therefore, **do not learn PHP as your first development language**. **Start with C# or Java**.

Based on the large experience of the authors' collective we advise you to begin programming with C# and ignore languages such as C, C++ and PHP until the moment you have to use them.

Why Not C or C++?

Although this is also debatable, the **C and C++ languages are considered rather primitive, old and decaying**. They still have their uses and are suitable for low-level programming (e.g. programming for specialized hardware devices), but we do not advise you to pursue them.

You can program in pure C, if you have to write an operating system, a hardware device driver or if you want to program an embedded device, because of the lack of alternatives and the need to control the hardware very carefully. **The C language is morally old** and in no way do we advise you to begin programming with it. A programmer's productivity under pure C is many times lower compared to their productivity under modern general-purpose programming languages like C# and Java. A variant of C is used among Apple / iPhone developers, but not because it is a good language, but because there is no decent alternative. Most Apple-oriented programmers do not like Objective-C, but they have no choice in writing in something else.

C++ is acceptable when you have to program applications that require very **close work with the hardware** or that have special **performance requirements** (like 3D games). For all other purposes (like web applications development or business software) C++ is phenomenally inadequate. We do not advise you to pursue it, if you are starting with programming just now. The reason it is still being studied in some schools and universities is hereditary, because these institutions are very conservative. For example, the International Olympiad in Informatics (IOI) continues to promote C++ as the only language permitted to use at programming contests, although **C++ is rarely used in the industry**. If you don't believe this, look through some job search site and count the percentage of job advertisements with C++.

The C++ language lost its popularity mainly because of the inability to quickly write quality software with it. In order to write high-quality software in C++, you have to be an incredibly smart and experienced programmer, whereas the same is not strictly required for C# and Java. **Learning C++ takes much more time** and very few programmers know it really well. The productivity of C++ programmers is many times lower than C#'s and that is why C++ is losing ground. Because of all these reasons, **the C++ language is slowly fading away** and therefore we do not advise you to learn it.

Advantages of C#

C# is an **object-oriented** programming language. Such are all modern programming languages used for serious software systems (like Java and C++). The advantages of object-oriented programming are brought up in many passages throughout the book, but, for the moment, you can think of object-oriented languages as languages that allow working with objects from the real world (for example student, school, textbook, book and others). Objects have properties (e.g. name, color, etc.) and can perform actions (e.g. move, speak, etc.).

By starting to program with C# and the .NET Framework platform, you are on a **very perspective track**. If you open a website with job offers for programmers, you'll see for yourself that the demand for C# and .NET specialists is huge and is close to the demand for Java programmers. At the same time, the demand for PHP, C++ and other technology specialists is far lower than the demand for C# and Java engineers.

For the good programmer, the language they use is of no significant meaning, because they know **how to program**. Whatever language and technology they might need, they will master it quickly. Our goal is **not to teach you C#, but rather teach you programming!** After you master the fundamentals of programming and learn to think algorithmically, when you acquaint with other programming languages, you will see for yourself how much in common they have with C# and how easy it will be to learn them. Programming is built upon principles that change very slowly over the years and this book teaches you these very principles.

Examples Are Given in C# 5 and Visual Studio 2012

All examples in this book are with regard to **version 5.0** of the C# language and the **.NET Framework 4.5** platform, which is the latest as of this book's publishing. All examples on using the Visual Studio integrated development environment are with regard to version **2012** of the product, which were also the latest at the time of writing this book.

The Microsoft **Visual Studio 2012** integrated development environment (IDE) has a free version, suitable for beginner C# programmers, called Microsoft **Visual Studio Express 2012** for Windows Desktop. The difference between the free and the full version of Visual Studio (which is a commercial software product) lies in the availability of some functionalities, which we will not need in this book.

Although we use C# 5 and Visual Studio 2012, most examples in this book will work flawlessly under .NET Framework 2.0 / 3.5 / 4.0 and C# 2.0 / 3.5 / 4.0 and **can be compiled under Visual Studio 2005 / 2008 / 2010**.

It is of no great significance which version of C# and Visual Studio you'll use while you learn programming. What matters is that you learn **the principles of programming and algorithmic thinking**! The C# language, the .NET Framework platform and the Visual Studio integrated development environment are just tools and you can exchange them for others at any time. If you read this book and VS2012 is not currently the latest, be sure almost all of this book's content will still be the same due to backward compatibility.

How To Read This Book?

Reading this book has to be accompanied with **lots and lots of practice**. You won't learn programming, if you don't practice! It would be like trying to learn how to swim from a book without actually trying it. There is no other way!

The more you work on the problems after every chapter, the more you will learn from the book.

Everything you read here, you would have to try for yourself on a computer. Otherwise you won't learn anything. For example, once you read about Visual Studio and how to write your first simple program, you must by all means download and install Microsoft Visual Studio (or Visual C# Express) and try to write a program. Otherwise you won't learn! In theory, everything seems easy, but **programming means practice**. Remember this and try to solve the problems from this book. They are carefully selected – they are neither too hard to discourage you, nor too easy, so you'll be motivated to perceive solving them as a challenge. If you encounter difficulties, look for help at the **discussion group** for the "C# Programming Fundamentals" training course at Telerik Software Academy: http://forums.academy.telerik.com (the forum is intended for Bulgarian developers but the people "living" in it speak English and will answer your questions regarding this book, don't worry). Thousands students solve the exercises from this book every year so you will find many solutions to each problem from the book. We will also publish official solutions + tests for every exercise in the book at its web site.

Reading this book without practicing is meaningless! You must spend much more time on writing programs than reading the text itself. It is just like learning to drive: no one can learn driving by reading books. To learn driving, you need to drive many times in different situations, roads, cars, etc. To learn programming, you need to program!

Everybody has studied math in school and knows that learning how to solve math problems requires lots of practice. No matter how much they watch and listen to their teachers, **without actually sitting down and solving problems, they won't learn**. The same goes for programming. You need lots of practice. You need to write a lot, to solve problems, to experiment, to endeavor in and to struggle with problems, to make mistakes and correct them, to try and fail, to try anew and experience the moments when things finally work out. You need lots and lots of practice. This is the only way you will make progress.

So people say that to become a developer you might need to write at least 50,000 – 100,000 lines of code, but the correct number can vary a lot. Some people are fast learners or just have problem-solving experience. Others may need more practice, but in all cases **practicing programming is very important**! You need to solve problems and to write code to become a developer. There is no other way!

Do Not Skip the Exercises!

At the end of each chapter there is a considerable list of **exercises**. **Do not skip them!** Without exercises, you will not learn a thing. After you read a chapter, you should sit in front of the computer and **play with the examples**

you have seen in the book. Then you should set about solving all problems. If you cannot solve them all, you should at least try. If you don't have all the time necessary, you must at least attempt solving the first few problems from each chapter. Do not carry on without **solving problems after every chapter**, it would just be meaningless! The problems are small feasible situations where you apply the stuff you have read. In practice, once you have become programmers, you would solve similar problems every day, but on a larger and more complex scale.

You must at all cost strive to solve the exercise problems after every chapter from the book! Otherwise you risk not learning anything and simply wasting your time.

How Much Time Will We Need for This Book?

Mastering the fundamentals of programming is a crucial task and **takes a lot of time**. Even if you're incredibly good at it, there is no way that you will learn programming on a good level for a week or two. To learn any human skill, you need to read, see or be shown how it is done and then try doing it yourselves and practice a lot. The same goes for programming – you must either read, see or listen how it is done, then **try doing it yourself**. Then you would succeed or you would not and you would try again, until you finally realize you have learned it. Learning is done step by step, consecutively, in series, with a lot of effort and consistency.

If you want to read, understand, learn and acquire thoroughly and in-depth the subject matter in this book, you have to invest **at least 2 months for daylong activity** or at least 4-5 months, if you read and exercise a little every day. This is the minimum amount of time it would take you to be able to grasp in depth the fundamentals of programming.

The necessity of such an amount of lessons is confirmed by the free trainings at Telerik Software Academy (http://academy.telerik.com), which follow this very book. The hundreds of students, who have participated in trainings based on the lectures from this book, usually learn all subjects from this book within **3-4 months of full-time work**. Thousands of students every year solve all exercise problems from this book and successfully sit on programming exams covering the book's content. Statistics shows that anyone without prior exposure to programming, who has spent less than the equivalent of 3-4 months daylong activity on this book and the corresponding courses at Telerik Academy, fails the exams.

The main subject matter in the book is presented in more than **1100 pages**, which will take you a month (daylong) just to read them carefully and test the sample programs. Of course, you have to spend enough time on the exercises (few more months); without them you would hardly learn programming.

Exercises: Complex or Easy?

The exercises in the book consist of about **350 problems** with varying difficulty. For some of them you will need a few minutes, for others several hours (if you can solve them at all without help). This means you would need a month or two of daylong exercising or several months, if you do it little by little.

The exercises at each chapter are ordered in **increasing level of difficulty**. The first few exercises are easy, similar to the examples in the chapter. The last few exercises are usually complex. You might need to use external resources (like information from Wikipedia) to solve them. Intentionally, the last few exercises in each chapter require **skills outside of the chapter**. We want to push you to perform a search in your favorite search engine. You need to **learn searching on the Internet!** This is an essential skill for any programmer. You need to learn how to learn. Programming is about learning every day. Technologies constantly change and you can't know everything. To be a programmer means to **learn new APIs, frameworks, technologies and tools every day**. This cannot be avoided, just prepare yourself. You will find many problems in the exercises, which require searching on the Internet. Sometimes you will need the skills from the next chapter, sometimes some well-known algorithm, sometimes something else, but in all cases **searching on the Internet is an essential skill** you need to acquire.

Solving the exercises in the book takes a few **months**, really. If you don't have that much time at your disposal, ask yourselves if you really want to pursue programming. This is a very serious initiative in which you must invest a really great deal of efforts. If you really want to learn programming on a good level, **schedule enough time** and follow the book or the video lectures based on it.

Why Are Data Structures and Algorithms Emphasized?

This book teaches you, in addition to the basic knowledge in programming, proper **algorithmic thinking** and using basic **data structures** in programming. Data structures and algorithms are a programmer's most important fundamental skills! If you have a good grasp of them, you will not have any trouble becoming proficient in any software technology, development tool, framework or API. That is what the most serious software companies rely on when hiring employees. Proof of this are job interviews at large companies like Google and Microsoft that rely exclusively on **algorithmic thinking** and knowledge of all basic **data structures and algorithms**.

The information below comes from **Svetlin Nakov**, the leading author of this book, who passed software engineering interviews at Microsoft and Google in 2007-2008 and shares his own experience.

Chapter 1. Introduction to Programming

In This Chapter

In this chapter we will take a look at the **basic programming terminology** and we will **write our first C# program**. We will familiarize ourselves with programming – what it means and its connection to computers and programming languages.

Briefly, we will review the different **stages of software development**.

We will introduce the C# language, the .NET platform and the different Microsoft technologies used in software development. We will examine what tools we need **to program in C#**. We will use the C# language to write our first computer program, compile and run it from the command line as well as from Microsoft **Visual Studio** integrated development environment. We will review the MSDN Library – the documentation of the .NET Framework. It will help us with our exploration of the features of the platform and the language.

What Does It Mean "To Program"?

Nowadays computers have become irreplaceable. We use them to solve complex problems at the workplace, look for driving directions, have fun and communicate. They have countless applications in the business world, the entertainment industry, telecommunications and finance. It's not an over-statement to say that computers build the neural system of our contemporary society and it is difficult to imagine its existence without them.

Despite the fact that computers are so wide-spread, **few people know how they really work**. In reality, it is not the computers, but the programs (the software), which run on them, that matter. It is the **software** that makes computers valuable to the end-user, allowing for many different types of services that change our lives.

How Do Computers Process Information?

In order to understand what it means to program, we can roughly compare a computer and its operating system to a large factory with all its workshops, warehouses and transportation. This rough comparison makes it easier to imagine the level of complexity present in a contemporary computer. There are many processes running on a computer, and they represent the workshops and production lines in a **factory**. The hard drive, along with the

files on it, and the operating memory (RAM) represent the warehouses, and the different protocols are the transportation systems, which provide the input and output of information.

The different types of products made in a factory come from different workshops. They use raw materials from the warehouses and store the completed goods back in them. The raw materials are transported to the warehouses by the suppliers and the completed product is transported from the warehouses to the outlets. To accomplish this, different types of transportation are used. Raw materials enter the factory, go through different stages of processing and leave the factory transformed into products. Each factory converts the raw materials into a product ready for consumption.

The computer is a machine for information processing. Unlike the factory in our comparison, for the computer, the raw material and the product are the same thing – information. In most cases, the input information is taken from any of the warehouses (files or RAM), to where it has been previously transported. Afterwards, it is processed by one or more processes and it comes out modified as a new product. Web based applications serve as a prime example. They use HTTP to transfer raw materials and products, and information processing usually has to do with extracting content from a database and preparing it for visualization in the form of HTML.

Managing the Computer

The whole process of manufacturing products in a factory has many levels of management. The separate machines and assembly lines have operators, the workshops have managers and the factory as a whole is run by general executives. Every one of them controls processes on a different level. The machine operators serve on the lowest level – they control the machines with buttons and levers. The next level is reserved for the workshop managers. And on the highest level, the general executives manage the different aspects of the manufacturing processes in the factory. They do that by issuing orders.

It is the same with computers and software – they have many levels of management and control. The lowest level is managed by the **processor** and its registries (this is accomplished by using machine programs at a low level) – we can compare it to controlling the machines in the workshops. The different responsibilities of the **operating system** (Windows 7 for example), like the file system, peripheral devices, users and communication protocols, are controlled at a higher level – we can compare it to the management of the different workshops and departments in the factory. At the highest level, we can find the **application software**. It runs a whole ensemble of processes, which require a huge amount of processor operations. This is the level of the general executives, who run the whole factory in order to maximize the utilization of the resources and to produce quality results.

The Essence of Programming

The essence of programming is to control the work of the computer on all levels. This is done with the help of "orders" and "commands" from the programmer, also known as programming instructions. To "program" means **to organize the work of the computer through sequences of instructions**. These commands (instructions) are given in written form and are implicitly followed by the computer (respectively by the operating system, the CPU and the peripheral devices).

To "program" means writing sequences of instructions in order to organize the work of the computer to perform something. These sequences of instructions are called "computer programs" or "scripts".

A sequence of steps to achieve, complete some work or obtain some result is called an **algorithm**. This is how **programming is related to algorithms**. Programming involves describing what you want the computer to do by a sequence of steps, by **algorithms**.

Programmers are the people who create these instructions, which control computers. These instructions are called **programs**. Numerous programs exist, and they are created using different kinds of **programming languages**. Each language is oriented towards controlling the computer on a different level. There are languages oriented towards the machine level (the lowest) – **Assembler** for example. Others are most useful at the system level (interacting with the operating system), like **C**. There are also high level languages used to create application programs. Such languages include **C#**, Java, C++, PHP, Visual Basic, Python, Ruby, Perl, JavaScript and others.

In this book we will take a look at **the C# programming language** – a modern high level language. When a programmer uses C#, he gives commands in high level, like from the position of a general executive in a factory. The **instructions** given in the form of programs written in C# can access and control almost all computer resources directly or via the operating system. Before we learn how to write simple C# programs, let's take a good look at the different stages of software development, because programming, despite being the most important stage, is not the only one.

Stages in Software Development

Writing software can be a very complex and time-consuming task, involving a whole team of software engineers and other specialists. As a result, many methods and practices, which make the life of programmers easier, have emerged. All they have in common is that the development of each software product goes through several different **stages**:

- Gathering the **requirements** for the product and creating a task;
- **Planning** and preparing the architecture and design;

- **Implementation** (includes the writing of program code);
- Product trials (**testing**);
- **Deployment** and exploitation;
- **Support**.

Implementation, testing, deployment and support are mostly accomplished using programming.

Gathering the Requirements

In the beginning, only the idea for a certain product exists. It includes a list of **requirements**, which define actions by the user and the computer. In the general case, these actions make already existing activities easier – calculating salaries, calculating ballistic trajectories or searching for the shortest route on Google maps are some examples. In many cases the software implements a previously nonexistent functionality such as automation of a certain activity.

The **requirements** for the product are usually defined in the form of documentation, written in English or any other language. There is no programming done at this stage. The requirements are defined by experts, who are familiar with the problems in a certain field. They can also write them up in such a way that they are easy to understand by the programmers. In the general case, these experts are not programming specialists, and they are called **business analysts**.

Planning and Preparing the Architecture and Design

After all the requirements have been gathered comes **the planning stage**. At this stage, a technical plan for the implementation of the project is created, describing the platforms, technologies and the initial architecture (design) of the program. This step includes a fair amount of creative work, which is done by software engineers with a lot of experience. They are sometimes called **software architects**. According to the requirements, the following parts are chosen:

- The **type of the application** – for example console application, desktop application (GUI, Graphical User Interface application), client-server application, Web application, Rich Internet Application (RIA), mobile application, peer-to-peer application or other;
- The **architecture** of the software – for example single layer, double layer, triple layer, multi-layer or SOA architecture;
- The **programming language** most suitable for the implementation – for example C#, Java, PHP, Python, Ruby, JavaScript or C++, or a combination of different languages;
- The **technologies** that will be used: platform (Microsoft .NET, Java EE, LAMP or another), database server (Oracle, SQL Server, MySQL, NoSQL

database or another), technologies for the user interface (Flash, JavaServer Faces, Eclipse RCP, ASP.NET, Windows Forms, Silverlight, WPF or another), technologies for data access (for example Hibernate, JPA or ADO.NET Entity Framework), reporting technologies (SQL Server Reporting Services, Jasper Reports or another) and many other combinations of technologies that will be used for the implementation of the various parts of the software system.

- The **development frameworks** that will simplify the development, e.g. ASP.NET MVC (for .NET), Knockout.js (for JavaScript), Rails (for Ruby), Django (for Python) and many others.
- The number and skills of the **people** who will be part of the development team (big and serious projects are done by large and experienced teams of developers);
- The **development plan** – separating the functionality in stages, resources and deadlines for each stage.
- Others (size of the team, locality of the team, methods of communication etc.).

Although there are many rules facilitating the correct analysis and planning, a fair amount of intuition and insight is required at this stage. This step predetermines the further advancement of the development process. There is no programming done at this stage, only preparation.

Implementation

The stage, most closely connected with programming, is the implementation stage. At this phase, the program (application) is implemented (written) according to the given task, design and architecture. **Programmers** participate by **writing the program** (source) code. The other stages can either be short or completely skipped when creating a small project, but the implementation always presents; otherwise the process is not software development. This book is dedicated mainly to describing the skills used during implementation – creating a **programmer's mindset** and building the knowledge to use all the resources provided by the C# language and the .NET platform, in order to create software applications.

Product Testing

Product testing is a very important stage of software development. Its purpose is to make sure that all the requirements are strictly followed and covered. This process can be implemented manually, but the preferred way to do it is by **automated tests**. These tests are small programs, which automate the trials as much as possible. There are parts of the functionality that are very hard to automate, which is why product trials include automated as well as manual procedures to ensure the quality of the code.

The testing (trials) process is implemented by **quality assurance engineers (QAs)**. They work closely with the programmers to find and correct errors (bugs) in the software. At this stage, it is a priority to find defects in the code and almost no new code is written.

Many **defects** and **errors** are usually found during the testing stage and the program is sent back to the implantation stage. These two stages are very closely tied and it is common for a software product to switch between them many times before it covers all the requirements and is ready for the deployment and usage stages.

Deployment and Operation

Deployment is the process which **puts a given software product into exploitation**. If the product is complex and serves many people, this process can be the slowest and most expensive one. For smaller programs this is a relatively quick and painless process. In the most common case, a special program, called installer, is developed. It ensures the quick and easy installation of the product. If the product is to be deployed at a large corporation with tens of thousands of copies, additional supporting software is developed just for the deployment. After the **deployment** is successfully completed, the product is ready for **operation**. The next step is to train employees to use it.

An example would be the deployment of a new version of Microsoft Windows in the state administration. This includes **installation** and **configuration** of the software as well as **training** employees how to use it.

The deployment is usually done by the team who has worked on the software or by trained **deployment specialists**. They can be system administrators, database administrators (DBA), system engineers, specialized consultants and others. At this stage, almost no new code is written but the existing code is tweaked and configured until it covers all the specific requirements for a successful deployment.

Technical Support

During the exploitation process, it is inevitable that **problems will appear**. They may be caused by many factors – errors in the software, incorrect usage or faulty configuration, but most problems occur when the users change their requirements. As a result of these problems, the software loses its abilities to solve the business task it was created for. This requires additional involvement by the developers and the **support experts**. The support process usually continues throughout the whole life-cycle of the software product, regardless of how good it is.

The support is carried out by the development team and by specially trained **support experts**. Depending on the changes made, many different people may be involved in the process – business analysts, architects, programmers, QA engineers, administrators and others.

For example, if we take a look at a software program that calculates salaries, it will need to be **updated** every time the tax legislation, which concerns the serviced accounting process, is changed. The support team's intervention will be needed if, for example, the hardware of the end user is changed because the software will have to be installed and configured again.

Documentation

The documentation stage is not a separate stage but accompanies all the other stages. **Documentation** is an important part of software development and aims to pass knowledge between the different participants in the development and support of a software product. Information is passed along between different stages as well as within a single stage. The **development documentation** is usually created by the developers (architects, programmers, QA engineers and others) and represents a combination of documents.

Software Development Is More than Just Coding

As we saw, software development is much more than just coding (writing code), and it includes a number of other processes such as: requirements analysis, design, planning, testing and support, which require a wide variety of specialists called **software engineers**. Programming is just a small, but very essential part of software development.

In this book we will focus solely on programming, because it is the only process, of the above, without which, we cannot develop software.

Our First C# Program

Before we continue with an in depth description of the C# language and the .NET platform, let's take a look at a simple example, illustrating how a **program written in C#** looks like:

```
class HelloCSharp
{
   static void Main(string[] args)
   {
      System.Console.WriteLine("Hello C#!");
   }
}
```

The only thing this program does is to **print the message "Hello, C#!"** on the default output. It is still early to execute it, which is why we will only take a look at its structure. Later we will describe in full how to compile and run a given program from the command prompt as well as from a development environment.