Some reasons to choose this book

In their first form of existence, computer programs are textual descriptions written according to the rules of a programming language. This book explains how computer programs are written using the C# programming language. Pronounced "see sharp", C# is a new language that is gaining popularity these days. The main purpose of this book is, however, to teach computer programming in general terms. After you have studied with this book, you'll know how computers operate; you'll be familiar with the fundamentals of programming and the essentials of object-oriented programming; and you'll be prepared to study other books related to computing and programming languages.

This book is written for people who are beginners in the field of computer programming, but experienced people may find it beneficial as well. This book does not assume any previous knowledge about computer programming. Logical operating principles of computers are explained before the actual studies of programming begin. In addition, this book promotes a programming style which prevents programming errors. All the examples of computer programs are written so that the reader encounters a lot of natural-language expressions instead of the traditional abbreviations of the computer world. This approach aims to make learning easier.

The pages of this book are designed to maximize readability and understandability. Examples of computer programs are presented in easy-to-read graphical descriptions. Pages are designed so that the reader does not need to turn them unnecessarily.

I have tried to write interesting examples of computer programs for this book. Actually most of the programs have already been published as programs written with the C++ programming language in my earlier book entitled *A Natural Introduction to Computer Programming with C++*. Now the programs are available as C# programs. If you choose to study with this book, you have an advantage regarding your future. If you some day, after having studied with this book, need to study the C++ programming language, that should be easy for you because there is a book which explains how familiar programs are written with C++.

In order to learn computer programming you must do programming exercises. Plenty of programming exercises are provided in the chapters of this book. Solutions to most of the exercises are available among the electronic material that can be downloaded via www.naturalprogramming.com.

If you want to know why I ended up writing these kinds of programming books, please read the Epilogue and the Preface of my C++ book. They are freely available at www.naturalprogramming.com.

Notes for teachers and other experienced programmers

This book differs from other programming books in that all its computer programs are written with so-called natural names. This means that all names (identifiers) of variables, constants, arrays, methods, classes, etc. consist of several natural words. The names are written without abbreviations, and they thus look like the following

```
character_index  integer_index  character_from_keyboard
given_value  number_of_characters_read
string_to_search  replacement_string
```

Learning should be easier because the programs are written with readable names like those above.

I have been teaching subjects related to computer programming for more than 10 years. My course materials have always contained only programs with natural names, and I have encouraged students to use this kind of naming style in their programming work. My experience is that those students who use only natural names in their programs tend to produce better programs. I have also experienced personally that inventing informative

natural names is a way to think during the process of program writing, and when you think clearly, you produce better programs and make less programming errors. For these reasons, I want to recommend the naming style that is used in this book. I know that there are experienced programmers who do not want to give up using abbreviations in programs, and people can have very strong opinions related to naming and other programming style issues. I would, however, like to urge people to try programming without abbreviations. Such programming style changes the physical appearance of programs and makes them slightly longer, but once you get accustomed to such programs you may experience something new in the process of program creation.

The names in the example programs of this book are such that underscore characters (_) are used to separate (or to join) the words of a name. This kind of naming style is, in fact, not in complete harmony with the naming guidelines that are provided in C# Language Specification (ECMA Standard No. 334, available at www.ecma.ch). The naming guidelines of the C# Language Specification recommend that names must be capitalized, i.e., the words of a name must be written so that an uppercase (capital) letter begins each word in a name. According to this naming style, names should look like

```
nameToBeKnown     integerFromKeyboard
FindSmallestNumberInArray     keyboardInputIsNumerical
```

while in this book these names are written in the following way

```
name_to_be_known    integer_from_keyboard
find_smallest_number_in_array    keyboard_input_is_numerical
```

I decided not to follow the naming recommendations of the C# Language Specification for the following two reasons

- The underscore character makes space between the words of a name, and as we are accustomed to the fact that there is generally space between the words of a text, the names that are formed with underscores can be considered more natural and readable than capitalized names. (ConsiderWhatThisBookWouldLookLikeIfItsBodyTextWouldBeWrittenThisWay.)
- The use of the underscore character helps to identify different types of names in this book. Because the names of the standard C# classes and methods are capitalized, they can be easily recognized as standard C# names when most other names contain underscore characters. For example, when the reader of this book finds method names like **`ReadLine`**, **`LastIndexOf`**, and **`Reverse`**, he or she knows that they are standard C# names because they are capitalized. On the other hand, names like **`given_file_name`**, **`line_number`**, and **`text_line_from_file`** are recognized as names invented by this author because they contain underscores.

This may sound like if I were a member of some secret underscoring society, but I am not the only person who is against abandoning underscores in programming. The underscore character serves an important purpose in this book, and it could be exploited in a similar way in other contexts.

Some advice for studying

Although I have tried my best while writing this book, and I believe this is a very good textbook for a person who is starting to learn computer programming, it is still a fact that computer programming can be a difficult subject to begin with. When you read about something new in this book, you may not understand it immediately. At the start it might be possible that things just won't begin to become clear. That has happened to me, and still I became quite an expert in computer programming. So my first suggestion is

Do not worry if you do not understand something immediately.

It often happens that some things which you're studying may seem difficult in the beginning, and are only understood later, perhaps the following day when you have re-read the text which had been so difficult. Therefore, it is of prime importance to learn to accept that there are difficult things which cannot be understood quickly.

As this book is written so that it should be read sequentially from the beginning towards the end, I advise you to do so. If you encounter something that is already familiar to you, it is up to you if you want to skip some pages. But I would like to remind you, that it may deepen your knowledge, and in any case it does not harm you at all, if you also read the pages that seem to contain something familiar.

While you are reading have a pencil or pen at hand, and do not hesitate to

- underline or otherwise mark those sentences which you think are important;
- write your own thoughts in the margins and empty spaces of the pages;
- add your own explanation texts near the example programs; and
- mark, for example with a question mark (?), those parts of the text which you did not understand, and which you need to read again later.

Because you learn also by doing, it is important that you write about the things you are studying. Sometimes it may even be helpful to write down the things you do not understand.You can write down on the pages of this book (provided that it is your own book), and you should also have a separate notebook for writing. In addition to writing, it is important that you discuss matters of computer programming orally with your friends and fellow students. Sometimes it may help you to understand a computer program when you try to explain it to another person.

You just can't learn computer programming without writing programs by yourself. There are plenty of programming exercises in parts II and III of this book. You should do at least part of those exercises with a computer. One possible approach for doing the exercises is that you first read the text of a section and then do the exercises in that section. You should also try to invent your own computer programs. It's nice to work with programs which nobody else has. In Part I of this book there are less exercises, but you should read that part before starting programming in the subsequent parts.

When you try programming exercises, you must use the keyboard of your computer. And in the future, you will probably use a computer and its keyboard daily in your work. To write effectively with a computer keyboard, and to prevent writing mistakes, I recommend that you learn to type with all 10 fingers. Computer keyboards are designed so that they can be used effectively with 10 fingers. The skill of typing with all 10 fingers can be learned through daily half-hour exercises over a few weeks. The time you spend learning this skill will be paid back hundreds of times in the future. There are special computer programs with which you can learn the 10-finger typing system. If you can't find a suitable program, you can borrow an old typing manual from a public library.

While you study this book and do exercises of computer programming, you will notice that you need a lot of information to write computer programs, and you cannot simply remember all the necessary information. The needed information may be in this book but you do not know on which page the information is located. To find information, you

need to learn to use the index that is at the end of this book. The index lists keywords and phrases which are used on certain pages. It is normal that you may not find a certain piece of information by checking the page of your first search word. In such situations you just need to be patient. The index is a useful tool to search for information, but you must use ingenuity to invent several search words which can be looked up in the index. This advice applies to the indexes in other books as well.

Some pieces of information are regularly needed when computer programs are being written. For example, you may often have to check things in Appendix A. One possible way to make it easier to find frequently-used data is to take copies of the important pages and hang the copies on the wall near the place where you study or use your computer. Another possibility is to make the frequently-used pages easy to find by attaching pieces of tape or paper clips to them. With these kinds of little arrangements it is possible to make learning and computer programming somewhat easier for you.

The purpose of this book is to be your first book about computer programming. In the beginning, this book should be enough. However, as you proceed towards the end of this book, you shall become more and more experienced in computer programming, and you may not consider yourself a beginner any more. You may want to write longer computer programs or try other fields of computer programming. When you become more experienced, I recommend that you also acquire other programming books. I have personally found out that what cannot be found in one book, may be found in another book. After studying this book, you should be able to read books that are written for more experienced programmers. Especially if you start working as a computer programmer, you or your employer should not hesitate to invest in computer programming books. An appropriate book may easily save many working hours, and the price of the book is usually not much more than the price of a single working hour.

It might be a good idea to re-read this introductory section after you have done some computer programming.

The structure of this book

This book is a textbook, a book for studying purposes. This is very much like other textbooks. The text and other elements of the book are organized in chapters. Then there are some things that do not belong to chapters. The overall structure of this book is the following

- introductory pages (this page belongs to these)
- chapters (there are 16 of them)
- appendices (these are identified with letters A, B, ...)
- index (this is very important to find information contained in the book)
- some useful tables for computer programmers

When you are learning computer programming, it is important that you learn to understand various structures of a text. This book has a certain structure which is described here. Like this book, computer programs are written according to certain structural rules.

Most of the text of this book is found in the chapters. Each chapter has a number. Chapters consist of sections which are numbered 1.1, 1.2, ... 2.1, 2.2, ... 3.1, 3.2, ... etc. Each section in this book consists of

- body text which is present in every section
- zero or more figures which are referred to in the body text by, for example, "Figure 3-2 shows how ..."
- zero or more tables which are referred to in the body text by, for example, "... is summarized in Table 6-3."
- zero or more program descriptions which are referred to with a file name in the body text by, for example, "Program **Weddingdates.cs** is an example ..."
- zero or more information boxes which give interesting information related to the subject discussed in the section
- zero or more exercise boxes which present exercises related to the subject of the section.

Figures and tables are numbered so that the chapter number is mentioned. For example, when Figure 10-4 is discussed, you know that it is the fourth figure in Chapter 10 and there are three preceding figures in that chapter. Figures and tables are usually numbered this way in textbooks.

Program descriptions are a unique feature of this book. They form a systematic way to present computer programs, and to show how programs behave when they are executed on a computer. The structure of program descriptions is explained in the following introductory section.

Example of an information box

In some sections you can find boxes like this one. There are two types of boxes:

- An information box usually presents information that is related to the subject discussed in the section, but the contents of an information box are usually not so important as the matters discussed in the body text of the section.
- An exercise box contains exercises which you should do after reading the text of the section. After the table of contents of this book, you can find a list which helps you to find those pages which contain exercise boxes.

The structure of program descriptions

Computer programs are presented in a unique and exact manner in this book. These presentations are called program descriptions. The program descriptions resemble figures but they are not figures because their captions are different, and they do not follow the numbering system of figures. The first program descriptions are included in Chapter 2, and they are used throughout parts II and III of the book. The program descriptions may look a little bit strange at first, but after you become familiar with them, you'll find out that they are a very informative way to describe computer programs and their operation. This introductory section explains the notations used in program descriptions. You might read this section after you have studied some source programs.

The following is the simplest form of a program description that describes an executable program:

Program descriptions contain text "balloons" in which the structure and behavior of the program in question is explained. You are now reading a text balloon. The arrow points to that part of the program which is being explained by the balloon. The arrow of this balloon points to the beginning of the actual program text. It is important to note that the balloons are not part of the program, they just explain it.

There can be several balloons with varying sizes in a single source program description, and several arrows can originate from a single balloon. The arrow of this balloon points to the last line in the program text. The balloons make the program descriptions look a little bit like cartoons. That should make this book fun to read.

```
//  Filename.cs

using System ;

class  Filename
{
   static void Main()
   {
      // The actual program is always written by using
      // this Courier font. In the Courier font all
      // characters have the same physical width.

      // This program is an example which does nothing.
   }
}
```

Filename.cs - 1. Caption which describes the source program.

```
What appears on the screen of the computer when the program is
executed is shown in this kind of box.
```

Filename.cs - X. Caption which says something about the execution of the program.

A simple program description consists of two separate sub-descriptions which both have their own captions, i.e., some text below the descriptions:

- The source description presents the source program text of the program. The caption of the source description begins **Filename.cs - 1. ...**
- The execution description shows what happens on the screen of a computer when the program is executed. The caption of the execution description is identified by the letter X and it begins **Filename.cs - X. ...**

The key idea here is that both a program, and its program description, are identified with a file name. You can use the same file name that is used in a caption of a program description to search for the program among the electronic material associated with this book. For example, because the captions of the first program description on the first pages of Chapter 2 begin **First.cs - ...**, you know that you can find the actual program in a file named **First.cs**.

When program descriptions are referred to in the body text of this book, they are identified using the file name. For example, when I write "Program **First.cs** is the first example of a C# program", I refer to all descriptions which have captions which begin **First.cs - ...**

Many examples of computer programs in this book are so long that their text does not fit into a single source description. In these cases several source descriptions are needed to present the program. For example, a program description in Chapter 15 consists of four sub-descriptions with the captions

Events.cs - 1: The declaration of ...

Events.cs - 2: Using an ArrayList array ...

Events.cs - 3. Sorting and combining ...

Events.cs - X. Three lists printed ...

When there is a colon (:) written in the caption of a source description, it means that the program text continues in the following source description. In the final source description there is a full stop (.) after the number.

In some cases, it is necessary to explain just part of a program in a separate source description. In these situations, the plus sign (+) is used in the caption of a source description. For example, the first program in Chapter 6, **Largeint.cs**, has a caption that begins

Largeint.cs - 1.+ A program to find ...

The full stop (.) means that this is the final source description of the program, but the plus sign (+) means that some part of the text shown in this description is presented in more detail in another source description. The caption of the detailed description is equipped with a new level number and it begins

Largeint.cs - 1-1. The if constructs that ...

The captions used in program descriptions form a kind of language where numbers, the colon, the full stop, the plus sign, and the letter X convey certain meanings. When you see a caption like

Somefile.cs - 2:+ ...

you should be able to deduce that the entire program is presented in at least three separate source descriptions, and one or more parts of this second source description are explained in descriptions that have captions like

Somefile.cs - 2 -1 ...

Somefile.cs - 2 - 2 ...