

Contents

<i>List of Figures</i>	vii
<i>List of Tables</i>	viii
<i>About the Authors</i>	ix

PART ONE GETTING STARTED

1 Who is this Book for?	3
2 What is Programming?	7
3 Everyone Makes Mistakes	20

PART TWO YOUR FIRST PROGRAM

4 Before You Start	35
5 Writing Your First Program	48
6 The Nature of Errors	63

PART THREE YOUR NEXT PROGRAMS

7 The Many Ways of Programming	77
8 Writing Bigger Programs	89
9 Becoming a Detective	104

PART FOUR IT ALL GETS INTERESTING

10 Grappling with Design	123
11 Writing Your <i>n</i> th Program	133
12 Juggling All the Pieces	149

PART FIVE OTHER LANGUAGES

13	Why Do We Have Different Programming Languages?	159
14	Exploiting Your Programming Skills	167
15	Taking Programming Further	178

PART SIX HERE BE DRAGONS!

16	How Languages Differ	199
	<i>Notes</i>	227
	<i>Glossary</i>	229
	<i>Index</i>	256

Part One

Getting Started

1 Who is this Book for?

► Getting acquainted

Since you have opened this book we guess you have some interest in computer programming, even if it's just curiosity.

As authors, we cannot be sure exactly who you are but we might guess that you are a student either taking a programming course or thinking about taking one? Alternatively, you might be the parent of such a student, or maybe you teach programming? Still not right? Then perhaps you are just someone who is interested in finding out what programming is all about. Whichever of those groups you belong to, you are one of the people we are primarily writing for: students in the final two years of school, in Further Education colleges or the first year of university; and those who support such students. But we hope this book might also be useful to others outside those boundaries.

This book is about how to go about learning programming effectively. We should stress at this early stage that it is not a tutorial for a particular **programming language**. Instead, it focuses on some of the skills, attitudes and techniques required to study programming successfully. We won't assume that you know anything about programming already, but we'll begin by telling you what programming is and how to get started. We'll also explain how studying programming may be different from studying other subjects, what pitfalls to avoid, and much more.

The naming of programming languages

You'll find boxed sections like this throughout the book. We use them to give short additional explanations of things like new words and ideas.

If you are not already familiar with any programming languages then you will soon learn that there is a bewildering number of them, most with strange names – Algol, APL, **BASIC**, **C**, **Fortran**, **Java**, **Scheme** – to name just a handful.

Do the different names mean anything? Some do and some don't! For instance, Algol stands for Algorithmic Language and Fortran stands for Formula Translator. On the other hand, Java gets its name from the coffee drunk by its designers!

Throughout this book we will make use of lots of different languages. Remember that we aren't trying to teach you any particular language, but just to give you a feel for the ways in which programming languages support the task of programming. If you want to learn a specific programming language – as part of one of your courses, say – then you will find that there are plenty of textbooks available that will give you an in-depth guide to all the commonly available languages.

Who are we then, and why did we write this book? We are all computer scientists and teachers of programming. We enjoy programming and enjoy introducing it to others. When we were sitting around discussing our teaching, we discovered that even though we were teaching different programming topics, there were ideas and techniques that we all discussed with our students. These were common to all the programming courses but were different from what was needed in other subjects. When we looked around for a book that covered the material we were surprised not to find one; and so we set about writing this.

► **Picking your route**

Many books are designed to be read straight through, from page 1 of Chapter 1 to the end. You would not normally read a novel any other way, for instance, unless you are one of those who peek at the last page early! But, of course, books are rarely written in a linear fashion, and most novels have multiple themes that run through them, with characters or plots that come and go to provide variation of pacing and interest.

We hope you will find that there is a variety of ways you could read this book. If you wish, you can read each chapter in turn, from Chapter 1 through to Chapter 16. On the other hand, you might like to take the chapters in a pick-and-mix fashion – following a route that fits in with your existing knowledge and your individual approach to study.

In our minds there are some recurrent themes that run through this book – rather like the warp and weft fibres that run through a fabric. You can think of the book as being structured into five parts, each of which consists of

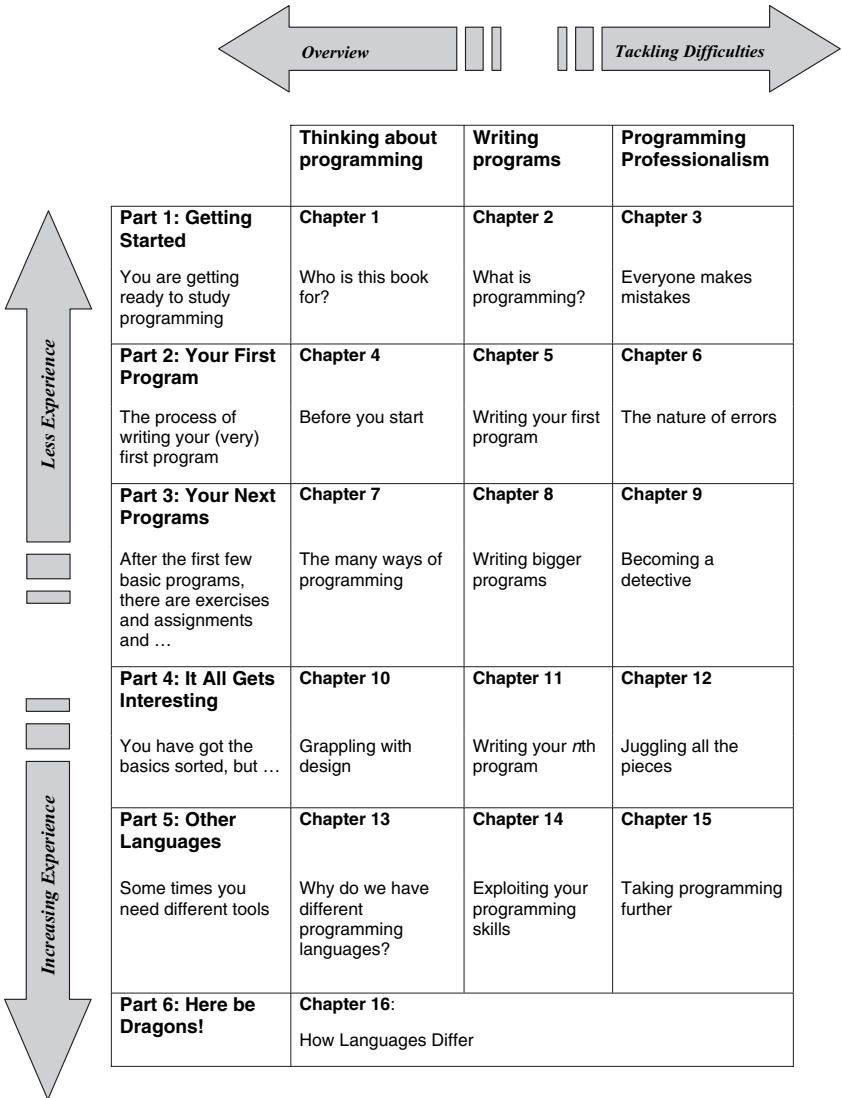


Figure 1.1 This book

three chapters. Each part relates to a stage in the development of a **programmer**, from *Getting Started* (Chapters 1–3), to moving on to *Your First Program* (Chapters 4–6), then exploring *Your Next Programs* (Chapters 7–9), to find that *It All Gets Interesting* (Chapters 10–12) and then learning about *Other Languages* (Chapters 13–15). These parts make good sense if read in

order but each can also be read independently of the others. For instance, if you've already written quite a few programs, you might want to start with *Your Next Programs* or *Other Languages*. You can dip in at the point that suits you. Whatever your level of experience you will probably find that there are ideas that you haven't come across before in most of the chapters.

Another way that we have structured the book is to use three themes that recur in each part: 'Thinking about Programming' (Chapters 1, 4, 7, 10 and 13), 'Writing Programs' (Chapters 2, 5, 8, 11 and 14) and 'Programming Professionalism' (Chapters 3, 6, 9, 12 and 15). So if you want to spend some time just thinking about programming issues you can dip into the first chapter of each part. Or you could focus on practical programming issues and take the second chapter in each of the five parts.

We have tried to capture the essence of these structures in the diagram shown in Figure 1.1. As you can see, the final chapter, 'Here be Dragons!' is a little different and stands outside the other structures. It takes you into the land beyond regular programming tasks and explores the fascinating world of programming **paradigms**. If you want to know what they are, you could even make that your starting point.

The recurrent themes we have highlighted serve to illustrate an important principle about any subject, not just programming: as you learn more and more about a subject, you will find yourself again and again revisiting topics that you have already encountered, and refining and deepening your understanding of them. You will never stop learning.

So, that's it with the first overview. We hope you enjoy the rest of the book!

Index

Note: numbers in **bold** indicate glossary definitions.

- abstraction, 131, **229**
- Ada, 163, **229**
- agile methods, 101, **229**
- Algol, 3–4
- algorithms, 78, 185, 190, **230**
- alien code, 155, **230**
- applications, **230**
- arguments, 212, 214, 215, **230**
- Ariane 5 rocket, 27–8, 155
- ArrayList, 73, **230**
- arrays, 117, 166, **230**
- artificial intelligence, **230**
- atomic data, 134–5, **230**
- AT&T Bell Labs, 162, **230**
- attributes, 131, **230**
- audit trails, 153, **231**
- automobiles, software in, 96
- average, algorithm for calculating, 104–5, 107–9, 118–19

- backtrack, 210–11, **231**
- bar, 126
- BASIC, 3, 14, 17–18, 165, 200, **231**
 - see also* Visual Basic (VB)
- begin, 50
- behaviour, 205, **231**
- big projects, *see* large projects
- black box testing, 116, **231**
- blazon, the language of heraldry, 9–10, 22
- BlueJ, 54, 55, **231**
- Boehm, Barry, 101, 103, 251

- book, represented in Java, 58–61, 135–8, 146–8
- Boolean variables, 142, 144, **231**
- bottom-up, 130, 138, **231**
 - see also* design; top-down
- boundary conditions, 115–16, 144, **231**
- brackets
 - curly brackets { }, 19, 59, 71, 246
 - round brackets (), 19, 212, 214, 216, 246
 - square brackets [], 209
- brainos, 71, **231**
- breakpoints, 110, 111, **232**
- bridge failures, 26–7
- bugs, 30, 65, **232**

- C, 162, **232**
 - flexible data structures, 139
 - ‘hello, world’ program, 49
 - if statements, 16, 19
 - security holes caused by errors, 139
- C++, 162, 165–6, 204, **232**
- capitalization, 61
- careers in programming, 193–4
 - deciding about a programming career, 178–9
 - job advertisements, 167–8, 194
 - mathematical skills, 179–80
 - myths about programming and programmers, 179–83
 - other careers related to programming, 188
 - ratio of males and females, 181–2

- cascading errors, 66, 68, **232**
- case sensitive, **232**
- cause–effect chasms, 106, **232**
- cells, 169, **232**
- central processing units (CPUs), 12, **232**
- characters, 201, **232**
- choices, making, 11, 15–16
- choice statements, *see* *if* statements
- `class`, 60, 135–6
- class definitions, 58–9, 203, 204–7, **233**
- class diagrams, 92–3
 - see also* Unified Modelling Language (UML)
- classes, **232**
- Climate Orbiter, 28, 30, 149
- COBOL (Common Business Oriented Language), 160–1, 182, **233**
- code generators, 170–1, **233**
- code management systems, 95, 152, 153, **233**
- code (source code), **233**
- code walkthroughs, 113, **233**
- commands, 25, **233**
- comments, 90–2, **233**
 - characters and symbols, 112
 - commenting out during debugging, 109, 111–12, 114
 - examples, 50
 - and maintenance of code, 90–2
 - punctuation, 59, 68, 71
 - and readability of code, 87–8
 - rule of thumb for use, 92
- Communicating Sequential Processes (CSP), 163
- compilation, 52–3, 56, 104, **233**
- compilers, 46–7, 50, 52–3, 62, **234**
 - see also* Integrated Development Environments (IDE)
- compile-time errors, 64, 152, **234**
 - see also* syntax errors
- component-based approach, 171–2, **234**
- components, **234**
 - class definition libraries, 207
 - e-commerce, 172–4, 175–6
 - integration, 154
 - interfaces, 151–2, 170
- computer architecture, **234**
- computer programs, **234**
- computers, **234**
 - elements of, 11–13
 - predictability, 7
- computer science education and courses, 189–93
 - A-level classes, 189
 - business-oriented computing courses, 191
 - continued and advanced study, 190–1
 - degree-level courses at university, 189–93
 - final-year projects, 192, 193
 - joint honours programmes, 189–90
 - other subjects that include programming, 191–2
 - work placements, 192–3
 - see also* learning strategies; skills needed in programming
- concurrency, 13, **234**
- concurrent programming, 219–25, **235**
 - see also* occam
- Concurrent Versioning System (CVS), 153, **235**
- `const`, 141
- constants, 134–5, 141, 146, **235**
- control structures, 144, **235**
- CPU, *see* Central Processing Units (CPUs)
- creative aspects of programming
 - art and music, 213–16
 - converting ideas into programming language, 57
 - design skills, 63, 185
 - lateral thinking, 184
 - program design, 37–8
 - programmer satisfaction and, 48, 104, 176, 180, 194
- CVS, *see* Concurrent Versioning System (CVS)

- data, **235**
 - atomic data, 134–5
 - constants, 134–5, 141, 146, **235**
 - data values, 12, 13
 - fixed value holders, 141
 - kinds of, 133–40
 - ‘one or more’ structures, 138–40
 - program structure and data structure, 133, 140
 - structured data, 135–6
 - structures within structures, 136–8, 139
 - ‘zero or more’ structures, 139, 142, 144
- databases, 171, 173–4, 207–11, **235**
- data processing, 160–1, 182, **235**
- data structures, 78, 117, 190, **235**
 - see also* structured data
- data types, *see* types
- debuggers, 109–11, **235**
 - breakpoints, 110, 111, **232**
 - state of program, 110
 - stepping into code, 110–11
- debugging, 30, **235**
 - code walkthroughs, 113
 - commenting out, 109, 111–12, 114
 - expert help, 113–14
 - hand tracing, 107–8, 111, **252**
 - printing results and flow of control information, 108–9
 - simplifying code, 111
 - talking through problems, 113
 - see also* debuggers
- declarative languages, 207–11, **236**
 - see also* Prolog
- declare, **236**
- decompose, 130–1, **236**
- design, 123–32, **236**
 - alternative ways to write programs, 63–4, 81–2
 - benefits of working away from the computer, 41, 45
 - bottom-up, 130, 138, **231**
 - design blocks, types of, 124
 - designing for efficiency, 83–4, 94, 184, 190
 - designing for robustness, 84, 117, 163, 186
 - four-stage problem solving process, 127–9
 - requirements, understanding, 77–8
 - requirements, understanding of, 125–7
 - requirements elicitation, 125–7
 - top-down, 130, 138, **252**
- digit, **236**
- distance travelled by falling object,
 - algorithm for calculating, 134–5, 141, 145–6
- division sign, 18
- do, 16–17
- double (double-precision floating point numbers), **236**
- DrScheme, 216, **236**
- dynamic web pages, 161, 173, 185

- Eats, Shoots and Leaves* (Lynne Truss), 23–4
- e-commerce (electronic commerce), 172–4, 175, 176, **236**
- edit-compile-run cycles, 53, 54, 63, **236**
- editors, 50–2, 62, **236**
 - ed, 51
 - Emacs, 51, **236**
 - Notepad, 51, 52
- education, *see* computer science
 - education and courses
- else, 15–16, 19
- Emacs, 51, **236**
- embedded systems, 176, **237**
- empty lists, 139, 214–15, **237**
- end, 50
- engineering errors and mistakes, 26–7
- entities, 131, 204–5, 207, **237**
- equals sign (=), 18
- Erlang, 163, **237**
- error messages, 25, **237**
 - confusing compiler error messages, 67
 - Java compiler message, 66, 67, 72
 - production by compiler, 53
 - reading and understanding, 65–8, 105–6

- errors, 20–31, 63–74
 - avoiding errors, 70–1
 - cascading errors, 66, 68, **232**
 - compile-time errors, 64, 152, **234**
 - fixing syntax errors, 68–9
 - generic computing errors, 63
 - grammatical errors, 23, 64, **239**
 - guarding against errors, 117–18
 - invalid data, 115, 117
 - large projects, errors and mistakes in, 25–9
 - learning from mistakes, 20–1, 25, 26–7, 29–30, 63, 71
 - legacy programs as sources of error, 28
 - logical errors, 65, 70, 104–5, 117, 152, **243**
 - misuse of data structures, 117
 - runtime errors, 64–5, 104–5, **249**
 - semantic errors, 22–3, 25, 64–5, 104, **250**
 - source code example with many errors, 69, 72–3
 - spelling errors, 24–5, 53, 64, 68
 - syntax errors, 22–3, 25, 64, 66–70, 104, **252**
 - testing for errors, 105, 114–16
 - see also* debugging; error messages
- evolutionary development, **237**
- evolving code, 155, **237**
- exception errors, 105
- executable code, 52–3, 62, **237**
- execute, **237**
- exhaustive testing, 56
- extensibility, 90, **238**
- Extensible Markup Language (XML), **237**
- Extreme Programming (XP), 100, 101–3, **238**
- fast languages, 94, 103
- fast programs, 93–4, 103
- final, 141
- fixed value holders, 141
 - see also* constants
- floating-point numbers, **238**
- flow charts, **238**
- fonts, 43, 51, **238**
- foo, 126
- for loops, 16, 108, 165, 166
- formally proved programs or statements, 163, **238**
- Fortran, 4, 162, 168, 192, **238**
- Frye, Arthur, 29
- functional programming languages, 212–16, **239**
 - see also* Scheme
- functional testing, 116
- functions, 212, **238**
- general-purpose languages, 54, 161, 176, **239**
- generic computing errors, 63
- gizmos, 126, **239**
- glossaries, **239**
- grammatical errors, 23, 64, **239**
- Graphical User Interfaces (GUIs), 151, 161, 170–1, **239**
- graphics, **239**
- gravity, use in calculations, 134–5, 141, 146
- GUI, *see* Graphical User Interfaces (GUIs)
- hacking and hackers, 80, **239**
- hash tables, **239**
- HCI, *see* human–computer interaction (HCI)
- ‘hello, world’ programs, 48–50, 52, **239**
 - in C, 49
 - in HTML, 174, 175
 - in Pascal, 50
 - Wikipedia entries, 49
- heraldry, 9–10, 22–3
- heuristics, 127, **240**
- high-fidelity prototypes, 97, **240**
- high-level languages, **240**
- Hopper, Grace Murray, 30, 182
- HTML, *see* HyperText Markup Language (HTML)
- human–computer interaction (HCI), **240**
- Humphreys, Walt, 99

- HyperText Markup Language (HTML),
10, 174–5, 176, **240**
- IDE, *see* Integrated Development
Environments (IDE)
- identifiers, 14, 53, **240**
see also variables
- if statements, **240**
components of, 15
in C programming language, 16
sample code, 15–16, 17, 19, 20–1
- imperative programming, **240**
- implementation, 150–2, **241**
- indentation, 51, 61, 223
- information design, 123, **241**
- input, **241**
- instructions
repeating instructions, 16–17
sequences of instructions, 14–15, 17–18
see also statements
- int, 49, 60
- integers, 49, 60, **241**
- Integrated Development Environments
(IDE), 54–6, 63, **241**
see also BlueJ; DrScheme; Visual
Basic
- integration, 151, **241**
- integration testing, 154
- interaction diagrams, 86, **241**
- interfaces, 151–2, **241**
communication between teams, 150–2
differences between operating
systems, 95
Graphical User Interfaces (GUIs), 151,
161, 170–1, **239**
implementing to, 150–1, 154
between program components, 151–2
user interface design, 97, 123, **253**
- Internet, **241**
see also e-commerce
- interpreters, 176, **241**
- iteration, 16, **242**
see also loops
- iterative development, 100, 150, **242**
- iterators, 69, 72–3
- Java, **242**
book, represented in Java, 58–61,
135–8, 146–8
class definitions, 58–9, 204–7
compiler messages, 66, 67, 72
flexible data structures, 139–40
meaning of name, 3–4
memory operations, 14, 18
portability, 208
runtime messages, 105
structured data, 135–6
- JavaScript, 217, **242**
- keywords, 53, 61, **242**
- languages, 8–11, 13–17, **242**
see also natural languages;
programming languages; scripting
languages
- large projects
communication, 150–1
complex software, writing and
managing, 90, 94, 95, 97–8, 103
errors and mistakes in large projects,
25–9
extensibility, 90, **238**
number of lines of code, 89–90
Y2K (year 2000) problem or
Millennium Bug, 28–9, 87, 155
see also teams and programming
- large projects, errors and mistakes in,
25–9
- learning strategies
active attention, 38–9, 45
asking questions, 39, 40, 45
breaking down complex problems,
36–7, 79
copying programs from textbooks, 48,
50, 52, 62
cyclic or spiral learning processes, 36,
46
exercises and example programs, 39,
40, 42, 43, 45
follow up and review, 39–40, 44
getting bogged down, 46

- learning strategies – *continued*
 - learning programming vs. other kinds
 - of learning, 35–8
 - in lectures, 40–1
 - note taking, 39
 - obtaining software, 47
 - online material and CDs, 42, 44
 - in practical classes, 40
 - practise, 35–6, 124, 140, 183
 - preparing for lecture or class, 38
 - programming toolkit, 46–7
 - in seminars, problem classes and revision classes, 41
 - sticking points and getting unstuck, 46, 112–14
 - studying programming by yourself, 44–6
 - use of typographical and visual material in textbooks, 43
 - working at and away from the computer, 44, 45
 - see also* programming textbooks; skills needed in programming
- legacy code, 155–6, **242**
- legacy program, 28
- legacy systems, 156, **242**
- LEGO® Mindstorms™, 219–5, **242**
 - see also* occam
- LET, 14, 17, 18
- libraries, 73, 207, **242**
- LISP, 139, 161, **243**
- lists, 139, 142–4, **243**
 - empty, 139, 214–15, **237**
 - sorting, 171–2, 190–1, 212, 217–18
 - variables, 142–4
- load, 180, **243**
- logbooks, 67, 71, 84–5, 106
- logical errors, 65, 70, 104–5, 117, 152, **243**
 - see also* runtime errors; semantic errors
- loops, 16–17, **243**
 - components, 16
 - linear structures and, 144
 - for loops, 16, 108, 165, 166
 - stepper variables in, 142
 - while loops, 16–17, 142–4
 - see also* iteration
- lower case, 61, 201, **243**
- low-fidelity prototypes, 97, **243**
- machine code, 53, 161, 179–80, **243**
- macros, 170, **243**
- main, 49
- maintenance of programs, 84, 90, 100, 156, 162, 185
- markup languages, **243**
 - see also* Extensible Markup Language (XML); HyperText Markup Language (HTML)
- Mars Climate Orbiter, 28, 30, 149
- mathematical proof, 163, **243**
- mathematical symbols
 - division sign, 18
 - division sign (*/*), 18
 - equals sign (=), 18
 - greater than (>), 113
 - greater than or equal to (>=), 19, 20–1
 - plus sign (+), 18
- memory, **244**
 - memory operations, 12, 14, 17–18, 180
 - numerical addresses, 14
- meta-data, **244**
- Microsoft Word, 51, 153, 170, 217
- Millennium Bridge, 26–7
- mistakes, *see* errors
- modules, 13, 150, 168, **244**
- Moisseiff, Leon, 27
- Moore, Gordon, 93
- Moore’s Law, 93
- Mozilla, 96
- multi-tasking, 11, 84, **244**
- music
 - art and music in programming, 213–16
 - musical notations, 8–9
 - music generation with Scheme, 213–16
- names
 - memory locations, 14; *see also* variables

names – *continued*

- of programming languages, 3–4
- and readability of code, 87
- nasty surprises, 163, **244**
- natural languages, 37, **244**
 - punctuation in, 71
 - similarity to programming, 37, 57–8
- \n character pair, 49
- nerds, 182–3
- nest, 17, 212, 213, **244**
- neural networks, **244**
- niche languages, 160, **244**
- non-programming languages
 - blazon, the language of heraldry, 9–10
 - heraldry, simple errors, 22–3
 - knitting, common errors, 21
 - knitting notations, 9
 - musical notations, 8–9
 - task-oriented languages, 8–11
 - see also* natural languages
- Notepad, 51, 52
- numerical applications, 162, **245**
- object orientation, 204, **245**
- object-oriented programming
 - attributes, 131, **230**
 - behaviour, 205, **231**
 - class definitions, 204–7, **233**
 - class diagrams, 92–3
 - entities, 131, 204–7, **237**
 - objects, 136–7, 148, 205–7, **245**
- objects, 136–7, 148, 205–7, **245**
- occam, 163, 219–25, **245**
 - see also* concurrent programming; LEGO® Mindstorms™
- off-by-one bugs, 144, **245**
- one-offs, **245**
- open-source programs, 90–1, 96, 194–5, **245**
- operating system (OS), 94–5, 99, **245**
- operators, 25, 53, 161, **245**
- otherwise, 15
- output, **246**
- packages, 168, 169–70, 173–4, **246**
- pair programming, 102, 149–50, **246**

paradigms, 199–226, **246**

- concurrent paradigm, 219–25
 - declarative paradigm, 207–11
 - functional paradigm, 212–16
 - imperative paradigm, 200–4
 - object-oriented paradigm, 204–7
 - scripting language paradigm, 216–19
- parameters, *see* arguments
- parentheses (round brackets; singular, parenthesis), **246**
 - use in *if* statements, 19
 - use in Scheme, 212, 214, 216
- Pascal, 176, 200, **246**
 - ‘hello, world’ program, 50, 52
 - memory operations, 14, 18
- patterns, 143–5, **246**
- Perl, 161, 192, 216–19, **246**
 - see also* scripting languages
- Personal Software Process (PSP), 98–100, **246**
- PHP (Hypertext PreProcessor), 161–2, **247**
- pictures
 - for clarifying code, 90, 92–3, 97, 220
 - data values, 12
 - see also* Unified Modelling Language (UML)
- plagiarism, 86, 88, 150, **247**
- platforms, 94, **247**
- plus sign (+), 18
- Polya, George, 127
- portable programs, 95, 208, **247**
- positive feedback, 27
- Post-it® Notes, 29
- predefined words and symbols, 59–60
- print, **247**
 - debugging messages, 108–9
- printf, 49
- println, 43, 50, 142
- private, 147, **247**
- problem-solving strategies
 - four-stage problem solving process, 127–9
 - see also* strategies for developing programs
- processing units, 12, **247**

- program design, *see* design
- programmer-chosen words and symbols, 61–2
- programmers, 48, **247**
 - characteristics, 183–8, 226
 - myths about programming and programmers, 179–83
- programming, overview
 - key elements, 10
 - programming tasks in everyday life, 10–11
 - what it means to be programmed, 7–8
 - see also* skills needed in programming
- programming careers, *see* careers in programming
- programming languages, 159–66, **247**
 - Ada, 163, **229**
 - C++, 162, 165–6, 204, **232**
 - COBOL (Common Business Oriented Language), 160–1, 182, **233**
 - comparison with natural languages, 7, 37, 57–8
 - in data processing systems, 160–1
 - development, 164
 - differences between programming languages, 13
 - Erlang, 163, **237**
 - essential features, 13–14
 - families, 54, 130, 164, 165
 - formally proved programs or statements, 163, **238**
 - Fortran, 3–4, 162, 168, 192, **238**
 - glue, 161
 - HyperText Markup Language (HTML), 10, 174–5, 176, **240**
 - JavaScript, 217, **242**
 - language structures, 13–17
 - learning multiple languages, 167–8, 186, 188, 193
 - naming of, 3–4
 - in numeric applications, 162
 - occam, 163, 219–25, **245**
 - Perl, 161, 192, 216–19, **246**; *see also* scripting languages
 - PHP (Hypertext PreProcessor), 161–2, **247**
 - Prolog, 208–12
 - in safety-critical systems, 81, 116, 162–3, 164, **249**
 - similarities of languages, 168
 - simplicity of, 7, 13
 - special-purpose and incomplete languages, 174–6
 - SQL (Structured Query Language), 161, 174, 191
 - Structured Query Language (SQL), 161, 174, **251**
 - Visual Basic (VB), 110, 118–19, 161, 170, **231**
 - see also* C; object-oriented programming; scripting languages
 - programming textbooks, 41–4
 - see also* learning strategies
- Prolog
 - backtracking, 210–12, **231**
 - knowledge bases, 208
 - output variables, 211
 - queries, 208–10
 - rule bases, 207, 211
 - subgoals, 208–12
 - see also* declarative languages
- protocols, **247**
- prototypes, 97, 129, **247–8**
- prototyping, **248**
- pseudo-code, 128, **248**
- public, **248**
- punctuation, 19, 71, **248**
 - at end of statements, 14, 19, 71
 - error messages caused by, 68
 - in natural languages, 71
 - uses of, 59
- questions
 - as learning strategies, 39
 - requirements elicitation, 125–7
- quote marks as causes of errors, 66, 68
- RDBMS, *see* relational database management system (RDBMS)
- readability of code, 51, 61, 87–8

- reading code, 58–62
 - purposes, 58
 - reading programs, as a skill, 37, 42–3, 85–6
- real-time systems, 219, **248**
- recursion, 13, 129, **248**
- regression testing, 154–5, **248**
- relational database management system (RDBMS), 161, 174, **249**
- relational databases, **248**
- relationships, 87, 131, 208, **249**
- repeating instructions, 16–17
- repeat, 16
- report-writing packages, 172, **249**
- requirements, 77–8, 125–7, **249**
- requirements elicitation, 125–7, **249**
- return, 49, **249**
- Ritchie, Dennis, 162
- RoboCup*, 195
- Roman numerals, 159–60
- Royce, Winston, 103
- runtime errors, 64–5, 104–5, **249**
 - see also* logical errors; semantic errors
- runtime systems, 53, 63, 176, 218, **249**
- safety-critical, 81, 116, 162–3, 164, **249**
- Sajaniemi, Jorma 'Saja', 141, 143
- Scheme, 212–16, **249**
 - append, 213
 - arguments, 212, 214, 215
 - cons, 214, 215
 - define, 214
 - functions, 212–16
 - lists, 139
 - parentheses in, 212, 214, 216
 - structured data, 136
- scripting languages, 216–19, **249**
 - compilation, lack of, 54, 218
 - glue, 161, 175
 - interpretation, 161, 176
 - supporting environment, 161
 - see also* JavaScript; Perl; PHP (Hypertext PreProcessor)
- semantic, **250**
- semantic errors, 22–3, 25, 64–5, 104, **250**
 - see also* logical errors; runtime errors
- semicolons, 71
 - at end of statements, 14, 19, 59
 - and programming errors, 64, 67–8, 234
- sets, **250**
- Silver, Spence, 29
- skills needed in programming
 - breaking down complex problems, 36–7, 79, 130–1
 - creating a set of instructions, 8
 - creative design, similarity to programming, 37–8
 - learning natural languages, similarity to programming, 37, 57–8
 - mathematical skills, 36–7, 179–80
 - planning ahead, 38
 - practical skills, similarity to programming, 35–6
 - practise, 35, 36, 124, 140, 183
 - programming as a unified skill, 36, 46
 - reading programs, 37, 42–3, 85–6
 - see also* learning strategies
- slashes (/), 59, 146
- SLoC, *see* source lines of code (SLoC)
- Smalltalk, 54, 204, **250**
- software development, *see* Extreme Programming (XP); large projects; Personal Software Process (PSP); Waterfall Model
- software engineering, 190–1, **250**
- software packages, *see* packages
- software patterns, *see* patterns
- software tools, 118, 172, **250**
- sorts and sorting, 77–8, 171–2, 190, 212
- source code, 62, **250**
 - and compilers, 52–3
 - example with many errors, 69, 72–3
- source lines of code (SLoC), 90, **250**
- specification languages, 163, **250–1**

- specifications, 125, **250**
 - design, 117
 - problem, 123, 128
 - program, 116, 125, 152, 155
- spelling errors, 24–5, 53, 64, 68
- spiral model, 101, 103, **251**
- spreadsheets, 105, 115, 169–70, **251**
- SQL, *see* Structured Query Language (SQL)
- Squeak, 54
- statements, **251**
 - order of execution, 14–15
 - punctuation at end of statements, 14
 - sequences of instructions, 14–15, 17–18
 - see also* specific types of statements
- state of a program, 107–8, 111, 200–1, **251**
- stepper variables, 142, 144
- strategies for developing programs
 - breaking down complex problems, 36–7, 79
 - designing for efficiency, 83–4, 94, 184, 190
 - designing for robustness, 84, 117, 163, 186
 - discovery, process of, 78, 80
 - divide and conquer, 79
 - efficiency of programs, 78
 - good programming habits, 84–8
 - hacking, 80, 82
 - keeping good records or logbooks, 67, 71, 84–5, 106
 - meta-strategies, 84
 - planning and executing, 79
 - playing, 80
 - readability of code, 51, 61, 87–8
 - reading programs, 42–3, 85–6
 - selecting a problem-solving strategy, 81–4
 - talking about programming, 85
 - time requirements of programming, 82–3
 - understanding requirements of task, 77–8
 - visualizing, 86–7
 - working incrementally (*iterating*, *building in cycles*), 79
 - see also* writing programs
- String, 60
- strings, 60, **251**
- Stroustrup, Bjarne, 162
- structural testing, 116
- structured data, 135–6, **251**
 - in Java, 135–6
 - linear structures, 144
 - ‘one or more’ structures, 138–40
 - in Scheme, 136
 - structures within structures, 136–8, 139
 - ‘zero or more’ structures, 139, 142, 144
 - see also* data
- Structured Query Language (SQL), 161, 174, **251**
- study skills, *see* learning strategies; skills needed in programming
- subroutines, 92, 170, **251**
- supersets, 162, **251**
- symbols, **251**
- syntax, 25, **251–2**
- syntax errors, 22–3, 25, 64, 66–70, 104, **252**
 - see also* compile-time errors
- Tacoma Narrows Bridge, 26–7
- task-oriented languages, 8–11
- Tcl, *see* Tool Command Language (Tcl)
- teams and programming, 149–54
 - communication, 150–1
 - Extreme Programming (XP), 101
 - logbooks, 85
 - see also* pair programming
- technical requirements, 152, **252**
- test cases, 114–15, 153, 154, **252**
- test data, 115, **252**
- test harnesses, 153, **252**
- testing, 56–7, 114–16, **252**
 - black box testing, 116, **231**
 - boundary conditions, 115–16, 144, **231**

- testing *—continued*
 - classes of tests, 114
 - comparison to calculated results, 114
 - exhaustive testing, 56
 - finding runtime errors, 105
 - functional testing, 116
 - guarding against errors, 117–18
 - integration testing, 154
 - invalid data, 115, 117
 - ‘no known bugs’ vs. bug-free programs, 116
 - regression testing, 154–5
 - selection of test data, 115–16
 - structural testing, 116
 - techniques and strategies, 56–7
 - white box testing, 116, **254**
 - see also* unit testing
- test suites, 152–3, **252**
- text editors, 50–2, **252**
 - ed, 51
 - Emacs, 51, **236**
 - Notepad, 51, 52
 - see also* Integrated Development Environments (IDE)
- then, 15
- time requirements of programming, 82–3
- tokens, 53
- Tool Command Language (Tcl), **252**
- top-down, 130, 138, **252**
 - see also* bottom-up; design
- tracing, 107–8, 111, **252**
- types, **253**
- typos, 23, 67–8, 71, **253**

- UML, *see* Unified Modelling Language (UML)
- Unified Modelling Language (UML), 86–7, 92, 127, 168, **253**
- unit conversions, 28
- unit testing, 96, 102, **253**
- uppercase, 201, **253**
- usability, 97
- use-case diagrams, 87, **253**
- user interface design, 97, 123, **253**

- variables, **253**
 - for constants, 135, 141
 - fixed value holders, 141
 - most-recent holders, 143
 - placeholders, 133
 - roles, 140–3
 - stepper variables, 142, 144
 - structured data, 136–8
 - use in memory operations, 14
- VB, *see* Visual Basic (VB)
- VBA, *see* Visual Basic for Applications (VBA)
- version control systems, 95, **254**
 - see also* Concurrent Versioning System (CVS)
- vi, 51, **254**
- Visual Basic (VB), 110, 118–19, 161, 170, **231**
- Visual Basic for Applications (VBA), **254**

- Wall, Larry, 161, 246
- Waterfall Model, 100–1, 102, 103, 150, **254**
- web browsers, 96, 173–5, 176, 217, **254**
- web pages
 - dynamic, 161, 173, 185
 - languages used, 161, 174
 - see also* e-commerce
- web servers, 173–4
- while loops, 16–17, 142–4
- white box testing, 116, **254**
- white space, 61–2, **254**
- widgets, 126, **254**
- Wikipedia, 49, 174, 182, 229, **254**
- word processors, 51, 62, 174, **255**
- worksheets, **255**
- World Wide Web (WWW), 174, **255**
- writing programs
 - alternative ways to write programs, 63–4, 81–2
 - bigger programs, 89–103
 - good programming habits, 84–8
 - as a skill, 37
 - understanding requirements of task, 77–8

- writing programs – *continued*
 - using Integrated Development Environments (IDE), 54–6
 - using text editors, 50–2
 - your first program, 48–57
 - your nth program, 133–48
 - your second program, 57–8
 - see also* strategies for developing programs
- WWW, *see* World Wide Web (WWW)
- XML, *see* Extensible Markup Language (XML)
- XP, *see* Extreme Programming (XP)
- Y2K (year 2000) problem or Millennium Bug, 28–9, 87, 155
- Z, 163, **255**
- zero, dividing by, 65, 104–5, 119
- ‘zero or more’ structures, 139, 142, 144

