SECOND EDITION

simple PROGRAM *design*

a step by step approach

lesley anne robertson

SECOND EDITION

Simple Program Design

Lesley Anne Robertson

a step by step • approach •

Course Technology, Inc. One Main Street, Cambridge, MA 02142

Nelson I(T)P®
102 Dodds Street
South Melbourne 3205

Email nelsonitp@nelson.com.au
Website http://www.nelsonitp.com

Simultaneously published by Course Technology, Inc
One Main Street
Cambridge, MA 02142

Nelson I(T)P® ***an International Thomson Publishing company***

First published in 1993
10 9
05 04 03 02 01 00 99 98
Copyright © Lesley Anne Robertson

Nelson I(T)P® edition ISBN 0 17 008822 7

National Library of Australia
Cataloguing-in-Publication data

Robertson, Lesley Anne.
ISBN 0 17 008822 7.
Simple program design.
2nd ed.
1. Programming (Electronic computers). 2. Structured programming.
3. Pseudocode (Computer program language). I. Title.
005.12

Boyd & Fraser edition ISBN 0-87709-283-4

Cover designed by David Albon
Text designed by Christina Neri
Printed in China by L. Rex Printing Co. Ltd.

Nelson Australia Pty Limited ACN 058 280 149 (incorporated in Victoria) trading as Nelson ITP.

SECOND EDITION

Simple Program Design

a step by step ● approach ●

Preface

With the increased popularity of programming courses in our universities, colleges and technical institutions, there is a need for an easy-to-read textbook on computer program design. There are already dozens of introductory programming texts using specific languages such as PASCAL, BASIC, COBOL or C, but they usually gloss over the important step of designing a solution to a given programming problem.

This textbook tackles the subject of program design by using structured programming techniques and pseudocode to develop a solution algorithm. The recommended pseudocode has been chosen because of its closeness to written English, its versatility and ease of manipulation, and its similarity to the syntax of most structured programming languages.

Simple Program Design is designed for programmers who want to develop good programming skills for solving common business problems. Too often, programmers who are faced with a problem launch straight into the code of their chosen programming language, instead of concentrating on the actual problem at hand. They become bogged down with the syntax and format of the language, and often spend many hours getting the program to work. Using this textbook, the programmer will learn how to define the problem, how to design a solution algorithm, and how to prove the algorithm's correctness, before coding a single statement from any programming language. By using pseudocode and structured programming techniques, the programmer can concentrate on developing a well-designed and correct solution, and thus eliminate many frustrating hours at the testing phase.

The book is divided into eleven chapters, beginning with a basic explanation of structured programming techniques, top-down development and modular design. Then, concept by concept, the student is introduced to the syntax of pseudocode; methods of defining the problem; the application of basic control structures in the development of the solution algorithm; desk checking techniques; hierarchy charts; module design; parameter passing; object-oriented design methodology; and many common algorithms.

Each chapter thoroughly covers the topic at hand, giving practical examples relating to business applications, and a consistently structured approach when representing algorithms and hierarchy charts.

This second edition of *Simple Program Design* contains material which is an extension to that in the first edition. Nassi-Schneiderman (N-S) diagrams have been removed from the body of the book and are introduced and developed entirely in Appendix 1. Appendix 1 has been written for those programmers who prefer a more diagrammatic approach to algorithm design and solutions to all the pseudocode algorithms in Chapters 2, 3, 4 and 5 have been presented again in this appendix using N-S diagrams.

This second edition also contains a number of early examples and problems which do not involve file processing. Because the examples are interactive, many of them can be coded directly into a programming language, and executed, before the topic of file processing has been covered.

The concepts of arrays and parameter passing are introduced early in the book, with many examples provided. Object-oriented design methodology and information hiding are introduced in Chapter 8, and dynamic data structures such as queues, stacks and linked lists are introduced in Appendix 2.

I would like to thank Dr Malcolm Cook at the University of Western Sydney for his suggestions and assistance with this second edition; my husband, David, for editing the manuscript; and my brother, Rick Noble, for his amusing cartoons.

Lesley Anne Robertson

Contents

Preface ix

1 Program design

Describes the steps in the program development process, explains structured programming, and introduces algorithms and pseudocode.

1.1 Steps in program development 2
1.2 Structured programming 4
1.3 An introduction to algorithms and pseudocode 5
1.4 Chapter summary 7

2 Pseudocode

Introduces common words and keywords used when writing pseudocode. The Structure Theorem is introduced, and the three basic control structures are established. Pseudocode is used to represent each control structure.

2.1 How to write pseudocode 10
2.2 The Structure Theorem 13
2.3 Chapter summary 16

3 Developing an algorithm

Introduces methods of analysing a problem and developing a solution. Simple algorithms which use the sequence control structure are developed, and methods of manually checking the algorithm are determined.

3.1 Defining the problem 18
3.2 Designing a solution algorithm 22
3.3 Checking the solution algorithm 25
3.4 Chapter summary 30
3.5 Programming problems 31

4 Selection control structures

Expands the selection control structure by introducing multiple selection, nested selection, and the case construct in pseudocode. Several algorithms, using variations of the selection control structure, are developed.

4.1 The selection control structure 34
4.2 Algorithms using selection 37
4.3 The case structure 44
4.4 Chapter summary 47
4.5 Programming problems 48

5 Repetition control structures

Develops algorithms which use the repetition control structure in the form of DOWHILE, REPEAT...UNTIL, and counted repetition loops.

5.1 Repetition using the DOWHILE structure 52
5.2 Repetition using the REPEAT...UNTIL structure 61
5.3 Counted repetition constructs 66
5.4 Chapter summary 70
5.5 Programming problems 70

6 Pseudocode algorithms using sequence, selection and repetition

Develops algorithms to eight simple programming problems using combinations of sequence, selection and repetition constructs. Each problem is properly defined; the control structures required are established; a pseudocode algorithm is developed; and the solution is manually checked for logic errors.

6.1 Eight solution algorithms 74
6.2 Chapter summary 85
6.3 Programming problems 85

7 Modularisation

Introduces modularisation as a means of dividing a problem into subtasks. Hierarchy charts are introduced as a pictorial representation of program module structure. Several algorithms which use a modular structure are developed.

7.1 Modularisation 88
7.2 Hierarchy charts or structure charts 91
7.3 Steps in modularisation 97
7.4 Programming examples using modules 97
7.5 Chapter summary 108
7.6 Programming problems 109

8 Communication between modules

Defines elementary data items and data structures and introduces the concepts of inter-module communication, local and global data, and the passing of parameters between modules. Algorithms which pass parameters are developed. The concept of object-oriented design is introduced, and the terms associated with it are defined.

8.1 Program data 114
8.2 Communication between modules 115
8.3 Using parameters in program design 117
8.4 Object-oriented design 121
8.5 Chapter summary 123
8.6 Programming problems 123

9 Cohesion and coupling

Introduces the concepts of module cohesion and coupling. Several levels of cohesion and coupling are described, and pseudocode examples of each level are provided.

9.1 Module cohesion 126
9.2 Module coupling 132
9.3 Chapter summary 137
9.4 Programming problems 137

10 General pseudocode algorithms for common business problems

Develops a general pseudocode algorithm for five common business applications. All problems are defined; a hierarchy chart is established; and a pseudocode algorithm is developed, using a mainline and several subordinate modules. The topics covered include report generation with page break, a single-level control break, a multiple-level control break, a sequential file update program, and array processing.

10.1 Program structure 140
10.2 Report generation with page break 141
10.3 Single-level control break 143
10.4 Multiple-level control break 146
10.5 Sequential file update 151
10.6 Array processing 158
10.7 Chapter summary 161
10.8 Programming problems 161

11 Conclusion

A revision of the steps involved in good program design.

11.1 Simple program design 166
11.2 Chapter summary 168

Appendix 1 Nassi-Schneiderman diagrams

Covers Nassi-Schneiderman diagrams for those students who prefer a more diagrammatic approach to program design. Algorithms which use a combination of sequence, selection and repetition constructs are developed in some detail.

The three basic control structures 170
Simple algorithms which use the sequence control structure 172
N-S diagrams and the selection control structure 174
Simple algorithms which use the selection control structure 176
The case structure, expressed as an N-S diagram 180
N-S diagrams and the repetition control structure 181
Simple algorithms which use the repetition control structure 182

Appendix 2 Special algorithms

Contains a number of algorithms which are not included in the body of the book but may be required at some time in a programmer's career.

Sorting algorithms 188
Algorithms which manipulate arrays 190
Dynamic data structures 192

Glossary 197
Index 201

CHAPTER

1

Program design

OBJECTIVES

- ▨ To describe the steps in the program development process
- ▨ To explain structured programming
- ▨ To introduce algorithms and pseudocode

OUTLINE

1.1 Steps in program development
1.2 Structured programming
1.3 An introduction to algorithms and pseudocode
1.4 Chapter summary

1.1 STEPS IN PROGRAM DEVELOPMENT

Computer programming is an art. Many people believe that a programmer must be good at mathematics, have a memory for figures and technical information, and be prepared to spend many hours sitting at a terminal, typing programs. However, given the right tools, and steps to follow, anyone can write well-designed programs. It is a task worth doing, as it is both stimulating and fulfilling.

Programming can be defined as the development of a solution to an identified problem, and the setting up of a related series of instructions which, when directed through computer hardware, will produce the desired results. It is the first part of this definition which satisfies the programmer's creative needs: that is, to design a solution to an identified problem. Yet this step is so often overlooked. Leaping straight into the coding phase without first designing a proper solution usually results in programs that contain a lot of errors. Often the programmer needs to spend a significant amount of time finding these errors and correcting them. A more experienced programmer will design a solution to the program first, desk check this solution, and then code the program in a chosen programming language.

There are seven basic steps in the development of a program. An outline of these seven steps follows.

1 Define the problem

This step involves carefully reading and rereading the problem until you understand completely what is required. To help with this initial analysis, the problem should be divided into three separate components:

- the inputs,
- the outputs, and
- the processing steps to produce the required outputs.

A defining diagram as described in Chapter 3 is recommended in this analysis phase, as it helps to separate and define the three components.

2 Outline the solution

Once the problem has been defined, you may decide to break the problem up into smaller tasks or steps, and establish an outline solution. This initial outline is usually a rough draft of the solution which may include:

- the major processing steps involved,
- the major subtasks (if any),
- the major control structures (e.g. repetition loops),
- the major variables and record structures, and
- the mainline logic.

The solution outline may also include a hierarchy or structure chart. The steps involved in creating this outline solution are detailed in Chapters 2 to 6.

3 Develop the outline into an algorithm

The solution outline developed in Step 2 is then expanded into an algorithm: a set of precise steps which describe exactly the tasks to be performed and the order in which they are to be carried out. This book uses pseudocode (a form of structured English) to represent the solution algorithm, as well as structured programming techniques. Nassi-Schneiderman diagrams are also provided in Appendix 1 for those who prefer a more pictorial method of algorithm representation. Algorithms using pseudocode and the Structure Theorem are developed thoroughly in Chapters 2 to 6.

4 Test the algorithm for correctness

This step is one of the most important in the development of a program, and yet it is the step most often forgotten. The main purpose of desk checking the algorithm is to identify major logic errors early, so that they may be easily corrected. Test data needs to be walked through each step in the algorithm to check that the instructions described in the algorithm will actually do what they are supposed to. The programmer walks through the logic of the algorithm, exactly as a computer would, keeping track of all major variables on a sheet of paper. Chapter 3 recommends the use of a desk check table to desk check the algorithm, and many examples of its use are provided.

5 Code the algorithm into a specific programming language

Only after all design considerations have been met in the previous four steps should you actually start to code the program into your chosen programming language.

6 Run the program on the computer

This step uses a program compiler and programmer-designed test data to machine-test the code for both syntax and logic errors. This is usually the most rewarding step in the program development process. If the program has been well designed then the usual time-wasting frustration and despair often associated with program testing are reduced to a minimum. This step may need to be performed several times until you are satisfied that the program is running as required.

7 Document and maintain the program

Program documentation should not be listed as the last step in the program development process, as it is really an ongoing task from the initial definition of the problem to the final test result.

Documentation involves both external documentation (such as hierarchy charts, the solution algorithm, and test data results) and internal documentation which may have been coded in the program. Program maintenance refers to changes which may need to be made to a program throughout its life. Often these changes are performed by a different programmer from the one who initially wrote the program. If the program has been well designed using structured programming techniques, the code will be seen as self documenting, resulting in easier maintenance.

1.2 STRUCTURED PROGRAMMING

Structured programming helps you to write effective, error-free programs. The original concept of structured programming was set out in a paper published in 1964 in Italy by Bohm and Jacopini. They established the idea of designing programs using a Structure Theorem based on three control structures. Since then a number of authors, such as Edsger Dijkstra, Niklaus Wirth, Ed Yourdon and Michael Jackson, have developed the concept further and have contributed to the establishment of the popular term 'structured programming'. This term now refers not only to the Structure Theorem itself, but also to top-down development and modular design.

Top-down development

Traditionally, programmers presented with a programming problem would start coding at the beginning of the problem and work systematically through each step until reaching the end. Often they would get bogged down in the intricacies of a particular part of the problem, rather than considering the solution as a whole. In the top-down development of a program design, a general solution to the problem is outlined first. This is then broken down gradually into more detailed steps until finally the most detailed levels have been completed. It is only after this process of 'functional decomposition' (or 'stepwise refinement') that the programmer starts to code. The result of this systematic, disciplined approach to program design is a higher precision of programming than was possible before.

Modular design

Structured programming also incorporates the concept of modular design, which involves grouping tasks together because they all perform the same function (e.g. calculating sales tax or printing report headings). Modular design is connected directly to top-down development, as the steps or subtasks into which the programmer breaks up the program solution will actually form the future modules of the program. Good modular design aids in the reading and understanding of the program.

The Structure Theorem

The Structure Theorem revolutionised program design by eliminating the GOTO statement and establishing a structured framework for representing the solution. The theorem states that it is possible to write any computer program by using only three basic control structures. These control structures are:

- sequence;
- selection, or IF-THEN-ELSE; and
- repetition, or DOWHILE.

They are covered in detail in Chapter 2.

1.3 AN INTRODUCTION TO ALGORITHMS AND PSEUDOCODE

Structured programming techniques require a program to be properly designed before coding begins, and it is this design process which results in the construction of an algorithm.

What is an algorithm?

An algorithm is like a recipe: it lists the steps involved in accomplishing a task. It can be defined in programming terms as a set of detailed, unambiguous and ordered instructions developed to describe the processes necessary to produce the desired output from a given input. The algorithm is written in simple English and is not a formal document. However, to be useful, there are some principles which should be adhered to. An algorithm must:

- be lucid, precise and unambiguous;
- give the correct solution in all cases; and
- eventually end.

For example, if you want to instruct someone to add up a list of prices on a pocket calculator, you might write an algorithm like the following:

```
Turn on calculator
Clear calculator
Repeat the following instructions
    Key in dollar amount
    Key in decimal point (.)
    Key in cents amount
    Press addition (+) key
Until all prices have been entered
Write down total price
Turn off calculator
```

Notice that in this algorithm the first two steps are performed once, before the repetitive process of entering the prices. After all the prices have been entered and summed, the total price can be written down and the calculator can be turned off. These final two activities are also performed only once. This algorithm satisfies the desired list of properties: it lists all the steps in the correct order from top to bottom, in a definite and unambiguous fashion, until a correct solution is reached. Notice that the steps to be repeated (entering and summing the prices) are indented, both to separate them from those steps performed only once and to emphasise the repetitive nature of their action. It is important to use indentation when writing solution algorithms because it helps to differentiate between the three control structures.

What is pseudocode?

Flowcharts were once used to represent the steps in an algorithm diagrammatically, but they were bulky and difficult to draw, and often led to poor program structure. In contrast, pseudocode is easy to read and write, as it represents the statements of an algorithm in English. Pseudocode is really structured English. It is English which has been formalised and abbreviated to look very like high-level computer languages.

There is no standard pseudocode at present. Authors seem to adopt their own special techniques and sets of rules, which often resemble a particular programming language. This book attempts to establish a standard pseudocode for use by all programmers, regardless of the programming language they choose. Like many versions of pseudocode, this version has certain conventions, as follows:

1 Statements are written in simple English.
2 Each instruction is written on a separate line.
3 Keywords and indentation are used to signify particular control structures.

4 Each set of instructions is written from top to bottom, with only one entry and one exit.
5 Groups of statements may be formed into modules, and that group given a name.

Pseudocode has been chosen to represent the solution algorithms in this book because its use allows the programmer to concentrate on the logic of the problem.

An alternative pictorial method of representing algorithms using Nassi-Schneiderman diagrams is described in detail in Appendix 1.

1.4 CHAPTER SUMMARY

In this chapter, the steps in program development were introduced and briefly described. These seven steps are:

1 Define the problem.
2 Outline the solution.
3 Develop the outline into an algorithm.
4 Test the algorithm for correctness.
5 Code the algorithm into a specific programming language.
6 Run the program on the computer.
7 Document and maintain the program.

Structured programming was presented as a combination of three separate concepts: top-down development, modular design, and the use of the Structure Theorem when designing a solution to a problem.

An algorithm was defined as a set of detailed, unambiguous and ordered instructions developed to describe the processes necessary to produce the desired output from the given input. Pseudocode is an English-like way of representing the algorithm; its advantages and some conventions for its use were listed.