

Introduction to Maple Programming

Anthony G. O'Farrell



CLÓ LOIGHIC/ LOGIC PRESS
Kilcock, Co. Kildare

Anthony G. O'Farrell
Mathematics Department
NUI Maynooth
Co. Kildare
Ireland
e-mail: aof@maths.nuim.ie

©Logic Press, 2005

All rights are reserved including the specific rights to translate, reprint, recite, perform, broadcast or reproduce on microfilm or in any other way or to store in data banks all or any part of this book, without prior permission from the publisher.

ISBN: 0-9544269-4-0

Printed in Ireland

Available from bookshops, and by post from the publisher.

Tel: +353-1-628-7343

Fax: +353-1-708-3913

e-mail: logic@maths.nuim.ie



Published by:

CLÓ LOIGHIC / LOGIC PRESS

The Maws, Kilcock

Co. Kildare, Ireland

web: <http://logic.maths.nuim.ie>

Preface

Maple is a symbolic computation system, developed and distributed by Waterloo Maple, Inc. It is one of a number of such systems. These systems crossed a threshold in the 1980's, making possible a new kind of Mathematical experience, a new way of learning, exploring, and using Mathematics.

In response to this development, we at NUIM decided, over fifteen years ago, to pick one system, and make it available to all students, on as many platforms as possible. We reviewed the competing claims of the various systems, and decided, on a balanced assessment of their strengths, to adopt Maple as standard. Thereafter, my colleagues and I developed and deepened our understanding of Maple, and set about helping our students to use it. This book grew out of that experience.

The book is not a self-contained *reference* book on Maple. Maple does not need such a thing: the Maple help system supplies all the reference material a user needs. It is intended to be used as an aid to people who are beginning to write their own programs in Maple.

Maple allows the student to develop her understanding of Mathematics in an experimental way. Thinking algorithmically about Mathematics adds a new dimension. This book is designed to exploit these opportunities to revisit previously-studied Mathematics. It is packed with exercises, ranging from minor items up to major projects. These draw on a range of areas usually studied by undergraduate majors in Mathematics, pure and applied.

I have found that a typical student with no previous programming experience can be expected to master the core of the material covered here in one semester. At Maynooth, we run a course that involves the student in two tutorials or lectures, one intensive supervised practical, and a further seven hours unsupervised lab work per week, with access to a two-hour helpdesk and three office-hours per week. The students work in groups of three to complete eight graduated assignments, each involving 5 mathematical problems, and are examined by individual interviews. Most are in their third undergraduate year. This is rather labour-intensive, but the outcomes are

good.

As a prerequisite, students are assumed to have a basic familiarity with the Maple environment, such as may be acquired in a ten-contact-hour course. One source for this background material is the book *Maynooth Maple Manual*, by Aiden O'Reilly, Logic Press. In practice, one expects that some students will be out of practice, and will need to put in a substantial amount of time on revision during the first week of work. The material and exercises on the kernel and library procedures in Chapter 3 are useful for this purpose. By the time the student has worked through those, she is up to speed.

I have learned a great deal from my colleagues and students about Maple, and I would like especially to thank Derek Bell, Stephen Buckley, André Diatta, Catherine Hurley, Simon Kokkendorff, Christoph List, Janice Love, Sarah Mattingly, Gary McGuire, Aiden O'Reilly, and Alejandro Sanabria. In the course of many conversations and teaching collaborations with them, I picked up many of the tips and examples that appear here. I would also like to thank them, and George Huxley, for their comments on various drafts. Of course, all the errors that remain are my own responsibility.

I would also like to thank my long-suffering wife, Lise, for her unlimited support and encouragement.

Maynooth, Christmas 2004

Contents

1	Introduction	1
1.1	Programs	1
1.2	Outline of the Book	2
1.3	How to Read this Book	3
2	Computers and Maple Names	5
2.1	Introduction	5
2.2	Maple Names	6
2.3	The Big Blackboard	7
2.4	Difficulties	8
2.4.1	Nothing	9
2.4.2	Quoted Strings	10
3	Procedures	13
3.1	Introduction	13
3.2	What is a Procedure?	14
3.3	The Kernel Procedures	16
3.3.1	Aside: Using Operators in Prefix Form	20
3.4	The Library Procedures	21
3.4.1	Help on the Library Packages	22
3.4.2	Loading a Package	22
3.4.3	Example: Linear Algebra	23
3.5	User-defined Procedures	26
3.5.1	Procedure Names	26
3.5.2	Volume of a Cone	27
3.5.3	Thickness of a Book	27
3.5.4	Too Many Arguments	29
3.6	Procedures as Modules	29
3.6.1	Table for an Oil Tank	29

3.6.2	Gibbs' Phenomenon	31
3.6.3	Searching for a Minimum	34
3.6.4	Stirling's Formula	34
3.7	Difficulties	36
3.7.1	Types	36
3.7.2	Confusing Return-values and Side-effects	36
3.7.3	The Effect of <code>f(x) = proc(x)</code>	37
3.7.4	The Invisibility of the Current State	39
3.7.5	Using Procedures to Hold Values via Individual Assignment	40
3.7.6	The Debugging Tools	41
3.7.7	Everything is a Procedure	41
3.8	Exercises	42
4	The Fat Controller	51
4.1	The Reception Desk	52
4.2	The Messenger Pool	53
4.3	The Parsing Team	53
4.4	The Fat Controller	56
4.5	The Boards	56
4.5.1	The Big Board	57
4.5.2	The Reserved Word Board	58
4.5.3	The Protected Symbol Board	58
4.5.4	The Kernel Names Board	58
4.6	The Kernel Stand and Crew	58
4.6.1	The Kernel Foreman	58
4.6.2	The Kernel Mechanics	58
4.6.3	The Kernel White-collar Group	59
4.7	The Specialist Library Crews	59
4.8	Support Team	60
4.9	The Work Ethic	60
5	Some Points on Program Design	61
5.1	How to design a Program	61
5.2	Good Programming Practice	63
5.3	The Maple module	63
6	Case Study: Angle between Two Vectors	65
6.1	The Task	66
6.2	The Strategy	66

6.3	Implementation	66
6.4	Interpretation	67
7	Selection: if	71
7.1	An Introduction to Conditionals	71
7.1.1	Boolean Expressions	72
7.1.2	Boolean Procedures	72
7.1.3	Boolean Operators	74
7.1.4	Layout	74
7.1.5	Simple <code>if-then</code> Examples	75
7.2	The <code>else</code> statement	76
7.2.1	<code>is_even</code>	76
7.2.2	The Characteristic Function of $[0, 1]$	78
7.3	The <code>elif</code> statement	79
7.3.1	Tax Calculation	80
7.3.2	Exam Result	80
7.3.3	Home-made Absolute Value	82
7.3.4	Computing a Median	82
7.3.5	Day of the Week	83
7.4	Tricky Bits	84
7.5	Exercises	85
8	Geography and Satellites	91
8.1	Conversions	92
8.1.1	Degrees to Radians:	92
8.1.2	Radians to Degrees:	92
8.1.3	Latitude and Longitude to Spherical Polars	93
8.1.4	Spherical Polars to Latitude and Longitude	94
8.1.5	Spherical Polar to Cartesian Coordinates	94
8.2	Operations on 3-D Vectors	95
8.2.1	Length of a Vector	95
8.2.2	Vector Difference $A - B$	95
8.2.3	Unit Vector from A to B	95
8.3	Direction and Distance from Place to Place	96
8.3.1	Distance to New York	96
8.3.2	Direction to Astra	96
8.3.3	Direction to New York	98
8.4	Exercise	98

9	Recursion	99
9.1	Recursive Procedures	99
9.1.1	Home-made Factorial	99
9.1.2	Triangular Numbers	100
9.1.3	Limitations of Recursion	100
9.1.4	Sexagesimal (Babylonian) numbers	100
9.1.5	Exercises	101
10	Iteration: for	103
10.1	for Loops	103
10.1.1	A Sequence of Squares	104
10.1.2	Sequence of Powers of x	105
10.2	for loops with a by clause	105
10.2.1	Using by	106
10.2.2	Writing a Table	106
10.2.3	Selected Squares	106
10.2.4	Loop running Backwards	107
10.3	for with Nested if	108
10.3.1	Counting some Primes	108
10.3.2	Finding the Maximum of a List	108
10.4	Case Study: Searching for an Optimal Position for a Gas Network Node	109
10.5	Case Study: Finding Formulas for π	110
10.6	Iteration over Lists or Sets	112
10.7	Adding up Series	112
10.8	Case Study: Simulating a Random Walk	114
10.9	Case Study: Fibonacci Numbers	116
10.10	Replacing Recursion with Iteration	120
10.11	Exercises	121
11	Iteration: while	127
11.1	Introduction	127
11.1.1	Example	127
11.1.2	Examining $n^2 + n + 17$	128
11.1.3	Warning: Infinite Loops	129
11.1.4	Cannot evaluate Boolean	130
11.1.5	Random Walk until some Event Occurs	131
11.1.6	Exercise on $n^2 + n + 41$	131
11.2	Loop Conditions	132
11.2.1	Searching for an Item in a List	132

11.3	Case Study: The Greatest Common Divisor	134
11.3.1	A Simpler gcd Algorithm	135
11.3.2	The <code>remember</code> Option	137
11.4	Combining <code>for</code> and <code>while</code>	137
11.4.1	Example: First Negative Number in a List	137
11.4.2	Example: First Composite in a List	138
11.4.3	Inverse of a Function	138
11.5	Teaching Project: Inventing Homework	139
11.6	Project on Cardinality	140
11.6.1	Bijections between \mathbf{N} and \mathbf{Z}	140
11.6.2	Cantor Diagonalization	141
11.7	2-D Newton-Raphson Project	142
11.8	Exercises	143
12	More about Procedures	147
12.1	Introduction	147
12.2	Type Checking	147
12.2.1	Example: First <code>a</code> in <code>L</code>	148
12.2.2	Specifying Argument Types	149
12.2.3	Example	149
12.2.4	Checking the Number of Arguments	149
12.2.5	Try-Catch	150
12.3	Local and Global Variables, and Scope	150
12.3.1	Home-made Factorial	151
12.3.2	Scope	152
12.3.3	Example: A Mistake	153
12.3.4	Illegal Use of Formal Parameter	154
12.4	List and Set Algorithms	156
12.4.1	Finding the Second-biggest Element	156
12.4.2	Splitting a List or Set	157
12.5	Procedures that return Procedures.	158
12.5.1	Picking a Random Number from an Interval	159
12.5.2	Inverse Functions	159
12.6	Project: The Prime Number Theorem	160
12.7	Difficulties	162
12.7.1	Evaluation	162
12.7.2	Problems with Index Variables	163
12.7.3	Bottom-testing loops	165
12.7.4	Replacing Recursion with iteration	166
12.8	Exercises	169

A	Simulation Case Studies	173
A.1	Poker	173
A.2	Fly Fishing	175
A.3	Exercise: Roulette	181
B	Graphics Projects	183
B.1	Plotting Problems	183
B.1.1	Problems with the <code>plot</code> Expression	183
B.1.2	Problems with the <code>plot</code> Range	184
B.2	Famous Curves and Surfaces	185
B.3	Case Study: Rose Windows	186
B.3.1	North Transept, Notre Dame de Chartres.	186
B.3.2	West End, Gun Chapel at Maynooth	191
B.3.3	Exercises	201
B.4	Tilings	201
B.4.1	A Nonagonal Prototile	202
B.4.2	Building Blocks	204
B.4.3	Parity, and Attachment Tools	208
B.4.4	Snail	218
C	Print Formatting	221
C.1	<code>printf</code>	221
C.2	Worksheet	222
C.3	Exercises	224

Chapter 1

Introduction

1.1 Programs

A program is the written expression of an *algorithm*, i.e. a systematic, logical procedure for carrying out some task. Programs are orders that are intended to be executed by a computer. They are written in a language that the computer can be made to understand. There are many such *programming languages*. In this course, you use the Maple language. When Maple is running on your computer, then the computer understands Maple instructions.

To say that the computer can understand Maple instructions just means that if an instruction is given in Maple, then the computer will do what it is supposed to do when given that instruction (or crash). It does not mean that the computer will do what the user had in mind when typing the instruction. The computer does not know what you have in mind, nor does it really understand any Mathematics. It doesn't understand English, either. Most especially, it doesn't understand English. It just does what it is told to do, when told in Maple.

So a computer on which Maple is running is a compliant slave, and Maple is the language in which you give it orders.

The main structure you will learn to use is the *procedure*, implemented using Maple's `proc` command. Programs are written using procedures. More accurately, a program is made up of modules, and each module is implemented as a procedure.

In a very simple program, the algorithm consists of just a *sequence* of steps. In more complicated programs, some steps will need to be repeated several times (*iterated*). It may also happen that there will be choices:

alternative steps may have to be *selected*, depending on the situation. The three elements of *sequence*, *selection* and *iteration* are the fundamental ingredients in the *flow of control* in any program.

In Maple, the language elements governing the flow of control are the **if** structure, the **for** structure, and the **while** structure.

1.2 Outline of the Book

After preliminaries on the nature of computers, and on Maple *names*, we start with an account of the idea of a procedure, and the Maple `proc` structure.

We offer some suggestions on how to visualise a Maple computation, and how to design a program. Chapter 6 presents the first of many case-studies, an analysis of the simple problem of calculating an angle. The purpose of this is to illustrate some aspects of program design, and the nature of programming. In Chapter 7 we learn how to use the **if**, control structure, which is used for selection. Chapter 8 has another case-study, and Chapter 9 introduces the use of recursive procedure calls¹. Chapters 10 and 11 present the other two control structures, **for** and **while**, which are used for iteration. In Chapter 12 we complete the discussion of procedure use. Appendices A and B have case-studies on simulation and graphics. The last appendix provides some pointers on the formatted-printing procedure, `printf`.

By the time you finish, you should be able to do some interesting things with Maple. However, there is a great deal more to Maple than we cover in this book. It is a very rich and complex tool. To get some idea of what can be done, have a look at Doron Zeilberger's amazing web-page about Shalosh B. Ekhad XIV's Geometry Textbook ("... all of a sudden, to my amazement, I chanced on a website whose *last update* was Sept. 30, 2050, ..."). On a more mundane level, there are many good textbooks about Maple, that may be used for private study, and have fuller description and detail than we give about Maple's features. This book is intended to be used in conjunction with a course, where the student has access to verbal advice and help from instructors and other students, and also has access to an interactive Maple environment, complete with its extensive help resources.

¹Experience shows that some students struggle to master the use of recursion. To cope with this, an instructor might decide to omit the topic, or to defer it to later in the course. In this case, Chapter 9 may be omitted, or read later. The sections in later chapters that involve recursion are clearly labelled. Our preference is to keep the concept in the course, because of the power and essential simplicity of the recursive method. We also prefer to introduce it early on, to allow time for the student to digest it.

1.3 How to Read this Book

Like most computer manuals, this is not intended to be read once through in the given order. There are occasional forward references, so we assume that the reader will read the material at least twice. Also, it is possible to skip around. One could start by skimming each chapter, to see what is there, and then gradually work into each one.

It is essential to have a working Maple environment to hand when working on the book. Given the extensive help documentation in Maple, we have not taken up space with material that can be read there, so the text has many references to Maple help.

When this book went to press, the current Maple version was Maple 9. Most of the contents will work fine with any version after Maple 6.

The book is designed for students with no previous programming experience. Those with such experience could skip a lot of the material, and concentrate on the exercises and projects. Students with experience of typed procedural languages such as Java should pay particular attention to the sections on “difficulties” and “tricky bits” scattered throughout the text.