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

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

Λ

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 encouragment.

Maynooth, Christmas 2004

Contents

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 calls[1]. 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.

[1]Experience 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 should pay particular attention to the sections on “difficulties” and “tricky bits” scattered throughout the text.