MACMILLAN MASTER SERIES

MASTERING C PROGRAMMING

W. ARTHUR CHAPMAN

Mastering

C Programming

W. Arthur Chapman

MACMILLAN

First edition 1991

Published by
MACMILLAN EDUCATION LTD
Houndmills, Basingstoke, Hampshire RG21 2XS
and London
Companies and representatives
throughout the world

Printed in Hong Kong

British Library Cataloguing in Publication Data
Chapman, Arthur
Mastering C programming.
1. Computer systems. Programming languages: C language
I. Title
005.113
ISBN 0–333–49842–9 Pbk
ISBN 0–333–49843–7 Pbk export

Preface

This book is intended as a first course in C programming. It is suitable for those new to programming as well as for those already familiar with another programming language. Access to a computer running C is assumed. With this condition the text is suitable for use in self-study, directed study through open or distance learning as well as via a more traditional approach as a class text. All the example programs and functions have been tested using Turbo C version 2.0. However, with very few exceptions, no changes should be necessary if other C compilers are used.

The main aim of the book is to introduce C and to provide the essentials of the language. The standard used throughout is the draft ANSI standard, and its counterpart the draft British Standard Specification (ISO/IEC DIS 9899), which is summarised in the second edition of the classic text for C *The C Programming Language* by Brian W. Kernighan and Dennis M. Ritchie, Englewood Cliffs, NJ: Prentice Hall 1988.

The text begins by looking at problem solving in fairly general terms before moving on to a first C program in Chapter 2. These first two chapters form an important introduction to the main text and are especially intended for anyone new to programming. Subsequent chapters develop the C language, its syntax and semantics. The material is designed to take the reader step by step from the basics (Ch. 3) through control structures (Chs. 4–6) and data structures (Chs. 8 and 10) to the more advanced topics of lists and list processing (Ch. 11). New elements of the C language are illustrated by numerous examples of program fragments, functions and complete programs.

Throughout the book a number of rather more substantial programs are developed to provide a context for the use of C in rather larger projects. As and when appropriate, these programs, their design and implementation are discussed and functions developed. Three main programs are dealt with in detail; they are a calculator, a line editor and a simple bridge tutor. The calculator is discussed in detail, and the program is developed, in Chapter 7. We introduce the line editor in Chapter 1 and develop various parts of it throughout the book. The bridge tutor is introduced in Chapter 2 and provides a simple program to simulate shuffling, dealing, counting

points and suggesting opening bids. For the most part no knowledge of bridge is necessary but a familiarity with cards and card games such as whist would be helpful. These latter two programs are listed in full, and their functions are discussed in detail, in Appendices B and C respectively.

Most chapters conclude with a summary which highlights the main points covered in the chapter and which serves to act as a revision aid to the reader. In addition, most chapters contain exercises which are designed to reinforce the topics covered and to develop the readers understanding of C. Some of these exercises refer to the larger programs and as such the answers can be found in the relevant program listings.

As you work through the material presented here you should develop a good understanding of C and C programming. If by the time you have completed your study of this text you have a desire to continue programming in C, wish to move on to more advanced aspects of the language, and have even more importantly found that C programming is both challenging and also fun, then the book will have achieved its purpose.

This book developed out of an idea suggested by my friend and colleague Noel Chidwick and I would like to thank him for that original idea and his encouragement throughout the project. (Not to mention the late nights and early mornings which seemed to form an inevitable part of life in recent months!) Thanks are also due to many other friends and colleagues at Telford College, Edinburgh and further afield who have helped and supported me in various ways. I would also like to extend my gratitude to students who attended various classes given by me in recent years. They willingly tried out many of the ideas which finally found their way into this book and provided much helpful stimulation. I am pleased to be able to extend my thanks to Jane Wightwick at Macmillan for her help and support during the lifetime of this project and her understanding when deadlines were missed. Finally, and most importantly, I would like to thank my wife Judy and our children Emma, Lucy and Donald who have put up with an, even more than usual, bad-tempered fifth member of the household! Without their forebearance and encouragement the task of writing would have been much harder.

June 1990 W. Arthur Chapman

Contents

List of Figures and Tables

FIGURES

TABLES

1 Beginning with problems

"Some problems are just too complicated for rational logical solutions. They admit of insights, not answers." J. B. Wiesner

1.1 PRELIMINARIES

In this chapter we will be mainly concerned with the important topic of problem solving. We will be looking at ways in which problems can be tackled and the most productive ways of obtaining solutions – we will be concentrating on problems which can be solved and for which "rational logical solutions" can be found. In the process of working through this chapter you will be introduced to some techniques which enable well structured programs to be developed. This includes the idea of top-down design, the use of stepwise refinement and the writing of algorithms. The fundamental control structures of procedural languages will be introduced and their relevance for C indicated. A method of writing algorithms using pseudocode will be developed and will be applied to some programming tasks which we will be discussing in greater depth in later chapters.

Developing a computer program, whether in C, or in any other language, is a matter of devising a solution to a problem; it involves clear and logical thinking and requires the writing of careful and effective code. Computer programming is a mixture of an art and a science. A computer program can include clever solutions to a problem using obscure elements of the language but if the final program is to be understood, or even used, by someone else then it must have a clear structure and good documentation.

We will be looking at both the art and the science of programming so that by the end of this chapter you will be able to develop an outline solution to most problems. In future chapters these techniques will be extended to enable you to write programs in C. Let us begin, though, by forgetting about the details of computer programming and look first of all at problem solving in more general terms.

Computer programming ...

involves devising solutions to problems;

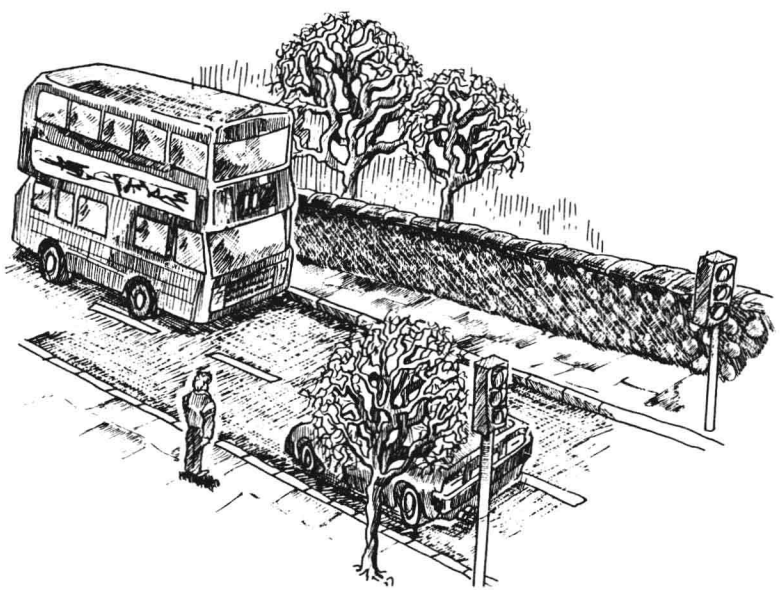
it requires clear and logical thought

1.2 PROBLEM SOLVING

The art of problem solving is difficult to define. However the task of problem solving, which is to find a solution to a particular problem, seems all too obvious. This appears easy enough until you start the process. Some problems are easy to solve, others are far more difficult. Problems come in all shapes and sizes. They cover such diverse tasks as: getting up in the morning (a problem to most people), preparing breakfast or making a cup of coffee, existing on unemployment benefit, achieving world peace, saving the tropical rain forests, or solving the all-embracing environmental problems.

We will concentrate on some more mundane tasks and will begin by taking a look at a reasonably simple everyday problem. You are on the pavement at the side of a busy road. You are in a hurry and you need to cross the road. A hundred metres away, in the opposite direction, is a set of traffic lights (see Figure 1.1). What do you do? Think about the possibilities, about what options are open to you (which ones are safest, which ones are quickest) before reading on.

Fig 1.1 ***Problem solving ... 'How do I cross the road?'***



Crossing the road

How did you approach this problem? Well, first of all there is no **right** answer; there are many possibilities, most of which have their good and bad points. Perhaps you decided to wait for a break in the traffic and then make a dash for it! This solution is not to be recommended, especially if you have young children or an elderly person with you. Alternatively you may have decided that the traffic was too heavy and so walking up to the traffic lights was the best option – you could afford to be a few minutes late rather than risk ending up in a hospital bed or worse. Again you may have decided that it was rather a silly problem and rather than try to solve it in advance you would wait until you next had to cross a road. Whilst this solution (putting it off) might be satisfactory in this case, it cannot be allowed in computer programming – problems need to be solved before they arise. However you may have decided that this problem was rather silly on the grounds that you were not given enough information. If you came up with this last point then give yourself a pat on the back.

One of the most important points which this seemingly simple problem should have highlighted is that often you are not given all of the necessary facts. For example:

- What day of the week is it?

  If it is a Sunday then possibly walking across the road would be the best option – observing the Green Cross Code of course.

- What time of day is it?

  The solution will obviously be different if it is the rush hour rather than 1.30 am.

- I neglected to tell you that there is a subway only a few metres away.

Even these few simple and obvious comments should help to underline the important point that a problem may not be well specified and that in deciding on a solution you may need to make some assumptions. If this is indeed the case then these assumptions must be made explicit from the outset. Discovering hidden assumptions, or making explicit assumptions which must be made are part of the task of understanding the problem. Another vital part of the process of understanding a problem involves drawing up a specification of the problem.

A second stage in problem solving is the all-important one of devising a solution. In computing this will generally mean devising an algorithm. We will be looking at one approach to this in the next section and will looking a little more closely at algorithms in 1.4. Once a solution has been devised it then has

to be implemented. In computer terms this will involve translating the algorithm into a computer program. Finally the plan is carried out and note taken of its successes (and failures). With a computer program this will mean running it and evaluating the accuracy of the results.

The stages in the problem solving process will generally be carried out in the order given above. However in practice the first two stages may be mixed up and a sufficiently detailed understanding of the problem may only be possible once the process of devising a solution has begun.

Problem solving involves ...

- ***understanding the problem***
- ***devising a solution***
- ***implementing the solution***
- ***evaluating the solution***

1.3 DEVISING A SOLUTION

There are various approaches to the task of devising a solution. One of the most common involves a 'top down' methodology. This means starting from the problem definition and working step by step towards a solution. At each step in the process the problem is broken up into smaller and smaller 'chunks'. This process of **stepwise refinement** is then continued until a set of easily-solved sub-problems has been arrived at.

Stepwise refinement ...

the process of breaking a problem into chunks which are then refined step by step

Charlie's desk

Charlie, a fresher of three weeks' standing, has been pondering the difficulties of working at a tiny table with less than stable legs and is out searching for a desk as a solution to 'all' his problems. Being of slender means (he is still awaiting

his grant) he drops into a shop littered with bric-à-brac and second-hand goods of all kinds.

While searching amongst the debris of bird cages, shooting sticks and battered suitcases, he discovers the answer to his prayers. There in the corner, in a dusty plastic bag, is a 'Student Desk', a self-assembly job at what he hopes is a knock-down price. Summoning the shop assistant he enquires the price. 'That's five pounds, sir' is the response to the vitally important question. So, dipping into a pocket of his tattered denims he pulls out five pound coins and, not believing his good fortune, walks out into the chilly October air with his newly acquired possession.

Arriving back at the flat he decides to celebrate his astounding good luck by having filtered coffee – there is just enough to make one last pot. Once the coffee is on he starts the process of unpacking his desk. He carefully lays out the pieces on the not very spacious floor and searches through the odds and ends for the instructions. At last, in a packet containing assorted screws he finds, somewhat tattered and torn, the crucial pieces of paper. He smooths them out and putting on his battered spectacles peruses the words of wisdom.

Charlie is devastated! With a crumpled, torn and incomplete set of instructions (Figure 1.2) how will he ever manage to build his desk? After pondering the problem for a few minutes he has a sudden flash of inspiration. 'Why not try stepwise refinement?' he says to himself. This wonderous method has only recently been introduced to him by his lecturer in programming techniques and now is the time to test out the theory on a real-life problem.

After sorting through the bits and pieces of the kit, checking the contents (luckily nothing appeared to be missing) and after an hour or so's work with the scraps of instructions he finally came up with what seemed like a usable set of instructions.

Charlie's instructions

1. Bookcase

Assemble the carcase using the $1\frac{1}{2}$" screws.
Fix the back to the carcase using six small nails.
Glue four dowels into the top of the bookcase.
Leave to set.

2. Cupboard

Glue four dowels into the top of the end panels.

Knock the drawer runners into the end panels.
Glue four dowels into the plinths (one into each end).
Fix the plinths between the end panels using the $1\frac{1}{2}$" screws.
Fix the back to the cupboard carcase using six nails.
Attach the front using $1\frac{1}{2}$" screws.

Cupboard Door

Screw the hinges to the door using the $\frac{1}{2}$" screws.
Screw the knob onto the door.
Attach the door with the hinges to the right hand panel using $\frac{1}{2}$"screws.

Drawer

Glue the drawer wrap at the joints and glue four dowels into the holes provided.
Glue the drawer front.
Assemble and leave until the glue sets (24 hours approx.).
When dry wipe over with a damp cloth to remove excess glue.
Fit handle.

3. Final Assembly

Place the top, upside down, on a clean, smooth surface.
Squeeze glue into the eight holes on the underside of the top.
Position the bookcase carcase and press firmly. (A slight tap with a mallet may be required to ensure that the dowels are firmly seated.)
Repeat for the cupboard.
Leave for approx. 24 hours to dry.

4. To finish

Wipe the entire desk with a damp cloth.
Cover all exposed screw heads with the screw covers provided.

With the help of his own instructions and after a few bouts of trial and error Charlie managed to complete the task and a day or two later was seen hard at work at his newly-acquired masterpiece.

The strategy which Charlie used to solve the problem of assembling the desk, and which the makers had also suggested, was that of stepwise refinement. The task was broken up into a number of jobs, each of which could be carried out separately. Once all the tasks had been completed the problem was solved and the desk finished.