

# Structured Programming into ANSI C


T-36

10

# Structured Programming into ANSI C

Chris Carter BSc, PhD, MIEE, MBCS, CEng

School of Computing and Management Sciences  
Sheffield City Polytechnic

Pitman 

PITMAN PUBLISHING  
128 Long Acre, London WC2E 9AN

A Division of Longman Group UK Limited

© C. Carter 1991

First published in Great Britain 1991

**British Library Cataloguing in Publication Data**

Carter, Chris

Structured programming into ANSI C.

1. Title

005.13

ISBN 0-273-03687-4

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording and/or otherwise without the prior written permission of the publishers or a licence permitting restricted copying in the United Kingdom issued by the Copyright Licensing Agency Ltd, 90 Tottenham Court Road, London W1P 9HE. This book may not be lent, resold, hired out or otherwise disposed of by way of trade in any form of binding or cover other than that in which it is published, without the prior consent of the publishers.

Reproduced and printed by photolithography  
in Great Britain by Biddles Ltd, Guildford.

# Preface

This book is suitable both for complete beginners and for those who are moving to C from another language such as Pascal. The book can be used as a stand-alone text, or to complement a lecture course.

Those who already know how to program may skim or skip Chapters 1 to 3, apart from Sections 3.1 and 3.2. If you are a beginner but would still prefer to get on with programming as quickly as possible you can do likewise. However, as you move from simple to more advanced programs, you will find that you need the information in the earlier Chapters, on how to design programs.

Some people think of C as a second rather than as a first language. In this book we concentrate on a sub-set of C that could almost as well be Pascal. The more 'techy' features of the language such as pre- and post-incrementation are left until towards the end of the book. Using this approach C proves to be no more difficult to teach, nor less educational, than Pascal. There are many good reasons for preferring C as a first language:

- It is an excellent general purpose language, which is in strong demand by employers and which is widely used and widely available.
- Whilst it is a structured language, structure is not imposed by the language, but by the programmer. This is helpful when using the language for more advanced applications.
- It is modular, self-consistent and concise.

The programs in this book were originally designed to be run on compilers which are ANSI C compatible. Should your compiler not yet be fully ANSI then slight modifications to the header files, the function declarations and the start of each function definition may need to be made. These minor modifications are detailed in Appendix 1 which contains a UNIX shell script to convert them automatically.

Should you need an (unprotected) copy on 3.5" disk of all of the programs (as detailed in the Program and Function Index) please quote the title of this book, your name and address and whether you require the IBM or MAC version and enclose £5 (world-wide) to cover costs. The address is Dr. Chris E. Carter, Sheffield City Polytechnic, CMS School, 100 Napier Street, Sheffield S11 8HD.

# Acknowledgements

The author would especially like to acknowledge the contribution of Mr. Bill McCausland, a retired school teacher who took a part-time degree at Sheffield Polytechnic at the age of 63. Bill has gone through the text and programs several times and has made many helpful suggestions. Almost at the other extreme of age range the author's 16 year old son Robin has also contributed ideas and suggestions including advice on computer game-play conventions. Others who have helped with program checking and proof reading include my friends Dave Denton and Phil Mouncey.

Thanks are also due to many students on courses throughout the Polytechnic ranging from the Advanced Certificate to the part-time MSc., who have had numerous exercises and drafts of parts of the book tried out on them. The technical staff of the Polytechnic, particularly John Leach, have also contributed much enthusiasm and technical support. The extensive computing facilities at Sheffield City Polytechnic have enabled the programs to be tested on a variety of systems.

John Cushion of Pitman has contributed many valuable suggestions from his wide experience of technical writing as well as consistent encouragement.

This book was produced using an Apple Macintosh computer, Superpaint and Word 4.

# Overture

On December 3rd. 1989 the Sunday Correspondent (an English Newspaper which has sadly since become defunct) ran a rather sensationalised story on the French nuclear reactor threat to the UK. The article questioned the inherent safety of the French reactors and pointed out that some of the reactors were only 60 miles from Southampton, a port on the south coast of England. After criticising the hardware the article went on to say that the reactor control systems were programmed in "a language which is notorious for allowing dangerous errors to slip in, say British experts". These British experts were also attributed as saying that "it is all too easy to write dangerous programs with C, yet difficult to spot the mistakes using safety analysis currently in use". The French, on the other hand, were said to be keen to give computers more control over nuclear reactors as a way of avoiding another Chernobyl. At the end of the article a French spokesman was attributed with the words "Yesterday we had a demonstration for visitors and everything worked fine". The article made me think of the following points with regard to computers.

- The safety or otherwise of a computer program has more to do with the design of the program than with the language employed.
- The language C is now very significant and is thought by some to be the most suitable language to use when the control of hardware and safety critical systems is involved.
- The English think that they know more about programming computers than the French. This is a curious view since the French computer industry is in at least as good a state as our own.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	What is Programming ?	1
1.2	What is C ?	2
1.3	ANSI C	2
1.4	The Compiler and Portability	3
1.5	The Significance of C	4
1.6	C++	5
1.7	Editors	6
1.8	The Keyboard	6
1.9	Error Messages	8
1.10	Backing up	8
1.11	The Operating System	8
1.12	Program Libraries	9
1.13	Programmer Accuracy	9
1.14	Summary	10
<b>2</b>	<b>Structured Programs</b>	<b>11</b>
2.1	Pseudo-code	11
2.2	A Note about the Algorithms	12
2.3	The Taps and Pipes Problem	13
2.4	Modified Goto	13
2.5	Structured Solution	15
2.6	Modified Structured Solution	16
2.7	Summary	17
2.8	Self-test	18
<b>3</b>	<b>Structured Data</b>	<b>19</b>
3.1	Memory and Variables	19
3.2	Variables	20
3.3	Arrays	21
3.4	Hand-testing Programs	22
3.5	A Letter Sorting Problem	23
3.6	A Specialised Solution	23
3.7	Six-letter Solution	25
3.8	A Loop Solution	27
3.9	Any Number of Letters	29
3.10	Finding the Earliest Letter	30
3.11	Summary	31
3.12	Self-test	32
<b>4</b>	<b>Programming into C</b>	<b>34</b>
4.1	What is a C Program ?	34
4.2	A First Program	36
4.3	Type	37

4.4	ASCII Conversion Program	40
4.5	Add Digits Program	41
4.6	Important Ideas	43
4.7	Temperature Conversion	44
4.8	Logical Variables	46
4.9	Operators	46
4.10	Unary Operators	49
4.11	Mixed Mode Arithmetic	50
4.12	Annotating Programs	50
4.13	Choosing Names	52
4.14	Summary	53
4.15	Program Ideas	54
<b>5</b>	<b>Compound Statements</b>	<b>55</b>
5.1	Introduction	55
5.2	If Then Else	56
5.3	Switch Case	58
5.4	While	60
5.5	For	61
5.6	Unofficial For Case	63
5.7	Break and Continue	64
5.8	Goto	65
5.9	Summary	65
5.10	Program Ideas	66
	<b>Syntax Diagrams</b>	<b>68</b>
<b>6</b>	<b>Writing Functions</b>	<b>75</b>
6.1	Using Functions	75
6.2	Function Parameters	76
6.3	Function Prototyping	77
6.4	Returned Values	78
6.5	Scope	80
6.6	Functions and Scope	81
6.7	Defensive Programming	82
6.8	Getting Any Character	86
6.9	Getting a Positive Integer	88
6.10	Coping with Keying Errors	91
6.11	Constants	91
6.12	Repeated Times Tables	94
6.13	Making a Grid	95
6.14	Summary	98
6.15	Program Ideas	99
<b>7</b>	<b>Using the Standard Library</b>	<b>100</b>
7.1	Borrowing a Function	100
7.2	The Library Stock	101
7.3	Casting	103
7.4	Modulus Operator and Primes	104



7.5	De Morgan's Theorem	106
7.6	Prime Number Programs	107
7.7	Sum of Primes	107
7.8	Printing with Formatting	108
7.9	List of Primes	110
7.10	Using Scanf()	110
7.11	Interest Rate Calculation	114
7.12	Getting Any Character	116
7.13	Using Macros	117
7.14	Unbuffered Input	121
7.15	Summary	122
7.16	Program Ideas	123
<b>8</b>	<b>Types</b>	<b>124</b>
8.1	Qualifiers and Modifiers	124
8.2	Modifiers	125
8.3	Mixed Precision Arithmetic	127
8.4	Type Conversion	128
8.5	Qualifiers	130
8.6	Enumerated Types	131
8.7	Static and Auto Variables	136
8.8	Implementation of Scope	136
8.9	Other Qualifiers	140
8.10	Summary	141
8.11	Program Ideas	141
<b>9</b>	<b>Arrays</b>	<b>144</b>
9.1	One-dimensional Arrays	144
9.2	Strings	145
9.3	Comparing and Copying Strings	146
9.4	The sizeof Operator	149
9.5	Passing Arrays to Functions	150
9.6	Selection Sorting	151
9.7	Bubble Sorting	154
9.8	Passing Strings to Functions	155
9.9	Reading a Character String	156
9.10	Temperature Conversion	159
9.11	Summary	164
9.12	Program Ideas	164
<b>10</b>	<b>Compound Types</b>	<b>166</b>
10.1	Two-dimensional Arrays	166
10.2	Abstract Data Types	171
10.3	Structures	174
10.4	Passing Arrays through Structures	177
10.5	Using Structures	178
10.6	Unions	180
10.7	Summary	182
10.8	Program Ideas	183

<b>11</b>	<b>Pointers</b>	<b>184</b>
11.1	An Analogy	184
11.2	Ordinary Variables	185
11.3	Ordinary Parameters	186
11.4	Pointer Variables	187
11.5	Pointers and Memory	188
11.6	Declaring Pointers	189
11.7	Ordinary and Pointer Variables	191
11.8	Pointers to Simple Variables	192
11.9	Pointers to Arrays	193
11.10	Getting an Answer	194
11.11	Sorting with Pointers	196
11.12	Pointers to Structures	198
11.13	Pointers to Functions	200
11.14	A More General Grid	201
11.15	Arrays of Pointers	205
11.16	Constant Pointers	206
11.17	Summary	207
11.18	Program Ideas	208
<b>12</b>	<b>Linear Data Structures</b>	<b>209</b>
12.1	Data Structures	209
12.2	Stacks	210
12.3	Stack Program	214
12.4	Queues	216
12.5	Circular Queues	217
12.6	Queue Implementation	219
12.7	Summary	225
12.8	Program Ideas	226
<b>13</b>	<b>Recursion and Trees</b>	<b>227</b>
13.1	Factorials	227
13.2	Towers of Hanoi	229
13.3	The Binary Tree	232
13.4	Fixed Binary Trees	234
13.5	Non-Fixed Binary Trees	238
13.6	Dynamic Memory with Arrays	240
13.7	Adding a Leaf to the Tree	246
13.8	Deleting a Node from a Tree	248
13.9	A Binary Tree Program	252
13.10	Summary	255
13.11	Program Ideas	256
<b>14</b>	<b>Dynamic Memory</b>	<b>257</b>
14.1	Dynamic Data Structures	257
14.2	Dynamic Storage	259
14.3	Dynamic Storage and Trees	262
14.4	Dynamic Memory in Practice	264

14.5	Summary	270
14.6	Program Ideas	270
<b>15</b>	<b>Shorthand Files and Bits</b>	<b>271</b>
15.1	Pre- and Post-Incrementation	271
15.2	Manipulation at Bit Level	273
15.3	Other Shorthand	275
15.4	Command Line Arguments	276
15.5	Conditional Compilation	277
15.6	Program Units	278
15.7	Data Files	280
15.8	Summary	282
15.9	Program Ideas	282
<b>CS0:</b>	<b>The Case Studies</b>	<b>284</b>
<b>CS1:</b>	<b>Code Maker</b>	<b>285</b>
<b>CS2:</b>	<b>Print-a-Graph</b>	<b>293</b>
<b>CS3:</b>	<b>Tabs-in-Tabs-out</b>	<b>300</b>
<b>CS4:</b>	<b>Risky Dice Game</b>	<b>310</b>
<b>CS5:</b>	<b>Tic Tac Toe</b>	<b>317</b>
<b>CS6:</b>	<b>TV Plan</b>	<b>330</b>
<b>A1</b>	<b>The Target Systems</b>	<b>339</b>
A1.1	Apollo and Sun Workstations	340
A1.2	Using a Hard Disk (personal computers)	341
A1.3	Obtaining Hard Copy	341
A1.4	Turbo C	342
A1.5	Microsoft C	343
A1.6	THINK C	343
<b>A2</b>	<b>Dealing with Errors</b>	<b>345</b>
A2.1	Finding Syntax Errors	345
A2.2	Run-Time Errors	346
A2.3	Debuggers	347
A2.4	Program Testing	349
<b>Program &amp; Function Index</b>		<b>351</b>
	Complete List of Programs	351
	Our Library (Functions Invented in this Book)	352
	Non-i/o Calls by Case Studies	352
	Non-i/o Calls by Other Programs	353
<b>General Index</b>		<b>354</b>

# 1 Introduction

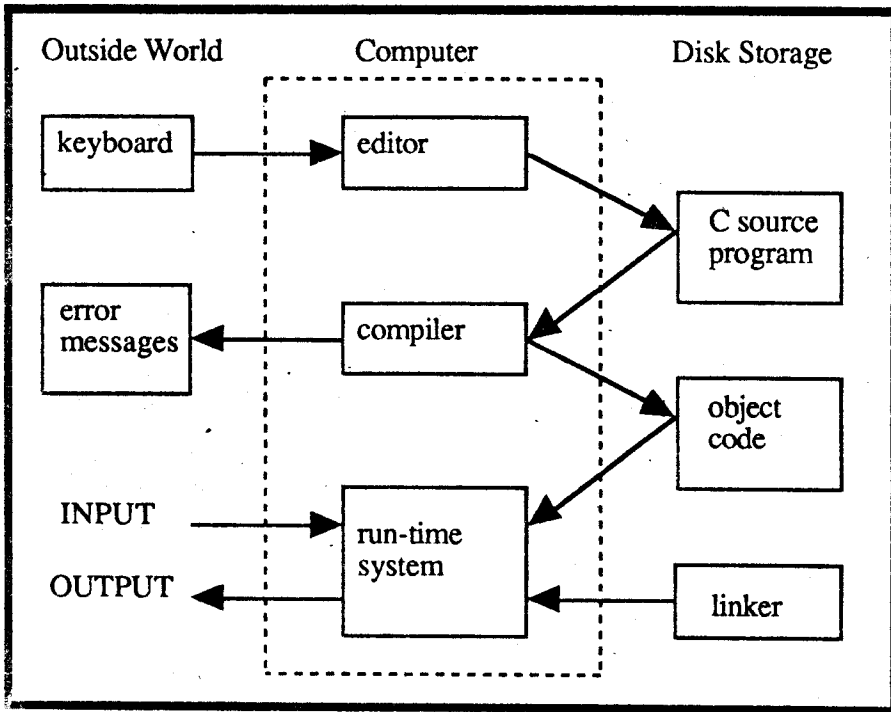


Fig. 1.1 The programming environment

## 1.1 WHAT IS PROGRAMMING ?

In order to type in and get a computer program working, written in the language C there are a number of steps that have to be gone through. These steps are illustrated in Fig. 1.1, and will be explained later in the Chapter. First it is necessary to place programming itself into a context.

Those new to computing often assume that programming is all to do with understanding a programming language and so it is, if by programming you mean translating a detailed design into something that a computer can process. This activity is called **coding**. Because computer code often looks unintelligible to the uninitiated it is tempting to assume that once you have mastered this activity most of your problems will be over. Whilst the value of having a thorough grasp of the programming language you are using should not be underestimated, most commercial problems arise because of poorly defined specifications and poor designs rather than incorrect coding.

Because programming is only one part of developing a satisfactory end product the term **software** is used to imply the entire process. Software design means correctly specifying a problem, designing the **algorithms**, or solution method to solve it, checking that the algorithms will work, coding them for the chosen language, testing the resulting program and conscientiously documenting every stage. One should think of programming into a language rather than in a language. This book is on C but the language is less important than the overall approach.

## 1.2 WHAT IS C ?

All computers can be programmed at two different levels. They can be programmed using a so-called **high-level** language or they can be programmed in a **low-level** language. Normally there is only one low-level programming language for any given computer and each kind of computer has a different low-level language.

Computers are normally programmed in a high-level language because:

- (1) High-level languages are easier for people to write in and the resulting programs are easier to follow;
- (2) Programs written in a high-level language are much shorter, since each high-level instruction is equivalent to about ten low-level instructions;
- (3) The same, or almost the same, program written in a high-level language will run on a range of different computers.

C is just one of a number of **procedural high-level languages**. Other examples are Pascal, ADA, Modula 2 and Basic.

## 1.3 ANSI C

The language C was originally designed and implemented by Dennis Richie at Bell Labs. in the early 70's. His main objective in designing the language was as a tool for systems programming: for writing such things as Compilers and Operating Systems. C therefore originated from the concept that a programming language is a tool for driving a machine (the computer) in the most efficient manner possible. Other languages treat the computer as an abstraction and are designed to protect the programmer from needing to understand how the machine works or even that the machine exists at all. It could be argued that ultimately that is their failing: sooner or later the programmer will not only want to dictate what problem the computer is to solve but also how the solution will be carried out.

For many years C remained a relatively esoteric language. The main reason that this changed is the rise of the UNIX operating system; the first operating system for mainframe computers that was not tied to a supplier of a particular computer. In 1983 ANSI<sup>†</sup> appointed a committee to provide a modern definition of C. The final version of the standard was not published till 1990. The significance of the new standard is mainly twofold. Firstly it includes most of the features that are expected in a modern language; this makes it possible to whole-heartedly commend it as a language for teaching programming. Secondly, with an agreed and detailed specification by which they can be judged, compilers can be checked independently to ensure that they conform to the standard.

In recent months there has been a flurry of activity as suppliers have brought their compilers up to ANSI or near-ANSI compatibility. The programs in this book should work on all ANSI compilers. They may not work on pre-ANSI compilers without minor modification. It would actually be possible to write programs that will run on any C compiler, but the code that we would need to use would be obscure and difficult to read. The new version of the language is still ingeniously simple in concept and satisfying to use. As a universal, go-anywhere do-anything language it can't be beaten and is unlikely to be beaten for many years to come.

#### 1.4 THE COMPILER AND PORTABILITY

A program written in C must be processed by another program called a **compiler** before it can be used. The compiler translates each group of high-level instructions into a sequence of low-level instructions. The compiler itself has to be written in a low-level language designed for the host machine and therefore you cannot use the same compiler program on different types of computer. However, in theory at least, you can run the same C program on a range of computers provided that you re-compile the program for each computer in turn. The compilation process takes a short while; less than a minute for even the longest program in this book.

Provided you avoid using facilities that are specific to one particular kind of computer there is no reason why programs written in the C language should not be **portable** (ie. be capable of working on a range of different computers). The programs in this book have been run on the following compiler/computer combinations:

- ✓ IBM PC running Turbo-C;

---

<sup>†</sup> ANSI is a trademark of the American National Standards Institute.

- ✓ IBM PC running Microsoft C;
- ✓ Apple Macintosh running THINK C;
- ✓ Sun workstation running the standard UNIX C compiler;
- ✓ Apollo workstation running the standard UNIX C compiler<sup>†</sup>.

An advantage of writing programs that can run on all five systems is that you can develop a program on an IBM PC, say, and then run it on a workstation. This can be helpful if the more expensive computer is not frequently available. It is usually possible to transfer the program electronically from one computer to another, so that you do not even have to type it twice.

In the case of the Sun and Apollo workstations there exist compilers that conform closely to the ANSI standard but they are optional products. This means that as in the case of the present author the ANSI version may not be available to you. Should this be the case then the standard C compiler (which does not conform to ANSI) will have to be used and that many of the programs will need to be modified. Appendix 1 contains a shell script (a program written in the UNIX language) which does this automatically for you. You just have to type it in and issue the command `sh convert myprg.c` or whatever.

## 1.5 THE SIGNIFICANCE OF C

The language C is rapidly becoming the most important of the procedural languages and is strongly in demand by students and employers alike. The reasons for this are:

- Its portability and the portability of the operating system UNIX with which it is often associated. The idea is that once having developed a program written in C that runs under UNIX, it will work on many other computers, including computers that have yet to be designed.
- Its versatility and suitability for controlling hardware and software almost as effectively as a program written in a low-level language.

---

<sup>†</sup> Turbo C is a trademark of Borland International Inc.

Microsoft is a trademark of Microsoft Corporation.

Apple Macintosh is a trademark of Apple Computer Inc.

THINK C is a trademark of Symantec Corporation.

UNIX is an operating system and a trademark of AT&T Bell Laboratories

Sun is a trademark of Sun Microsystems Inc.

Apollo is a trademark of Apollo Computer Corporation now Hewlett Packard.

- ☛ Its modularity and the consequent savings in program writing time and compilation time.
- ☛ Its use as a gateway to X-Windows<sup>†</sup> and other major software associated with user interfaces and communications.

Alas, the strengths of C can also be seen as weaknesses. It is a sophisticated language with a large number of features including an optional shorthand notation, which has earned it the reputation of being unreadable. The shorthand notation does not have to be used and is largely avoided in this book but explained in a later Chapter. The main benefit of the shorthand is that it can lead to faster execution of programs.

Attempting to describe all the features of the language in detail from the outset can overwhelm the reader. It also seems unnecessary since many only have a specialist usage. In this book I have stuck to describing only the most important features of the language. There are many more advanced books on C. You may find that you will need to consult not just one but several in order to satisfy all your eventual requirements.

After reading this book and working with the programming examples, you should find that you are confident enough to write quite advanced programs of your own, which use all the main features of the language. You should also find that you have mastered the essential design ideas which are needed for successful programming into any language.

## 1.6 C++

You may have seen reference to the language C++, which is currently much in vogue. C++ is really an extension to C and enables the ideas of Object Orientated Programming to be used more easily than in C. Pre-requisites to learning C++ are:

- (1) you are already conversant with C;
- (2) (even more importantly) you are familiar with the philosophy of object orientated programming.

Therefore, should you wish to learn C++, you should first read this book and then get a good book on C++, which explains about the Object Orientated philosophy before explaining about C++. At the time of writing, it would be unrealistic to expect portability from C++ programs, and it is arguable

---

<sup>†</sup> X-Windows is a file server graphics system based on C functions and is designed to make graphics independent of the computer, screen resolution and keyboard being used. X-Windows is a trademark of MIT (Massachusetts Institute of Technology).



whether C++ would form a suitable pathway to Object Orientated Programming, anyway. The concepts described in this book would form a good basis for moving to object orientated programming.

## 1.7 EDITORS

Before you can enter any of the programs in this book and run them on the computer that you are using, you will need to use an editor to actually type them in. Often you will have a choice of editors. For example on workstations you can use the editor supplied by the workstation vendor or a UNIX editor such as vi. In the case of personal computers, an editor is usually supplied along with the programming language or you can use your favourite editor or word processor. Should you choose the latter you should save your work as plain text.

Editors and their use are outside the scope of this book but it is worth mentioning that the global search-and-replace facilities that are available on almost all editors are very useful for changing the names of variables, and that copy-and-paste is useful for speeding up the entry of large numbers of `printf()` statements and ensuring that function names and parameter lists are the same in all places.

When you come to save your program on to disk you have to give it a name. Usually the name is an alphabetical word of say six to ten characters and is followed by `.c`. The `.c` is called a file extension and is partly there in order for you to distinguish between different types of **file** (the name used to describe a single set of information which is saved on to disk such as a computer program, a set of data or a piece of text such as this book). For most compilers the `.c` extension is obligatory or the compiler will not process your program at all. Of course, you can always re-name any file so that it has the right extension.

## 1.8 THE KEYBOARD

One of the first impediments to programming is finding your way round a keyboard. Programmers who can touch-type are at an advantage. By using one of the typing tutor programs on the market the skill can be mastered within three weeks by conscientiously performing two fifteen-minute exercises each day.

The alphabetic characters, space bar and digits are all standard among computers. There are a number of non-alphanumeric characters that are used in C. Since it can take a while for the beginner to find these keys, some of which vary from computer to computer, their positions are given in Fig. 1.2. There are two single quote characters on most keyboards and the diagram will