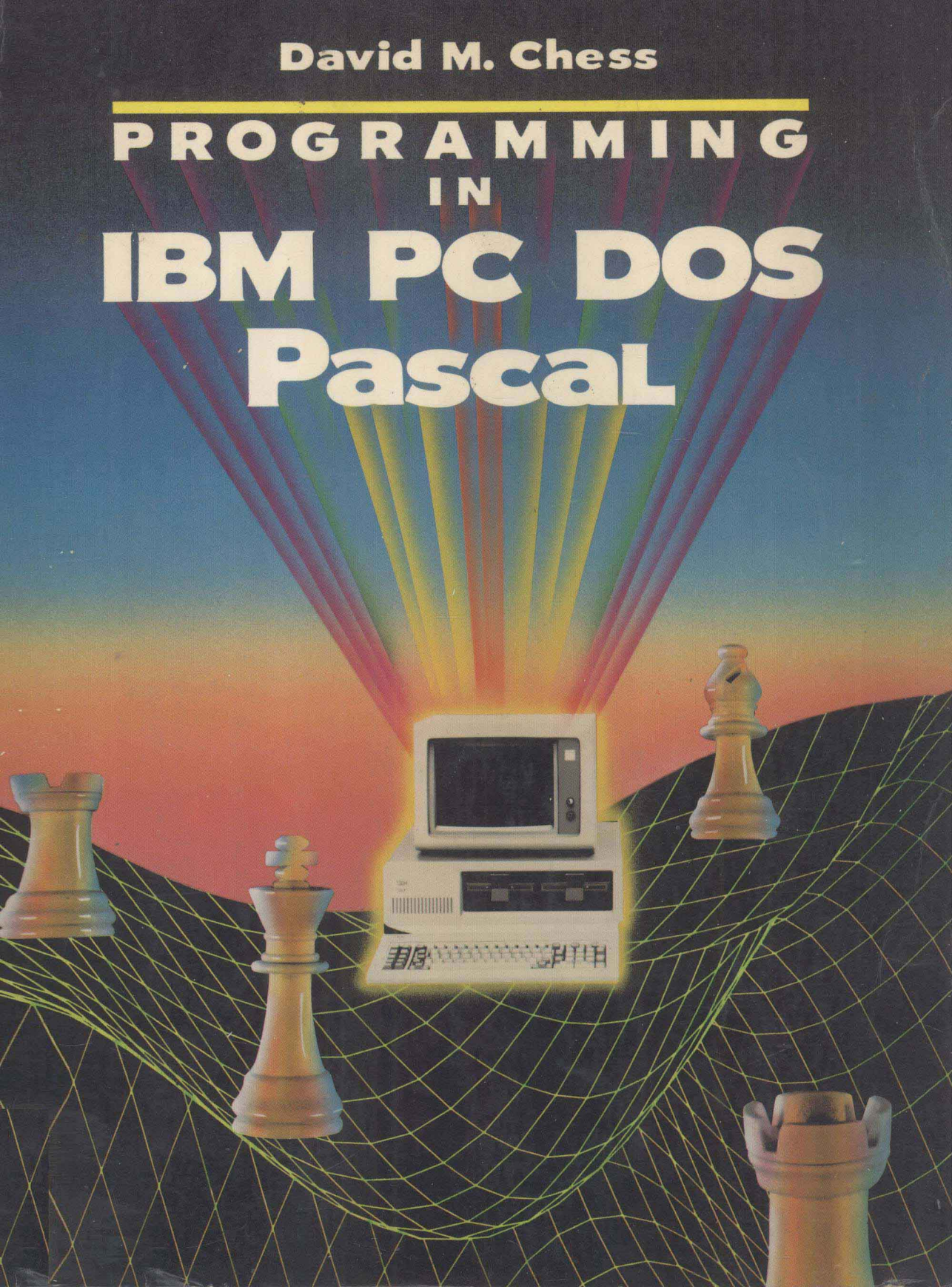
David M. Chess

PROGRAMMING IN IBM PC DOS Pascal



Programming in IBM PC DOS Pascal

David M. Chess
IBM Thomas J. Watson Research Center

Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632

Library of Congress Catalog Card Number 84-42860

Editorial/production supervision: **Karen Skrable Fortgang**
Cover design: **Photo Plus Art**
Manufacturing buyer: **Gordon Osbourne**

TO MARGARET

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-730292-4 01

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA PTY. LIMITED, *Sydney*
EDITORA PRENTICE-HALL DO BRASIL, LTDA., *Rio de Janeiro*
PRENTICE-HALL CANADA INC., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL OF SOUTHEAST ASIA PTE. LTD., *Singapore*
WHITEHALL BOOKS LIMITED, *Wellington, New Zealand*

CONTENTS

1. Introduction **1**
1.1 Organization of the Book 2

2. Background **4**
2.1 DOS, BIOS, and Pascal 4
2.2 About Compilation and Linking 5
2.3 A Quick Path 7
2.4 Compilation and Link Files 9
2.5 Onward 10

Part One -- Essential Pascal **11**

3. Basic Pascal I **11**
3.1 Pascal Programming 11
3.2 Identifiers 12
3.3 The Program Heading 12
3.4 Variable and Type Declarations 13
3.5 The Simple Types 14
3.6 Structured Types 18
3.7 Reference Types 26
3.8 Constants 29
3.9 Value Sections 30
3.10 Attributes of Variables 30
3.11 Notes on Declaration Sections 32
3.12 Subroutine Declarations 33
3.13 Conclusion 37

4. Basic Pascal II **38**
4.1 Statements 38
4.2 Assignment and Operators 40
4.3 Decision Statements 42
4.4 Control Flow 44
4.5 Miscellaneous Statements 48
4.6 Extended Statements 50

5. Built-in Power **53**
5.1 Procedures and Functions 53
5.2 Predeclared Mathematical 54

5.3 External Mathematical ... 58
5.4 String Related ... 63
5.5 Predeclared File Related ... 66
5.6 Low-Level File Related ... 68
5.7 Miscellaneous Predeclared ... 71
5.8 Miscellaneous External ... 76
5.9 Other Routines ... 78
5.10 Conclusion ... 80

Part Two -- The Details ... 81

6. Pascal Data Files ... 81
6.1 Pascal Files and DOS Files ... 81
6.2 Types of Files ... 83
6.3 Devices as Files ... 86
6.4 Operations on Files ... 87
6.5 More Features ... 94

7. Screen Handling ... 99
7.1 The PC's Displays ... 100
7.2 Techniques and Tradeoffs ... 100
7.3 Pascal Write ... 100
7.4 Using ANSI.SYS under DOS 2.0 ... 102
7.5 Using the BIOS ... 108
7.6 Writing Directly to Screen Memory ... 117
7.7 Multiple Displays ... 125
7.8 Some Speed Considerations ... 126
7.9 Summary ... 127

8. Keyboard I/O ... 128
8.1 Pascal Read ... 128
8.2 Using the Keyboard as a File ... 130
8.3 Scan Codes and Keystrokes ... 132
8.4 The DOS Call ... 134
8.5 Shift States and the BIOS Byte ... 135
8.6 Using ANSI.SYS to Redefine the Keyboard ... 138
8.7 Conclusion ... 139

9. DOS Services ... 140
9.1 DOSXQQ - Accessing DOS Functions ... 141
9.2 Useful DOS Functions ... 142
9.3 Other DOS Functions ... 152

10. Separate Compilation 153
10.1 Include Files 154
10.2 Modules 155
10.3 Units 158
10.4 Portability 161
10.5 Conclusion 162

11. The Assembler Interface 163
11.1 Assembler Tradeoffs 164
11.2 Calling Conventions 166
11.3 Parameter Passing 166
11.4 Function Values 170
11.5 Port I/O 172
11.6 Accessing Common Data 173
11.7 Cautions 176
11.8 Conclusion 177

Part Three -- PASCAL for Real Applications 178

12. Programming in Pascal 178
12.1 Elements of Style 179
12.2 Modularity 185
12.3 Top-Down Design 190

13. A Programming Project 194
13.1 Function 194
13.2 The Top Level 195
13.3 The Next Level 197
13.4 The Depths 200
13.5 Notes 203
13.6 Conclusion 207

A. Collected Facts and Reserved Words 208
A.1 Some Useful Pascal Facts Gathered in One Place 208
A.2 PC Pascal Reserved Words 209

B. PC Pascal and the ISO Standard 211
B.1 Notation 211
B.2 Types and Constants 212
B.3 Compilands 213
B.4 Expressions and Operators 213
B.5 Statements 214

B.6 Parameters and Attributes 214
B.7 File Handling 215
B.8 Other Functions 215
B.9 Metalanguage 215
B.10 Other Extensions 216

C. Keyboard Codes 217
C.1 Alternate-shift Alphabetic Keys 217
C.2 Function Keys 218
C.3 Cursor Control 218
C.4 Other Codes 218

D. Bibliography 220
D.1 Pascal 220
D.2 Programming, Algorithms, and Style 221

Index 223

1. Introduction

This book is a description of, and a learning guide for, the advanced features offered by Pascal under the IBM Personal Computer Disk Operating System (DOS).[1] It does *not* apply to the UCSD P-System Pascal (also sold by IBM). Versions 1.0 and 2.0 of PC Pascal are covered; differences, where they exist, are noted in the text.

The reader of this book should either know Pascal (or some comparable block-structured language, like PL/1), or have an introductory work at hand (the Bibliography gives several suggestions). A working knowledge of DOS and the PC is also recommended. The book is intended primarily for PC users looking for a good, powerful language, and for Pascal users new to the PC looking for ways to get to the power of the machine from Pascal.

The Pascal language was designed by Niklaus Wirth in the late 1960's as a simple, easy to understand language for teaching computer science. Since then, it has been implemented on a vast number of computers, from 8-bit micros to large industrial mainframes. The original language has been extended to include more powerful facilities for input/output, character and string manipulation and systems programming, and has become one of the most popular general-purpose programming languages. The International Standards Organization, which establishes standard definitions for common languages, has drafted a document defining a set of functions that are essential to the language. Most current Pascal implementations, including IBM PC Pascal, start with this basic set, and extend it in several directions.

---

1 IBM PC Pascal version 1 is IBM order number 6024010; version 2 is 6024128.

This book briefly describes the essentials of the Pascal language, with emphasis on the extensions PC Pascal provides. It also goes into detail on interfacing Pascal programs to the rest of the PC environment; the handling of disk files, access to the PC's displays, the keyboard, and various DOS and BIOS functions. Use of system utilities such as the library manager is not covered; these are described in the system manuals. The linker is described in only enough detail to illustrate a "fast path" through Pascal compilation.

1.1 Organization of the Book

The Background chapter discusses the PC as a microcomputer, and gives some general facts about the machine, its basic I/O system, the DOS operating system, and how they all relate to Pascal.

Part One covers the basic concepts of Pascal as PC Pascal implements them, describing in detail those features that are extensions of the Pascal standard. The reader who does not know Pascal, but does know another block-structured language, can treat this section as a quick Pascal tutorial.

Part Two consists of several chapters, each devoted to a different aspect of PC Pascal. As I will be saying several times in this book, standard Pascal is a "sparse" language. Each chapter of Part Two shows how PC Pascal offers some function that is not specified in the standard. External routines and linkage of separately compiled subroutines, access to DOS disk files in various modes, powerful screen and keyboard handling, direct access to some DOS function calls, and the interface to assembler language routines, are all covered. By the end of Part Two, the reader should be eager to apply these functions to a real problem, and to see them working together in a genuine application.

Part Three shows how Pascal can be used for real programming. In the rapidly expanding small computer market, Pascal has had a good deal of bad press: it has been called a toy language, good for teaching programming, but not for much else. This section of the book will discuss how Pascal's structure makes top-down design possible, and how top-down design can make it easy to construct and test full-blown application and systems programs. To give some teeth to the assertions, the last chapter of the book will illustrate the process of writing a real program; a simple file-browsing program that can be written in about two hours. I will also give suggestions as to how it might be expanded into a genuine full-function text editor.

The appendices include a compendium of useful Pascal facts, a list of reserved words, and a summary of the ways that PC Pascal extends the ISO standard.

As well as the code for the browser, sample programs and program fragments are given throughout the book; the importance of examples in teaching, particularly in teaching programming, cannot be stressed too strongly. The difference

between being told how to do something and seeing it done can be the difference between obscurity and understanding.

All the examples and hints given, as well as illustrating the point in question, stress the importance of structured coding and top-down design: techniques that Pascal was designed around, and that the Pascal language structures make natural. These principles should help the reader in learning and using other structured languages and in writing cleaner and more reliable programs in any language.

2. Background

This first chapter outlines the relationship between Pascal, Pascal programs, and the rest of the PC environment. The experienced PC user, eager to begin using the power of Pascal immediately, may want to proceed directly to the next section, and the description of the language. Readers not as familiar with the PC and its operating system are encouraged to read this chapter, and even the PC expert may want to return to it eventually, during an idle moment or a long compilation.

2.1 DOS, BIOS, and Pascal

When a Pascal program runs on any modern computer, it is not running by itself. There are other programs in the machine at the same time, keeping track of the time of day, accepting input from the keyboard and holding it until the program asks for it, carrying out requests from the program to write data onto the screen, and generally making the program's (and the programmer's) task easier. When the program completes, the machine does not come to a halt; there is another program in the machine ready to accept control, and ask the user "what next, boss?" These other programs are called, among other things, the "operating system," the "I/O system," and the "background."

The PC's Disk Operating System, usually called DOS, is the program that controls all the operations of the PC. You run a Pascal program by telling DOS the name of the program you want to run. DOS determines where in memory to load the program, reads it in from a disk or diskette, makes any necessary adjustments to allow the program to run where it has been loaded, and passes control to it. When

the program needs to communicate with the user through the keyboard or the display screen, it can call a routine in DOS to do it. When the program ends, it returns control to DOS, which displays its prompt to indicate to the user that it is ready for the next task.

Most of the operation of DOS is not visible to the Pascal programmer. Whenever a Pascal program uses the standard output routine `write`, for instance, to display data on the screen, Pascal calls DOS automatically; the programmer needn't worry about the details. The same applies to the function `read`; the programmer may use it to get data from the keyboard or from a diskette, without worrying about the details of keyboard buffers, single or double sided diskettes, or file allocation. Because Pascal knows to call DOS, and DOS keeps track of these things, the Pascal programmer can make use of the PC's hardware without knowing anything about the hardware-specific details.

Even DOS is not on its own in the machine. Underlying DOS is the I/O system, called the BIOS (for "Basic Input/Output System"). BIOS handles the details of input and output; when DOS needs to read data from a certain place on a diskette, it calls BIOS. BIOS determines just what commands to send to the diskette to accomplish the reading, deals with the timing and other hardware requirements of the device, and eventually returns the data to DOS. In similar ways, DOS calls BIOS to write data to the display, to read data from the keyboard, and to interface with the rest of the environment. When you press a key on the keyboard, it is BIOS that gets control, stores the data that you enter away in memory, and then returns control of the machine to whatever was running when you interrupted things by pressing a key.

Specific calls to DOS and BIOS will be mentioned in their places later on in the book. As we will see, there are times when it is useful to be able to call DOS or BIOS directly, or even to circumvent them entirely, to make a program faster or more powerful. The fact that, for instance, every character output to the display in a `write` statement causes a call to DOS and multiple calls to BIOS, means that, for applications where speed is vital, it is good to know faster paths. In general, however, it is simplest to write standard Pascal statements, let Pascal call DOS, and let DOS call BIOS, in the usual way.

2.2 About Compilation and Linking

The process of generating an executable program (in a file with extension EXE) from a Pascal source program (usually with extension PAS) is described in some detail in the Pascal manual. These sections contain a few words about what each phase of the compilation does, along with a description of the quickest way to get from source to executable. The various compiler and linker options will not be described in detail, because they are adequately covered in the respective manuals.

The four steps involved in generating an executable file are:

1. The creation of the source file. This is generally done with an editor of some kind. The EDLIN program that comes with DOS is a good line-oriented editor; many more powerful full-screen editors are available, such as the IBM Personal Editor or the IBM Professional Editor. Word processors that put special control characters into the files they create should *not* be used, since Pascal will not understand the control characters, and will report them as errors. A term often used to describe source files is "flat ASCII." If your editor can create flat ASCII files, consisting of lines of characters separated by carriage returns and line feeds, it can most likely produce acceptable Pascal source files.

2. The production of "intermediate code" from the source file. This is done by the PAS1 program on one of your Pascal diskettes. PAS1 analyzes your source program for errors, and produces output (in files `PASIBF.BIN` and `PASIBF.SYM`) that is ready for the next step. If PAS1 reports errors in your source, return to step 1 and correct them. The intermediate code files themselves are not very interesting, and are useful only in that they serve as input to the next step.

3. The production of object code from the intermediate code. This is done by the PAS2 program on another of your Pascal diskettes. PAS2 reads the intermediate files produced by PAS1, and writes an object file (with extension OBJ). The data in an object file is almost in executable form; most of the intermediate data has been converted into data that the PC can execute directly.

4. The production of an executable file from one or more object files and libraries. This is done by the LINK program that comes with both DOS and Pascal. LINK takes one or more object files, and optionally one or more libraries (extension LIB), and combines them to produce a file that DOS can load and execute. Object files may contain references to routines or data that are not actually contained in the file; for instance, if your Pascal program calls a built-in routine to compute some trigonometric function, the object file will contain a call to that routine. But the routine itself is in the Pascal library (file `PASCAL.LIB`), and not in the object file. LINK's job is to find the places in the object file that these external references occur, and to fill in the correct address, once the object file has been combined with all other object files and library members that it uses. The result of a successful LINK is a file with the extension EXE. Files with this extension may be executed simply by typing their names at the DOS prompt.

In addition to these steps, Pascal version 2.0 provides a PAS3 program that may be used to generate a pseudo-code file (see "Compilation and Link Files" later in this chapter). In version 1, this function is accomplished by PAS2.

The programs PAS1, PAS2, and LINK, as well as the editor that you use to create the Pascal program source in the first place, are all simply programs that run under DOS. When PAS1 needs to read in your source file, it calls DOS to get access to the file, and DOS calls BIOS to read the actual data from the diskette. When it needs to write to the display to report an error, it calls DOS, which in turn calls BIOS. In fact, a good portion of the programs PAS1 and PAS2 are actually written in Pascal! So as you can see, there is nothing special or privileged about the compiler or the linker; they all read one or more files, perform some computations based on the contents of the files, and produce one or more new files as output.

2.3 A Quick Path

The various compiler and linker options can often be useful, and the reader is urged to read the section in the Pascal manual that describes them. For most compilations, however, only a very few of the possibilities are actually used, and the simplest forms of the commands will do. In these cases, there is a simple and easy to remember "quick path" through the process.

This section will illustrate the quick path by way of an example. Assume that you have just created a file called `PROG1.PAS`, containing a small Pascal program. The simplest steps from here to execution are:

- Run PAS1 to produce the intermediate code. To do this, put a diskette containing PAS1 into your PC's "A" drive, put the diskette containing your program into the "B" drive, and execute the commands "`b:`" and "`a:pas1 prog1;`". This has the effect of making the "B" drive the default drive, which is where Pascal will put the intermediate files, and then running the PAS1 program on the source file. The semicolon following the second command indicates to Pascal that no special options are required for this run. This is what the screen might look like after executing these commands (the characters "`A>`" and "`B>`" in the following examples are the standard DOS prompts):

```
A>b:

B>a:pas1 prog1;
```

  PAS1 will run for awhile, notifying you of any errors found by displaying the incorrect lines and error messages on the screen, and then return control to DOS. If all went well, the last lines it will type will read "`Pass One  No Errors Detected..`" (If there were any errors in the program, they should be corrected, and PAS1 should be run again.)

  If you were to look at the contents of the "B" diskette at this point, you would see files called "`PASIBF.SYM`" and "`PASIBF.BIN`".

- Run PAS2 to produce the object code. To do this, put a diskette with PAS2 into the "A" drive, and enter the command "a:pas2" (no parameters are needed after the command, because PAS2 always reads the files PASIBF.SYM and PASIBF.BIN). Entering the command would look like

```
B>a:pas2
```

When PAS2 completes, it will issue the message

```
Pass Two   No Errors Detected
```

and return control to DOS. PAS2 erases the PASIBF files created by PAS1, and creates an object file; in this case, the object file is "PROG1.OBJ".

- Produce the executable file. To do this, put a diskette containing the LINK program and the Pascal library (in file "PASCAL.LIB") into the "A" drive. The command you will issue now depends on what version of LINK you are using. If you are using LINK version 1.0, enter the command "a:link", and respond to the first question with the name of the program, followed by a semicolon. This might look like:

```
B>a:link
IBM Personal Computer Linker
...
Object files [.OBJ]: prog1;
...
```

If you are using a later version of the linker (1.1 or later), issue the command "a:link prog1;". This might look like:

```
B>a:link prog1;
```

In both cases, the semicolon tells the linker that no special options are required. When the linker completes, it will return control to DOS, and the file "PROG1.EXE" will be on the "B" diskette. The linker does not erase object files, so "PROG1.OBJ" will still be there, also.

To summarize the commands in the quick path:

1. `b:`
2. `a:pas1 prog1;`
3. `a:pas2`
4. `a:link prog1;` (for linker version 1.1 or later).

After compiling, you may want to issue "a:" to restore the default drive to "A", if that is the way you usually operate. The EXE file produced by the compilation may be copied to any diskette or fixed disk, and executed by typing its name. The object file is no longer of any use. Later, in the chapter on Separate Compilation, we will see what OBJ files are good for, and why the OBJ file for a useful subroutine might be a good thing to keep around.

2.4 Compilation and Link Files

During the compilation and link steps, various files are produced. Some of them are always produced, and some require special options to the commands. This section is a quick summary of these files, where they come from, and what they are for.

- The Pascal source file - This is the file that contains the Pascal program in human-readable form. It is usually produced by a programmer, using an editor. It may have any name, but for the sake of keeping one's files in order, and taking advantage of compiler defaults, it should have the extension PAS.
- The intermediate files - These files are produced by PAS1 for use by PAS2, and are really of no interest in themselves. Their names are always "PASIBF.SYM" and "PASIBF.BIN".
- The object file - This is produced by PAS2 for use by the linker. It will have an extension of OBJ.
- The executable file - This is produced by the linker, and is the file that DOS can actually load and execute. Its extension is EXE.
- The listing file - This file is optionally produced by PAS1; if you use the quick path described above, it will not be produced. The Pascal manual describes how to invoke PAS1 in order to produce a listing. The listing file has an extension of LST, and contains a formatted listing of your source program, along with line numbers, titles, the time and date, descriptions of all parameters used by your subroutines, and other sometimes-useful information. The format of the file is described in detail in the manual.
- The pseudo-code file - This file is optionally produced by PAS2, at the request of PAS1 (in Pascal version 1.0), or by PAS3 (in Pascal 2.0). Again, see the Pascal manual for instructions on how to produce the pseudo-code file. This file has an extension of COD, and contains an approximation of the machine instructions that correspond to your program. The instructions in the pseudo-code file are in a sort of assembler language; as a last resort, an assembly-language programmer may refer to a pseudo-code file to determine where a Pascal program is going wrong.

2.5 Onward

The rest of this book will concern itself with PC Pascal as a language, rather than as a compiler. The LINK command and object files will be mentioned again in the chapters on Separate Compilation and the Assembler Interface, but the major concern of the book is how a Pascal program can be written to use the power of the PC and its I/O devices most effectively.