Designing Screen Interfaces in C

James L. Pinson

Network Specialist
University Computing and Network Services
University of Georgia

YOURDON PRESS
Prentice Hall Building
Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging-in-Publication Data

Pinson, James L.
Designing screen interfaces in C / James Pinson.
p. cm. -- (Yourdon Press computing series)
Includes bibliographical references and index.
ISBN 0-13-201583-8
1. C (Computer program language) 2. Computer interfaces.
3. Information display systems. I. Title. II. Series.
QA76.73.C15P583 1991
005.265--dc20

Editorial/production supervision
and interior design: *Harriet Tellem*
Cover design: *Lundgren Graphics*
Manufacturing buyers: *Kelly Behr/Susan Brunke*

The publisher offers discounts on this book when ordered
in bulk quantities. For more information, write:

Special Sales/College Marketing
Prentice-Hall, Inc.
College Technical and Reference Division
Englewood Cliffs, New Jersey 07632

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

ISBN 0-13-201583-8

Prentice-Hall International (UK) Limitd, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

Preface

This book is designed to aid C programmers in the creation of functional, intuitive screen interfaces.

We will cover in depth the construction of a screen and window function library, and use it to build menu systems similar to those found in popular commercial software packages.

Programs will be presented that will allow you to produce:

- A window-based screen interface.
- Moving light bar menus (as in Lotus 1-2-3).
- Multilevel light bar menus.
- Pop-up menus.
- Dialogue boxes.
- Pull-down menus.
- Field editors.
- Data input screens.
- List selection windows.
- Directory functions (for file selection).
- Context-specific help screens.
- Help screen editors.

In the process of building these systems, we cover concepts and techniques for:

- Performing BIOS interrupts.
- High-speed direct writing to the screen buffer.
- Avoiding snow on CGA displays.
- Producing "instant" screen updates with virtual screens.
- Virtual window creation and manipulation.
- Assuring compatibility with multitasking environments such as DESQview.
- User friendly menu design.
- Data input and verification techniques.
- Obtaining directory information for DOS.
- Help screen creation and display.

GOALS

The purpose of this book is to provide the reader with:

1. Programs that can be used as they are, or adapted for the individual programmer's needs.
2. The concepts behind each program.
3. Explanations of programming techniques used to build these programs.

PORTABILITY

Throughout this book we will strive for portability within the MS-DOS/IBM-DOS world.

For the user of our programs this means that the programs should work right the first time they are run. The programs require no special drivers such as ANSI.SYS, no alteration of config.sys files, and no need to buy special windowing operating systems or DOS shells. The software will work successfully with all popular video cards (CGA, MDA, Hercules, EGA, and VGA). Software based on this approach will have the widest possible audience, applicable to virtually every IBM PC and clone made.

For the programmer, portability means having a choice of compilers. We will avoid using compiler-specific graphic/window libraries. *Everything we need we will build.* The code created is memory-model independent; that is, all programs will compile under all Turbo C and QuickC memory models.

The programs are written entirely in C. No assembler is required. The commented source code may be customized to fit programmers' needs.

WHAT YOU WILL NEED TO USE THIS BOOK

1. A C compiler. Borland's Turbo C compiler was used to produce the examples in this book. The source code is also totally compatible with Microsoft's QuickC.
2. An IBM PC or compatible computer with enough memory to run the compiler.
3. IBM- or MS-DOS.
4. Experience programming with C. Advanced knowledge is not necessary.

WHAT OTHERS WILL NEED TO RUN YOUR SOFTWARE

1. An IBM PC or compatible.
2. 64K of available RAM. More may be needed if your applications are large.
3. IBM or MS DOS.

THE BOOK LAYOUT

This book is divided into seven chapters.

1. The first chapter is a review of video adapters, memory layout and access, and the use of pointers and BIOS interrupts.
2. The second chapter deals with the creation of a window-based screen function library. Topics covered include creating, removing, and writing to windows; creation of a flexible screen writing routine; and screen buffer "paging" techniques. We also discuss virtual screens and DESQview compatibility.
3. In Chapter 3 we will discuss menu systems. We will build Lotus 1-2-3 style menu bars, multilevel menu bars, pop-up menus, dialogue boxes, and pull-down menus. Of particular importance is the section on pop-up menus; here we establish the data structures, calling conventions, and menu design philosophies that will be used in all our menus.
4. Chapter 4 deals with data entry. We will build a field editor and will use it to construct a dBase style input screen, complete with color-coded fields and data verification.
5. In Chapter 5 we will build a list-select function that will allow the user to select from a virtually unlimited number of options, using point-and-shoot and speed search techniques.
6. In Chapter 6 we will create a function for selecting files from a directory, using the list-select function we created earlier.

7. In Chapter 7 we will build a help screen designer and will show how to add context-specific help to our programs.

Each major topic is subdivided into four categories:

- **Goals:** We discuss the general goals and features we wish to implement in our code. This discussion describes the overall "look and feel" of the product.
- **Application:** This is the practical part of the book. We discuss the individual functions and how they would be used within a program. Sample programs are presented as illustrations. A few tricks for getting the most out of the functions are described.
- **Techniques:** Within this section we discuss the design considerations and techniques used to produce the code. We look at possible programming approaches, pitfalls, and solutions. We place emphasis on aspects of the code that would not be intuitively obvious from examining the source code.
- **Source Code:** The code is highly commented. Every effort has been made to produce readable, useful code and comments. Meaningful variable names are used when possible (e.g., top = 1 instead of t = 1).

This modular approach should allow you to concentrate on the aspects of the book you find most useful. If you are interested in having a set of screen and menu tools, then you will probably prefer the Applications section. If you are interested in finding out how the code works, see the sections on Techniques and Source Code.

OBTAINING SOURCE CODE

Please see Appendix A for instructions for obtaining the source code from this book.

ACKNOWLEDGMENTS

I wish to thank Ed Yourdon for encouraging me to start this project. Constuctive critiques by two anonymous reviewers greatly improved the text, and John Hopkins and Steve Spencer helped test the code. I greatly appreciate the efforts of Paul Becker, Harriet Tellem, Noreen Regina, Robyn Goodale, and all the other helpful professionals at Prentice Hall.

Special thanks go to my wife Chantal, for all her encouragement, suggestions, editing, and proofreading. This book would never have been written without her.

Contents

Chapter 1

Accessing the Display Adapter

In this chapter we will review:

- The types of display adapters.
- The video buffer memory map.
- The use of pointers for accessing video memory.
- Default pointer types for compiler memory models.
- BIOS interrupts.
- The ANSI console driver.

In the first section we will discuss the display adapter, giving particular attention to the video buffer and the techniques used to access it.

DISPLAY ADAPTERS

The wide variety of monitors and text/graphic cards currently available for the IBM PC falls into two major groups, monochrome and color.

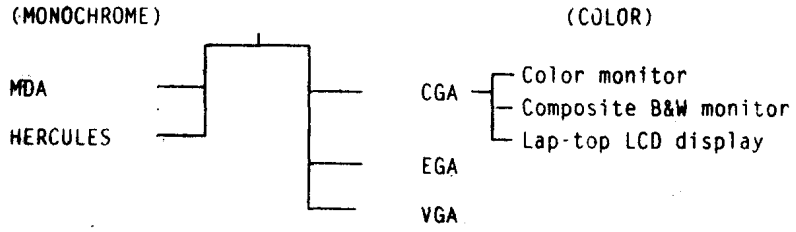

Figure 1–1 Evolution of major display adapters

An evolutionary tree would look somewhat like that in Figure 1–1. The major display adapters are described as follows:

- **MDA** = Monochrome Display Adapter. This card was intended primarily for business applications. It displays very well-defined text characters and is not capable of displaying graphics. The text is displayed in an 80 × 25 format (80 columns with 25 rows). This card requires the use of a monochrome monitor.
- **Hercules** = This card was designed to provide compatibility with the MDA, but is also capable of displaying proprietary high-resolution graphics. This card also requires a monochrome monitor.
- **CGA** = Color Graphics Adapter. This card is capable of displaying both text and graphics in color. The text display is limited to 80 × 25 (maximum).
- **EGA** = Enhanced Graphics Adapter. This card is compatible with the CGA but has increased text and graphics resolution. It supports text modes greater than 80 × 25.
- **VGA** = Video Graphics Adapter. Compatible with the EGA, but with still more text and graphic modes.
- **LCD** = Liquid Crystal Display. Used on laptop computers, this card is usually equivalent to a CGA adapter.

The newer, more advanced cards support the functions and modes found in the earlier ones. For example, VGA cards can run programs written for the CGA, and the Hercules card emulates the MGA. If we write our software so that it is compatible with the CGA and MDA, we gain compatibility with the newer adapters.

COMPOSITE DISPLAYS

Note that the CGA works with two display monitors: color and black-and-white (B&W). The black-and-white composite display tries to show color as shades of grey, with only limited success.

Color characters displayed on composite monitors are often unreadable. The LCD found on many laptops can be grouped with the composite displays since most LCD monitors use grey levels instead of colors.

Users of such B&W displays often use the DOS "mode" command to disable color, allowing only normal or intense text, which is more legible. Entering "mode bw80" at the DOS command line sets the mode to B&W with 80 columns.

Well-written software should check the mode via a BIOS call to ascertain the current mode the user has selected. Based on that mode, the software can select the proper text attributes (e.g., color or not) for that monitor. Unfortunately, if an application writes directly to the video buffer, it bypasses the BIOS and is not affected by the "mode" command, and therefore may write text with color attributes even though it is not appropriate for the user's monitor.

Although software may be enabled to detect what type of display adapter is present, it cannot "know" whether the type of monitor attached to the adapter is color or composite. The software can only ascertain whether the B&W mode has been set.

Later on, we will explore techniques and code for ascertaining the type of display adapter and the currently selected display mode. This information will be essential for producing a legible display.

THE VIDEO BUFFER

The IBM PC uses a section of memory as a video buffer. CGA and CGA compatibles use a buffer beginning at memory segment b800. Monochrome cards (MGA, Hercules and others) use a buffer beginning at b000. The first memory location contains the first character shown on the display, and the next location contains the attribute for that character. This arrangement is known as a *memory mapped display*, and any change made to the video buffer is immediately visible on the video monitor.

If you printed "hello" in the upper left corner of a CGA display, the memory map would appear as in Figure 1–2. The entire display on an 80-column, 25-line display would consist of 2000 (80 × 25) characters plus 2000 attributes. Using a column, row (x, y) screen-based numbering system, the upper left corner of such a display would be 1, 1 and the bottom right corner would be 25, 80. Knowledge of the video memory map will be essential later on as we develop our screen writing techniques.

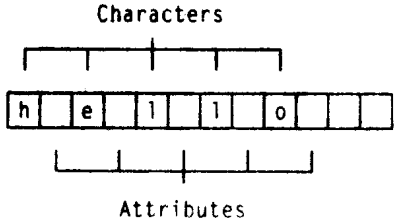

Figure 1–2 Memory map

TEXT ATTRIBUTES

Text attributes vary according to the type of video card in use. Attributes which may be used on monochrome systems include normal, intense, reverse, and underline. Our programs use the header file mylib.h which defines these attributes for us.

```
#define UNDERLINE 1 /* ATTRIBUTES FOR MONOCHROME CARDS
*/
#define NORMAL     7
#define HI_INTEN   15
#define REVERSE    112
```

On a color adapter the attribute byte can be used to set the foreground/background colors as well as the foreground intensity and blinking characteristics. The color attribute is mapped as in Figure 1–3.

The three primary additive colors—which are red, blue, and green—may be combined to create the so-called "subtractive" colors. For example, red and blue may be added to create magenta; red and green to create yellow. The colors magenta, cyan, and yellow can likewise be combined (subtracted, in this case) to form the primary colors. Cyan and yellow, for example, produce green. The "subtractive" color system is used in photographic enlargers. The relationship is shown in Figure 1–4 in the color wheel.

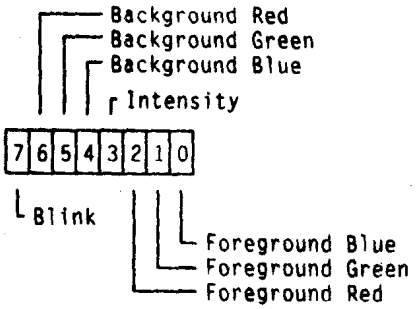

Figure 1–3 Bit map of color attributes

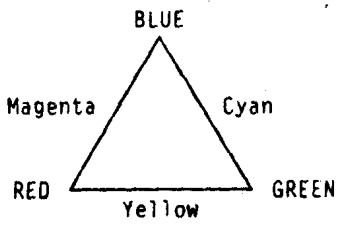

Figure 1–4 The color wheel

The color wheel is only a guide. The actual colors produced depend on the combination of red, blue, green, and intensity. For example, red and green produce brown if the intensity is not set. With intensity set ON, the result is yellow. Table 1–1 shows a bit map naming the colors that result when the primary color bits are turned on (intensity is set to OFF). To simplify the use of color attributes, the following definition is found in mydef.h:

```
#define BLACK     0   /* THESE ARE FOR COLOR CARDS */
#define BLUE      1
#define GREEN     2
#define CYAN      3
#define RED       4
#define MAGENTA   5
#define BROWN     6
#define WHITE     7
#define YELLOW   14   /* intensity set on */
```

The following function-like macro sets the foreground/background colors:

```
#define set_color(foreground,background)\

(((background)<<4) | (foreground))
```

Notice how the macro shifts the background 4 bits to the left, then combines it (bitwise 'or') with the foreground.

For example, using this macro, the variable **attribute** can be set to a foreground color of BLUE and a background color of BLACK with the call:

```
attribute=set_color(BLUE,BLACK);
```

The intensity is set HIGH for any attribute when its fourth bit is set ON (set to a 1). We may force this bit to be set ON for any attribute by use of the macro set_intense( ) which performs a bitwise 'or' with the decimal number 8 (which equals 00001000 in binary). This forces the bit to become a 1.

```
#define set_intense(attribute) ((attribute) |8)
```

TABLE 1-1. BIT MAP OF PRINCIPAL COLORS

R	G	B	Color	Attribute (decimal)
0	0	0	Black	0
0	0	1	Blue	1
0	1	0	Green	2
0	1	1	Cyan	3
1	0	0	Red	4
1	0	1	Magenta	5
1	1	0	Brown	6
1	1	1	White	7